The purpose of this publication is to describe the facilities provided by the IBM 1800 Time-Sharing Executive (TSX) System, and to explain the basic concepts and techniques underlying their use. It is intended as a reference and guide for customer systems personnel in the implementation of the TSX system.

The manual is divided into four sections. The first section serves as an overall introduction to the TSX system. The second and third sections describe the three main executive programs and discuss some of the important design considerations that bear on the use of standard TSX components. The final section provides selections of programming techniques covering a wide spectrum of TSX usage.

The general approach taken is to explain each concept as it is encountered, and, where possible, elucidate that concept by means of an example. Numerous sample problems are included to acquaint the programmer with recommended techniques of TSX programming. A detailed TSX Sample System is specially provided as a tutorial on all aspects of TSX design, usage and implementation.
PREFACE

This publication describes the facilities provided by the IBM 1800 Time-Sharing Executive System, and discusses the concepts and techniques underlying their use. It is intended as a reference and guide for customer systems personnel in the implementation of the TSX system.

The manual is written in four progressive sections where information in one section is sometimes necessarily related to information in another section. These comprise:

- Overview of the IBM 1800 Time-Sharing Executive System
- Functions of Executive Programs
- System Design Considerations
- Programming Techniques

The approach taken is to explain each concept as it is encountered. In some instances, a subject concept is necessarily included in a section prior to its definition later on in that section or a subsequent section. Sample problems are scattered throughout the text as illustrative examples designed to clarify concepts discussed and to familiarize the user with recommended techniques. They should not be construed as models.

The first section gives a rapid survey of the TSX system. It defines the executive system, its modes of operation and system requirements; discusses some of the basic TSX system concepts employed; and describes the various components of the system, and their inter-relationships to the total system.

The second section describes the three main executive programs (TASK, the System Director, and the Nonprocess Monitor) in terms of their functions and capabilities. Numerous examples are included as demonstration of sound programming practice and technique. Subjects discussed embrace: Program Scheduling, Handling of Interrupts, Use of Interval Timers, Use of Time-Sharing, Error Alert Control and Procedures, and Nonprocess Monitor Usage.

The third section discusses some of the important design considerations bearing on the use of standard TSX system components such as the System Loader, IBM Nonprocess System, Temporary Assembled Skeleton (TASK), and the System Director. Subjects discussed include: Assignment of Interrupt Levels and Restrictions, Level Work Areas, Disk System Configuration, and the System Skeleton.

The final section incorporates selections of programming techniques covering a wide spectrum of TSX uses. The purpose of this section is to aid the programmer, acquaint him with recommended techniques of TSX programming, and to help him to build on the fundamentals discussed in earlier sections of this manual. A detailed TSX Sample System at the end of the section touches on every facet of TSX design, use and implementation.

For details of TSX system generation procedures, System Loader assignment cards, TASK and System Director equate cards, and all Nonprocess Monitor control cards, the user is referred to IBM 1800 Time-Sharing Executive System, Operating Procedures, Order No. GC26-3754.

To derive maximum benefit from "Concepts and Techniques", the user should have a working knowledge of the following TSX support publications:

- IBM 1800 Data Acquisition and Control System, Functional Characteristics, Order No. GA26-5918
- IBM 1800 Assembler Language, Order No. GC26-5882
- IBM 1130/1800 Basic FORTRAN IV Language, Order No. GC26-3715
- IBM 1800 Operating Procedures, Order No. GC26-5753
- IBM 1130/1800 Plotter Subroutines, Order No. GC26-3755
- IBM 1800 Time-Sharing Executive System Subroutine Library, Order No. GC26-3723

Second Edition (June 1970)

This is a major revision of, and obsoletes, GC26-3703-0 and Technical Newsletters N33-8020, N33-8032, N33-8039, N33-8063, GN33-8074. Changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to version 3, modification 8, of IBM 1800 Time-Sharing Executive System and to all subsequent modifications until otherwise indicated in new editions or Technical Newsletters. Changes are periodically made to the specifications herein before using this publication in connection with the operation of IBM systems, consult the latest SRL Newsletter, Order No. GN26-1800, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader’s comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Department D78, Monterey and Cottle Roads, San Jose, California 95114.

© International Business Machines Corporation 1967, 1970
OVERVIEW OF THE IBM 1800 TIME-SHARING EXECUTIVE SYSTEM

Introduction ............................................. 1
Minimum System Requirements ......................... 1
Modes of Operation ..................................... 3
System Concept .......................................... 3
Role of the Skeleton Executive .......................... 3
Time-Sharing .............................................. 4
Versatility in System Configuration ..................... 5
Concept of Core Load ..................................... 5
Local Subprograms ........................................ 6
Common Areas ............................................ 6
Multi-Level Programming .................................. 7
System Components ....................................... 7
Control Programs ......................................... 8
Processing Programs ....................................... 9

FUNCTIONS OF EXECUTIVE PROGRAMS ............... 12
Temporary Assembled Skeleton (TASK) ............... 12
The System Director .................................... 13
Program Scheduling ....................................... 14
Handling of Interrupts .................................... 27
Use of Interval Timers ................................... 42
Use of the Operations Monitor ......................... 59
Error Alert Control ....................................... 59
The Nonprocess Monitor ................................. 71
Nonprocess Supervisor (SUP) ........................... 72
Disk Utility Program (DUP) .............................. 74
FORTRAN Compiler ....................................... 74
Assembler .................................................. 74
Simulator Program ........................................ 75
Summary of Nonprocess Monitor Control Cards ...... 77
Examples of Nonprocess Monitor Usage ............... 79

SYSTEM DESIGN CONSIDERATIONS ................. 115
Temporary Assembled Skeleton (TASK) ............... 117
Task Equate Cards ........................................ 117
Buffering of Messages to Disk ........................... 119
Calculating Task Core Size ............................. 120
The IBM Nonprocess System ............................. 121
System Loader Operation ................................ 123
Function of the *Assignment Cards ................. 126
The *DEDIT Control Card ............................... 127
Summary of Assignment Card Restrictions .......... 128
Sector Break Records for Absolute Programs ....... 128
System Director .......................................... 130
Size of System Director ................................ 131
Definition of Functions Required ..................... 133
Allocation of Internal and External Interrupt Levels 134
Number of Call Count Subroutines Required by User 147
Disk System Configuration .............................. 147
Disk Organization ........................................ 148

The DEFINE CONFIG Operation ..................... 151
Disk Cartridge Initialization .......................... 157
Summary of Disk Storage Requirements and Assignment Restrictions 159
System Skeleton .......................................... 161
Constitution of the System Skeleton ................... 161
Skeleton Core Size ....................................... 164
Calculating Skeleton Core Size ....................... 172
Use of *INCLD Control Cards ......................... 173
Summary of the Skeleton Build Process ............... 173

PROGRAMMING TECHNIQUES ......................... 181
Writing Assembler Language Subroutines ............. 181
LIBF Subroutines ....................................... 182
Input/Output Subroutines ................................ 182
Programming Subroutines Using Reentrant Coding .... 185
Need for Reentrant Coding ............................. 185
Concept of Level Work Areas ........................... 185
Mechanism for Reentrant Control ..................... 185
Masking Out the Interrupts ............................. 186
Programming Notes ...................................... 187
Writing User-Programs for Execution Under the TASK 187
Absolute Loader ........................................ 187
Program/Data Format ................................... 187
Absolute Loader Operation ............................. 188

Basic Concepts of Data Acquisition and Process Control
Systems (DACS) .......................................... 191
Introduction ............................................. 191
Data Acquisition Systems ................................ 193
Operator Guide/Supervisory Control ................... 194
Direct Digital Control ................................... 195
TSX Sample System ..................................... 195
System Design ............................................ 196
Periodic Program Scheduler ............................ 197
Sample System Error Design ............................ 197
Closed Loop Control .................................... 199
Operator Guide Control .................................. 200
System Design for Optimum Time-Sharing ............ 200
Process Operator's Console ............................. 200
System Documentation ................................... 203
Description of Sample System Flowchart ............ 203
Coding Techniques ....................................... 203
System Generation ....................................... 206
On-Line Output from the Sample System ............ 208

APPENDIX A. TSX SYSTEM COMPOSITION AND CAPABILITIES ................................. 291
APPENDIX B. SUMMARY OF TSX CALL STATEMENTS ........................................ 292
APPENDIX C. ASSEMBLER LANGUAGE TSX CALLS ........................................ 294
APPENDIX D. CONTENTS OF THE FIXED AREA OF CORE ................................. 297

INDEX ..................................................... 300
ILLUSTRATIONS

Figures

1. IBM 1800 Time-Sharing Executive System .......................... 2
2. A TSX On-Line System -- Illustrating the Skeleton Executive .................. 4
3. Four Types of Core Loads Commonly Used in TSX .................. 6
4. TASK Organization ...................................................... 8
5. Correspondence between TASK and the System Skeleton-Program or Event Sequence .................. 15
6. Illustrating Time Dependency ........................................... 16
7. Use of the CALL QIFON Statement ..................................... 22
8. Use of the CALL VIAQ Statement ....................................... 25
9. Initial Core Load ......................................................... 26
10. Mainline Core Load ALPHA ............................................ 26
11. Count Routine ............................................................ 27
12. Exit from MIC After an Interrupt has been Serviced ............ 36
13. Priority Interrupt Level Structure and Assignment ............... 38
14. Illustrating the Implementation of LOCALS ....................... 40
15. Initial Core Load. .......................................................... 42
16. Mainline Core Load ALPHA ............................................ 43
17. Count Routine PEROD .................................................... 47
18. Sequence of Characteristics of Process Interrupt Statements .................................................. 49
19. Illustrating a Method of Segmenting Mainlines Based on Scheduling Requirements .................. 54
20. Use of the CALL INTEX, CALL DPART, and RETURN Statements .................................................. 56
21. Action of MIC During an Interrupt ................................... 57
22. Action of MIC During an Interrupt ................................... 59
23. Exit from MIC After an Interrupt has been Serviced ............ 60
24. Timer Locations in Core Storage ....................................... 61
25. Subroutine A for Example 5 -- Queuing an Analog Scan Program .................................................. 63
26. General Problem Logic Flow -- Example 2 ........................... 65
27. Action of EAC when an Error Occurs .................................. 67
28. Effects of Restart and Reload on Timers and Interrupts .......... 70
29. The Nonprocess Monitor ................................................ 72
30. Illustrating Nonprocess Monitor Action During Time-Sharing .................................................. 74
31. Assemble/Compile and Execute a Nonprocess Core Load ........ 75
32. Assemble and Execute a Nonprocess Program from the Temporary Area ........................................ 77
33. Compile and Store a Nonprocess Program in the Relocatable Program Area (User Area) on Disk ........... 79
34. Compile and Execute a Nonprocess Program from the Core Load Area ........................................ 81
35. Delete a Process Mainline, Combination, or Interrupt Core Load from the Core Load Area ............ 83
36. Replacing a Nonprocess Core Load in the Core Load Area ........ 85
37. Replace a Relocatable Program in the Relocatable Program Area .................................................. 87
38. On-Line Rebuilding of Process Core Loads ......................... 89
39. Reserving a File Area in the Core Load Area ....................... 90
40. Illustrating Various Card Arrangements in Dumping a Program/Data to Nonprocess Working Storage, Punched Cards, and the List Printer .................................................. 92
41. Reloading Core Loads to User Sequence .............................. 93
42. Dumping a Relocatable Program from the User Area ............. 94
43. Moving a Data File within the Core Load Area .................... 96
44. Reloading a Program to Nonprocess Working Storage ......... 98
45. Showing the Relationship of Local Groups or Blocks to Associated Core Load within the Core Load Area on Disk .................................................. 100
46. Illustrating the Implementation of LOCALS ....................... 102
47. Repacking User Area on Disk Drive 1 ............................... 104
48. Repacking the Relocatable Subroutine Area Following a Removal of Various Portions of the TSX Subroutine Library .................................................. 106
49. Production of Cards ....................................................... 108
50. Dump LET/FLET of Disk Drives 0, 1 and 2 ........................ 109
51. LET Entries ............................................................... 111
52. FLET Entries .............................................................. 112
53. Cold Start for an On-Line System ..................................... 114
54. Relationship of Physical Disk Drive Units to Logical Number .................................................. 116
55. Cold Start for an Off-Line System .................................... 118
56. Preparing a GUARD or Dummy Interrupt Core Load .......... 120
57. Illustrating Logic of Console Interrupt ............................. 122
58. Illustrating Perpetual Time-Shared Nonprocess Monitor Operation .................................................. 124
59. System Generation Overview .......................................... 126
60. TASK Source Deck and TASK Equate Cards ....................... 128
61. A Set of TASK Equate Cards for the TSX Sample System (see Programming Techniques) ................ 130
62. The IBM Nonprocess System ........................................... 132
63. Sequence of Control Cards at System Load Time ................. 134
64. Disk Drive 0 after a System Load Operation ....................... 136
65. System Director Source Deck and EQUATE Cards ............... 138
66. Example of a Set of System Director Equate Cards ............... 140
67. Mainline Core Load Queue Table ..................................... 142
68. Example of Interrupt Level Status Word Assignment .......... 144
69. Layout of a Level Work Area ......................................... 146
70. Interrupt Core Load Table ............................................. 148
71. Disk Storage Unit Conversion Factors ............................... 150
72. Disk Layout of a Single Disk Drive TSX System .................. 152
73. Overview of the DEFINE CONFIG Operation (Disk Drive 0) .... 154
74. Illustrating Direction of Disk Configuration ........................ 156
75. Establishment of System Areas at High Address End of a Disk .................................................. 158
76. Establishment of Message Buffer Area at System Load Time .... 160
77. Illustrating Redefinition of the Message Buffer Area .......... 162
78. Disk Layout of Disk Drive Zero for Example 1 .................... 164
79. Definition of a Three-Drive TSX On-Line System for Example 2 .................................................. 166
80. Definition of a Three-Drive TSX Off-Line System for Example 3 .................................................. 168
81. Constitution of the System Skeleton ................................ 170
82. A Partial Dump following a Skeleton Build to Illustrate the Program Name Table and the Executive Transfer Vector .................................................. 172
83. On-Line (Time-Sharing) System ....................................... 174
84. On-Line (Non Time-Sharing) System ................................. 176
85. Off-Line System .......................................................... 178
86. Illustrating Relationship of DP I/O Devices to Associated Function and Conversion Subroutines ....... 180
87. Layout of the System Skeleton as it would appear at
Skeleton Build Time in NPWS and the Skeleton Area .... 174
| Core Map for Initial and Rebuilt Skeleton ............. 178
IBM 1800 Data Acquisition and Control System ......... 193

Tables
1. Table of Available Timer Time Bases ................. 43
2. Comparison of Timers ................................ 48
3. On-Line EAC Error Type Codes ......................... 61
4. On-Line EAC Errors and Recovery Procedures ......... 62
5. Monitor Control Cards ................................ 77
6. Loader Control Cards .................................. 77
7. DUP Control Cards ..................................... 78
8. FORTRAN Control Cards ................................ 79

Program Listings
1. .......................................................... 51
2. .......................................................... 89
3. .......................................................... 92
4. .......................................................... 104
5. .......................................................... 168
6. .......................................................... 170
7. .......................................................... 183
8. .......................................................... 189
9. .......................................................... 221
10. .......................................................... 280

Examples
Use of Timers ............................................. 44
Use of Non-process Monitor ............................... 79
1. Assemble and execute a non-process program from the Temporary User Area .... 80
2. Compile and store a non-process program in the Relocatable (or User) Area .... 81
3. Compile and execute a non-process program from the Core Load Area .......... 82
4. Delete a mainline, combination or interrupt core load from the Core Load Area .... 84
5. Replace a non-process core load in the Core Load Area .................. 85
6. Replace a relocatable program in the User Area .................. 85
7a. On-line rebuilding of process core loads ................ 86
7b. Changing the sequence of linked core loads (using $SISKCH) .......... 86
8. De-bugging the Simulator ................................ 87
9. An example program which uses the Simulator ................. 91
10. Reserving a file area in the Core Load Area ............. 97
11. Dumping a program or data file from the Core Load Area .......... 98
12. Loading a previously dumped core load back into the Core Load Area .......... 98
13. Dumping a program from the Relocatable (or User) Area .......... 98
 Moving a data file (or files) within the Core Load Area .......... 99
 Loading a core load back into Non-process Working Storage .......... 99
16. A program using LOCAL sub-programs ................. 102
17. Packing the User (Relocatable Program) Area .............. 107
18. How to reproduce cards ................................ 107
19. Dumping a LET/FLET Table ............................. 107
20. How to call for a process core load externally ............ 107
21. How to initiate a Nonprocess Monitor operation .......... 110
22. How to terminate a Nonprocess Monitor operation ........ 111
23. Preparing a "guard" (or "dummy") interrupt core load .......... 111
24. Use of the Console Interrupt ............................. 111
25. Preparing a mainline for perpetual Time-Shared Nonprocess Monitor operation .......... 112

Non-synchronous Periodic Scheduling ..................... 26
How to use masks to "set up" designated levels ............ 40
1. How to mask levels .................................. 40
2. How to unmask levels ................................ 40
3. How to unmask all levels .............................. 40
4. How to mask some levels and unmask all others .......... 40

The use of LUN Numbers/IAC Codes ....................... 126
Disk Configuration Operations ............................ 154
1. .......................................................... 154
2. .......................................................... 155
3. .......................................................... 156
Disk Cartridge Initialization, using DWRAD ................ 157
4. .......................................................... 158
5. .......................................................... 158
6. .......................................................... 158
Inclusion of Subroutines in the Skeleton .................... 164
1. .......................................................... 167
2. .......................................................... 169
CALL-type Subroutines ................................ 186
LIBF-type Subroutines ................................ 186
Masking and Unmasking to interrupts ...................... 186
Re-entrant Subroutines ................................ 187
Writing User Programs for execution under TASK Absolute Loader .......... 187
INTRODUCTION

With few exceptions, real-time applications are distinguished from other applications by two chief characteristics: 1) some process or operation going on outside the computer system normally has a continuous need for on-line communication with the system, 2) there is a requirement for the computing system to keep pace with the associated process or operation. These characteristics of the application place some unique and stringent requirements on real-time processing systems for use in the real-time environment.

Recognizing the formidable programming task associated with a system of this scope, IBM has developed the 1800 Time-Sharing Executive System (TSX) which relieves the user of much of the required programming effort by freeing him to concentrate on the primary task of problem solution. TSX is a FORTRAN-oriented disk-resident operating system which permits the user to make optimum use of an IBM 1800 Data Acquisition and Control System (DACS) for its primary purpose, the control of processes and similar complex environments, as well as providing him with an effective off-line monitor system for data processing and scientific computation. TSX improves greatly the versatility of a Data Acquisition and Control System (DACS) computer by making it possible for background jobs to be processed when the real-time foreground task relinquishes control of the processor-controller. It is through time-sharing that idle computer time is minimized or eliminated. Programs may be written in FORTRAN and/or symbolic Assembler language.

Figure 1 introduces the capabilities of TSX in generalized form.

MINIMUM SYSTEM REQUIREMENTS

To assist users in performing their initial system generation, a standardized "starter" called System Generation (SYSGEN) TASK is provided with each installation which contains the basic elements necessary for system generation in a form that will be directly usable by a majority of users. SYSGEN TASK is supplied in assembled object format as part of the IBM Nonprocess System and consists of the following:

- Nonprocess Monitor Linkages
- Skeleton Builder Linkages
- Absolute Loader

It is designed to support the following minimum devices:

- 1 IBM 1801 or 1802 Processor-Controller with a minimum of 8K words of core storage
- 1 IBM 2310 Disk Storage Unit with one disk drive
- 1 IBM 1442 Card Read Punch
- 1 IBM 1816 Keyboard Printer (that is, printer portion only) or
- 1 IBM 1053 Printer

The user may employ additional I/O devices on his system, but he must satisfy the above machine configuration requirements to use SYSGEN TASK. For example, if he substitutes a 1443 Printer for a 1053 Printer or an 1816 Keyboard Printer, a card assembly of the TASK source deck to include this provision becomes mandatory. As the "starter" system is a limited version of the Temporary Assembled Skeleton (TASK), it will neither buffer 1053 Printer messages to disk, nor does it contain the trace and dump utility functions.

Machine Features Supported

In addition to the above, the following optional machine units and features are supported by the TSX system:

- Additional core storage (up to a maximum of 32,768 words)
- Additional disk drives for IBM 2310 Disk Storage Unit -- up to a maximum of three disk drives
- Additional IBM 1442 Card Read Punch Unit (up to a maximum of 2)
- Additional IBM 1816 Printer Keyboard (up to a maximum of 2)
- Additional IBM 1053 Printer Units (up to a total of eight 1053s and 1816s)
System Director

Supervises Execution of Process Programs. It includes:
- Time-Sharing Control
- Program Sequence Control
- Master Interrupt Control
- Interval Timer Control
- Error Alert Control

SUBROUTINE LIBRARY

Arithmetic, Input/Output and Conversion
Process, Input/Output

NONPROCESS MONITOR

Supervises Execution of Nonprocess Programs. It includes:
- Nonprocess Supervisor
- Disk Utilities
- Fortran Compiler
- Assembler
- Simulator

USER'S NONPROCESS PROGRAMS

IBM Programs

User-Written Programs

Figure 1. IBM 1800 Time-Sharing Executive System

- Additional Data Channels (up to a total of 9)
- Additional Interrupt Levels (up to a maximum of 24)
- Multiplexer Unit (Solid state and Relay)
- Analog-Digital Converter (up to a total of 2)
- Digital-Analog Output
- Digital Input

- Comparator
- IBM 1443 Printer Unit
- IBM 2401-2402 Magnetic Tape Units (maximum of 2)
- IBM 1627 Plotter Unit
- IBM 1054 Paper Tape Reader
- IBM 1055 Paper Tape Punch
MODES OF OPERATION

The IBM 1800 Time-Sharing Executive System consists essentially of two main parts: (1) a Skeleton Executive and (2) a Nonprocess Monitor. It is through the Skeleton Executive that process control and data acquisition applications are serviced in the on-line mode, while the Nonprocess Monitor operates either in the time-shared mode or as an independent programming system to provide data processing functions in a standard off-line mode. Each of these modes is brought into play by an appropriate and corresponding system generation procedure. The user elects the option of constructing an on-line or off-line system tailored to individual requirements.

On-Line Mode

In real-time processing, inputs arrive randomly from a process being monitored to the processor-controller. The computer rapidly responds to each input usually by conveying an output back to the process. This is in contrast with conventional batch processing where groups of input data are processed by passes through the computer. The notion of real-time usually implies that a processor-controller is responding to inputs as they occur in the physical world.

TSX operates in this mode under the control of the Skeleton Executive. In an on-line environment, user-written programs may monitor and/or control a process operation at any time. The machine is also permitted to be shared by process and nonprocess work; that is, batch work may be interleaved with other work. Whenever variable core is not required for a process program, the Nonprocess Monitor may be brought into service. All core loads and/or programs executed are accessed from the system resident disk cartridge.

Off-Line Mode

The off-line TSX system operates in this mode under the control of the Temporary Assembled Skeleton (TASK) as a dedicated Nonprocess Monitor System. Typical off-line operations are assemblies, compilations, disk utility functions, and the execution of data processing programs.

An off-line system can be used to test problem programs before they are permanently stored and catalogued on the system cartridge, to execute problem programs that require the full capacity of available disk drives for data files, or to execute problem programs that are used so infrequently that their on-line storage is not justified. It is also used to build an on-line disk resident system.

SYSTEM CONCEPT

ROLE OF THE SKELETON EXECUTIVE

The Skeleton Executive constitutes the framework of an on-line TSX system, and must be resident in permanent core storage before any continuous and coordinated real-time processing can take place. Other portions of the system are brought into core from disk storage as they are required to perform specific functions.

The Executive is extremely flexible and can be assembled at system generation time so that no core is wasted by selecting any of the numerous options available. The user may include frequently-called subroutines, fast response interrupt servicing routines, and other user-written programs in the skeleton to make the most effective use of his control system.

A typical skeleton executive might consist of the following parts as shown in Figure 2. The function of each individual component will now be described.

Skeleton I/O. This is a set of input-output subroutines which provides a rapid and easy method for the user to reference the various data processing input-output devices (e.g., card read punch, disk, printer) for input or output of data. It includes:

- DISKN (Disk Storage Subroutine - performs all reading from and writing to the IBM 2310 Disk Storage Unit)
- TYPEN/WRTYN (Printer-Keyboard Subroutine - transfers data to and from the IBM 1053 and IBM 1816 Printer-Keyboard)
- PRNTN (Printer Subroutine - handles all print and carriage control functions relative to the IBM 1443 Printer)

These and other basic system routines make up the Skeleton I/O package which corresponds to an identical set of input-output subroutines used by TASK. A description of each subroutine will be found elsewhere in the TSX Systems Reference Library.
INSKEL COMMON. A uniquely labelled common area in the skeleton set aside for communications among the various types of core loads used in the system. It can be referenced by any process or nonprocess program under the on-line system.

System Director. This is the nucleus of the Skeleton Executive and controls all facets of process monitoring. It directs the handling of interrupts in a priority fashion determined by the user; supervises the execution of any number of mainline core loads or programs dictated by the process; services all error conditions with a minimum of disturbance to most processes under control; maintains the 1800 interval timers; and makes the system available to the Nonprocess Monitor.

User-Written Programs. The user has the option to include as many programs and subroutines as possible in the skeleton for reasons of frequent usage, rapid response, and optimum utilization of disk space. These may take the form of:

- Interrupt subroutines
- Timer subroutines
- Count subroutines
- Special trace and error subroutines
- IBM-supplied arithmetic, I/O, and other subroutines
- Any other user-written subroutines

These are first compiled/assembled in relocatable format and stored on disk; at skeleton build time, they are relocated into the Skeleton Executive.

TIME-SHARING

In many industrial installations, the real-time control system will not utilize all the computer time; therefore, time will be available to perform background jobs. Time-sharing techniques can thus be employed when idle processor-controller time is available in a given system environment to offer the user the kind of service he requires. The notion of time-sharing also implies the sharing of computer resources, since not only time but primary and secondary storage as well as most input-output facilities are also shared.

When idle time is available in the IBM 1800 TSX System, control can be automatically transferred to an independent Nonprocess Monitor System which is identical to any stack-job monitor system. All assembling, compiling, simulating, and other system activities can now be executed under the control of the Nonprocess Monitor. Performing such jobs time-shared has a distinct advantage in that any time not required for process control functions can be used for data processing functions. Also, since process control programs and strategies tend to change, time-sharing makes it extremely desirable to be able to modify these programs and strategies at the on-line installation without taking the computer off-line. It is through the time-sharing feature that the utilization of the 1800 system is best optimized.
VERSATILITY IN SYSTEM CONFIGURATION

A modern real-time operating system must be geared to change and diversity. The TSX system itself can exist in an almost unlimited variety of machine configurations: different installations will typically have different configurations as well as different applications. Moreover, the configuration at a given installation may frequently change. If we look at application and configuration of an operating system, we see that the operating system must cope with an unprecedented number of environments. All of this puts a premium on system modularity and flexibility.

TSX gives the user the ability to define his configuration according to his exact needs: he is therefore never bound to a fixed system. Furthermore, after having specified and generated a particular system, he is still free to move process and/or nonprocess portions of his system from one disk storage device to another.

In general, the input-output capability of the IBM 1800 Data Acquisition and Control System can be backed up. For example, under program control, a 1053 Printer can have its messages automatically switched to a back-up 1053 Printer; disk storage drives can be logically switched or removed from the system; and any device may be removed from service if it continues to fail. This dual capacity indicates that an installation may suffer from the failure of one or more input-output devices, and remain "on the air" under the most stringent usage conditions. Hand-in-hand with this back-up capability, a history of hardware device failures can be examined at any time for maintenance purposes.

CONCEPT OF A CORE LOAD

In practice, the core storage size of a data acquisition and control system is not sufficient to contain (nor does it need contain) all of the instructions required for the execution of all functions at any one time. Thus, the entire set of instructions must be broken down into smaller pieces, and these pieces be made available for immediate loading. To facilitate rapid loading, they should be stored on disk in executable core image format.

The technique of program segmentation is employed in the TSX system where programs are formed into smaller units called core loads. A core load is, by definition, an executable program or portion of a program which performs some user function. It is not necessarily a program in its entirety because this program may well be too large to fit into variable core in one piece for execution. The core load is unique in that it is stored on disk in core load core image format to facilitate rapid loading when it is called for execution.

Figure 3 illustrates the four types of core loads commonly used in TSX. A core load may contain other subroutines that are not associated with the main program - that is, subroutines not otherwise available in core (either included in the skeleton, or in the form of load-on-call subprograms). A typical core load may consist of a mainline or interrupt program, in-core interrupt subroutines, and all other required subroutines that are not included with the Skeleton Executive.

Core loads are important in real-time systems for the following reasons:

- Real-time linkages are automatically built
- The core-load is permanently built and stored on disk for rapid execution
- Core loads are called by name
- No compiling/assembling is needed at execution time.

LOCAL SUBPROGRAMS

TSX provides a facility for loading subroutines at the time they are called for in the execution of a program. Such a subroutine is known as a LOCAL (load-on-call). All LOCALs called by the same mainline program in a core load use the same area of core storage by overlaying one another as they are called. A copy of each LOCAL subprogram used with a core load is kept on disk in core-image format together with that core load (see Figure 3).

LOCALs thus allow the user to have, effectively, a larger program executed in core than would otherwise be possible if all the subroutines were loaded into core at the same time. There is no theoretical limit to the number of LOCALs which the user wishes to implement. This means a virtual extension of variable core. Other advantages of this feature are (a) the ability to read in subroutines, and (b) the breakdown of core loads to the subroutine level.
Reentrant Coding

One of the basic problems that arises in multi-level programming is that different levels of operation require the use of the same subroutine. If a method of reentrant coding is not used, there is often a danger that intermediate results might be destroyed. To allow one subroutine to be entered at any time and from any interrupt level, without loss or degradation of results, a method of reentrant coding which uses level work areas is devised for TSX.

All subroutines that are required on multiple levels in TSX are fully reentrant. That is, they can be called repeatedly by different interrupt subroutines at different levels; they are automatically reenterable; and they automatically keep guard of the partial results acquired when they were interrupted. A single subroutine can be used simultaneously at all 26 levels, while it is servicing any other level. The automatic accounting of the partial results of the subroutines is a very significant step forward which is made possible through the programming structure of TSX.

Some of the advantages of TSX reentrant coding may be summarized as follows:

- All levels of operation may execute any given reentrant subroutine
- The size of the overall system in core and on disk is reduced. This is because (1) core loads do not have to contain subroutines that are included in the Skeleton Executive, and (2) subroutines included in core loads are smaller than non-reentrant subroutines
- All subroutines can share the same area of core
- The system overhead time is reduced during the loading of core loads because (1) core loads are now smaller, and (2) they are performing more functions

Common Areas

Three unique areas of core storage are used for FORTRAN COMMON storage within TSX. These are

1. INSKEL COMMON
2. Normal COMMON
3. Interrupt COMMON
INSKEL COMMON has already been defined. To assign a variable to this area, a special FORTRAN statement, COMMON/INSKEL/, must be used. All other attributes of the COMMON statement remain the same. This area must be used for communications between

- Core loads of a different type
- Interrupt core loads
- Combination core loads (if either is executed as an interrupt core load)
- A special core load and the mainline core load that calls it
- A mainline core load (which called a special core load) and the core load that restores it
- A skeleton subroutine and any other subroutine or core load

The normal COMMON area located at the high-address end of core storage can be referenced only by mainline or nonprocess core loads. The normal COMMON statement in a mainline, special, or nonprocess core load is used to refer to this area. This area is saved and restored when special core loads or time-sharing operations are initiated or terminated; i.e., communication between nonprocess core loads is possible.

The third area for COMMON is used only for interprogram communication for programs that form an interrupt core load or, between combination core loads when they are executed on the mainline level. The normal COMMON statement in an interrupt or combination core load is used to refer to this area. The highest addressed location of this area must be assigned by the user at system generation time, and must be an even number. This assigned location is the high-address boundary of the variable core storage area that is saved when an interrupt core load is loaded for execution. Thus, it is necessary to save only the area specified by the user for interrupt core loads (not the whole variable area).

MULTI-LEVEL PROGRAMMING

The interrupt structure of the 1800 system consists of a total of 24 hardware levels with up to 16 interrupt signals per level. These can, of course, be processed in a true priority sequence. A higher level interrupt subroutine will always interrupt a lower level interrupt subroutine, but beyond this, the Skeleton Executive permits interrupts to be "recorded" now for later processing.

The interrupt scheme within the Skeleton Executive also provides a great amount of flexibility and frees the user from most of the problems of servicing interrupts. Interrupt servicing subroutines may be assigned in the following ways:

1. An interrupt subroutine which must be executed immediately under any condition whatsoever can be made a permanent part of the skeleton. That is, the subroutine will always be in high-speed core storage and will be executable in the order of microseconds.
2. Those subroutines which are associated with a given mainline program can be assigned in such a way that they are always read into core storage with that mainline when it is loaded from disk. The response time of a mainline interrupt routine is almost the same as that of a skeleton interrupt routine only if the mainline core load containing the interrupt routine is in core when the interrupt occurs.
3. For low-priority subroutines, a core overlay technique allows the user to call an interrupt core load, bring it into core storage, save what was in core storage, and on completion of the interrupt process, restore core storage to its original state. These multiple operations of sequencing, saving, and replacing of core storage is automatically handled by the Skeleton Executive. All that is required of the user is to assign the priority. It should be mentioned that the priority interrupt sequence can be changed, at will, under program control.

The interrupt core load response time depends on the size of the core load and the disk layout. It is slower than the skeleton or mainline core load interrupts.

SYSTEM COMPONENTS

TSX components can be considered under two separate group-headings: (1) control programs and (2) processing programs.

In general, control programs govern the order in which processing programs are executed, and provide services that are required in common by the
processing programs during their execution. A key control program is the System Director which is loaded into main storage (as part of the resident Skeleton Executive) and remains there indefinitely to ensure continuous coordinated operation of the system. Other parts of the system are brought into main storage from secondary storage as they are required to perform specific functions. Processing programs consist of language translators and service programs that are provided by IBM to assist the user, as well as problem programs that are user-written and incorporated as part of the TSX system. Both IBM and user programs have the same functional relationship to the control programs.

CONTROL PROGRAMS

There are three control programs within the TSX system:

Temporary Assembled Skeleton (TASK)
System Director
Nonprocess Supervisor

Temporary Assembled Skeleton (TASK)

TASK is a stand-alone disk oriented monitor program from which an on-line or off-line TSX system is constructed. It performs three distinct functions:

- Supervises the generation of a disk oriented TSX operating system according to user specifications.
- Supports a full monitor capability so that TSX can be used as a data processing monitor system.
- Allows the user to load absolute programs into core for execution, or to store them on disk.

Since real-time process control installation requirements vary from installation to installation, it follows that each installation must be defined or tailored to the specific system functional requirements and input-output configuration of that installation. The tailoring function, defined as system generation, is carried out by TASK which provides the facilities for the creation and maintenance of a monitor system composed of IBM and user-written programs. The user specifies his system through the medium of equate cards.

Figure 4 illustrates TASK organization in simplified form.

The System Director

This control program forms the heart of the TSX system and resides in core storage at all times as part of the skeleton where all permanent areas are storage-protected to ensure that they are not inadvertently violated or altered.

The System Director directs the handling of process and data processing input-output interrupts, provides timer control over the process, is responsible for the orderly transfer of control from one core load to the next, and handles the transfer of control between the on-line and off-line modes. All process core loads are in core-image format on disk and are accessed at disk read speed.

The Director is read from disk only during a cold start or reload (EAC) operation. Primary entry to the System Director results from 1) internal and external hardware interrupts, 2) TSX calls from user's programs, and 3) errors.
The Nonprocess Supervisor

The Nonprocess Supervisor directs the execution of all nonprocess core loads which may be either IBM-supplied as part of the TSX package or user-written. It normally operates in the time-sharing mode under the control of the System Director, but it may also be run as a dedicated off-line monitor system under TASK.

Its main function is to recognize certain system control cards and transfer control to the processing program specified. It also initializes the nonprocess system whenever a job control card is identified.

PROCESSING PROGRAMS

Processing programs consist of service programs and language translators broken down as follows:

Service Programs
Cold Start Program
System Loader
Core Load Builder
Skeleton Builder
IBM TSX Subroutine Library
Disk Utility Program (DUP)

Language Translators
Assembler
FORTRAN Compiler
Simulator

Service Programs

Service programs include a group of loaders and builders which serve as system generation aids, as well as a disk utility program and a comprehensive IBM TSX Subroutine Library.

Cold Start Program

This program loads the Skeleton Executive into core, storage protects it, starts the real-time clock and calls the user's initial core load for execution. This operation places the System Director in control of the on-line system.

System Loader

The primary functions of the System Loader are to load the initial IBM TSX system onto the disk, build an interrupt assignment table from user-supplied control records, and prepare the disk layout for system operation. System assignment cards are used to inform the System Loader of interrupt level assignment of I/O devices, interval timers, and process interrupts. The loader makes entries in a directory called the Location Equivalence Table (LET) for each component part of the IBM TSX system.

Core Load Builder

The Core Load Builder program combines a user-written relocatable program together with all referenced subroutines not included in the Skeleton Executive into an executable core load for storage in the Core Load Area on disk. Core loads may be of several types: process mainline, combination, interrupt, or nonprocess.

All process core loads must be built and stored on disk prior to execution under control of an on-line TSX system. Input to the Core Load Builder is supplied by the user in the form of control records which contain the names of relocatable mainline programs, interrupts to be recorded, data files used, interrupt routines included as part of the core load, and LOCAL (load-on-call) subprograms.

Using the data provided by the System Loader and the Skeleton Builder, as well as information from programs and subroutines, the Core Load Builder establishes all subroutine linkages, hardware interrupt servicing linkages, and the creation of certain communications areas which are integrated with instructions to make up a core load.

Skeleton Builder

The Skeleton Builder program obtains its input from user-assigned control records, programs, subroutines, and information from the System Loader to construct the System Skeleton in core-image format which is then stored on disk. The skeleton is read into core for execution by a cold start operation. The rebuilding of the skeleton is required whenever routines are added or deleted, or other modifications are made. It is the System Skeleton which constitutes the Skeleton Executive.

IBM TSX Subroutine Library

This comprises a comprehensive set of reentrant subroutines as well as a select set of non-reentrant subroutines designed to aid the user in making efficient use of the IBM 1800 Data Acquisition and
Control System. The library contains the following:

- Data processing and process input-output subroutines
- Conversion subroutines
- Arithmetic and functional subroutines
- FORTRAN input-output subroutines
- Miscellaneous subroutines

Data Processing and Process I/O Subroutines. Data processing (printers, punches, etc.) and process input-output (P I/O) subroutines provide a quick and straightforward method for the programmer to reference the various data processing, digital and analog I/O devices for input or output of data. All I/O routines may be called directly from FORTRAN; data processing I/O subroutines may be called indirectly by the use of FORTRAN I/O.

Conversion Subroutines. The design and operation of the numerous input-output devices is such that many of them impose a unique correspondence between character representations in the external medium and the associated bit configurations within the computer. Conversion subroutines convert inputs from these devices into a form in which the computer can operate and to prepare computed results for output on various devices.

Arithmetic and Functional Subroutines. The arithmetic and functional group of subroutines includes a selection of twenty-seven basic routines which are most frequently required because of their general applicability. The arithmetic library contains both the routines that are visible to the FORTRAN programmer, as well as the many routines that are used by the FORTRAN compiler generated object code, which may also be used by the Assembler programmer.

FORTRAN I/O Subroutines. FORTRAN I/O subroutines provide a link between the FORTRAN object code and the I/O devices. They support both standard and extended precision.

Miscellaneous Subroutines. The miscellaneous group provides the user with the ability to perform certain machine operations using the FORTRAN language. These include real-time, selective dump, trace, and overlay routines.

Real-time subroutines are operational control routines which service the Skeleton Executive in an on-line environment. Examples are TIMER (specify one of two hardware interval timers for some periodic activity), LEVEL (set one of twenty-four levels for programmed interrupt use), and MASK (inhibit selectively one or more levels of interrupt).

Selective dump subroutines allow the user to print chosen areas of core storage during the execution of an object program. For example, DUMP will output on the list printer, in hexadecimal or decimal format, a certain portion of core storage; DUMPS will print the status of the 1800 (that is, status indicators, contents of registers, and work areas).

The user may exercise the option of writing his own mainline trace interrupt routine which can be included in a core load to process a trace interrupt. He might, for example, design such a routine to monitor a number of conditions. The subroutine TRPRT is available for use in tracing routines which print a specified number of characters on the 1053/1816 Keyboard Printer or 1443 Printer.

The TSX Subroutine Library also contains an overlay routine called FLIP which serves to call LOCAL (load-on-call) subprograms into core storage. All LOCALs in a given core load are executed from the same core storage locations; each LOCAL group overlays the previous group.

In order to permit entry from multiple programs and interrupt levels before completing computations from a previous call, the arithmetic and functional subroutines, and most of the I/O subroutines, are designed to be reentrant. That is, they can be entered from a different level of machine operation despite the fact that they may not have completed operation on a previous level. Non-reentrant versions of the arithmetic, functional, and conversion subroutines are also supplied.

Disk Utility Program (DUP)

The Disk Utility Program is a comprehensive group of generalized utility and maintenance routines designed to aid the user in the day-to-day operation of the TSX system. By this means, the most frequently required services of disk and data maintenance can be
performed with a minimum of effort. The TSX DUP philosophy is to provide as much assistance as possible to the user. DUP is a component part of the Nonprocess Monitor.

DUP is called into service by the Nonprocess Monitor Supervisor (SUP) whenever it recognizes a DUP monitor control card. It is also automatically summoned after the successful completion of an assembly or FORTRAN compilation. DUP functions can be summarized as follows:

1. It permits the user to store, modify, and refer to programs and data using the compact and economical direct-access disk storage facilities of the system without regard to specific input-output configurations.
2. It allows the free interchange and use of programs and data among programmers.
3. It provides a systematic method to identify and locate programs and data, and systematic methods to reference data after it is located.

All of these functions can be carried out while the TSX system is on-line, as well as in the off-line mode. Examples of DUP facilities include the following:

- Change sequence of execution of core loads
- Replace a core load with another core load
- Create disk files
- Replace an object program already stored on disk
- Build core loads (in conjunction with the Core Load Builder)
- Define the disk configuration
- Dump data/program from one medium to another
- Delete a program, core load, or a data file from the disk
- Pack a file on the disk to eliminate unused areas, thus minimizing disk storage requirements
- Modify core loads on-line

Language Translators

Language translators assist a programmer by enabling him to define a problem or an application in a language form that can be readily learned and understood. In the TSX system, the user may define his problem solution or application

- In a flexible easy-to-use symbolic language – Assembler language, and/or
- In a form of mathematical notation – FORTRAN

Assembler

The Assembler program is a one-for-one disk oriented symbolic type translator which assembles object programs in machine language from source programs written in symbolic language. It normally resides on disk. The assembler accepts control records and source programs in card form only. Upon a successful assembly, the object program in relocatable format is moved to disk where it is permanently stored, or, alternatively, called for execution. The Assembler Language is fully described in the publication IBM 1800 Assembler Language, Form C26-5882.

FORTRAN Compiler

The FORTRAN Compiler translates programs written in the FORTRAN language into executable machine language. The real-time TSX FORTRAN Compiler permits the user to make the most of the digital and analog I/O features using a higher level language, while at the same time allowing background jobs to be executed. Since FORTRAN is easily understood by technical personnel, its availability in the TSX system reduces significantly the programming effort required. For a full description of the FORTRAN language, see IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715.

Simulator

The Simulator is designed as a debugging aid which allows the user to check out or test process and/or nonprocess programs without disrupting normal TSX system operation - that is, without taking the system off line. It functions under the control of the Nonprocess Monitor.
FUNCTIONS OF EXECUTIVE PROGRAMS

This section describes the functions of the three main executive programs which constitute an IBM 1800 Time-Sharing Executive System, namely,

Temporary Assembled Skeleton (TASK)
System Director
Nonprocess Monitor

and discusses the concepts underlying their use. Sample programs and coding are interspersed throughout the text as demonstration of good programming practice and technique. Since the Temporary Assembled Skeleton (TASK) is the first program with which the user becomes involved in the creation of an on-line or off-line TSX system, it is discussed at the outset.

TEMPORARY ASSEMBLED SKELETON (TASK)

It has already been mentioned that TASK (Temporary Assembled Skeleton) is a builder or "tailor" card monitor system with strong disk capabilities from which an off-line or on-line TSX system is constructed. The use of TASK, therefore, constitutes the intermediate stage in system generation towards placing a system on-line. In an on-line TSX system, TASK control ceases at cold start time when the System Skeleton has been loaded into core storage. In an off-line TSX system, TASK itself functions in much the same fashion as a System Skeleton with permanent time-sharing.

For simplicity, TASK can be considered in two parts (see Figure 4):

- Skeleton I/O
- TASK Program Set

Skeleton I/O

The Skeleton I/O is a collection of input-output and general supporting subroutines that the TSX system requires to be in core at all times. It is that portion of a user-configurated TASK which corresponds exactly to the Skeleton I/O on an on-line TSX system.

Figure 5 illustrates this correspondence, as well as the core layout, at two time periods of an on-line and an off-line system.

![Diagram of TASK and System Skeleton](Figure 5. Correspondence between TASK and the System Skeleton)
The I/O routines used by TASK form the basis of the Skeleton I/O. These consist of the following:

- **DISKN** - Disk subroutine as used by TSX Operating System
- **TYPEN/WRTYN** - Printer/Keyboard subroutine as used by TSX Operating System
- **PRNTN** - Printer subroutine as used by TSX Operating System
- **CARDN** - TASK only Card I/O subroutine

A description of each of the above subroutines will be found in the TSX Systems Reference library.

Since the TSX system requires that at least one disk be present on the 1800, DISKN must be in core at all times. If the user has assigned a 1053 or 1816 to his machine, TYPEN/WRTYN must also reside permanently in core. Although CARDN is in core during TASK execution, it does not normally form a part of the Skeleton I/O. The user must therefore define whether or not CARDN is to be a component part of his skeleton by means of the TASK equate card CDINS. If it is not, CARDN automatically becomes a part of the TASK Program set. It is through the Skeleton I/O that an off-line system obtains full monitor capabilities.

**TASK Program Set**

The TASK Program Set is that integral part of the Temporary Assembled Skeleton which functions in a similar manner to the System Director. It consists of:

- **TASK Master Interrupt Control (TMIC)**
- **TASK Director**
- **TASK Error Alert Control (TEAC)**
- **Absolute Loader function**
- **Load Monitor function**
- **Skeleton Build function**
- **TASK Conversion routines**
- **TASK Utilities**

TMIC directs all I/O interrupts to their corresponding servicing routines and resets all process interrupts, while TEAC processes errors that have been found by other parts of TASK. The TASK Director initializes TASK and directs the execution of the Absolute Loader function, Load Monitor function, and the Skeleton Build function.

The Absolute Loader gives the user a facility to load absolute assembled programs from cards to core for execution. It can also be used for the storing of user-written programs or data on disk. The use of this function is discussed later in some detail (see Programming Techniques). The Load Monitor function serves to initialize the TSX Nonprocess Monitor for execution. There are two conversion routines: (1) TASK HOLEB converts hollerith input to one or two EBCDIC characters per word output, while (2) TASK EBPRT converts two characters per word EBCDIC input to two characters per word, system, list, or EAC printer code.

A complete utility package comprised of full trace, check/stop trace, four utility programs, and a utility monitor can be included within TASK at assembly time. The user elects this option through equate cards.

Except in the case of a skeleton builder option, a TASK disk load, or a cold start, TASK is loaded with a 4-card TASK high core loader.

For a more complete description of TASK functions and system generation procedures, the user is referred to the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Other considerations affecting the use of TASK are discussed under System Design Considerations.

**THE SYSTEM DIRECTOR**

The System Director is the nucleus of the skeleton executive of a TSX system, and always resides in core as part of the skeleton to direct the handling of interrupts, to load and execute core loads, to expand usage of interval timers, and to process errors. Primary entry to the System Director derives from internal and external hardware interrupts, TSX calls from user's programs and errors. Its principal component parts comprise the following:

Master Interrupt Control (MIC). This is a reentrant control program which automatically directs all internal, I/O, external, and programmed interrupts to their proper interrupt servicing routines. Control returns to MIC as long as unserviced interrupts exist.
**Program Sequence Control (PSC).** The Program Sequence Control Program is responsible for orderly transfer of control from one user-specified core load to the next. A core load may also temporarily be saved on disk pending the processing of another core load. All PSC functions are restricted to process mainline core loads.

**Time-Sharing Control (TSC).** This controls the time-sharing of variable core between process and nonprocess core loads by a core exchange method. TSC is entered selectively from the execution of a CALL SHARE statement or automatically by a CALL VIAQ statement when the queue is empty.

**Interval Timer Control (ITC).** ITC services all interrupts involving three machine timers A, B, and C, nine programmed timers, and a programmed real time clock. The programmed timers and the real time clock are based on timer C. Timer C is reset by the subtraction of a fixed value; accurate timing is therefore kept, even when the response to the timer interrupt itself may be delayed. It also services the "no-response routine" for the 1053/1816 printers in the Skeleton I/O. As an option, it also serves the Operations Monitor during nonprocess execution. Periodic interrupts are generated from interval timers rather than from the real time clock. The programmed timers interrogate the Interrupt Core Load Table (ICLT), but only skeleton count routines are entered into. If there is no such routine, the condition is recorded for later servicing.

**Error Alert Control (EAC).** The EAC program resides in core at all times, and is called to process all error conditions whenever an error develops.

- optionally saves core for future reference
- optionally branches to a user-written error subroutine (which may be included with each process core load) for further error analysis
- prints an error diagnostic message, and
- executes one of four possible error recovery procedures

**Mainline Core Load Queue Table.** This is a stack or pushdown list of names of mainline core loads (and their respective priorities) that have been queued (that is, put in line) for future execution.

Although the Queue Table forms part of the System Director, the real-time TSX queue-calling statements (e.g., QUEUE, UNQ, QIFON and VIAQ) are designed as subroutines which may be included in the skeleton or with the mainline at the user's discretion. Processing of a mainline is not suspended as a result of queueing a higher priority mainline.

**Level Work Areas.** A level work area of 104 words (in the skeleton) is required for

- each interrupt level used
- process mainlines
- nonprocess core loads, and
- internal errors

A level work area contains interrupt level instructions, MIC linkages, and work areas. It is used to allow recursive entry to those programs supplied by IBM.

Each of the following System Director functions will now be explained in some detail:

- Program Scheduling
- Handling of Interrupts
- Use of Interval Timers
- Use of Time-Sharing
- Use of the Operations Monitor
- Error Alert Control

**PROGRAM SCHEDULING**

Control processes may be classified under three basic headings:

- Program or event sequence
- Time dependence
- Interrupt initiation

In practice, a process may be a combination of all three categories, but is usually weighted more heavily towards one. Rarely does a process lend itself to only one.
Figure 6 is a simplified version of a process based totally on program sequence. An example might be a crude-oil distillation unit in an oil refinery. A scan is made to see what the present status is, tests and calculations are made to verify the information, optimization towards a given product mix is applied, required changes to process variables are effected, data is recorded, etc. Each event thus depends on the completion of previous events.

A process based on time is illustrated by Figure 7. This classification could be applied to a process involving a solitary engine test stand. For example, a given throttle position and resistance load are set up. At specified time increments, one or more variables are recorded, such as manifold pressure, RPM, fuel flow, fuel level, oil temperature, oil pressure, etc. When all the variables have been recorded, the throttle position and/or load resistance are changed and a subsequent timing cycle initiated. Finally, when all specified combinations of throttle and load resistance settings have been tested, the system is initialized for another engine. Each event in this situation depends on time.

Note that in practice, the servicing of a process as depicted in Figure 7 is not necessarily sequential in nature. Also, it is the actual time period that schedules the servicing of an event. The manner in which servicing takes place is not dictated by the type of program (e.g., mainline, interrupt routine) which initiated the event.

The third classification is illustrated by Figure 8. An example might be the input phase of a hospital information system. With no input information, the system switches over to the time-sharing mode or remains idle. When, however, a patient enters the hospital, certain historical information is punched into cards. An interrupt is then initiated by an operator. The interrupt recognition routine sets up the card read program and the patient data enters the system -- the system then returns to time-sharing or to an idle condition. When, later, a doctor requests medication for a certain patient, in a specified quantity, at certain time increments and duration, he sets up the proper information on a manual entry unit and initiates an interrupt. The interrupt recognition routine again calls the appropriate program which reads in the manual entry, verifies the information, enters it in the specified files, and once again returns the system to the time-sharing mode. In a similar fashion, other input information such as records and/or schedules for dietary, patient status, laboratory, surgery, etc., are entered. Events thus classified are initiated by interrupts.
Figure 8. Interrupt Initiation

It is obvious from the foregoing that if each application illustrated were expanded to its complete operating requirements, it would most likely consist of all three classifications to some degree. For example, in Figure 6, an inventory log of input and output material is given every hour. This is required to update inventories, product costs, etc. Also, an interrupt will occur whenever a heating unit goes out of range. This will immediately initiate a program to rectify the situation.

Multi-Level Control. A control system must be able to immediately recognize certain situations of a physical process. It must also be able to ignore certain functions until they occur. In practice, the first requirement is more critical. In either case, the normal sequence of events will be interrupted until some action is taken. The situation is further complicated if a second interrupt, more critical than the first, occurs during the action phase of the first interrupt. The servicing of the first interrupt must obviously be suspended while attention is given to the more critical interrupt. Such a chain of interrupts may continue through several iterations as shown in Figure 9. Upon completion of the required action of each interrupt, the previously interrupted action must be continued until the system returns to normal. From this brief picture of multi-level operations, we see that program scheduling now becomes more complex. The user must now have the capability to take immediate action, record the occurrence for later action, or arrange for action to be taken as soon as possible, but on a less critical level. To do this, the user defines what is to be recognized on each level and sets this up by machine configurations. Later his program sets up when action is to be taken and at what level.

Figure 9. Example of Multi-Level Processing
Program Scheduling Requirements

In a control system application, the scheduling of programs to be executed on the normal or mainline level constitutes the main problem. During certain phases of a control system, the user will execute programs in a set sequence. This type of sequence may be set up by a program condition, an interrupt, or a given time period. Sequencing or chaining of programs may or may not be required depending on the user's specific application.

A direct sequence or chain of programs is required for two separate situations. The first situation is a set of programs whose functions must be in a given order that cannot be interrupted except for critical conditions. The second derives from a program that is too large for core size available, so it must be segmented into several separate programs. These programs will now overlay each other, and must therefore be scheduled in a fixed sequence.

As illustrated in Figure 6, special sequences of programs may also be required on the mainline level under certain special conditions. These special sequences are required under three conditions which come under normal operation. The first requirement is a sequence or chain of events that is common to several different phases of a system. This is logically equivalent to a subroutine which is called by several programs: the main difference is that a chain of programs is now being scheduled instead of a subroutine. The next requirement occurs when a situation is bordering a critical point, but is still within the limits defined by the user. In this event, the user may want a warning, but has no real need to be alarmed. The third situation is similar except that the user is alarmed and cannot therefore proceed with the present sequence of programs until certain conditions are met. This is a common situation in process control where process inputs are not acceptable and a special scan is set up until valid variables are obtained. As a result, the normal calculation, optimization, etc., are delayed but will be resumed as soon as possible.

The requirements stated thus far are categorized under program sequence since they have a definite relationship and order. Three commands are used to implement sequence control:

1. CALL CHAIN (specify the next program to be executed)
2. CALL SPECL (terminate the program, save it on disk, and execute the next program).
3. CALL BACK (return control to a program which was partially executed).

Multi-process control, however, presents a new scheduling problem. Since one control system is used to control two or more processes, the definite relationship and order of programs is normally applicable within each process but not between processes. However, each process must be able to schedule its own programs in such a manner that the control system can handle all schedules. Also, because each process will normally contain its own unique program sequences, one type of scheduling problem does not necessarily eliminate another. It should also be understood that multi-level processing does not always dictate unrelated program scheduling: all possible combinations must be considered by a given program scheduling situation. The queueing technique itself will not produce such a system, but when combined with the priority technique, the system becomes flexible enough for any control system's requirements. Four commands permit this form of control:

1. CALL QUEUE (enter a core load into a waiting queue)
2. CALL UNQ (remove a core load from a waiting queue)
3. CALL VIAQ (call the highest priority core load waiting in the queue to be executed next in sequence).
4. CALL QIFON (interrogate recorded interrupts)

Program Sequence Control (PSC)

The center of the scheduling system is the Program Sequence Control (PSC) Program which is permanently resident in core in an on-line TSX system working under control of the System Skeleton. PSC is a means by which mainline core loads are loaded to core, and control transferred from one core load to another, according to user specifications. The
user sets up his requirements when he uses a chain or sequence type CALL or a queueing-type CALL statement. PSC performs the following functions:

- Loads all mainline core loads
- Saves and reloads the special core load
- Initializes the ICL Table for each core load
- Tests for errors in calls to load programs

Chaining or Sequence Technique

Chaining or sequence-type call statements permit the programmer to control the order in which tasks are performed, interrupts serviced, and off-line jobs allowed. This control is important since the various levels of control are necessarily carried out in sequence and the order is critical. For example, an optimizing routine too large for core storage can be executed in segmented parts if the programmer has control over their sequence. Three call sequences are used in chaining: 1) CALL CHAIN, 2) CALL SPECL, and 3) CALL BACK. Note that core load names referenced by the CALL statement must also be specified in a FORTRAN EXTERNAL statement. A core load name cannot be the name of a component subprogram of that core load. Figure 10 illustrates the use of these call sequences.

Such statements may be freely embedded within process programs written in FORTRAN or in the Assembler language. Through the use of these commands, within programs, the programmer can control the frequency and order in which the various levels of control are performed. Even when the various levels are not performed on a regular basis, these commands allow control over the sequence. Of equal importance is the ease by which sequence is changed as the process control problem changes with time.

CALL CHAIN -- Normal Call

When a given core load is called for execution, the user sets up the following statement:

```
CALL CHAIN (NAME)
```

where

```
CHAIN = Entry to PSC
and NAME = Name assigned by user to the next sequential core load to be executed
```

This normal call transfers control to PSC, thereby terminating the current mainline core load at its last logical statement. PSC then sets up a disk function to read in the next mainline core load specified by NAME into variable core, overlaying the present core load that contained the CALL CHAIN statement. The new core load thus destroys the previous core load. Once the core load is in core, the disk I/O routine reverts to PSC, which in turn passes control to the new core load.

CALL SPECL -- Special Call

The second type of core load sequence is similar to the CALL CHAIN, except that the current core load and its associated parameters must be saved. This is set up as follows:

```
CALL SPECL (NAME)
```

where

```
SPECL = Special entry to PSC
and NAME = Name assigned by user to a special core load to be executed next
```

The special call suspends execution of the current mainline core load and transfers control to PSC which saves the present variable core area and the return address, but no registers. This information is written to the Special Save Area on disk. Once the save operation is complete, the disk I/O routine returns control to PSC. The operation proceeds from this point exactly as in a CALL CHAIN.

Note that only one mainline core load can be saved. Thus, if a CALL SPECL is used in a core
load that was referenced by another CALL SPECL, the mainline core load saved originally is lost. A core load called by a CALL SPECL may, however, chain to other core loads as long as these core loads do not contain a CALL SPECL (see Figure 10).

**CALL BACK -- Return Saved Mainline**

In order to return to the saved core load, a third call statement becomes necessary. This is used only in conjunction with the special sequence function. It is set up as follows:

```
CALL BACK
```

where

```
BACK = Special entry to PSC
```

CALL BACK transfers control to PSC which, in turn, initiates a disk read operation to load variable core with the information stored in the Special Save Area on disk as the result of a CALL SPECL. When the read operation is complete, the disk I/O routine returns control to PSC, which in turn transfers control to the restored core load, which continues execution at the saved return address (that is, the statement following the CALL SPECL statement).

It should be noted that a CALL BACK statement is required only if the saved core load is to be restored and continued. The user may well initiate a new core load as the result of a special core load. This new core load could then be referenced by a CALL CHAIN or a CALL SPECL.

A core load is terminated or suspended as the result of any of the three calls: CALL CHAIN, CALL SPECL, or CALL BACK. CALL CHAIN and
CALL BACK are the last logical statements executed in a core load. However, a CALL SPECL will not be last logical statement of a core load if a CALL BACK has been executed to restore the saved core load, and to continue execution following the CALL SPECL statement.

Queueing and Priority Techniques

Queueing techniques normally use statistical methods to control the number of queues. The rule that governs the input and output order in which waiting requests are serviced is usually based on an ordered-queue discipline -- that is, first-come, first-served. Since we are considering the use of only one queue, a first-come, first-served control is only valid for a given priority. Therefore, as several priorities are, in practice, required by most control system applications, a priority technique must be enforced. A priority level is one of the most common ways of classifying interrupt requests according to their urgency. Note, however, that the urgency may change as a function of the condition of the servicing system. For example, a request may be given a higher level as waiting-time increases. Priorities are assigned by the user to programs, processes, and functions. The queueing and priority control techniques employed combine to provide a flexible method completely acceptable for scheduling unrelated core loads. Although the call sequences to be described are referred to as queueing calls, both queueing and priority control are implied.

CALL QUEUE -- Insert into Queue

The first of four calls is used to place a core load entry in the Core Load Queue Table (see System Design Considerations: System Director), and to continue with the execution of the present function. The format of the call is:

\[
\text{CALL QUEUE (NAME, P, E)}
\]

where

- **QUEUE** = Name of the subroutine that places the specified core load in the Queue Table.
- **NAME** = Name of user-assigned core load entry to be entered in the Queue Table (and in FLET).
- **P** = Integer expression, specifying queue priority of core load NAME. This may be 1-32767. One (1) is the highest priority number.

- **E** = Designated error procedure to be taken if the queue is full. In each case, an appropriate error message is printed (see Table 4: On-line EAC Errors and Recovery Procedures). The parameter is user-assigned as follows:
  - **E = 0**. Ignore this call, and continue execution as if the core load had been queued.
  - **E = 1 through 32766**. Replace the lowest priority entry currently in the queue with the name and priority specified in this call, if the priority of that current queue entry is lower (that is, numerically larger) than E. If there is no queue entry with a lower priority, execute the restart core load specified for this core load.
  - **E = 32767**. Execute a restart core load.

Note: When two or more entries have the same priority, the entry that is at the lowest core storage location will be executed first.

In practice, E is always set to zero. The size of the Queue Table should be redefined by the user if it becomes saturated. The options listed under E (above) are provided by the Error Alert Control (EAC) program (described later).

Figures 11 and 12 illustrate the use of these functions. In Figure 12, an example is given of a series of mainlines which, if executed serially without interruption, would not allow queue testing for an inordinate amount of time. In order to be able to check the queue in some user-specified time period to see if any high priority core loads need to be executed, a program of the priority of the current executing program is queued; a CALL SPECL is then made to a core load that exits via a CALL VIAQ. The VIAQ routine then checks the queue for the highest priority program and executes it. When the executed program is the core load queued by core load A, a CALL BACK is performed which restores the original calling core load to execution status. This technique is commonly employed to break up the execution of a long program.
When a CALL QUEUE statement is executed, control is transferred to the real-time QUEUE routine which tests for an entry in the Queue Table with the identical name and priority as that specified in the user calling statement. If such an entry exists, a further entry will not be made -- a given core load and priority cannot, by definition, appear more than once in the Queue Table. However, the same core load with varying priorities may appear once for each unique priority.

If the entry is already in the queue, control is passed to the next executable instruction following the CALL QUEUE statement. If this is not the case, the QUEUE routine tests for a Queue-Table-full condition. If the table is full, control passes to EAC which executes the function specified by the E parameter. If the Queue-Table-full condition test is not satisfied, the QUEUE routine will place the core load entry in the Queue Table, and transfer control to the next instruction following the CALL QUEUE statement.

CALL QUEUE may be executed in a program that was initiated by an interrupt or a specified time interval, or as the result of a program decision. It should never be used as the last logical statement of a core load since the QUEUE routine returns control to the instruction immediately following the CALL QUEUE. A CALL ENDT (see Use of Time-Sharing) statement is normally used in conjunction with CALL QUEUE for time-sharing systems. The main uses of CALL QUEUE can be summarized thus:
Problem: Repeated execution of queued core loads during a given core load.

Solution: (The encircled numbers specify the sequence of operations.)

CALL
CHAIN (B)

Note 1: The CALL SPECl statements cause core load A to be saved before transferring to core load E via lines 3 and 8. The CALL BACK statement in core load R causes core load A to be restored before the return is made via lines 6 or 11.

CALL QUEUE (R,2,0)
CALL SPECl (E)

CALL QUEUE (R,4,0)
CALL SPECl (E)

CALL BACK

Note 2: Between lines 4 and 5 all core loads of priorities 1 and 2 will be executed; between lines 9 and 10 all core loads of priorities 1 through 4 will be executed.

Figure 12. Illustrating a Method of Segmenting Mainlines Based on Scheduling Requirements
• To queue a core load from any program
• To queue a core load from any hardware operational level
• To queue a core load when the user is unaware what is presently in progress on any one machine level
• To queue a core load when the user is unaware what machine levels are in progress, and
• To queue a core load that is not related to all other core loads.

This is a very flexible command since related or unrelated core loads can be scheduled on the basis of time, a program decision, an interrupt, and from any hardware operational level.

CALL UNQ -- Delete from the Queue

The reverse of queueing a core load entry is to remove such an entry from the Queue Table in the system. The statement which gives this ability is:

CALL UNQ (NAME, P)

where

UNQ = Name of the subroutine that removes the specified mainline core load entry from the Queue Table
NAME = User-assigned name of mainline core load entry to be removed
P = Priority status of user-assigned core load NAME. This may be in the range 1-32767.

Upon execution of a CALL UNQ statement, control is transferred to the UNQ subroutine which searches the Queue Table for a similar entry of the same name and priority. If such an entry is detected, it is removed (that is, deleted) from the Queue Table. If the table does not contain a matching entry, the Queue Table remains unchanged. In either case, the UNQ subroutine returns control to the instruction immediately following the CALL UNQ statement. Like CALL QUEUE, CALL UNQ may be executed at any time and from any level of machine operation. Note that no error parameter is required.

CALL QIFON -- Queue Core Load if Indicator is On

The third queueing-type call is the CALL QIFON statement.

CALL QIFON (NAME, P, L, I, E)

where

NAME = User-assigned name of a mainline core load
P = Priority status of each NAME, in the range 1-32767.
E = Error parameter, as described for CALL QUEUE
L = Interrupt priority level indicator
I = PISW bit position indicator or CALL COUNT indicators

In TSX, a unique L and I combination parameter is set up for each process interrupt, program-settable interrupt, and program interval timer routine. The significance of this combination (which is dependent on the user's machine configuration) is given below:

<table>
<thead>
<tr>
<th>L</th>
<th>I</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-23</td>
<td>0-15</td>
<td>Process interrupts</td>
</tr>
<tr>
<td>0-23</td>
<td>(-)n</td>
<td>Programmed interrupts</td>
</tr>
<tr>
<td>(-)n</td>
<td>0-31</td>
<td>Subprogram number for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALL COUNT statements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(see Interval Timers)</td>
</tr>
</tbody>
</table>

Minus (-)n above refers to any minus number.

The CALL QIFON function is required only when any of the above mentioned interrupts are set up to be recorded (for delayed servicing). In general, most interrupts call for immediate action, or as soon as their appropriate servicing program can be read from disk to variable core. Some interrupts, however, must be recognized immediately, but do not require action until a later time. The function of delaying servicing is termed "recording": the interrupt is then said to be "recorded". CALL QIFON thus provides the user with the ability to interrogate recorded interrupts only when he so desires. It is the only way a recorded interrupt can be serviced. Figure 13 illustrates the use of this function.

The core load entries are queued only if their respective interrupt record indicators are on. When an indicator is on, the QIFON routine sets up the
proper information and then executes a CALL QUEUE. If the Queue Table is not full, or the replace error option is utilized, the QUEUE routine returns control to QIFON which proceeds with the interrogation of indicators until the QIFON call is completed. A recorded interrupt indicator is automatically turned off (that is, cleared) whenever the QIFON routine interrogates a program indicator. Control is then passed to the next executable instruction following the CALL QIFON statement, or as specified for error conditions under E.
CALL QIFON may be used from any level of machine operation. It should never be used as the last logical statement of a core load.

CALL VIAQ -- Execute Highest Priority Core Load

The fourth and last queueing statement is

```
CALL VIAQ
```

where

```
VIAQ = Name of the subroutine that determines the highest priority core load entry in the Queue Table.
```

The CALL VIAQ statement, like CALL CHAIN, and CALL BACK, is used as the last logical statement of a core load. When executed, control is transferred to the VIAQ routine which interrogates the Queue Table. If the table is empty, the process is considered to be in an idle condition (that is, the process does not require any action at this time.) Since variable core is not utilized in this case, by process core loads, control is passed to the Time Sharing Control (TSC) program for nonprocess work if there is work to do. The Nonprocess Monitor indicates that it has batch work to perform by the execution of the Console Interrupt button, with sense switch 7 on. When the operator places a job stack in the card hopper, he turns on sense switch 7 and depresses the Console Interrupt button. This informs TSC that batch work is to be performed.

At the end of the job, the // END OF ALL JOBS card indicates no more batch work is to be performed until the Console Interrupt button is again depressed. This feature is provided to reduce the amount of disk activity, and to give faster response to the process whenever there is no nonprocess work for execution.

The time-sharing operation, thus initiated, will continue for the duration of time specified at system generation time, or until it is terminated by a CALL ENDTLS statement. Note that a CALL VIAQ is automatically performed when time-sharing terminates. If, therefore, an interrupt program has previously placed a name in the queue, the named core load will then be immediately executed (see also Use of Time-Sharing). Figure 14 illustrates the use of this calling statement.

Problem: All programs of a given priority must be executed before a certain core load.

Solution:

```
Figure 14. Use of the CALL VIAQ Statement
```

In normal operations, the queue might not be empty, in which case the VIAQ routine obtains the name of the entry with the highest priority. If several entries have the same (highest) priority, the first entry of that priority will be selected.

The VIAQ routine then sets up the proper information for a CALL CHAIN with the core load name derived from the Queue Table, and passes control to PSC to execute the CHAIN function exactly as if a CALL CHAIN had been executed. Note that a core load containing a CALL CHAIN statement is destroyed by the core load it calls; a core load containing a CALL VIAQ is, therefore, similarly overlaid in core. The CALL VIAQ and CALL CHAIN commands are similar except for the method of obtaining the name of the core load to be called. Both calls, however, have their own useful unique functions.
Example of Non-synchronous Periodic Scheduling

The following example illustrates a simple technique frequently used in a process control environment whereby core loads can be executed on some periodic time basis. This is known as non-synchronous periodic scheduling. The test case is not intended as a model: it serves only to demonstrate program scheduling techniques. The example is given in three easy steps:

1. The Initial Core Load -- This is the initial mainline core load named START which is read into core by a cold start operation. The core load first unmasks the system because cold start enters the initial core load in an all-level masked condition; it then sets a programmed timer to initiate a continuous cycle of operations (by calling the count routine #0).

   Figure 15 illustrates this core load. The use of CALL CHAIN to call in another core load (that is, ALPHA) is also shown.

2. Mainline Core Load ALPHA -- This is the ALPHA core load called by the initial core load. It is a mainline core load which prints out the time of day (see Figure 16).

   Figure 16 also shows the use of CALL VIAQ to check the queue. If there is nothing in the queue, the system establishes the time-sharing mode (that is, the Nonprocess Monitor is called).

   If an // END OF ALL JOBS has just been executed, the VIAQ routine will wait until an interrupt occurs to check the queue. Every time this wait is interrupted, the operations monitor will be reset.

   If time-sharing is in process (that is, the Nonprocess Monitor is occupied), core is exchanged and the Nonprocess Supervisor is read into core, or alternatively, the interrupted non-process program is brought into core.

3. Count Routine PEROD -- This is the count routine named PEROD which is included in the System Skeleton at system generation time.

   It is entered by way of the Interval Timer Control (ITC) program when the time period specified in the initial core load START, or from its own call, that is CALL COUNT (0, 1, 5), has elapsed.

   The function of PEROD is to end time-sharing and to load ALPHA into the queue, so that when time-sharing is ended and the queue is checked, ALPHA will print out the time. It also restarts the timer to repeat this cycle of operation (that is, it starts the count again).
Basically, in all on-line real time control systems, the processor-controller behaves in very much the same fashion as a radar system. The real-time computer reacts to input data from a real world environment and provides input data to correct or control that environment. For example, a computer system controlling a chemical process monitors the inputs from measuring devices and instrumentation on the operator's control panel. Later, the computer updates the control mechanisms and indicators to maintain safe and efficient operation. Emergency conditions are also sensed and appropriate action initiated. Instrument status sensing, data computation, and reaction control must occur within a specified interval of time to prevent disruption of the process. How well it is able to respond generally determines the maximum capability of the on-line system. A significant component in the responsive ability of any real time system is the inclusion of a powerful and flexible multi-priority interrupt program.

### Purpose of I/O Interrupts

There are two main reasons for I/O interrupts:

1. To reduce system cost by reducing control circuitry in I/O devices
2. To speed up job throughput, which is relatively slow when compared with internal processing.

Consider a normal computer operation without interrupts. Since the computer is basically a sequential machine, it functions sequentially (or serially, performing one job at a time). In the simple example below,

\[
\text{INPUT1 - PROCESS1 - OUTPUT1 - INPUT2 - PROCESS2}
\]

when PROCESS1 is completed, the user must wait until OUTPUT1 and INPUT2 are accomplished before he can begin PROCESS2. This could be extremely time-consuming.

Since the input device waits idly during PROCESS1 and OUTPUT1 time, the question arises: why should this idle interval of time not be used to read in INPUT2? This could be obviated with the use of I/O interrupts. The I/O interrupt is based on the concept of keeping I/O devices active, thus, hopefully, eliminating process delay caused by these devices. The following sequence of events illustrates the type of action that might be taken:

1. A mainline program initiates an I/O device operation.
2. The program proceeds with its processing while the I/O device is sending (or receiving) information.
3. When the I/O device has transferred its information, an interrupt signal is sent to the Process Controller.
4. This interrupts the mainline program.
5. The interruption is serviced; that is, further data is requested or sent.
6. The mainline resumes processing at the point of interruption.
7. The cycle repeats itself during the execution of the program.
1800 Multi-Interrupt Priority Scheme. In the IBM 1800 Time-Sharing Executive System, the essential elements of the multi-interrupt priority control scheme consist of:

- A hardware priority structure
- Core store data areas for each interrupt level
- A Master Interrupt Control Program (MIC) which recognizes, controls, and directs the servicing of interrupts

The hardware priority structure provides for 3 fixed and up to 24 additional interrupt levels which are assignable by the user to I/O, process, or programmed interrupts, as shown in Figure 18.

The interrupt philosophy can be explained in the following way. Because of the large number and widely varying types of interrupt requests, it is often not desirable to cause a branch to a unique address for each condition. For the same reasons, it is not desirable to initiate one branch for all interrupt requests and to require the program to determine the individual requests requiring service. Grouping the numerous request lines into a number of priority levels (see Figure 18) accomplishes two aims:

1. It allows all interrupt requests common to a specific interrupt level to have the privilege of interrupting immediately, if the only requests present are of a lower priority level. Conversely, it permits interrupt requests connected to a higher priority level to temporarily terminate the servicing on a lower level and to immediately interrupt to the higher priority level. Service is returned to the initial request only after all higher level requests have been serviced.

<table>
<thead>
<tr>
<th>INTERRUPT</th>
<th>PRIORITY LEVEL</th>
<th>DECIMAL ADDRESS</th>
<th>ILSW</th>
<th>PISW ASSIGN'T</th>
<th>MASK &amp; UNMASK</th>
<th>PROGRAM INTERRUPT</th>
<th>I/O, TIMER, PROCESS INTERRUPT: ASSIGNMENT ALLOWED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal</td>
<td>1</td>
<td>8</td>
<td>Yes</td>
<td>-</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Trace</td>
<td>26</td>
<td>9</td>
<td>No</td>
<td>-</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CE</td>
<td>27</td>
<td>1</td>
<td>No</td>
<td>-</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Assigned</td>
<td>0</td>
<td>2</td>
<td>Yes</td>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Levels</td>
<td>1</td>
<td>3</td>
<td>Yes</td>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>最基本</td>
<td>2</td>
<td>4</td>
<td>Yes</td>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>Yes</td>
<td>5</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>Yes</td>
<td>6</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
<td>Yes</td>
<td>7</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>8</td>
<td>Yes</td>
<td>8</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>9</td>
<td>Yes</td>
<td>9</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>10</td>
<td>Yes</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>11</td>
<td>Yes</td>
<td>11</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>12</td>
<td>Yes</td>
<td>12</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>13</td>
<td>Yes</td>
<td>13</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>14</td>
<td>Yes</td>
<td>14</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>15</td>
<td>Yes</td>
<td>15</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>16</td>
<td>Yes</td>
<td>16</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>17</td>
<td>Yes</td>
<td>17</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>18</td>
<td>Yes</td>
<td>18</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>19</td>
<td>Yes</td>
<td>19</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>20</td>
<td>Yes</td>
<td>20</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>21</td>
<td>Yes</td>
<td>21</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>22</td>
<td>Yes</td>
<td>22</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>23</td>
<td>Yes</td>
<td>23</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>24</td>
<td>Yes</td>
<td>24</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>25</td>
<td>Yes</td>
<td>25</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. NOTE: 1 Highest priority
    27 Lowest priority
2. 24 PISW's Basic IBM 1800.
3. Manually masked and unmasked by switch.
4. Return address in I-counter stored in decimal address 0010, but hardware-generated BSC addresses decimal address 0001.

Figure 18. Priority Interrupt Level Structure and Assignment
2. Since a unique branch can be defined for each interrupt priority level, it is possible to combine many requests on a common priority level and thereby use a common interrupt subroutine to service many requests.

Each interrupt request line is thus positioned into a table order of priority; the highest priority being closest to the output, while the lowest priority is farthest away. An interrupt request received at a given level automatically causes the level to shift from an uninterrupted to an interrupted state. If no higher priority level is presently being served, the scheme permits the request line to be activated. At this time, a unique address associated with this level is supplied to the system, which transfers control to a core location determined by this address. The mainline return address is now preserved and entry made to the Master Interrupt Control Program to direct the servicing of this interrupt. At completion of servicing, control is returned to the point of departure (see Figure 9).

In this way, every interrupt request is obeyed immediately, provided no priority request is presently in execution. The biggest advantage of this method of priority level control is a near-optimum priority response. To guarantee minimum response time to alarm conditions, most process interrupt servicing routines should be in core at all times.

Characteristics of Interrupts

Interrupts can be classified into three broad types:

- I/O
- External (that is, process), and
- Programmed

Skeleton-resident interrupts operate on a true priority basis from the 24 levels available. An interrupt is, by definition, a hardware feature -- it is the machine hardware, not the Master Interrupt Control Program which determines what level the interrupt is on. As far as the problem programmer is concerned, he has no control over the time of occurrence of process interrupts. He has, however, indirect control of their time response through masking, recording, and the allocation of priority levels. In general, interrupts are distinguishable from one another only in the manner in which they are serviced (see also System Design Considerations).

Priority Assignments. Some important considerations affecting priority assignments can be summarized thus:

- Each of the 24 levels can interrupt the mainline program.
- Level 0 is the highest priority.
- Higher priority levels can interrupt lower priority levels. Lower priority levels cannot interrupt higher levels. This permits fast access devices to interrupt slower ones.
- Hierarchy of machine operation:

<table>
<thead>
<tr>
<th>Highest</th>
<th>Interrupt level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>23</td>
</tr>
</tbody>
</table>

Process Mainline

Nonprocess Mainline

- Interrupt levels may be masked by programming means. Masking inhibits interrupts to the 1800. The user is thus allowed to inhibit or permit specified levels of interrupts, and to allow determination of the status of interrupt levels -- that is, whether they are inhibited or not -- at any time. Through selective use of masking, data channels can be kept in operation for the transmission of data into and out of core storage while process interrupts are prevented from occurring. This gives an increased efficiency of execution of programs.

When a request line is unmasked, the Processor-Controller is interruptible. Note that although a level may be masked, the fact that the interrupt has occurred is not lost. The function of masking is used to delay recognition of an interrupt.
In practice, priorities must be assigned using the interaction of functions with each other as a primary basis. See also System Design Considerations: System Director.

Types of Servicing Subroutines Used

An interrupt servicing subroutine may be

- An I/O device subroutine
- An interrupt subroutine included in the skeleton
- An interrupt routine included with a mainline
- A mainline core load
- An interrupt core load

The different options are provided to permit flexibility in terms of both core storage and response time requirements.

I/O Device Subroutines. An I/O device routine is a routine that performs the second level of sensing of a Device Status Word (DSW) or a Process Interrupt Status Word (PISW). The first level of sensing the Interrupt Level Status Word (ILSW) is carried out by MIC. This means that any bit on the ILSW that requires sensing at the second level may be executed by an I/O device routine.

The majority of the I/O devices in the 1800 have IBM-supplied device routines (e.g., disk, card/read punch). Those that require sensing by the user at the second level include the following:

- RPQ devices
- Special PISW's that the user may wish to sense himself (e.g., multiple PISW groups per level)
- Any other I/O device (e.g., System/360 Channel Adaptor)

These routines are entered with a BSC; they exit by an indirect branch through word (90)10. The appropriate entry reflecting the ILSW will be assigned by the user on *Assignment control cards to the System Loader at system generation time.

Subroutines that are entered from the I/O device routine comprise count, timer, and process I/O subroutines. They perform specific limited tasks associated with the event that is occurring within the I/O device (e.g., elapsed time on a particular timer). Entry to the subroutine is made by a BSI; the routine exits to its return statement by a BSC I through the entry point. These subroutines are included in the skeleton by *INCLD control cards.

Interrupt Subroutines included in the Skeleton. The shortest response time (that is, the minimum time before an interrupt servicing routine is entered after the interrupt has been recognized) is obtained by placing the routine in core with the System Skeleton. The interrupt routine is included by specifying a control card (*INCLD) at skeleton build time. Like the interrupt core load, the in-core interrupt routine performs a limited task. It is masked only for short periods of time by the system during the execution of certain reentrant coded routines. This period of time is normally of the order of 20–30 instructions.

These routines are entered with a BSI; they exit through a CALL INTEX statement. Some of the important factors governing their inclusion in the skeleton area are discussed in detail in System Design Considerations: System Skeleton.

Interrupt Routines included with a Mainline. Next in length of response time to skeleton interrupt routines are in-core routines loaded with the core image mainlines. These are entered almost as quickly as skeleton routines loaded with the core image mainlines. These are entered almost as quickly as skeleton routines provided the mainline is in progress when the interrupt occurs, but may be forced to wait if the mainline is not in core. This will be the case if a lower level interrupt routine has been read over the mainline. The length of delay involved would then be the balance of the reading of the interrupt routine and the execution of that routine and the read-back of the mainline. No immediate exchange to obtain the mainline is done. If the interrupt that occurs has a routine in the mainline and the interrupt is at a higher or equal level to the interrupt being processed, the interrupt core load assigned to this interrupt will be read directly into core upon completion of the interrupt core load being serviced.

An interrupt core load is always required before any servicing of a process interrupt in-core with the
mainline can take place. If an interrupt core load is not available, the event will be recorded even if an interrupt servicing route is included with the mainline. The Master Interrupt Control (MIC) Program first ascertains if an interrupt core load is available; if it is, the ICL table is checked to see if the routine is in the mainline; if it is not available, the event is recorded. The interrupt routine is always serviced with the same masked status as an interrupt core load.

Interrupt routines included with a mainline are always entered by an indirect branch (BSI); they exit through a CALL INTEX.

Mainline Core Loads. External (that is, process) interrupts whose occurrences are recorded are serviced with mainline core loads. The mainline core load performing the servicing action is identical to any other mainline core load, except that it is queued for execution by a CALL QIFON statement. Since it is a queued core load, it should have a CALL VIAQ as its last logical statement. It could, of course, be the first core load of a special series, in which case it would end with a CALL CHAIN to obtain the next core load in sequence, but a CALL VIAQ must ultimately be executed.

Note that the only major difference between an interrupt core load and a mainline core load used for the servicing of recorded interrupts is in the last logical statement used. This must be a CALL INTEX for an interrupt core load and a CALL VIAQ for a mainline core load.

If a process interrupt is immediately serviced on some occasions and recorded on other occasions, it would require two core loads (one for each function) which would be identical in all respects except for their last logical statement. To eliminate this duplication of core loads, a special combination exit statement (CALL DPART) is provided (see Exit Procedures from Interrupt Servicing Routines). An interrupt or mainline core load which terminates with a CALL DPART is, by definition, a combination core load.

The combination core load should not violate restrictions placed on either mainline or interrupt core loads. That is, mainline interrupt subroutines are not allowed as part of this core load; only statements allowed in both mainline and interrupt programs are permitted. See also Appendix B, Summary of TSX Statements.

Interrupt Core Loads. The user may create interrupt core loads which are brought into core over the mainline when the interrupt occurs. Interrupt core loads are essentially disk-resident routines where the response time is not a problem. They are required for those interrupts that meet either of the following conditions:

1. The user has specified the interrupt servicing routine to be out-of-core.
2. The user has specified the interrupt servicing routine to be in-core as part of a mainline core load.

When this type of interrupt servicing routine is executed, the area of core that the routine will occupy is saved on disk before reading in the interrupt core load. The time for this save operation, in addition to the time for the disk read operation needed to get the interrupt core load, causes this method of interrupt servicing to have the longest response time. Once an interrupt servicing core load has begun, it may be interrupted by a higher level routine, only if the interrupt routine for this higher level is in the skeleton on a higher level.

The use of interrupt core loads is normally restricted to the performance of a particular task at a time, or the initiation of a task on a mainline level which does not take an inordinate amount of time. A typical example is the queueing of a sequence of mainline core loads to accomplish the task that originated an interrupt. The user should remember that if his problem program is time-consuming, he will, in the normal course of events, execute this on the mainline level. The reason for this is that interrupt core loads cannot, by definition, interrupt other interrupt core loads. This system restriction is because of the disk exchange time that would be required.

Interrupt core loads are built and assigned to a particular process interrupt bit (PIISW) on programmed interrupt level. The core load then performs the servicing task or sets in motion the task that will be required when this specific bit is activated.

Note that this type of interrupt servicing routine does not contain an Interrupt Status Table (IST). The reason is that the IST is used for updating the Interrupt Core Load Table (ICLT), and the ICL table is only updated from mainline core loads or from combination core loads when these are executed at
the mainline level. For the same reason, the interrupt core load cannot include other routines within it. Another explanation is that the programs that might be included in an interrupt core load are masked off during its execution.

An interrupt core load is not necessarily the length of variable core: it has a defined length (see System Design Considerations: Disk System Configuration). Hence, in contradistinction to mainline core loads, all of variable core is not needed because of the limited function performed by this type of core load. To increase execution speed, therefore, the Interrupt Save Area can be made smaller.

Note also that interrupt core loads can communicate with mainline core loads (and combination core loads when these are executed as interrupt core loads) only through INSKEI COMMON. The interrupt core load itself contains a COMMON which is located at the end of the Interrupt Save Area.

Figure 19 gives a summary of the types, characteristics, and location of process interrupt servicing routines.

<table>
<thead>
<tr>
<th>Type of Routine and Location</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skeleton Interrupt Routine</td>
<td>Permanently in core.</td>
</tr>
<tr>
<td>Core Storage Location</td>
<td>Normally high priority.</td>
</tr>
<tr>
<td>Skeleton Area Variable Area</td>
<td>Can immediately interrupt lower priority routines, and</td>
</tr>
<tr>
<td></td>
<td>Interrupt Core loads if no Interrupt Core load is</td>
</tr>
<tr>
<td></td>
<td>assigned to that level.</td>
</tr>
<tr>
<td></td>
<td>Fastest interrupt response.</td>
</tr>
<tr>
<td></td>
<td>Must CALL INTEX as last logical statement.</td>
</tr>
</tbody>
</table>

| Mainline Interrupt Routine   | Available almost as quickly as Skeleton interrupt routines, if the mainline |
| Core Storage Location        | is in-core. |
| Skeleton Area Variable Area  | Once execution is started, only interruptable by Skeleton Interrupt Routine |
|                              | or internal interrupt. |
|                              | Can be different with each mainline core load. |
|                              | Interrupt core load is required. |
|                              | Must CALL INTEX as last logical statement. |

| Interrupt Core Load          | Large core area available. |
| Core Storage Location        | Once execution is started, only interruptable by Skeleton Interrupt Routine |
| Skeleton Area Variable Area  | or internal interrupt. |
|                              | Mainline or nonprocess program in operation at time of interrupt is saved |
|                              | before and restored after Interrupt Core Load operation. |
|                              | CALL INTEX is last logical statement used. Cannot include interrupt |
|                              | routines for other interrupts. |

| Mainline Core Load           | Large core area available. |
| Core Storage Location        | Can include Interrupt routines. |
| Skeleton Area Variable Area  | Queued for execution if record indicator is on when named in QIFON statement. |
|                              | If mainline core load is always queued, last logical statement should be |
|                              | CALL VIAQ. |

| Combination Core Load        | Cannot violate any rules governing interrupt and mainline core loads. |
| Core Storage Location        | Large core area available. |
| Skeleton Area Variable Area  | Queued for execution if record indicator is on when named in QIFON |
|                              | statement. |
|                              | CALL DPART is last logical statement used. |

Figure 19. Summary of Characteristics of Process Interrupt Servicing Routines
Exit Procedures from Interrupt Servicing Routines

Three forms of exiting are used:

- CALL INTEX
- RETURN
- CALL DPART

CALL INTEX -- Interrupt Exit

All interrupt routines serviced on an interrupt level must return control to MIC through a

CALL INTEX

statement. INTEX is the symbol for INTerrupt EXit. CALL INTEX must be used as the last logical Statement in skeleton interrupt routines. It can also be used in interrupt core loads.

RETURN

Subprograms called by user-written interrupt servicing routines must use a

RETURN

statement to return to the interrupt routine or may return control directly to MIC.

CALL DPART -- Departure

CALL DPART causes the level of operation to be tested for the following conditions:

- If the present level is an interrupt level, a CALL INTEX is executed.
- Otherwise a CALL VIAQ is executed.

Thus CALL DPART eliminates duplication of core loads. An interrupt that is sometimes directly serviced, and sometimes recorded, can now be serviced with the same core load. This core load operates from an interrupt level when servicing is specified; it is queued and operates from the mainline level when the interrupt is specified as recorded.

Figure 20 illustrates the use of the two exit CALL and RETURN Statements.

Master Interrupt Control

Once an interrupt has been detected at the hardware level, a reentrant control program, the Master Interrupt Control (MIC) program, takes over the control and servicing of that interrupt. The interrupt is first recognized by the interrogation of certain indicators on a level.

The MIC routine is assembled as part of the System Director at which time it origins out those tables and coding not used by the system to user specifications. MIC resides in core at all times in an on-line TSX system when the computer is operating under control of the System Skeleton. It is designed to:

- Save the interrupted registers whenever an interrupt is processed on the appropriate work level
- Direct the interrupt to its servicing routine
- Restore the FORTRAN I/O buffers if required
- Restore the interrupted registers, and
- Return to the point of departure in the interrupted program.

Detailed Action of MIC when an Interrupt Occurs

Consider the train of events that follows when a process interrupt is generated by an event within a process control environment. Let us assume that this interrupt was originally assigned (at system generation time) by the user on an NB (System Director) equate card to level zero. Remember that an interrupt is, by definition, a hardware feature, and that the user has limited control over the time of occurrence of process interrupts, except by masking, recording, and the allocation of priority levels. Figures 21, 22 and 23 illustrate this action in simplified form.

Entry to MIC.

1. In the 1800, an interrupt request is recognized at the completion of the current instruction
being executed within a mainline program. When this happens, an indirect branch (BSI) to a fixed-word (location 11) in core takes place. This word contains the start address of a level work area associated with level 0 (see System Design Considerations: System Director). A set of instructions within this area then sets the level busy, saves Index Registers 1, 2 and 3, and sets Index Register 3 as a pointer to this work level (at entry point + 8). It is through the level work area that an interrupt formally enters MIC -- from now on, all references to the work area and saved information is made through the Index Register 3 address.

The sequence of operations (specified by the encircled numbers) can be either 1, 2, 3, 4, 5, 6A, 6B, 6C, 8, 9, 10, or 1, 2, 3, 4, 5, 7A, 7B, 8, 9, 10.

---

**Figure 20.** Use of the CALL INTEX, CALL DPART, and RETURN Statements.
2. MIC is the entry point at which all process (and I/O) interrupts enter the Master Control Program for processing. The accumulator, the status word, and the pseudo-accumulator, are now saved for the particular level of interrupt being processed. The previous (that is, last) work level address is also saved, and the new (that is, current) work level address set up for use by reentrant coded subroutines so that they are aware of the address of the particular work level they are required to use at this particular time. Now that the registers of the interrupted level have been saved and the new level (0) address set up, the question of determining which of 16 possible interrupts is to be serviced on this level remains. This is done by sensing the ILSW. If no bits are "on" in the ILSW, a check is made to see whether a programmed interrupt has been selected for this level; if it has been, a transfer is made to (A) in Figure 22, and the processing procedure proceeds as for a process interrupt. If no programmed interrupt is present, an exit from MIC is made via (B) -- see Figure 23.

3. If a bit is on, a branch is made via the level work area to the Interrupt Branch Table within the mainline core load to determine whether the interrupt is a process or I/O interrupt.

Each core load (mainline, combination, interrupt, or nonprocess) must contain an Interrupt Branch Table which provides the means of routing each I/O, process, or programmed interrupt to its appropriate servicing routine. The table, built in reverse order as shown in Figure 21, consists of single-word entries, each of which contains either an entry address to an I/O device servicing routine for an I/O interrupt, or a fixed address within the Skeleton for a process interrupt. The table is initially built by the Skeleton Builder and Core Load Builder to the specifications of the System Loader. Its size is determined by the number of bits on all interrupt levels used.

Since we are concerned with a process interrupt (LEVEL BIT 0 = PISW, see Figure 21) level 0 will contain the entry point PRIE (that is, the reentry point to MIC). (Note that if an I/O interrupt were present instead, the I/O servicing routine is entered. The case of an I/O interrupt occurrence is discussed later).

The PISW derived from the work level is now sensed. If no bits are on (that is, no event has taken place within the process control environment) the exit route (from MIC) is taken via (B).

If a bit is on, it is reset, and the address of the ICL table associated with this particular interrupt set up.

5. Now that the process interrupt is correctly known, the option of processing must be interrogated and executed — that is, we must now determine what type of servicing this particular process interrupt requires. Various tests are performed to determine:

- Whether the interrupt is to be recorded
- Whether the interrupt servicing routine is in core with the skeleton
- Whether the interrupt is to be serviced by an out-of-core interrupt core load, or
- Whether the interrupt servicing routine is in core with the mainline

in conjunction with entries made in the ICL Table (see System Design Considerations: System Director).

The first test ascertains whether this particular interrupt is to be recorded. If it is, a subroutine records the interrupt. If it is not to be recorded, a check is made to see if the interrupt servicing routine is included with the skeleton. If it is, it is serviced by that subroutine. The next test determines whether an interrupt core load has been loaded to the disk to service this interrupt. If it has not, the interrupt is automatically recorded. If it has, all interrupt levels serviced by out-of-core routines will be masked. This also prevents a user from unmasking any level that is associated with out-of-core interrupts.

A test is now made to determine if the interrupt servicing routine is in core with the mainline program. If it is in core with the mainline, the mainline itself is in core, and we are not in an exchange of variable core; the Index Register is then set to the transfer vector, and the entry point of the interrupt servicing routine is located in the Interrupt Status Table. Entry points to interrupts in core with the mainline are situated in a table known as the Interrupt Status Table (IST). The format of the table consists of:

- One word indicating the length of the table for each level
- One word for interrupts that are in core with the mainline
Figure 21. Action of MIC During an Interrupt
- One word for interrupts which are to be recorded on a particular level followed by as many words as are necessary to contain the start address of interrupts in core with the mainline. The size of the table is determined by the user when he defined his system.

If the interrupt is an out-of-core interrupt, I/O must be completed in the mainline area prior to either exchanging core, or, if we are in an exchange, prior to reading in the interrupt core load. Once the interrupt core load is read into core, Index Register 3 is set to the transfer vector and the interrupt entered for execution. An exchange means that variable core has been saved in the Interrupt Save Area on disk. The area exchanged will be the size of the largest interrupt program specified by the user.

Note that due to cycle stealing I/O, some area may be either modified or recorded at the time the process interrupt occurred. This means that out-of-core interrupts must always be assigned to a priority level lower than all I/O devices.

Exit from MIC. All process interrupt routines terminate by a return CALL INTEX statement to MIC. INTEX is the address to which interrupt servicing routines return upon completion of their processing. If the servicing routine just executed was an out-of-core routine, all out-of-core interrupts are unmasked at this point to allow other out-of-core interrupts to occur, so that it is not necessary to carry out an exchange of variable core for the servicing of that particular interrupt. For both in-core-with-Skeleton and out-of-core routines a common exit from MIC is taken via (C) and (B) - - see Figure 21 and 23 - - if this is the last servicing required (no further PISW bits on). Note that this is also the exit path for all I/O interrupt routines.

If additional process interrupts are indicated

---

Figure 22. Action of MIC during an Interrupt (Continued)
(that is, more bits for PISW sensing are on) the exit routine proceeds to (A) -- see Figure 22 -- and the procedure continues in the normal fashion of a process interrupt. A closed loop is thus maintained until all process interrupts have been serviced, finally exiting through the common exit point (C) for all categories of interrupts. At (C) a test is made to see if a programmed interrupt is requesting service on this level. If yes, the user’s interrupt servicing routine will be executed. This routine will eventually exit to INTEX, and the test at (C) will be made again. Since there can be only one programmed interrupt per level, this time there will be a branch to (B). Here it is decided whether variable core is to restored or not. If variable core has been saved as the result of an interrupt on the same interrupt level as the current level, all out-of-core levels are masked and variable core restored to its proper status which existed prior to the interrupt. The system is then unmasked to the user’s level. If the current interrupt level is of higher priority than the level on which variable core was saved, no restoring is performed since this will be done when the current level has been left, and the lower priority interrupt level is allowed to resume execution. (See Figure 23).

Figure 23. Exit from MIC After an Interrupt Has Been Serviced
The Case of I/O Interrupts. When an I/O device interrupt occurs, a similar procedure to that discussed in 1), 2), and 3) is adopted. In 3), it was mentioned that in the case of an I/O interrupt, the I/O servicing routine will be entered through its entry point in the Interrupt Branch Table (IBT). Some of the important aspects of the I/O device routine are discussed elsewhere in this section.

The last instruction in an I/O device interrupt subroutine is an indirect branch BSCI 90 back to MIC. Before an exit is made through the common exit point (B) -- see Figures 21 and 23) for all categories of interrupts, a check is performed to determine the presence of a programmed interrupt within the two groups of possible programmed interrupts -- group 1 (levels 0-13) and group 2 (levels 14-23). Only the bit associated with a level is tested. If a programmed interrupt is present, a branch is made to (A) and processing proceeds as for process interrupts. The I/O device interrupt, otherwise, undertakes to exit from MIC through the common route (B).

At this point, the FORTRAN I/O buffers are restored to their former state. All interrupt levels are masked, Index Registers 1, 2, and 3, and the accumulator, and words 54 and 55 are restored and the system is unmasked to the user's level. Programmed interrupts are now turned on (they were previously turned off) and a return is made to the interrupted mainline program.

Masking, Servicing, and Recording of Interrupts

An interrupt may occur at any time, but it will not be recognized by MIC until the level on which it is assigned is unmasked and of a higher priority than the current level of machine operation. It is the 1800 hardware, not MIC, that determines which level the interrupt is on. Interrupt levels are user-specified at system generation time. The user may delay any interrupt from being recognized by masking the level on which that interrupt has been assigned. For example, it may be to his advantage to delay the servicing of an interrupt to minimize core exchanges such as when it is known that a program is short and the interrupt can wait. In another situation, he may desire to prevent interrupts entirely from occurring, such as when a routine cannot be reentrant and may be called from more than one level. Once an interrupt has been recognized, MIC will determine if it is to be (1) serviced immediately or (2) recorded for servicing at a later time. Servicing an interrupt may be delayed by the user by simply setting a record option on that interrupt. The options of recording or servicing interrupts immediately may be changed from one mainline core load to another. This designation is made when the core load is initially built. MIC also services interrupts (a maximum of 384) in an optimized sequence within the user's specifications.

Masking of Interrupts

Interrupts can be prevented from occurring by masking. This is accomplished by using four real-time subroutines provided in TSX:

- CALL MASK
- CALL UNMK
- CALL SAVMK
- CALL RESMK

Call Mask. CALL MASK can be used to lock out for some time period those designated interrupt levels on which the user does not want interrupts to occur during some time-dependent programs. This routine gives him the facility to inhibit or mask out groups of interrupt levels (0-13; 14-23) or selectively chosen interrupt levels. The status of levels not designated remain unchanged. The format of this statement is:

```
CALL MASK (I, J)
```

Where I and J are integer expressions which designate the level(s) to be masked. Bits 0-13 of I refer to levels 0-13. Bits 0-9 of J refer to levels 14-23. Each one bit specifies a level to be masked. Both parameters are always required.
EXAMPLE 1. In this and following examples, DATA statements are used in conjunction with the CALL MASK and CALL UNMK statements to set up designated levels. See IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715.

The problem is to mask levels 5, 7, 11, 12, 21, 22 and 23.

```
DATA I, J/Z0518, Z01C0/
CALL MASK (I, J)
```

Call Unmask. CALL UNMK gives the user the ability to unlock an interrupt level -- that is, it allows interrupts to be recognized on a level. Thus, he may, if he wishes, selectively allow or unmask interrupts, one level at a time. This is a required routine (and procedure) for the initial core load -- the first core load called into the system by the Cold Start program. The statement format is

```
CALL UNMK (I, J)
```

Where I and J are integer expressions which designate the levels to be unmasked within the two groups of levels as for CALL MASK.

EXAMPLE 2. The problem is to unmask levels 1, 2, 3, 5, 12, and 21.

```
DATA I, J/Z7408, Z0100/
CALL UNMK (I, J)
```

From Examples 1 and 2 we see that

- Levels 1, 2, 3, 5, 12, and 21 are unmasked,
- Levels 7, 11, 22, and 23 are masked.
- Levels 4, 6, 8, 9, 10, 13-20 are unchanged.

The mask and unmask subroutines maintain a current record of the interrupt level mask status. This is necessary since the system sometimes masks all levels and then restores the status of these levels according to this record. The user should always mask and unmask via these routines to keep this record current.

EXAMPLE 3. The problem is to unmask all levels (as at cold start time).

```
CALL UNMK (-1, -1)
```

Call Save Mask. CALL SAVMK allows the user to save the masked condition (that is, the contents of the current mask words) that existed prior to his calling for masking. The statement format is:

```
CALL SAVMK (I, J)
```

Where I and J are integer variables that will receive the contents of the retained mask words.

For example, a mainline has just masked certain levels of interrupts. The user may not be aware of this condition -- that is, he may not know which bits are on (masked). So, he executes a CALL SAVMK to save this condition prior to masking those levels of interrupt he plans to have masked. When he is ultimately ready to unmask these levels, he executes a CALL RESMK which restores or returns the masked register to its original condition. This acts, effectively, as a mask and unmask routine and is closely analogous to the saving and restoring of registers, etc., during the handling of an interrupt.

Call Restore Mask. CALL RESMK is used to perform a mask and unmask operation to restore the interrupt mask register to its previously saved condition. The variables used as parameters are normally those named in a previous CALL SAVMK statement. Its format is:

```
CALL RESMK (I, J)
```

Where I and J are as for CALL MASK, except that each one bit specifies a level to be masked; each zero bit specifies a level to be unmasked.

EXAMPLE 4. The problem is to mask levels 5, 7, 9, 10, and 12; unmask all other levels.

```
DATA I, J/Z0568, Z0/
CALL RESMK (I, J)
```

Restrictions. It is not possible to unmask an out-of-core interrupt level:

1. while an out-of-core interrupt level specified on the System Director equate cards ICLL1-2 is being serviced,
2. while a mainline core load is being loaded by the Program Sequence Control (PSC) program -- e.g., by CALL CHAIN, CALL BACK, CALL SPECL.
Servicing of Interrupts

In the servicing of interrupts, the answers to three vital questions must be known:

1. What caused the interrupt?
2. How fast is its response?
3. How often does it occur?

In practice, the service action taken depends to a large extent on the frequency of occurrence of an interrupt, and the time required to service it – that is, its servicing time span. There are, in general, four approaches in servicing interrupts:

- The servicing routine may reside in the skeleton.
- It may be located on disk as an interrupt core load.
- The user has the option to include the servicing routine as an integral part of a mainline core load.
- The user has the option to record the interrupt. That is, he may delay its servicing until it is cleared by a CALL CLEAR or serviced by a CALL QIFON.

CALL CLEAR -- Clear Recorded Interrupts

The CALL CLEAR Statement is used to ignore or clear interrupts which have occurred but which were recorded for later servicing. The statement format is:

CALL CLEAR (M, L, I, L, I, ...)

Where M = an integer constant which specifies the number of parameters to follow. If M = 0, all indicators specifying the recorded status are changed to indicate "not recorded".

L and I = as for CALL QIFON (see Program Scheduling).

CALL CLEAR can be used in any process program.

The above four general approaches provide a variety of ways of handling a specific interrupt. For example, an INSKEL interrupt routine may set up a programmed interrupt for a level which is serviced by an out-of-core interrupt core load. This core load may, in turn, be made to queue a mainline core load or a series of mainline core loads to alter, say, the entire user control strategy.

Consider another example. A mainline core load may begin a chain of operations by setting up a programmed interrupt for a specific level. This interrupt may be recorded, or it may be immediately serviced.

The user will always obtain rapid and immediate servicing of interrupts if he (1) includes his interrupts as part of the System Skeleton, (2) does not record these interrupts. Interrupts that reside in core with the skeleton never require an exchange, while those that are included with a mainline core load may require an exchange if a nonprocess program is in memory on a time-sharing operation. If, however, time-sharing is not being used (that is, the mainline core load is in memory) or another interrupt serviced by an interrupt core load is not in progress, interrupts in core with the mainline core load will be serviced almost immediately.

In general, therefore, interrupt servicing routines should be short in execution time. The reason for this is that the 1800 hardware locks out lower priority level interrupts for whatever time that is involved on that level. That portion of the interrupt routine that is not required for execution at this priority level should, therefore, be carried out either at the mainline level or at a lower priority level.

If mainline core loads are used to service interrupts through the queueing technique, then the user must ensure that his mainline core loads do not remain in execution for a period of time that is unacceptable to him prior to checking the Queue Table. A mainline core load may be interrupted by a CALL SPECL in such a core load (see Program Scheduling).

Recording of Interrupts

In general, interrupts may be recorded, that is, deferred service, under any of three different sets of circumstances:

1. When the user has one or more mainline core loads that must be executed within a certain time span.
2. When the user is adjusting or optimizing the process control and creating conditions which would cause interrupts to occur, and he elects to ignore them.
3. The user may wish to record interrupts for later servicing, but he prefers to do this through a CALL QIFON procedure rather than have them serviced on an interrupt level.

Interrupts to be recorded are entered on a *RCORD control card (in any order) and assembled at core load build time. The data set up in the card is later placed into the Interrupt Core Load Table from the Interrupt Status table (within each core load) by PSC.

The action of MIC when an interrupt occurs and the procedural flow through its servicing has already been described elsewhere in this section.

Rules Governing the Servicing of Interrupts

1. If an interrupt is serviced by a subroutine located in the variable area, it must be at a lower priority level (higher number) than the I/O device. This applies to:

   Interrupt and combination core loads
   Interrupt subroutines included with the mainline

The exception to this rule is that an interrupt must be on a level of priority lower than the I/O device it intends to use except for the disk and the 1053 typewriter. DISKN and TYPEN are so written that if either the disk or the typewriter detects that its call was executed from a level with a higher priority, it will remain in itself until the servicing operation is completed. This is achieved by sensing the appropriate Device Status Word (DSW).

2. If a servicing routine does not use any I/O device, it may be on any level, but the routine must be in the skeleton -- not in the variable area of core.

3. Interrupts on levels that are serviced by out-of-core interrupt core loads are serviced in the masked mode so that they cannot be interrupted by another interrupt serviced by an out-of-core routine. Only one level of exchange is maintained.

USE OF INTERVAL TIMERS

In most industrial control installations, some portion of the control of the user's system will require response in time -- that is, the user may want to schedule his programs periodically or at a specific time of day. For example, he may wish to print a shift log on a synchronous basis, say at 8 a.m., 4:30 p.m., and midnight each day; or he may take periodic scans of his process instrumentation once every five minutes; or there may be certain loops to time out.

An interval timer is, by definition, a clocking device which cycles a value contained in a full word of main storage. It thus provides a computer system with the ability to read elapsed time in second or millisecond increments, and to inform the system when a specified period of time has passed.

A simple cyclic timer serves, in effect, both as a basic interval counter and clock. In order to measure an elapsed time interval, a predetermined total count is loaded into the counter word storage by program control and a count down to zero is initiated. As the particular counter reaches zero, an internal interrupt signal is sent to the system.

Information about elapsed time and local time is often required by control computer systems to initiate hourly logs, to time the period between control actions on the process, for process data updating, etc. The time of day is required for printing logs, alarm records, and so on.

Clock interrupts can be used to start a scheduled computer operation. For example, in the control of a complex distillation plant process, periodic interrupts have been used to initiate the recalculation of the reflux ratio required to maintain a desired separation in the tower. In this situation, control of a dependent process quantity is possible through a periodic reexamination of process conditions requiring extensive computer time.

To accomplish the above, the Interval Timer Control (ITC) program provides for FORTRAN language control of three hardware interval timers, A, B, and C which operate on various user-specified time bases (see Table 1). Timers A and B are available to the user, while Timer C is used exclusively by TSX for time-sharing control purposes and as a real-time clock. Furthermore, Timer C is expanded into nine additional programmed interval timers -- thus making available to the user a total of 11 interval timers. As shown in Figure 24, each interval timer is assigned a fixed location in core storage.

ITC also performs three additional functions:

- Resets the Operations Monitor during time-sharing
- Tests for no response from 1053 printers
- Performs end of time-sharing
The establishment of the two principal time bases, the Primary (or Interrupt) Time Base and the Secondary (or Programmed) Time Base, and their relationships to the system are discussed in the section, System Design Considerations: System Director.

Each timer is assigned to a wired-in time base by the user at system generation time, selectable from the table of available time bases given in Table 1.

The .125ms time base is available only on a 2usec machine; the 128ms time base, only on a 4usec machine. Each timer is assigned a permanent time base by the user. Note that a different time base can be selected for each timer, but all three timers (A, B, and C) must be assigned to the same interrupt level. In order to schedule programs based on hours, minutes, or seconds, the wired-in time base for interval timer C must be an even divisor of one second (e.g., .5, 1, 2, 4, 8). The servicing of all interrupts is controlled by ITC.

Table 1. Table of Available Timer Time Bases

<table>
<thead>
<tr>
<th>Core Storage Cycle Times</th>
<th>Available Time Bases (in Milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 usec</td>
<td>.125, .25, .5, 1, 2, 4, 8, 16, 32, 64</td>
</tr>
<tr>
<td>4 usec</td>
<td>.25, .5, 1, 2, 4, 8, 16, 32, 64, 128</td>
</tr>
</tbody>
</table>

Hardware Timers A and B

CALL TIMER

In order to use timers A and B, the system provides a basic call statement:

CALL TIMER (NAME, I, INT)

where

NAME = Name of the user's subprogram that is executed when the specified time elapses. Note that NAME must also appear in a FORTRAN EXTERNAL statement (see IBM 1130/1800 Basic FORTRAN IV Language, Form No. C26-3715).

I = An integer expression whose value must be:
   1 for Timer A (word 00004)
   2 for Timer B (word 00005)

INT = A user-assigned positive integer expression which specifies the number of interval counts before the user's subprogram is executed.

The subprogram specified in a CALL TIMER statement must be in core storage when the interrupt generated by the timer is recognized. The interrupt occurs when the time specified has elapsed, but it is only recognized

1. When the level of current operation is lower than the timer interrupt level, and
2. If the timer level is unmasked.
At the end of the elapsed time, the timer resets itself. Note that, when zero time has been reached, the timer continues to operate -- that is, zero is not a not-busy condition.

In the section System Design Considerations: System Director, it is pointed out that it is the user's responsibility to ensure that the mainline program containing the call statement remain in core until the end of the elapsed specified time -- that is, until the timer times out. He achieves this either by

1. including the subroutine named in the Call Timer Statement in the Skeleton, or
2. masking out all out-of-core interrupt levels, and forbidding a core load exit until the timer interrupts.

Unless previously loaded with the System Skeleton, the subprogram is automatically loaded with the calling mainline core load.

In addition, periodic programs (that is, programs initiated by interval timers) should not, as a rule, be executed on the timer level: they should make use of the programmed interrupt technique.

The following examples assume that the timers specified are called from only one level. If possible, it is preferable not to share timers among two different programs.

EXAMPLE 1. Assume hardware Timer A is wired for the .125ms time base.

CALL TIMER (SCAN1, 1, 35)

When this statement is executed, ITC initializes Timer A (by setting it to -35) and returns control to the next executable instruction following the CALL TIMER statement. When the Primary (or Interrupt) Time Base (= 35 X .125 = 4.375ms) elapses, an interrupt occurs and control passes to the subprogram named SCAN1.

EXAMPLE 2. Assume hardware Timer A is wired for the 1ms time base.

If we assume that the Primary Time Base (= 500ms) for statement 1 in the above coding has not elapsed, the Timer A interrupt will occur 2ms after execution of statement 2 when subprogram MILL2 will be executed. Subprogram MIL5H will never be executed because Timer A was reset before the 500ms time elapsed. Although this condition can be prevented (see Example 3), its logic can prove useful under certain practical conditions.

EXAMPLE 3. Assume identical conditions as for Example 2. This example illustrates the use of the LD functional subroutine in testing for a timer-busy condition.

The format of this function is:

\[ \text{LD}(I) \]

where

\( I \) = A user-assigned integer expression that specifies a core storage address. The contents of this address are moved to the accumulator. This permits a test for busy, etc., of known locations outside of the program area. Timer storage locations are given in Figure 24.
Statement 2 tests if Timer A is busy. If it is busy (that is, negative in core location 00004), a programmed loop is activated until Timer A is no longer busy (that is, when subprogram MIL5H is executed) at which time statement 3 is processed.

EXAMPLE 4. Another example is given to illustrate the use of the LD subroutine function for a test for timer-busy condition.

This test is required if subprogram SUBR7 is not in the skeleton and time-sharing is utilized.

In this example, statement 12 tests if Timer A is busy, and waits until subprogram SUBR7 has been executed before passing to the CALL VIAQ statement.

NOTE: The execution of a machine interval timer busy-test using the LD (I) functional subroutine in an IF statement may fail to indicate the correct busy status if (1) the timer interrupt occurs immediately after the loading of the timer not-busy indication (a zero), and (2) in servicing the interrupt, the timer is reinitialized on another level.

Thus, when a timer is shared by different levels, a solution (see below) would be to follow the first busy-test by a second busy-test in order to prevent an interrupt out of the busy-test.

Sample Coding Form

Sample Coding Form

Real-Time Clock

ITC also provides a programmed real-time clock which keeps time on a 24-hour basis and is updated each time Timer C decrements to zero (that is, it is incremented from 00.000 to 23.999; then returns to 00.000). The clock accuracy is a function of the Primary (or Interrupt) Time Base discussed in the section System Design Considerations: System Director.

CALL SETCL -- Set-up Programmed Real-time Clock

Note that the clock is set at cold start time (a user option), but if it is required to be set at any other time through a user program, the following statement is provided.
CALL SETCL (I)

where

\( I = \) A user-assigned integer expression specifying the time of day setting desired in hours and thousandths of hours (e.g., 8 a.m. = 08000; 10.45 a.m. = 10750)

CALL CLOCK -- Read Programmed Real-time Clock

If the user desires to read the clock, say, for time-recording of his output to the printer, disk, etc., he does so through a

CALL CLOCK (I)

where

\( I = \) A user-assigned integer variable which indicates the core location where the readout time is stored.

Note that the clock is also used by the Error Alert Control (EAC) Program to time-stamp error messages.

Programmed Timers

The mechanism of programmed timers is covered in the section System Design Considerations: System Director.

CALL COUNT

Programmed interval timers are controlled by the following statement.

CALL COUNT (IN, I, INB)

where

\( IN = \) A user-assigned integer constant or integer variable that specifies the number of the program to be executed or recorded when the specified time elapses. The number must be in the range which is established by the System Director NITPI and and NITP2 EQU cards. Its maximum size can be between 0 and 31. The number is assigned at System Skeleton build time. Program numbers are used instead of names to provide the record interrupt option.

\( I = \) An integer expression, identifying the number (1-9) of the programmed timer.

\( INB = \) A user-assigned expression that specifies the number of interval counts before the called program is executed. This number is a function of the Secondary (or Programmed) Time Base.

An additional programmed timer is used as the time-sharing control timer for the allocation of time slicing for non-process operations (see Use of Time Sharing).

EXAMPLE 5. The problem is to queue an analog scan program every five minutes with a priority of 7 if JTEST (a programmed indicator in INSKEL COMMON) is set to zero; if it is non-zero, queue the same program every minute with a priority of 1. Assume the following:

1. Subroutine 19 is SUBROUTINE A which was included in the Skeleton at Skeleton build time by an include card

   *INCLD A/2703

   thus assigning it as count routine number 19.

2. Primary Time Base = 8ms (Timer C wired time base) X 125 (user-assigned number) = 1 second
   Secondary Time Base = 1 (Primary Time Base) X 15 (user-assigned number) = 15 seconds

To solve the problem, a CALL COUNT statement must be given in a mainline core load, thus:

```
SAMPLE CODING FORM
```

<table>
<thead>
<tr>
<th>SAMPLE CODING FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-10</td>
</tr>
<tr>
<td>11-20</td>
</tr>
<tr>
<td>21-30</td>
</tr>
<tr>
<td>31-40</td>
</tr>
<tr>
<td>41-50</td>
</tr>
</tbody>
</table>
This designates that subroutine 19 is to be called in 5 minutes; thereafter, the subroutine calls itself within the specified time period. Its coding is shown in Figure 25.

SCAN is the name of a mainline core load that will be executed at mainline level as the result of a CALL VIAQ when SCAN is the highest priority entry in the queue.

In order to effect immediate execution of the scan routine, the CALL QUEUE statement may be replaced by a CALL LEVEL statement to cause an interrupt on a lower level. This allows the user the flexibility of executing the SCAN routine either as an interrupt core load, an INSKEEL interrupt servicing routine, or as a routine included with a mainline. The advantage is that the timer interrupt level is not tied up. It also gives the user the ability to call other I/O devices within the SCAN routine.

If the time-sharing mode is not used, the CALL ENDTKS statement has no effect. If it is used, the time-sharing clock is set to zero and a return made to the calling program. See Use of Time-Sharing for further action.

A further example is given elsewhere in this section (see Program Scheduling).

Table 2 provides a ready comparison of the salient features in the usage of interval timers and programmed timers.

**USE OF TIME-SHARING**

In many industrial control installations, the user will have a large amount of time that is not utilized by the process being controlled. To allow him to make effective use of this time, the time-sharing feature of the TSX system gives him the ability to compile, assemble, and simulate without taking the system off-line. In this manner, low-priority jobs are automatically interrupted whenever the need arises to execute a higher-priority task. In addition, the inclusion of this feature gives the user the capability of modifying the logic of his control strategy.

---

**SAMPLE CODING FORM**

<table>
<thead>
<tr>
<th>I-10</th>
<th>11-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
<th>51-60</th>
<th>61-70</th>
<th>71-80</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBROUTINE A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMBER 19, IS ASSIGNED TO THIS SUBROUTINE AT SKELETON BUILD TIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXTERNAL SCAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J(C,TST=18,23,18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J(TST,INDICATOR, IN,INSKEEL COMMON)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J(18, JPER=1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPER-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPER-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPER-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET UP FOR PERIOD OF 1 MINUTE AND PRIORITY OF 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60 TO 34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL COUNT (18, A, JPER)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET UP FOR PERIOD OF 5 MINUTES AND PRIORITY OF 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL QUEUE (SCAN, JPER=1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUEUE PERIODIC SCAN CORE LOAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL ENDTKS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TERMINATE TIME-SHARING IF ON</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETURN TO JTC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 25. Subroutine A for Example 5 -- Queuing an Analog Scan Program
Table 2. Comparison of Timers

<table>
<thead>
<tr>
<th>INTERVAL TIMERS 1 &amp; 2</th>
<th>USED WHEN SHORTER TIME BASE IS SPECIFIED</th>
<th>SUBROUTINE CALLED MUST BE IN CORE (IN SKELETON OR INCLUDED WITH MAINLINE) WHEN TIMER ELAPSES</th>
<th>CANNOT BE A RECORDED INTERRUPT</th>
<th>SUBROUTINE IS EXECUTED ON INTERRUPT LEVEL OF INTERVAL TIMERS</th>
<th>EXIT WITH A RETURN STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAMMED TIMERS</td>
<td>USED WHEN LONGER TIME BASE IS NEEDED (E.G., HOURS)</td>
<td>SUBROUTINE MAY OR MAY NOT BE IN CORE WHEN TIMER ELAPSES</td>
<td>IF SUBROUTINE IS NOT IN CORE, IT IS HANDLED AS A RECORDED INTERRUPT</td>
<td>MAY BE EXECUTED ON INTERRUPT LEVEL OR MAINLINE LEVEL</td>
<td>EXIT WITH A RETURN OR CALL VIAQ</td>
</tr>
</tbody>
</table>

Methods of Initiating Time-Sharing

Time-sharing can be initiated in two ways: selectable method (CALL SHARE) and automatic method (CALL VIAQ).

Selectable Method -- CALL SHARE

The user will know at some predetermined point in his program that he wishes to discontinue being in the process mode for a specific period of time. He therefore enters the time-sharing mode by the execution of a CALL SHARE (that is, he gives up control to the Nonprocess Monitor via the CALL SHARE). This statement may be part of the user's process program intended for those special applications where time-sharing is desired without the use of the queueing technique. Its format is as follows:

CALL SHARE (I)

Where I is an integer expression which specifies the number of time intervals allowed for the nonprocess program operation. The basic time interval is assigned by the user at system generation time (see System Design Considerations - System Director; also Use of Interval Timers).

The meaning of the I parameter is clarified by the following example.

EXAMPLE 1. Assume that the secondary time base is 15 seconds (see Use of Interval Timers). Then

<table>
<thead>
<tr>
<th>Time-Sharing Interval Requested</th>
<th>Required Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 minute</td>
<td>CALL SHARE (4)</td>
</tr>
<tr>
<td>5 minutes</td>
<td>CALL SHARE (20)</td>
</tr>
<tr>
<td>30 seconds</td>
<td>CALL SHARE (2)</td>
</tr>
<tr>
<td>1.75 minutes</td>
<td>CALL SHARE (7)</td>
</tr>
</tbody>
</table>

The time-shared operation is terminated whenever the time interval specified by the user has elapsed; it is usually not terminated before. Thus, if 1 minute of time-sharing is indicated, it is usually 1 minute before control is returned to the next executable instruction following the CALL SHARE statement. The exchange time is not part of the 1 minute specification. This 1 minute is the length of the time in the share mode. All interrupt time is allotted against this 1 minute span.

Note that the Nonprocess Monitor will perform a WAIT operation if there are no off-line jobs for execution. Also, interrupts will be serviced as they occur. If an interrupt routine recognizes a need for the process program to resume operation, it can terminate the time-sharing mode by executing the following call:

CALL ENDTX

CALL ENDTX can be used only in an interrupt routine where it sets the time-sharing clock to indicate zero time. The first Timer C interrupt that occurs after control is returned to the nonprocess program causes the time-sharing operation
to be terminated; control then reverts to the process mainline program. Note also that whenever time-sharing is not in force the CALL ENDTS statement is ineffective.

Automatic Method -- CALL VIAQ

The second method uses the queueing technique to load a mainline or combination core load when the Core Load Queue Table is empty, by executing a CALL VIAQ (See Program Scheduling).

Note that a CALL VIAQ (when referenced) forces a CALL SHARE statement for execution when the queue is empty only if the user has indicated through the use of the Console Interrupt button, with sense switch 7 on, that batch work is to be carried out. As a result, the process core load which is in progress, or which has just been completed, is saved on disk and control transferred to the Non-process Monitor (or the nonprocess core load if one had been interrupted and stored on disk). The period of time allocated to time-sharing is specified by the user in a System Director equate card, TISHA, at system generation time. The computer remains in the nonprocess mode for this specified period unless a CALL ENDTS is executed by an interrupt routine.

At the completion of the specified time, another CALL VIAQ is automatically forced by the system. If, in the meantime, a core load has been queued, it is then executed. If the queue remains unchanged (that is, nothing has been added to it), another time-sharing operation will be triggered.

If, at the end of a nonprocess job, the // END OF ALL JOBS card indicates that there is no further nonprocess work for execution, the VIAQ routine will WAIT until either some addition has been made to the queue or the Console Interrupt (C.I.) button is again depressed for the commencement of a new nonprocess job.

This method of entering time-sharing is, in practice, preferred to CALL SHARE. CALL SHARE may, however, be desirable in certain special situations.

Two additional functions performed by the Time-Sharing Control (TSC) program are CALL LINK and CALL EXIT when these are referenced from nonprocess programs.

EXAMPLE 2. (See Program Listing No. 1). In order to illustrate some of the many TSX usages without complex FORTRAN/Assembler language coding, the following example was devised. Note that in this example, the system and list printers have been defined as the same device (1443). In actual practice, the system printer would be a 1053; the list printer, a 1443 or another 1053.

Three analog inputs, A, B, and C, are to be read at 15-second intervals. After C has been read, linear interpolation is used between point A and point B, and between point B and point C. The values A, B, and C are temperatures: the temperatures between A and B, and B and C are linear. The point at which temperature A is taken is 25 feet away from the point where temperature B is taken; similarly for B and C.

A temperature histogram showing temperature versus distance is to be printed on the list printer. A nonprocess program is to be written which simply lists numbers: this program is to be executed in the time-sharing mode.

Timer 2 is used to produce an interrupt every 15 seconds so that one of the three analog inputs may be read.

The skeleton contains a timer service subroutine for Timer 2, called SCAN, which calls programmed interrupt level 7 when 15 seconds have elapsed (that is, SCAN executes a CALL LEVEL (7)). Timer 2 has a base (TBASE) of 1 millisecond.

The problem was solved under TSX using the in-skeleton subroutine SCAN and the following five core loads:

- COLDC
- WAITC
- READC
- CALCC
- SHOWC

Figure 26 illustrates the general problem logic flow:

COLDC (referred to at execution time as C/L #1). This is a mainline core load which is directly called by the cold start program. Its primary function is to unmask all interrupt levels, set timer to 15 seconds, and chain to core load WAITC.

WAITC (referred to at execution time as C/L #2). This core load merely calls VIAQ which results in either a queued program being executed, or the beginning of time-sharing.

READC (referred to at execution time as C/L #3). This is the solitary interrupt core load which is
executed on level 7. The SCAN routine in skeleton executes a programmed interrupt to level 7 each time the 15-second interval elapses. The *STORECI control card for this core load contains level and bit indicators equal to 2407 -- which indicates programmed interrupt level 7.

When this core load is executed, an indicator named ICNT, which is in INSKEL COMMON, is interrogated. If this indicator is 1, the first point A is read, timer 2 is reset (for another 15-second interval), and the core load exits by way of a CALL INTEX.

If the indicator is 2, the second point B is read, the timer is reset, and the core load exits.

If the indicator is 3, the third point C is read, the timer is reset, two core loads CALCC and SHOWC are queued, time-sharing is terminated, and the core load exits via a CALL INTEX.

CALCC (referred to at execution time at C/L #4). CALCC takes the three analog readings, A, B, and C, which have been stored in INSKEL COMMON, interpolates and stores the 51 results back into INSKEL COMMON.

The core load is executed by a CALL VIAQ.

SHOWC (referred to at execution time as C/L #5). SHOWC takes the 51 interpolated results from INSKEL COMMON and outputs a scaled histogram on the list printer. It then calls VIAQ.

NOTE: Each core load prints a message on entry to and on exit from the core load itself. This message identifies the core load as C/L 1, C/L 2, C/L 3, C/L 4, or C/L 5.

This diagnostic message is accomplished by a CALL-type FORTRAN subroutine which is included in the skeleton. Its format is as follows:

CALL ENT (I, J)

where ENT is the name of this subroutine. Either of two messages, depending on the parameters I and J, will be printed:

A) ENTERED C/L NO. . . .
B) EXITED C/L NO. . . .

ENTERED will be printed when I = 1. EXITED will be printed when I = 2.

J is the core load identification number as follows:

J = 1 = COLDC
J = 2 = WAITC
J = 3 = READC
J = 4 = CALCC
J = 5 = SHOWC

The on-line results on the list printer (Program Listing No. 1) also clearly indicate when time-sharing has taken place.

Figure 26. General Problem Logic Flow -- Example 2
PROGRAM LISTING NO. 1: EXAMPLE 2

FLET

```
PACK LABEL
00000

*FIDS 001B 03AC /EPDN TFFFF 038H /EPSV 0780 0422 /INSV 2280 0428 /NP 0098 0488
DUMMY 0092 04AB DUMIN 005A 0489 NONPR 00F0 048A NP 0098 0488
P/SVS 4000 05AC *SKEL 0038 05ED *EPAS 0022 0618 /CLST 0780 063A *E 00F0 0488

FUNCTION COMPLETED
```

(Note: This is the state of FLET before compilations begin.)

```
// JOB
// FOR COLDP
*I0CS(1443PRINTER)
*LST ALL

EXTERNAL SCAN,WAITC
COMMON/INSKEL/I1, I2, I3, INCNT
CALL UNMK(-1,-1)
CALL ENT(I2,1)
INCNT=1
CALL TIMER(SCAN,2,15000)
CALL PNT(2,1)
CALL CHAIN(WAITC)
END

VARIABLE ALLOCATIONS
I1 =FFFF = 12 =FFF5= 13 =FFF0 = INCNT=FFFF

FEATURES SUPPORTED
ONE WORD INTEGERS
IDCS

CALLED SUBPROGRAMS
SCAN WAITC UNMK ENT TIMER CHAIN PNTN EBPR

INTEGER CONSTANTS
1=0004 2=0005 15000=0006

CORE REQUIREMENTS FOR COLDP
COMMON 0 INSKEL COMM 4 VARIABLES 4 PROGRAM 40

END OF COMPILATION

COLDP

FUNCTION COMPLETED

// DUP
*STORECM M COLDP COLDP COLD
*GEND

CLB, BUILD COLDC

CORE LOAD MAP

TYPE NAME ARG1 ARG2

*CDW TABLE 4002 000C
*LIB TABLE 4006 001D
*PID TABLE 402B 0010
*ETV TABLE 4038 000F
*TVT TABLE 404A 001E
*IST TABLE 4058 0036
*PNT TABLE 409E 000C

MAIN COLDP 40B1
PNT COLDC 40B3
PNT COLDP 40A4
CALL UNMK 40D6
CALL ENT 4130
CALL TIMER 415C
PNT WAITC 40A8
LIBF SUBIN 41B2 404A
LIBF COMGO 41EC 4040
LIBF MWRT 43C8 4050
LIBF MD0 447E 4053
LIBF MCUMP 4455 4056
LIBF INR 487A 4059
CALL I0FIX 4932

Functions of Executive Programs 51
This is a genuine TSX warning message. It indicates that core load WAITC was not built at this stage.

// JOB
// FOR WAITP
*LST ALL
*IOCS (1443PRINTER)

CALL ENT(1,2)
CALL ENT(2,2)
CALL VIAQ
END

FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS

CALLED SUBPROGRAMS
ENT VIAQ PANTN EBPR

INTEGER CONSTANTS
1=0001
2=0001

CORE REQUIREMENTS FOR WAITP
COMMON 0 INSKEL COMMON 0 VARIABLES 0 PROGRAM 12

END OF COMPILATION

WAITP
DUP FUNCTION COMPLETED
// DUP
*STORECIM M WAITC WAITP COLDC
*CCEND

CLB, BUILD WAITC

CORE LOAD MAP
TYPE NAME ARG1 ARG2
#COW TABLE 0002 000C
#IST TABLE 000E 0010
#FIO TABLE 0028 0010
#ETV TABLE 0038 000F
#VTV TABLE 0048 001E
#IST TABLE 0068 0036
#PNT TABLE 008E 0008
MAIN WAITP 00A8
PNT WAITC 00A0
PNT COLDC 00A4
CALL ENT 00CF
CALL VIAQ 00EE
LIBF SUBIN 414E 404A
LIBF COMGU 4188 404D
LIBF MWAT 4364 4050
LIBF MIOI 441A 4053
LIBF MCOMP 43F1 4056
LIBF IDU 4816 4059
CALL IDFIX 488E
CALL BTIB 48FE
CALL SAVE 486A
LIBF ADRCK 4962 405C
LIBF FLOAT 4984 405F
LIBF IFIX 49C6 4062
LIBF NORM 49FC 4065
CORE 4A2A 3506

CLB, WAITC LO XO
DUP FUNCTION COMPLETED
EXTERNAL SCAN,CALCC,SHOWC
COMMON/INSKEL/IA1,IA2,IA3,ICNT
CALL ENT(1,3)
L=ICNT
GO TO (5,10,15),L
5 K=76
GO TO 20
10 K=79
GO TO 20
15 K=127
20 CALL AIP((0),JTEST)
GO TO (25,30),JTEST
25 GO TO 20
30 CALL AIP((10000),ITEMP,K)
70 CALL AIP((0),JTEST)
GO TO (71,72),JTEST
71 GO TO 70
72 GO TO (35,40,45),L
35 IA1=ITEMP
GO TO 50
40 IA2=ITEMP
GO TO 50
45 IA3=ITEMP
50 WRITE(*,100) ICNT
100 FORMAT'I,3',ICNT
ICNT=ICNT+1
CALL TIMER(SCAN,2,15000)
GO TO (55,55,55,60),ICNT
55 CALL ENT(2,3)
CALL INTEX
60 ICNT=L
CALL QUEUE(CALCC,1,0)
CALL QUEUE(SHOWC,2,0)
CALL ENTS
CALL ENT(2,3)
CALL INTEX
END
VARIABLE ALLOCATIONS
IA1 =FFFF* IA2 =FFE* IA3 =FFD* ICNT =FFC* L =0000  K =0001  JTEST=0002ITEMP=0003
STATEMENT ALLOCATIONS
100 =0000  S =0023 10 =0029 15 =002F 20 =0033 25 =003D 30 =003F 70 =0044 71 =004E 72 =0050
35 =0057 40 =005D 45 =0063 50 =0067 55 =0081 60 =0087
FEATURES SUPPORTED
ONE WORD INTEGERS
IDCS
CALLED SUBPROGRAMS
SCANDLCC SHOWC ENT AIP TIMER INTEX QUEUE ENTS COMGD MWRT NCOMP MIDI PRNTN EBPRRT
INTEGER CONSTANTS
1=0004 3=0005 76=0006 79=0007 127=0008 0=0009 1000=000A 2=000B 15000=000C
CORE REQUIREMENTS FOR READP
COMMON 0 INSKEL COMMON 4 VARIABLES 4 PROGRAM 156
END OF COMPILATION
READP
DUP FUNCTION COMPLETED
// DUP
*STORECIM 1 READC READP 2407
*CCEND
CLB, BUILD READC
ROC ANINT 0023 LEV.0
CORE LOAD MAP
TYPE NAME ARG1 ARG2
READC is an interrupt core load responding to a programmed interrupt on level 07.
*CD# TABLE 4002 000C
*IB# TABLE 400E 0010
*FD# TABLE 402B 0010
*ET# TABLE 403B 000F
*VT# TABLE 404A 0021
*PN# TABLE 406C 000C
MAIN READP 408C
PN# READP 406E
CALL ENT 4135
LIB# COMGD 4154 404A
CALL AIP 41A6
LIB# MWRT 435C 404D
LIB# MI01 4412 4050
LIB# MCMP 43E9 4053
CALL TIMER 480E
CALL QUEUE 4864
PN# CALCC 4072
PN# SHOWC 4076
CALL ENDS 4926
LIB# SUBIN 4930 4056
CALL QZ010 496A
CALL QZERQ 498E
LIB# AIPTN 49CC 4059
LIB# IOU 444E 405C
CALL IOFIX 4806
CALL BT1BT 4836
CALL SAVE 44A2
LIB# ADRCX 409A 405F
LIB# FLOAT 40EC 4062
LIB# IFIX 40EB 4065
CALL GAGED 4C34
CALL UNGAG 4C45
CALL ANINT 4C5D
LIB# NORM 4090 4068
CORE 40BE 1242
CLB, READC LD XQ
D +5 CORELOADS NOT FOUND
CALCC SHOWC
DUP FUNCTION COMPLETED

// JOB
// FOR CALCP
#LIST ALL
#IOCS (1443PRINTER)

DIMENSION N(51)
COMMON/INSKEL/J1,J2,J3,ICNT,N
CALL ENT(1,4)
WRITE (3,6) J1,J2,J3
  6 FORMAT (16 READING$7,3I10)
      N(I) = J1
      N(26) = J2
      N(51) = J3
DO 4 I = 1,2,25
  4 N(I) = N(I) + ((N(26) - N(I,1,25) * (I-1)
DO 5 I = 27,50
  5 N(I) = N(51) + ((N(26) - N(51))/25) * (51-I)
WRITE (3,7) (N(I),I=1,51)
  7 FORMAT (12110)
CALL ENT(2,4)
CALL VIAQ
END

VARIABLE ALLOCATIONS
 J1 = FFFF* J2 = FFFE* J3 = FFFD* ICNT = FFFC* N = FFFB* 1 = 0002
STATEMENT ALLOCATIONS
 6 = O00E 7 = 0017 4 = O03E 5 = O06A
FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS
CALLED SUBPROGRAMS
ENT VIAQ ISTOX MWRT MCMP MIOIX MIOI SUBSC PRNTN EBPRT

54
INTEGER CONSTANTS
1=0006 3=0008 5=000A 2=0009 25=000B 27=000C 50=000C 51=000D

CORE REQUIREMENTS FOR CALCP
COMMON 0 INSKEL COMMON 56 VARIABLES 6 PROGRAM 170

END OF COMPILATION

CALCP
DUP FUNCTION COMPLETED

CORE LOAD MAP
TYPE NAME ARG1 ARG2
#CDW TABLE 4002 000C
#I8T TABLE 4006 0010
#FIO TABLE 4028 0010
#ETV TABLE 4038 000F
#VTV TABLE 404A 0027
#IST TABLE 4071 0036
#PNT TABLE 40A8 0008
MAIN CALCP 40CA
PNT CALCC 40AA
PNT COLDC 40AE
CALL ENI 4170
LIBF MRT 4326 404A
LIBF MIO1 43DC 4040
LIBF MDCMP 43B3 4050
LIBF ISTOY 4708 4053
LIBF SUBSC 4788 4056
LIBF MIOX 43EB 4059
CALL VIAQ 4824
LIBF SUBIN 4888 405C
LIBF COMGO 48BE 405F
LIBF I0U 4910 4062
CALL IOFIX 49C8
CALL BT1BT 49F8
CALL SAVE 49H4
LIBF ADRC 4A5C 4065
LIBF FLOAT 4AAE 4068
LIBF IFIX 4ACA 4068
LIBF NORM 4AF6 406E
CORE 4824 340C

CLB, CALCC LD X0
DUP FUNCTION COMPLETED

// JOB
// FOR SHOWP
#IOCS (1443PRINTER)

DIMENSION N(51),M(51),L(120)
COMMON/INSKEL/II,12,13,ICNT,N
CALL ENI(3,5)
DO 2 L=I,1,120
2 L(I)=0
DO 3 J=I,1,51
M(J)=N(J)/300
3 M(J)=ABS(M(J))
DO 4 J=I,1,51
K=M(J)/2
4 WRITE (3,100) J,M(J),I=1,K
100 FORMAT (13,1X,98E12) CALL ENI(2,5)
CALL VIAQ
END

VARIABLE ALLOCATIONS
I1 *FFFE* I2 *FFFE* I3 *FFFD* ICNT *FFFC* N *FFFB* M =0032 L =00AA 1K =00AB 1 =00AC M1 =00AD
J =00AE K =00AF

Functions of Executive Programs 55
STATEMENT

ALLOCATIONS
100 =00B8A 2 =00C7 3 =00E9 4 =0109

FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS

CALLED SUBPROGRAMS
ENT IABS VIAQ ISTOX MWRT MCOMP MIDIX MIDI SUBSC PRNTN EBPRT

INTEGER CONSTANTS
1=00B2 5=00B3 120=00B4 0=00B5 51=00B6 300=00B7 2=00B8 3=00B9

CORE REQUIREMENTS FOR SHOWP
COMMON 0 INSKEI COMMON 56 VARIABLES 178 PROGRAM 128

END OF Compilation

SHOWP
DUP FUNCTION COMPLETED
// DUP
*STORECIM SHOWC SHOWP COLDC
*CGEND

CLB, BUILD SHOWC

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 4002 000C
*IEL TABLE 400E 0010
*FIO TABLE 4028 0010
*ETV TABLE 403B 000F
*YIV TABLE 404A 0027
*IST TABLE 4071 003A
*PNT TABLE 4084 0008
MAIN SHOWP 416F
PNT SHOWC 40AA
PNT COLDC 40AE
CALL ENT 41FF
LIBF SUBSC 421E 404A
LIBF ISTOX 424A 404D
CALL IABS 426A
LIBF MWRT 4405 4050
LIBF MIDI 444C 4053
LIBF MIDIX 444D 4056
LIBF MCOMP 449B 4059
CALL VIAQ 48CO
LIBF SUBIN 4920 405C
LIBF COMGO 495A 405F
LIBF ADTCK 49AC 4062
LIBF I0Q 49FE 4065
CALL I0FIX 4A86
CALL BTIB 4A86
CALL SAVE 4A52
LIBF FLOAT 4A6A 406B
LIBF IFIX 4B46 406B
LIBF I0Q 4B86 406E
CORE 4BC0 3440

CLB, SHOWC LD X0

DUP FUNCTION COMPLETED
*DUMPLET F

FLET

PACK LABEL
000000
*FIDS 0018 03A0 /EPDM 7FFF 0388 /EP5V 0780 0422 /INSV 2280 0428 /NP5V 4000 0444 /MESS 0010 0478
DUMMY 0092 0488 DUMIN 005A 0489 NDWPR 00F0 048A NP 0098 048B COLDC 0ABC 048C WATC 028 0495
READC 08BC 049E CALLC 0822 0449 SHOWC 08BE 0482 90UMY 00BC 048C /SP5V 4000 0578 /PRSV 4000 05AC
*SKEL 0038 05EO /EPRG 0022 0618 /CLST 0780 063A ,E 00F0 0488

DUP FUNCTION COMPLETED
Time-Sharing begins here.

Interrupt core load on level 07 takes precedence over nonprocess job. Programmed interrupt level 07 initiated from in-skeleton timer routine called SCAN.

During time-sharing, a nonprocess job is executed and prints out a pattern of numbers in an increasing order of magnitude, as shown. This list of numbers is interrupted by core loads (mainline process or interrupt process) at a higher level.

Third entry of core load REAC calls end time-sharing.
Time-sharing terminates the next time timer C interrupts.

Core load 4 is executed from the QUEUE.

Core load 5 is executed from the QUEUE.

Core load 5 (SHOWC) prints histogram.

Functions of Executive Programs 57
During the printing of the histogram, interrupt core load
READC is brought into core and executed.

The cycle of events repeats itself.
USE OF THE OPERATIONS MONITOR

The Operations Monitor is an optional watch-dog type timer device which warns the user when the processor-controller is not executing a predicted sequence of instructions. This may be caused by power failure, computer hang-up, or computer looping.

The watch-dog timer works on the principle that a contact closes upon completion of a preset time-out period. When this occurs, a separately-powered alarm or indicator (supplied by the user) is brought into play. The time-out period is settable within the range 5-30 seconds. Note that the time interval selected must be greater than the secondary time base specified by the Interval Timer Control (ITC) program.

The user may also exercise the option of manual or automatic reset of the Operations Monitor. This option is specified in the OPMON equate card at system generation time (see System Design Considerations: System Director). Automatic resetting is undertaken by ITC during time-sharing operations. If the Operations Monitor is used, it is the user's responsibility to ensure that a reset (XIO) instruction is executed frequently enough in his program so as to prevent timeout during normal operation. If the reset command is not given during the selected interval, timeout occurs and the alarm circuit is closed.

The Call Operations Monitor subroutine is used to reset the monitor. Its format is:

```
CALL OPMON
```

Consider the following example. A particular program (say, a logging program) has been designed for execution every 15 seconds, and therefore ideally suited for Operations Monitor reset. If the program is not, for some reason, executed within this allowed time span, the Operations Monitor is set, causing an alarm in the warning device the user has attached to the Operations Monitor.

ERROR ALERT CONTROL

Error procedures in the IBM 1800 Time-Sharing Executive System are provided by a program package called the Error Alert Control (EAC) Program which is designed to analyze errors that are:

1. Basic to the hardware, and
2. which may result from incorrect use of software programs.

Since errors affect all real-time systems, from the largest to the smallest, the policy adopted towards all errors is to keep the system on-line if at all possible, and to minimize operator decisions.

NOTE: On all entries to EAC, the system is placed in a fully masked state. On exit from EAC, the system mask registers are restored by the calling program, or PSC, or MIC, or on a system reload, by the cold start core load. All interrupts that occurred while the system was masked are then serviced.

Since EAC masks the whole system, the I/O operations performed by EAC (saving of the part of variable core where the disk resident parts of EAC are to be read in, reading in EDP and error decision program, error update program if hardware error, restoring of the saved area of variable core, printing of the error message on the EAC printer) cannot be overlapped. This means that the system can be masked for quite long. For example, on an error like "1443 not ready" the system will be masked for about 3.5 seconds (about 6 seconds if all of core is dumped to the /EPDM area on disk for later error analysis) with a 1053 typewriter as EAC printer. With a 1443 as EAC printer these times are about 2.5 and 4.0 seconds respectively. These figures are for a 32K system with fast-access disk units.

Features of EAC

Error Conditions Serviced

The Error Alert Control program provides error recovery for the following conditions:

- An input/output error which persists despite repeated corrective action by an I/O subroutine.
- Occurrence of an internal machine error (e.g., invalid operation code, parity, storage protect violation)
- Other control subroutine error conditions (e.g., QUEUE, FORTRAN I/O)

Error Analysis Provisions

Provision is also made for the following features.

Dump of Core Storage to Disk. An optional dump of all core storage to disk is provided if this option is elected through the System Director equate card DUMP1 at system generation time. If, for example, DUMP1 is equated to 1, the DUMP routine is included (at System Director assembly time) with the EAC program package. This feature is only applicable to subroutine type errors.

The DUMP routine writes core into the EDP DUMP AREA (/EPDM) on disk. Before writing this routine tests word 3 of fixed core. If it is negative, core will not be dumped to /EPDM since a negative value means either that the user has not set it (it is initialized to -1) or that core has already been...
dumped to /EPDM on a previous entry to EAC. If word 3 is non-negative but not equal to the actual error code, no saving of core is performed. If word 3 equals the actual error code, it is set negative to prevent /EPDM from being overlaid by later entries to EAC, and core is dumped to /EPDM. Since permanent core may be storage protected, and the disk routine must insert the sector address at the start of each sector to be written, the dump routine moves blocks of six sectors of permanent core to variable core and copies it to disk. After all of permanent core has been copied, that portion of variable core used is restored.

The copied data on disk can now be dumped to an output device by the DUP *DUMP function. The data can also be dumped to cards by the EDMP program for analysis by the dump analysis program DUP. The program is in core.

Error Subroutine. In a process program, EAC branches to a user-written error subroutine if this is included with the mainline core load. This action is bypassed for internal machine errors, if an error subroutine is not included and if a nonprocess program is in core.

A user-written error subroutine can be optionally included with each process core load. The purpose of this subroutine is to allow the user to have control before EAC overlays the variable area with the disk portion of EAC. For example, there may be special data or other information that the user wants to save. Output, such as special core dumps, messages, or contact operate functions, can also be executed. The error subroutine cannot be written in FORTRAN and must be a CALL type subroutine.

Before entering the user's error subroutine, error identification data is placed in words 00115-00119. These words will contain the following:

**Input/Output Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00115</td>
<td>Error type code</td>
</tr>
<tr>
<td>00116</td>
<td>Address of illegal call or address of the device table</td>
</tr>
<tr>
<td>00117</td>
<td>Address of level work area</td>
</tr>
<tr>
<td>00118</td>
<td>Address of originating call</td>
</tr>
</tbody>
</table>

**Queue Overflow**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00115</td>
<td>Error type code</td>
</tr>
<tr>
<td>00116</td>
<td>Word count of core load named in CALL QUEUE</td>
</tr>
<tr>
<td>00117</td>
<td>Sector address of core load named in CALL QUEUE</td>
</tr>
<tr>
<td>00118</td>
<td>Priority of core load named in CALL QUEUE</td>
</tr>
<tr>
<td>00119</td>
<td>Error parameter of core load named in CALL QUEUE</td>
</tr>
</tbody>
</table>

The meaning of on-line EAC error type codes is given in Table 3. Table 4 contains a description of all on-line errors serviced by EAC, the format of each EAC message printout, and corrective action specifications.

A standard recovery procedure is executed by EAC according to the type of error (see Table 4). User options are specified in the same table (see USER OPTION column). However, under certain conditions, EAC overrides the user option. The EAC option is always executed if an error subroutine is not used or the user does not specify an option. Options can be specified by the user before returning to EAC by loading the A-register with -10 for S (RESTART) or -1 for I & R (CONTINUE).

The last logical statement in the error subroutine must be a BSC I entry to the error subroutine.

The core load named for the restart option can be an error analysis core load, or it can be the first of a new series of core loads. If queueing techniques are used, the restart core load can be simply a CALL VIAQ statement (CALL QUEUE can be executed in the restart core load or the error subroutine).

The statements listed below cannot be used in an error subroutine:

- CALL BACK
- CALL QIFON
- CALL CHAIN
- CALL RESMK
- CALL DPART
- CALL SAVMK
- CALL ENDTS
- CALL SHARE
- CALL EXIT
- CALL SPECL
- CALL INTEX
- CALL UNMK
- CALL LEVEL
- CALL VIAQ
- CALL LINK
- CALL MASK

Update Error Counters Maintained on Disk. For each I/O unit on the system, a hardware counter is maintained on the disk for printout to the Customer Engineer for maintenance purposes.

Back-up Capability for D.P. I/O Units. The option of including backup units for the 1053 and the 1816, as well as the logical removal of the 1443 from service, can be specified at system generation time. If backup is not provided, the 1053 printer will be automatically removed from service when multiple failures occur without taking the system off-line.

Backup for the EAC printer is achieved by defining multiple EAC printers at TASK assembly time (if the EAC printer is defined as a 1053). When an output error occurs, or if the unit is not ready (that is, interrupt response is not received), EAC will logically disconnect the unit in error and substit-
tute the backup unit. When backup is initiated because of a hardware malfunction, the message in progress on the failing unit is not continued on the backup device. When the error condition is corrected, the unit can be restored to its original status by using the C. E. Interrupt routine. See C. E. Interrupt Routine in the publication IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

**EAC Program Breakdown**

EAC can be considered in terms of four component parts; each component functions as a separate subprogram, the four parts remaining interdependent insofar as the status information of the error (detected) is shared by all routines concerned. In addition, EAC sets up a level work area for the use of reentrant coded programs when it is processing.
### Table 3. On-Line EAC Error Type Codes

#### EAC MESSAGE FORMAT

<table>
<thead>
<tr>
<th>*INN CL.OCK AC-M PNAME LOCN</th>
<th>* - INDICATES PROCESS CORELOAD IN CORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLANK</td>
<td>BLANK - INDICATES NON-PROCESS CORELOAD IN CORE</td>
</tr>
<tr>
<td>I</td>
<td>I - GENERAL I/O</td>
</tr>
<tr>
<td>P</td>
<td>P - PROCESS I/O</td>
</tr>
<tr>
<td>F</td>
<td>F - FORTRAN</td>
</tr>
<tr>
<td>Q</td>
<td>Q - QUEUE</td>
</tr>
<tr>
<td>M</td>
<td>M - MASK</td>
</tr>
<tr>
<td>X</td>
<td>X - MISCELLANEOUS</td>
</tr>
<tr>
<td>NN</td>
<td>NN - TWO DIGIT NUMBER INDICATING TYPE OF ERROR</td>
</tr>
<tr>
<td>CL.OCK</td>
<td>CL.OCK - TIME IN THOUSANDS OF AN HOUR</td>
</tr>
<tr>
<td>AC</td>
<td>AC - AREA CODE FOR THE ASSOCIATED I/O DEVICE</td>
</tr>
<tr>
<td>M</td>
<td>M - MODIFIER IF MORE THAN ONE FOR THAT AREA CODE</td>
</tr>
<tr>
<td>PNAME</td>
<td>PNAME - NAME OF THE PROGRAM IN CORE AT THE TIME OF THE MESSAGE (NOT NECESSARILY THE ONE WHICH ORIGINATED THE CALL LEADING TO THE ERROR CONDITION)</td>
</tr>
<tr>
<td>LOCN</td>
<td>LOCN - LOCATION OF THE CALL</td>
</tr>
</tbody>
</table>

#### USER ERROR TYPE CODES FOR DP I/O

| I01 | PARITY |
| I02 | STORAGE PROTECT |
| I03 | ILLEGAL CALL |
| I04 | NOT READY |
| I05 | //BLANK CARD |
| I06 | FEED CHECK |
| I07 | READ-PUNCH CHECK |
| I08 | DATA OVERRUN |
| I09 | WRITE SELECT |
| I10 | NO PRINT RESPONSE |
| I11 | DATA ERROR |
| I12 | INVALID MESSAGE ON DISK |
| I13 | FILE PROTECT ERROR |
| I14 | TAPE ERROR |
| I15 | EXCESSIVE TAPE ERRORS |
| I16 | END OF TAPE |
| I17 | INVALID CALL TO ERROR ROUTINE |
| I18 | NO RESPONSE FROM DISK |
| I19 | INVALID DISK ADDRESS |

#### USER ERROR TYPE CODES FOR PROCESS I/O

| P01 | PARITY DATA OR COMMAND REJECT |
| P02 | STORAGE PROTECT VIOLATION |
| P03 | ILLEGAL CALL |
| P04 | PARITY CONTROL |
| P05 | OVERLAP CONFLICT |
| P17 | INVALID ERROR CODE |

#### USER ERROR TYPE CODES FOR QUEUING

| Q01 | ERROR OPTION IS ZERO - CALL IGNORED |
| Q02 | ERROR OPTION NOT ZERO - NO LOWER PRIORITY IN QUEUE |
| Q03 | QUEUE ENTRY REPLACED BY NEW CALL QUEUE |
| Q04 | QUEUE CALL NOT HONORED - RESTART INITIATED |
| Q17 | INVALID ERROR CODE |

#### USER ERROR TYPE CODES FOR FORTRAN

| F90 | ILLEGAL ADDR COMPUTED IN AN INDEXED STORE |
| F91 | ILLEGAL INT USED IN A COMPUTED GO TO |

### USER ERROR TYPE CODES FOR FORTRAN (CONTINUED)

| F92 | FILE NOT DEFINED |
| F93 | RECORD NUMBER TOO LARGE, ZERO OR NEGATIVE |
| F94 | INPUT RECORD IS IN ERROR |
| F95 | RANGE OF NUMERICAL VALUES IS IN ERROR |
| F96 | OUTPUT FIELD TOO SMALL TO CONTAIN THE NUMBERS |
| F97 | ILLEGAL UNIT REFERENCE |
| F98 | REQUESTED RECORD EXCEEDS ALLOCATED BUFFER |
| F99 | WORKING STORAGE AREA INSUFFICIENT FOR DEFINED FILES |
| F17 | INVALID ERROR CODE |

#### UNEDITED I/O

| F87 | ILLEGAL UNIT REFERENCE |
| F88 | RECORD DOES NOT EXIST FOR READ LIST ELEMENT |

#### USER ERROR TYPE CODES FOR MASK ROUTINES

| M01 | ILLEGAL CALL RESMK |
| M02 | ILLEGAL CALL UNMK |
| M17 | INVALID ERROR CODE |

#### USER ERROR TYPE CODES FOR PROGRAM SEQUENCE CONTROL

| X01 | ILLEGAL CALL BACK |
| X02 | INTERRUPT LEVEL ERROR |
| X03 | CORELOAD NOT LOADED ON DISK |
| X04 | RESTART CORELOAD NOT LOADED ON DISK |
| X17 | INVALID ERROR CODE |

Functions of Executive Programs 61
### Table 4. On-Line EAC Errors and Recovery Procedures

<table>
<thead>
<tr>
<th>ERROR CODE DEC</th>
<th>TYPE CODE EAC</th>
<th>EAC STANDARD EXIT</th>
<th>USER OPTION</th>
<th>ERROR MESSAGE AND COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1053/1816 Printer/Keyboard</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00 00</td>
<td>103</td>
<td>S</td>
<td>N</td>
<td>ILLEGAL CALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103 CL.OCK PNAME LOCN 1053</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>USER MUST CORRECT CALL IN PROGRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1053 NOT READY</td>
</tr>
<tr>
<td>01 01</td>
<td>104</td>
<td>R,S</td>
<td>S*</td>
<td>USER MUST CORRECT CALL IN PROGRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>104 CL.OCK AC-M PNAME 1053 NOT READY</td>
</tr>
<tr>
<td>03 03</td>
<td>104</td>
<td>R,S</td>
<td>R,S</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1816 KEYBOARD NOT READY</td>
</tr>
<tr>
<td>04 04</td>
<td>102</td>
<td>L</td>
<td>N</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STORAGE PROTECT VIOLATION FROM 1816</td>
</tr>
<tr>
<td>05 05</td>
<td>101</td>
<td>S</td>
<td>R</td>
<td>USER MUST CHECK PROGRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KEYBOARD PARITY ERROR</td>
</tr>
<tr>
<td>06 06</td>
<td>101</td>
<td>I</td>
<td>N*</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101 CL.OCK AC-M PNAME 1053 PARITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LAST CHARACTER TYPED MAY BE INVALID</td>
</tr>
<tr>
<td>07 07</td>
<td>110</td>
<td>R</td>
<td>N*</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101 CL.OCK AC-M PNAME NO PRINT RESPONSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NO OP COMPLETE HAS BEEN RECEIVED</td>
</tr>
<tr>
<td>08 08</td>
<td>112</td>
<td>R</td>
<td>N</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>INVALID MESSAGE ON DISK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112 CL.OCK AC-M PNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>THIS MESSAGE IS NOW LOST</td>
</tr>
<tr>
<td><strong>1442 Card Read-Punch</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 0A</td>
<td>103</td>
<td>S</td>
<td>N</td>
<td>ILLEGAL CALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103 CL.OCK PNAME LOCN 1442</td>
</tr>
<tr>
<td>11 08</td>
<td>101</td>
<td>R</td>
<td>S</td>
<td>USER MUST CORRECT CALL IN PROGRAM</td>
</tr>
<tr>
<td>12 0C</td>
<td>102</td>
<td>L</td>
<td>N</td>
<td>LAST CARD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PARITY ERROR</td>
</tr>
<tr>
<td>13 OD</td>
<td>102</td>
<td>L</td>
<td>N</td>
<td>NON-PROCESS RUN OUT, RELOAD UN-READ CARDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STORAGE PROTECT VIOLATION</td>
</tr>
<tr>
<td>14 OE</td>
<td>106</td>
<td>R</td>
<td>S</td>
<td>USER MUST CHECK PROGRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FEED CHECK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>106 CL.OCK AC PNAME 1442 NOT READY</td>
</tr>
<tr>
<td>15 0F</td>
<td>108</td>
<td>R</td>
<td>S</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NON-PROCESS RUN OUT, RELOAD UN-READ CARDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DATA OVERRUN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>108 CL.OCK AC PNAME 1442 NOT READY</td>
</tr>
<tr>
<td>16 10</td>
<td>107</td>
<td>R</td>
<td>S</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NON-PROCESS RUN OUT, RELOAD UN-READ CARDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>READ-PUNCH CHECK</td>
</tr>
<tr>
<td>17 11</td>
<td>105</td>
<td>S</td>
<td>N</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>105 CL.OCK AC PNAME 1442 NOT READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>//BLANK CARD</td>
</tr>
<tr>
<td>19 13</td>
<td>104</td>
<td>R</td>
<td>S</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>104 CL.OCK AC PNAME 1442 NOT READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1442 NOT READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CONTROL CARD ENCOUNTERED - CHECK DECK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1442 NOT READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PRESS START ON UNIT</td>
</tr>
<tr>
<td><strong>1054/1055 Paper Tape Reader/Punch</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 14</td>
<td>103</td>
<td>S</td>
<td>N</td>
<td>ILLEGAL CALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>103 CL.OCK PNAME LOCN 1054</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>USER MUST CORRECT CALL IN PROGRAM</td>
</tr>
<tr>
<td>21 15</td>
<td>101</td>
<td>S</td>
<td>I</td>
<td>PUNCH PARITY ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101 CL.OCK AC PNAME 1055 PARITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LAST CHARACTER TYPED MAY BE INVALID</td>
</tr>
<tr>
<td>22 16</td>
<td>104</td>
<td>R,S</td>
<td>S</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>READER NOT READY</td>
</tr>
<tr>
<td>23 17</td>
<td>104</td>
<td>R,S</td>
<td>S</td>
<td>MAKE READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>104 CL.OCK AC PNAME 1054 NOT READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1055 NOT READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1054 NOT READY</td>
</tr>
</tbody>
</table>

**Legend for EAC Standard Exit and User Option:**

1 - CONTINUE AT THE POINT OF INTERRUPT
R - RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S - RESTART
L - RELOAD
N - NO OPTION - MUST TAKE EAC EXIT
* - INTERNAL BACKUP ATTEMPTED
### Table 4. On-Line EAC Errors and Recovery Procedures

<table>
<thead>
<tr>
<th>ERROR CODE</th>
<th>TYPE CODE</th>
<th>EAC</th>
<th>USER OPTION</th>
<th>ERROR MESSAGE AND COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested</td>
<td>Error</td>
<td>Standard</td>
<td>Exit</td>
<td></td>
</tr>
<tr>
<td>1054/1055 PAPER TAPE READER/PUNCH (Cont'd)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>101</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>102</td>
<td>L</td>
<td>N</td>
</tr>
<tr>
<td>1810 DISK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>103</td>
<td>S</td>
<td>N</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>104</td>
<td>R</td>
<td>S</td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>108</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>109</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>111</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>102</td>
<td>L</td>
<td>N</td>
</tr>
<tr>
<td>1627 PLOTTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>101</td>
<td>S</td>
<td>I</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>104</td>
<td>R, S</td>
<td>S</td>
</tr>
<tr>
<td>1443 PRINTER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>103</td>
<td>S</td>
<td>N</td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>110</td>
<td>R, S</td>
<td>R, S</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>101</td>
<td>S, 1</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>104</td>
<td>R, S</td>
<td>R, S</td>
</tr>
</tbody>
</table>

**Legend for EAC Standard Exit and User Option:**
- **I** - CONTINUE AT THE POINT OF INTERRUPT
- **R** - RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
- **S** - RESTART
- **L** - RELOAD
- **N** - NO OPTION - MUST TAKE EAC EXIT
- ***** - INTERNAL BACKUP ATTEMPTED

Functions of Executive Programs 63
### Table 4. On-Line EAC Errors and Recovery Procedures

<table>
<thead>
<tr>
<th>ERROR CODE DEC</th>
<th>TYPE CODE</th>
<th>EAC STAND., EXIT</th>
<th>USER OPTION</th>
<th>ERROR MESSAGE AND COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANALOG INPUT BASIC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60 3C P03 S</td>
<td>N</td>
<td>ILLEGAL CALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61 3D P02 L</td>
<td>N</td>
<td>STORAGE PROTECT VIOLATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62 3E P04 S</td>
<td>N</td>
<td>PARITY CONTROL ERROR OR A CAR CHECK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63 3F P01 S</td>
<td>N</td>
<td>PARITY ERROR ON DATA OR CONTROL CYCLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 40 P05 S</td>
<td>N</td>
<td>PARITY DATA ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65 41 S</td>
<td>N</td>
<td>INTERMEDIATE INTERRUPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66-68 P17 S</td>
<td>N</td>
<td>DATA TRANSFER COMPLETED DURING CHAIN OPERATION NOT USED</td>
<td></td>
<td>INVALID ERROR CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P17 CL.OCK PNAME AIN INVALID ERROR CODE FROM EAC</td>
</tr>
<tr>
<td><strong>DIGITAL INPUT BASIC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70 46 P03 S</td>
<td>N</td>
<td>ILLEGAL CALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71 47 P01 S</td>
<td>N</td>
<td>PARITY ERROR OR COMMAND REJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72 48 P02 L</td>
<td>N</td>
<td>DATA TRANSMITTED INCORRECY OR ILL. REQUEST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>73 49 S</td>
<td>N</td>
<td>STORAGE PROTECT ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74-79 P17 S</td>
<td>N</td>
<td>PARITY CONTROL ERROR OR A CAR CHECK</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DIGITAL AND ANALOG OUTPUT BASIC</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 50 P03 S</td>
<td>N</td>
<td>ILLEGAL CALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81 51 P01 S</td>
<td>N</td>
<td>PARITY ERROR OR COMMAND REJECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82 52 S</td>
<td>N</td>
<td>DATA TRANSMITTED INCORRECTLY OR ILL. REQUEST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83-89 P17 S</td>
<td>N</td>
<td>INTERMEDIATE INTERRUPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2402 MAG TAPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90 5A I03 S</td>
<td>N</td>
<td>ILLEGAL CALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91 5C I02 L</td>
<td>N</td>
<td>STORAGE PROTECT VIOLATION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92 5C I02 L</td>
<td>N</td>
<td>WRITE INTO MEMORY PROTECTED LOCN ATTEMPTED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LEGEND FOR EAC STANDARD EXIT AND USER OPTION:**
- **R** - Return to the routine which detected the error
- **S** - Restart
- **L** - Reload
- **N** - No option — must take EAC exit
- **I** - Continue at the point of interrupt
- ***** - Internal backup attempted

(Continued)
### Table 4. On-Line EAC Errors and Recovery Procedures

<table>
<thead>
<tr>
<th>ERROR CODE DEC</th>
<th>TYPE CODE</th>
<th>EAC STAND, EXIT</th>
<th>USER OPTION</th>
<th>ERROR MESSAGE AND COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2402 MAG TAPE (Cont'd)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93 5D 113</td>
<td>S</td>
<td>R</td>
<td>COMMAND REJECT</td>
<td></td>
</tr>
<tr>
<td>94 5E 115</td>
<td>S</td>
<td>R</td>
<td>ILL MT OPERATION REQUESTED. USER CHECK PROGRAM</td>
<td></td>
</tr>
<tr>
<td>95 5F 114</td>
<td>R</td>
<td>S</td>
<td>EXCESSIVE TAPE ERRORS</td>
<td></td>
</tr>
<tr>
<td>96-97</td>
<td></td>
<td></td>
<td>TOO MANY FAILS ON THIS REEL. MOUNT NEW REEL</td>
<td></td>
</tr>
<tr>
<td>98 62</td>
<td>S</td>
<td></td>
<td>TAPE ERROR</td>
<td></td>
</tr>
<tr>
<td>99 63</td>
<td>S</td>
<td></td>
<td>PARITY ERROR OR OTHER FAIL CONDITION</td>
<td></td>
</tr>
<tr>
<td>100 64</td>
<td>P90</td>
<td>S</td>
<td>USER OPTION</td>
<td></td>
</tr>
<tr>
<td>101 65</td>
<td>P91</td>
<td>S</td>
<td>INVALID ERROR CODE</td>
<td></td>
</tr>
<tr>
<td>102 66</td>
<td>P92</td>
<td>S</td>
<td>NOT READY</td>
<td></td>
</tr>
<tr>
<td>103 67</td>
<td>P93</td>
<td>S</td>
<td>END OF TAPE</td>
<td></td>
</tr>
<tr>
<td>104 68</td>
<td>P94</td>
<td>S</td>
<td>LOAD CONTAINS FOR FILE RECO</td>
<td></td>
</tr>
<tr>
<td>105 69</td>
<td>P95</td>
<td>S</td>
<td>WORKING STORAGE AREA INS</td>
<td></td>
</tr>
<tr>
<td>106 6A</td>
<td>P96</td>
<td>S</td>
<td>REQUESTED OUTPUT FIELD TOO SMALL</td>
<td></td>
</tr>
<tr>
<td>107 6B</td>
<td>P97</td>
<td>S</td>
<td>REQUESTED RECORD EXCEEDS ALLOCATED BUFFER</td>
<td></td>
</tr>
<tr>
<td>108 6C</td>
<td>P98</td>
<td>S</td>
<td>UNFORMATED I/O</td>
<td></td>
</tr>
<tr>
<td>150 96</td>
<td>F87</td>
<td>S</td>
<td>UNFORMATED I/O</td>
<td></td>
</tr>
</tbody>
</table>

**FORTRAN**

<table>
<thead>
<tr>
<th>ERROR CODE DEC</th>
<th>TYPE CODE</th>
<th>EAC STAND, EXIT</th>
<th>USER OPTION</th>
<th>ERROR MESSAGE AND COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>93 5D 113</td>
<td>S</td>
<td>R</td>
<td>COMMAND REJECT</td>
<td></td>
</tr>
<tr>
<td>94 5E 115</td>
<td>S</td>
<td>R</td>
<td>ILL MT OPERATION REQUESTED. USER CHECK PROGRAM</td>
<td></td>
</tr>
<tr>
<td>95 5F 114</td>
<td>R</td>
<td>S</td>
<td>EXCESSIVE TAPE ERRORS</td>
<td></td>
</tr>
<tr>
<td>96-97</td>
<td></td>
<td></td>
<td>TOO MANY FAILS ON THIS REEL. MOUNT NEW REEL</td>
<td></td>
</tr>
<tr>
<td>98 62</td>
<td>S</td>
<td></td>
<td>TAPE ERROR</td>
<td></td>
</tr>
<tr>
<td>99 63</td>
<td>S</td>
<td></td>
<td>PARITY ERROR OR OTHER FAIL CONDITION</td>
<td></td>
</tr>
<tr>
<td>100 64</td>
<td>P90</td>
<td>S</td>
<td>USER OPTION</td>
<td></td>
</tr>
<tr>
<td>101 65</td>
<td>P91</td>
<td>S</td>
<td>INVALID ERROR CODE</td>
<td></td>
</tr>
<tr>
<td>102 66</td>
<td>P92</td>
<td>S</td>
<td>NOT READY</td>
<td></td>
</tr>
<tr>
<td>103 67</td>
<td>P93</td>
<td>S</td>
<td>END OF TAPE</td>
<td></td>
</tr>
<tr>
<td>104 68</td>
<td>P94</td>
<td>S</td>
<td>LOAD CONTAINS FOR FILE RECO</td>
<td></td>
</tr>
<tr>
<td>105 69</td>
<td>P95</td>
<td>S</td>
<td>WORKING STORAGE AREA INS</td>
<td></td>
</tr>
<tr>
<td>106 6A</td>
<td>P96</td>
<td>S</td>
<td>REQUESTED OUTPUT FIELD TOO SMALL</td>
<td></td>
</tr>
<tr>
<td>107 6B</td>
<td>P97</td>
<td>S</td>
<td>REQUESTED RECORD EXCEEDS ALLOCATED BUFFER</td>
<td></td>
</tr>
<tr>
<td>108 6C</td>
<td>P98</td>
<td>S</td>
<td>UNFORMATED I/O</td>
<td></td>
</tr>
<tr>
<td>150 96</td>
<td>F87</td>
<td>S</td>
<td>UNFORMATED I/O</td>
<td></td>
</tr>
</tbody>
</table>
Table 4. On-Line EAC Errors and Recovery Procedures

<table>
<thead>
<tr>
<th>ERROR CODE</th>
<th>TYPE CODE</th>
<th>EAC</th>
<th>USER OPTION</th>
<th>ERROR MESSAGE AND COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FORTRAN (Cont'd)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>151 97 F88 S N</td>
<td></td>
<td></td>
<td>READ LIST EXCEEDS LENGTH OF WRITE LIST F88 CL, OCK PNAME LOCN LOCN CONTAINS THE ADDRESS WITHIN THE I/O AREA WHERE THE ERROR OCCURRED LIST IN READ STATEMENT IS LONGER THAN LIST IN CORRESPONDING WRITE STATEMENT RECORD DEP'S NOT EXIST FOR READ LIST ELEMENT F89 CL, OCK PNAME LOCN LAST PHYSICAL RECORD OF LOGICAL RECORD HAS BEEN EXHAUSTED</td>
<td></td>
</tr>
<tr>
<td>152 98 F89 S N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MISCELLANEOUS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110 6E X01 S N</td>
<td></td>
<td></td>
<td>PSC CALL BACK ERROR X01 CL, OCK PNAME LOCN CALL BACK TRIED BEFORE CALL SPECIAL CORELOAD NOT LOADED ON DISK X03 CL, OCK PNAME COREN COREN - CORELOAD NOT LOADED RESTART CORELOAD NOT LOADED ON DISK X04 CL, OCK PNAME COREN COREN - CORELOAD NOT LOADED INVALID ERROR CODE X17 CL, OCK PNAME CLB INVALID ERROR CODE FROM EAC QUEUE CALL IGNORED ERROR OPTION ZERO Q01 CL, OCK WC SA P WC - 5 DIGIT WORD COUNT SA - 5 DIGIT SECTOR ADDRESS P - 5 DIGIT PRIORITY</td>
<td></td>
</tr>
<tr>
<td>120 78 Q02 S N</td>
<td></td>
<td></td>
<td>QUEUE CALL NOT HONORED - NO LOWER PRIORITY IN QUEUE ERR OPTION 1 TO 32766 Q02 CL, OCK WC SA P QUEUE CALL HONORED - CALL ENTERED IN QUEUE ERR OPTION 1 TO 32766 Q03 CL, OCK WC SA P REPLACES WC SA P</td>
<td></td>
</tr>
<tr>
<td>120 78 Q04 S N</td>
<td></td>
<td></td>
<td>QUEUE CALL NOT HONORED - RESTART INITIATED ERR OPTION 32767 Q04 CL, OCK WC SA P INVALID ERROR CODE Q17 CL, OCK PNAME QUE INVALID ERROR CODE FROM EAC CALL RESMK ERROR M01 CL, OCK PNAME LOCN ATTEMPT TO UNMASK OUT OF CORE INTERRUPT LEVEL WHILE IN AN OUT OF CORE INTERRUPT PROGRAM CALL UNMk ERROR M02 CL, OCK PNAME LOCN ATTEMPT TO UNMASK OUT OF CORE INTERRUPT LEVEL WHILE IN AN OUT OF CORE INTERRUPT PROGRAM INVALID ERROR CODE M17 CL, OCK PNAME MSK INVALID ERROR CODE FROM EAC INTERRUPT LEVEL ERROR X02 CL, OCK PNAME LOCN ATTEMPT TO CALL LEVEL UNDEFINED FOR SYSTEM INVALID ERROR CODE X17 CL, OCK PNAME LEV INVALID ERROR CODE FROM EAC</td>
<td></td>
</tr>
</tbody>
</table>

**LEGEND FOR EAC STANDARD EXIT AND USER OPTION:**

1 - CONTINUE AT THE POINT OF INTERRUPT
R - RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S - RESTART
L - RELOAD
N - NO OPTION - MUST TAKE EAC EXIT
* - INTERNAL BACKUP ATTEMPTED
### Table 4. On-Line EAC Errors and Recovery Procedures

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTERNAL ERRORS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAR CHECK ERROR</td>
<td>996</td>
<td>CL.OCK PNAME OPTION SKELETON - RELOAD VARIABLE - RESTART</td>
</tr>
<tr>
<td>OP CODE VIOLATION</td>
<td>997</td>
<td>CL.OCK PNAME OPTION SKELETON - RELOAD VARIABLE - RESTART</td>
</tr>
<tr>
<td>STORAGE PROTECT VIOLATION</td>
<td>998</td>
<td>SKELETON - RELOAD VARIABLE - RELOAD</td>
</tr>
<tr>
<td>PARITY ERROR</td>
<td>999</td>
<td>CL.OCK PNAME OPTION SKELETON - RELOAD VARIABLE - RESTART</td>
</tr>
<tr>
<td>OPTION WILL BE RELOAD (IF ERROR IS IN SKELETON), RESTART (VARIABLE CORE - ABORT OF NONPROCESS JOB, OR USER'S RESTART CORE LOAD IF PROCESS), OR COLD START (REQUIRED IF EAC IS UNABLE TO RELOAD SYSTEM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULTIPLE ENTRANCE TO EAC MLPT EAC</td>
<td></td>
<td>AN ERROR HAS OCCURRED WHILE EAC WAS PROCESSING A PREVIOUS ERROR. MUST GO TO A COLD START.</td>
</tr>
<tr>
<td>NORMALLY THIS ERROR INDICATES THAT THE SYSTEM DISK IS DOWN. THIS ERROR WILL ALSO OCCUR IF AN ERROR OCCURS IN EAC WHILE EAC IS ATTEMPTING TO PROCESS A SYSTEM ERROR.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LEGEND FOR EAC STANDARD EXIT AND USER OPTION:**

- **I** - CONTINUE AT THE POINT OF INTERRUPT
- **R** - RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
- **S** - RESTART
- **L** - RELOAD
- **N** - NO OPTION - MUST TAKE EAC EXIT
- ***** - INTERNAL BACKUP ATTEMPTED

(Continued)
an error. Note that the user cannot call from his error subroutines any routine that utilizes more than 14 words of the subroutine work area (a portion of the level work area). This area is principally used for those calls to disk and output printers used by EAC. It may be increased in size if the user elects to remove this restriction.

The EAC program is entered whenever an error occurs or a condition arises that calls for operator intervention. An error message is then given on the EAC printer and the program takes one of five possible exits after proper analysis has determined which exit may be taken for the error in question. Where more than one exit pertains to a given error condition, the user has the option of specifying the exit desired from his (user) error subroutine.

The four component parts are described below.

EAC In-Core. The in-core component of EAC is an integral part of the System Director and resides in core storage at all times. Its main function is to channel one of the several possible types of errors to a specific entry such that information relating to this particular error is passed on correctly to the analysis section. It also saves the current machine status so that after an error has been processed, the exit routine can return the machine back to the user without loss of information. EAC in-core also has the ability to dump variable core to disk if this is specified by the user at System Director assembly time (see System Design Considerations: System Director). This program also determines conditions such as process or nonprocess mode, invalid operation code, parity errors, and user error subroutine availability.

Error Disk Program (EDP). EDP resides permanently on disk, except when it is called to core by the EAC in-core program. Once EDP is in core, it takes the error information from the fixed area and determines what type of error has arisen, the approximate address at which it occurred, and the appropriate error processing subroutine; prior to this, the correct entry addresses for the conversion and error routines are initialized. When the error processing routine has completed its task, certain information such as perform a Cold Start or Restart, or this error was not corrected but we are continuing the process, or this error has been successfully corrected, etc., are passed to the Exit component.

Error Decision Subroutines. These subroutines reside on disk at all times until called to main core by EDP to process a particular error. After the error processing has taken place, a decision is made on the type of recovery procedure required (e.g., Continue processing, Restart, Reload). This information is then passed to the Exit component of EAC for execution.

EAC Exit. This is the means by which a branch is made to the recovery exit prescribed by the Error Decision Subroutine. Note that there is no normal exit from EAC.

Action of EAC When an Error Occurs

Consider the train of events that takes place when an error occurs, as shown in the simplified block diagram, Figure 27. The error may be an Internal Machine Error, a C.E. Interrupt, or a Miscellaneous Subroutine Error which may be an error or condition requiring outside intervention. Depending on the type of error, one of three possible entries is made to EAC, as follows:

- Internal Errors: EAC00
- C.E. Interrupt: EAC01
- Miscellaneous Error: EAC02

The explanatory paragraphs that follow are given in an alphabetic sequence which corresponds exactly to blocks within Figure 27.

A. An Internal Machine Error may be the result of:

- Parity
- An invalid operation code
- A storage protect violation, or
- A Channel Address Register (CAR) check

When such an error occurs, the hardware generates a BSI indirect to EAC00 through word 8 where the processing procedure begins. The return address, the status of the accumulator and its extension, the type of error and certain registers are now saved, and the machine put in a fully masked state. For each
Figure 27. Action of EAC when an Error Occurs
error level, EAC then sets up a work area within the Fixed Area. Note that the user's error routine (if included with his core load) will be ignored on all internal machine errors and nonprocess programs. A direct branch is then made to the variable core procedure (G).

B. A C.E. Interrupt routine forms part of a TSX on-line system to allow the user to check and modify system unit assignments of 1053 output printers, 1443 printers, and 2310 disk drives, and to initiate backup procedures if and when this becomes necessary. This is normally a Customer Engineer responsibility. When the C.E. LEVEL INTERRUPT toggle switch, located on the C.E. panel, is activated, a C.E. interrupt occurs, forcing a BSI to that level, and, after some processing, another BSI is generated to an entry EAC01 in EAC. On entry to EAC01, the return address, the status of the Accumulator and its extension, and certain registers are now saved, and the machine put in a fully masked state. A direct branch is then taken to G.

C. A Miscellaneous Error which is neither an internal machine error nor a C.E. interrupt may be either an error or a condition that requires outside intervention. For example, a not-ready condition on an I/O device has arisen. This condition has been detected by the I/O routine, which then sets up an error code and additional parameters in locations 115 through 119, and finally executes a BSI through location 120 to an entry EAC02 in EAC. On entry to EAC02, the return address, the status of its accumulator and its extension, and certain registers are now saved, and the machine put in a fully masked state.

D. If the dump-core-to-disk option was specified by the user at System Director assembly time, and the Selective Dump Word (word 3 of the Fixed Area) matches the EAC Error Code, the whole of core is written to the Error Dump Area on disk for interrogation at a later date, and the Selective Dump Word is overlayed (replaced by -1). Only the latest error is kept since there is only one Save Area.

E. A determination is now made if the core load in error is a process core load.

F. If it is, a branch is made to the user's error subroutine, if it has been included in the core load, to allow him to perform such processing as he may require for his particular system. This also permits him to modify some system options. Upon return from this routine, any indicator that may have been set is saved. If it is not a process core load, an exit is made to the common variable core procedure at G.

G. At this point, DISKN is called in from the Skele-
Continue. The error is noted, but it is not of such a nature as to interfere with the program in progress. For example, the entry to EAC may have been a C.E. Interrupt or a request to print a message for outside intervention reflecting a not-ready state, a non-fatal error, or a printer parity.

The "continue" recovery action implies that the program proceeds at the point of interruption. Consider an I/O device which has just completed its operation -- an interrupt is generated. This will transfer control to the I/O routine which then determines the correct error condition, and branches to EAC. This exit option bypasses the I/O routine and returns control to the point in the program at which the interrupt developed.

Reload. The Reload recovery routine is brought into core whenever it is suspected that some of the non-storage protected words in permanent core may have been destroyed.

The routine saves the tables necessary for the completion of previous core loads, having first verified that these tables have not been destroyed. The Skeleton (from disk) is then read by sectors into a buffer area. Each word in the buffer is compared with the corresponding word in core. If these words are not identical, the word in core is overlayed by the word in the buffer.

Upon completion of the Skeleton reload, various conditions and indicators are initialized and the routine exits by way of a CALL CHAIN to the initial core load.

If a storage protected word is different to its corresponding word on disk, because of a programming operation or hardware error, the Reload recovery routine prints a message which gives the address of the protected word, at the same time indicating that a cold start must be performed.

Note that if an error has occurred outside of the Skeleton Area, the present core load is aborted and the initial core load is read into main core for execution.

The CAR error may be caused by incorrect usage of the "XIO" instruction or incorrect chaining of data tables, etc. This is always a reload condition.

Restart. An error has occurred which prohibits the present core load from continuing. Three types of "restart" are used:

1. If the error, such as an illegal call, occurred in a process core load, the program in progress is aborted and its restart core load is called into core for execution.
2. If the error occurred in a nonprocess core load, the job is aborted by calling in the Non-process Supervisor.
3. If the error occurred in an interrupt core load, this core load is aborted, and the restart core load of the current process core load is called for execution. This means that the user's restart routines must be written in such a way as to analyze his system and determine what program will be called for execution.

Exit through an Interrupt Level. A restart condition has arisen on a level other than the mainline level. The level on which the error occurred is terminated and the Restart procedure taken when the mainline level is reached.

THE NONPROCESS MONITOR

The Nonprocess Monitor (NPM) is an independent programming system which is designed to function in one of two possible modes within a TSX system:

- In the on-line mode, it operates under the control of the System Director.
- It can also be run in the off-line mode as a dedicated monitor system under TASK control.

The user elects either system (that is, an on-line or off-line system) at system generation time (see System Design Considerations).

The NPM serves a three-fold purpose:

<table>
<thead>
<tr>
<th>Changes to ( )</th>
<th>on a RESTART</th>
<th>on a RELOAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer A, Timer B</td>
<td>If the subroutine to be entered when the timer times out is included in the Skeleton, Timer A and Timer B are left ON, otherwise they are turned OFF.</td>
<td>Turned OFF</td>
</tr>
<tr>
<td>Timer C</td>
<td>Left ON</td>
<td>Set to -1 (shortest time possible) and turned ON</td>
</tr>
<tr>
<td>Real-time clock</td>
<td>Unchanged</td>
<td>Incremented by the approximate reload time (3400ms)</td>
</tr>
<tr>
<td>Programmed timers 1 - 9</td>
<td>Unchanged</td>
<td>Set to zero</td>
</tr>
<tr>
<td>Programmed interrupts</td>
<td>Unchanged</td>
<td>Unchanged</td>
</tr>
<tr>
<td>External interrupts</td>
<td>Unchanged</td>
<td>Turned OFF</td>
</tr>
<tr>
<td>Started interrupt routines</td>
<td>Aborted</td>
<td>Aborted</td>
</tr>
</tbody>
</table>

Figure 27.1. Effects of Restart and Reload on Timers and Interrupts.
1. It permits better computer utilization through time-sharing.

2. It allows the user to compile, assemble, store, delete, and modify programs with extreme flexibility. Because the system programs are resident on disk, only source statements and data cards are required to be read in.

3. It provides for job stacking at the Card Reader, which is fast because less card handling is required. A stacked-job environment permits automatic and uninterrupted operation.

The primary function of the Nonprocess Monitor is to provide continuous processor-controller operation during a sequence of jobs that might otherwise involve several independent programming systems. The monitor coordinates the processor-controller activity by establishing a common communications area in core storage, which is used by the various programs that make up the monitor. It also guides
the transfer of control between monitor programs and the user's programs. Operation is continuous and setup time is reduced to a minimum, thereby effecting a substantial time saving in processor-controller operation and allowing greater programming flexibility.

Figure 28 illustrates the five distinct but interdependent programs which make up the Nonprocess Monitor.

NONPROCESS SUPERVISOR (SUP)

The Nonprocess Supervisor directs and controls the execution of all nonprocess programs which may be either IBM-supplied as part of the TSX package (e.g., FORTRAN Compiler, Assembler, Core Load Builder, Disk Utility Program and Simulator) or user-written. It is composed of several separate but closely-related routines; its two principal components are:

- The Skeleton Supervisor, and
- The Monitor Control Record Analyzer

Skeleton Supervisor. This contains the requisite direction and control logic for the orderly transition of one program to another. The Skeleton Supervisor is read into core storage whenever monitor system operation is initially started, and provides the communications link between monitor programs and user programs.

Monitor Control Record Analyzer. This component of the Nonprocess Supervisor reads the monitor control record, prints its contents on the list and/or System Printer, and calls the appropriate monitor program.

Analysis of monitor control records extends over columns 1-5 only, except for the // JOB card. Invalid control records result in an error message and cause an abort. Blank cards are bypassed and not stacker-selected. The card I/O routine, CARDN, in the skeleton is used; if CARDN is not included by the user, the monitor program uses its own card I/O routine. The // JOB control record resets the abort indicator and the effective address for the Nonprocess Working Storage on disk. It can also specify which of logical disk drives 1 and 2 are expected to be operational, and, accordingly, checks the labels on their disk packs when indicated. The // END control record directs the Nonprocess Supervisor into a wait state.

Specifically, the Nonprocess Supervisor performs the following functions:

1. Analyzes all monitor control records (e.g., // JOB, // ASM, // FOR)
2. Performs JOB initialization
3. Calls and transfers control to the requested monitor program (e.g., FORTRAN Compiler, Assembler)
4. Performs PAUS (that is WAIT) and END OF ALL JOB functions when requested
5. Also analyzes control records for the Core Load Builder following the // XEQ, *STORECI and *SIMULCI.

Method of Operation

The Nonprocess Supervisor, including all monitor programs, must reside on logical disk drive zero where it occupies 21 sectors (see System Design Considerations: IBM Nonprocess System). The first 168 words of the Disk Communications Area (DCOM) of sector 00000 on this disk (the system disk pack) contains the Nonprocess Communications Area which provides the logical linkages between monitor programs and user programs. This area holds address words, error indicators (used by DUP, FOR, ASM, SUP, etc.), the name of the program or core load being executed, as well as a loader for the monitor programs.

DCOM is always brought into core each time a // JOB control record is read. Certain words are then initialized to reflect the current status of the disk as reflected by LET/FLET. Note that recognition of a // JOB control record by the Nonprocess
Supervisor also removes all temporary entries from LET. Whenever a // END or // XEQ control card is encountered, DCOM is written back to disk.

Entry to the Nonprocess Supervisor occurs through a) Console Interrupt, b) a CALL SHARE (or CALL VIAQ) statement in a process mainline, c) FORTRAN Compiler, d) Simulator, or e) Disk Utility Program.

In an on-line TSX system, process interrupts are serviced as they occur, the interrupt servicing time being applied against the time specified by the user for nonprocess operations. As an example, assume a process mainline calls for one minute of time-sharing. This one-minute span is the length of time in the share mode. If, during this period, ten seconds are used up for process interrupt servicing, only fifty seconds are actually available for nonprocess work.

If all nonprocess jobs are completed before the end of the user-specified time, the Nonprocess Supervisor program performs a WAIT operation for the remainder of the time allotted. In other words, if the CALL SHARE statement specified one minute of time-sharing, control is not returned to the process program until one minute has elapsed, or alternatively, a CALL ENDTTS statement is executed by an interrupt routine (see Use of Time-Sharing).

Figure 29 illustrates, in simplified form, Nonprocess Monitor action during time-sharing.

If a nonprocess job is not completed before the specified time is up, it is saved on the disk. When the next CALL SHARE statement is executed, operation of the nonprocess job is resumed at the point of termination.

When an unfinished job is waiting, the CALL SHARE statement causes it to be read in and executed. Otherwise, the Nonprocess Supervisor program is read into core and determines, by checking a program indicator located within the System Director, if any time-sharing operations are to be performed. This indicator is turned on by the execution of a special console interrupt routine, supplied with the system.

The following example illustrates a typical use of the Nonprocess Monitor whenever nonprocess jobs are ready for execution.

1. Operator stacks jobs in Card Reader and starts Reader.
2. Time-Sharing is typically initiated by an operator interrupt, with a coded number set up in the console switches to indicate a time-sharing request.
3. Interrupt routine sets a program indicator to a process mainline.
4. Process mainline calls for time-sharing when it is idle. It specifies the time interval.
5. Nonprocess programs may be interrupted and later continued by an external (that is, process) interrupt or timer interrupt. This will involve an exchange to the disk save area if the interrupt program is not in core, or if the shared period has timed out.
6. Nonprocess jobs are completed in sequence until no jobs remain (program ends on a WAIT instruction) or until // END OF ALL JOBS control record is reached.
7. During time-sharing, the Skeleton Supervisor will be in transient core, identifying monitor control records and initiating monitor programs.
DISK UTILITY PROGRAM (DUP)

DUP is a set of routines designed to aid the user in the day-to-day maintenance of data and programs on disk packs. That is, it has the capabilities of storing, deleting, and outputting user's programs as well as defining system and machine parameters. It also updates the location equivalence table (LET) and maintains other communications areas. The Disk Utility Program is called into operation by a // DUP monitor control record; it can be used on-line or off-line.

The // DUP monitor control record must be followed by at least one DUP control statement that selects the desired routine. DUP control statements are identified by an asterisk in column 1. Columns 2 through 10 contain a symbolic code which identifies the routine (e.g. *STORE, *DELETE, *SEQCH). The columns following the coded routine name provide additional information used by the routine itself.

Like the Nonprocess Supervisor, DUP must reside on logical disk drive zero where it occupies 68 sectors. Primary entry to DUP derives from a) Nonprocess Supervisor, b) FORTRAN Compiler, c) Assembler, and d) Core Load Builder.

DUP uses the card I/O routine, CARDN, if this is included in the skeleton; otherwise, it uses its own card I/O routine. Blank cards are skipped and stacker-selected when searching for control records. Non-DUP or non-monitor control records result in an error message. All DUP control records and messages are printed on both the System and List printers.

Essential data for most DUP functions to communicate with a disk pack include the following:

- Disk sector addresses
- Numeric label in word 0, sector 0
- Disk Communications Area (DCOM) -- This provides information on the size and location of work storage areas, LET for the Relocatable Program Area and FLET for the Core Load Area.
- Valid entries in LET/FLET

A list of all DUP functions is given in the Summary of Nonprocess Monitor Control Records. See also Examples of Nonprocess Monitor Usage.

FORTRAN COMPILER

The TSX FORTRAN Compiler is a disk-resident version of the 1800 card compiler, and occupies 103 sectors on logical disk drive zero. Provision is also made for the user to easily make use of input-output, conversion and arithmetic subroutines that are a part of the TSX subroutine library. The FORTRAN language is described in IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715.

The // FOR monitor control record is used to call the FORTRAN compiler into operation, and to name the mainline program. The compiler reads the control records and source program in card form only. After a successful compilation, the object program in relocatable format is moved to the temporary area on disk, and an entry (name and disk block count) is made in LET. It can, henceforth, be called for execution by an // XEQ control record, or it can be stored permanently in the Relocatable Program Area by a DUP (*STORE) operation. All FORTRAN programs are compiled in relocatable format. A list of FORTRAN control records is given in the summary at the end of this section.

ASSEMBLER

The Assembler program for the 1800 TSX system is a disk-resident version of the 1800 card assembler. It is designed to translate source program statements written in a symbolic format into a binary format which may be stored and/or dumped by the Disk Utility Program (DUP), or executed directly from the Nonprocess Work Storage on disk. The Assembler Language is fully described in IBM 1800 Assembler Language, Form C26-5882.

The Assembler program resides on logical drive zero and occupies seven cylinders. Entry to it is obtained via a // ASM monitor control record. The Assembler accepts control records and source programs in card form only. Upon a successful assembly, the object program in relocatable format is moved to the temporary area on disk where it can be called for execution by a // XEQ control record or stored permanently in the Relocatable Program Area by a DUP (*STORE) function. A list of Assembler control records is given in the summary at the end of this section.
SIMULATOR PROGRAM

The simulator is designed as a debugging aid which allows the user to checkout or test process and/or nonprocess programs without disrupting the normal operations of the TSX system — that is, without taking the system off-line. It functions under the control of the Nonprocess Monitor.

Each instruction in the object program is analyzed for a valid operation code and format before its operation is simulated. In addition, addresses of store and branch instructions are checked to ensure that the instruction would not alter anything outside of the areas of the defined program, COMMON, INSKEI COMMON, or the level work area, if they are actually executed on-line. Process input values may be read from cards or derived from a random number generator. Since System Skeleton routines are used during simulation, it is mandatory that the skeleton area be built before simulation of process core loads can be performed.

Since the primary function of the Simulator is to detect programming errors in the object project, several optional debugging features are available to aid the user. These include Snapshot, Branch Trace and Dump. Simulated COMMON can be dumped on cards so that a run can be executed in several different parts. In addition, the branch and arithmetic trace provided by the FORTRAN Compiler can be operative in the simulator mode.

Simulation runs for process programs are called by a DUP control record, *SIMULCI; runs for nonprocess programs are called by a // SIM monitor control record. Details of operating procedures and stacked-input for a typical simulation run are described in IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Subroutines

General Input/Output

Each time the Simulator encounters a user-called sequence to an I/O subroutine, the location of the calling sequence and the subroutine name are printed on the List printer. Each time the Simulator encounters a subroutine test function (I/O function digit = 0), the following occurs: the first time a test is encountered, a busy return is made; the second time, a not busy return is made. Succeeding entries alternately cause busy and not-busy returns.

Listed below are the general input/output subroutines (IBM-supplied) recognized by the Simulator, and corresponding operations which the Simulator performs:

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARDN (Simulated card subroutine)</td>
<td>Read a card, feed a card, simulate punch a card</td>
</tr>
<tr>
<td>DISKN (Simulated disk subroutine)</td>
<td>Read disk, write disk, simulate disk seek</td>
</tr>
<tr>
<td>MAGT (Simulated magnetic tape subroutine)</td>
<td>Simulates all read, write, and control functions relative to 2401 and 2402 magnetic tape units</td>
</tr>
<tr>
<td>PAPTN (Simulated paper tape subroutine)</td>
<td>Simulate reading paper tape, simulate punching paper tape</td>
</tr>
<tr>
<td>PLOTX (Simulated plotter subroutine)</td>
<td>Simulate plotter output</td>
</tr>
<tr>
<td>PRNTN (Simulated printer subroutine)</td>
<td>Print a line, simulate a carriage operation</td>
</tr>
<tr>
<td>TYPEN or WRYTN (Simulated printer keyboard subroutine)</td>
<td>Simulates printing on 1816 printer keyboard or 1053 printer</td>
</tr>
</tbody>
</table>

The Simulator requires that the card reader, disk, and List printer be physically present on the system.

Process Input/Output

Call sequences which specify input from pulse input points, digital input points, process contact points, and analog input, may obtain input from two sources: cards and a random number generator.

Data cards are used if samples of specific values are desired; the points can be read in a nonprocess program and punched into cards to be read by the Simulator. Any value can be simulated when using cards, but in order to obtain the desired results, the input data must be sequenced according to the flow of the process input subroutines called. In other words, the card feature requires careful ordering of the card deck.

A random number generator, within the Simulator program, produces numbers that fall into a user-specified range. With this option, the user can employ a wide variety of input data to check program operation. A pseudo-process input environment can also be created through the use of a random number generator. All input values are printed on the list printer as they are called.

In the program being simulated, call sequences that specify output for the contact operate, pulse output, digital output, and digital-to-analog output fea-
tures are printed when they are encountered. Input call sequences, error messages, and data are included in the printed output. This provides a complete chronological record of all that occurred during the simulation.

IBM-supplied process input/output subroutines are functionally simulated; that is, the subroutines' call parameters are analyzed according to specifications supplied in the form of control records. The routine name, calling parameters, and data are printed on the List printer. Listed below are the process input/output subroutines recognized by the Simulator, and corresponding operations which the simulator performs. Special-condition returns are also simulated.

**SUBROUTINE**

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIPTN or AIPN (Simulated analog input point)</td>
<td>Simulates the read of a single analog point</td>
</tr>
<tr>
<td>AIRN (Simulated analog input random)</td>
<td>Simulates reading random analog input points</td>
</tr>
<tr>
<td>AISQN or AISN (Simulated analog input sequential)</td>
<td>Simulates reading sequential analog data points</td>
</tr>
<tr>
<td>DAOP (Simulated digital-analog output)</td>
<td>Simulates the transfer of digital or analog information</td>
</tr>
<tr>
<td>DICMP (Simulated digital input read compare)</td>
<td>Simulates the reading in of digital input values under program control and compares these values to a table of user-supplied values. Only the first compare error is detected. A single entry to the special routine is made with appropriate indication. The end-of-table interrupt will not occur if a comparator error occurs.</td>
</tr>
<tr>
<td>DIEXP (Simulated digital input read expand function)</td>
<td>Simulates the reading in of a digital input value and expands it into 1, 2, 4, 8, or 16 words.</td>
</tr>
<tr>
<td>DINP (Simulated digital input hardware functions)</td>
<td>Simulates the reading in of digital input values</td>
</tr>
</tbody>
</table>

**Arithmetic and Conversion Subroutines**

Copies of the IBM-supplied arithmetic and conversion subroutines are contained within the Simulator. It is these copies that are executed when a call to an arithmetic or conversion subroutine is encountered. The requested operations are performed in a pseudo-processing environment maintained under control of the Simulator.

**General TSX Subroutines**

When a call to a TSX control subroutine is recognized by the Simulator, the subroutine name and its calling sequence parameters are printed. There are two categories of subroutines designed for control and communication with the TSX system: the termination class and the functional simulate class.

The following subroutines comprise the termination class, and when encountered, cause the Simulator to execute the close-job procedure:

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACK</td>
<td>PAUSE</td>
</tr>
<tr>
<td>CHAIN</td>
<td>SPECL</td>
</tr>
<tr>
<td>DPART</td>
<td>STOP</td>
</tr>
<tr>
<td>INTEX</td>
<td>VIAQ</td>
</tr>
<tr>
<td>LINK</td>
<td>EXIT</td>
</tr>
</tbody>
</table>

The subroutines listed below comprise the functional simulate class, and when encountered, cause the Simulator to simulate the function, i.e., they analyze the call parameters for validity and print the routine name, the calling parameters, and the data contained within the subroutine.

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>REMSK</td>
</tr>
<tr>
<td>COUNT</td>
<td>SAVMK</td>
</tr>
<tr>
<td>ENDTS</td>
<td>SETCL</td>
</tr>
<tr>
<td>LEVEL</td>
<td>SHARE</td>
</tr>
<tr>
<td>MASK</td>
<td>TIMER</td>
</tr>
<tr>
<td>OPMON</td>
<td>UNMK</td>
</tr>
<tr>
<td>QIFON</td>
<td>UNQ</td>
</tr>
<tr>
<td>QUEUE</td>
<td></td>
</tr>
</tbody>
</table>

**User-Written Subroutines**

User-written subroutines are simulated in the same manner as mainline programs.

**Common Area**

The simulated COMMON area can be dumped on cards whenever a program being simulated is terminated. The output cards can be used for input to reload COMMON, thus providing communication from one core load to another.
Restrictions

Restrictions placed upon the use of the Simulator program are listed below:

1. Nonprocess work storage must be used if actual data is to be transferred between disk and core.
2. Link or chain jobs must be simulated by presenting one core load at a time.
3. The Simulator utilizes LIBF and CALL instructions for special purposes. When analyzing post-mortem dumps, the contents of LIBF and CALL locations should be ignored by the user.
4. All I/O must be performed by Simulator subroutines. An execute I/O (XIO) instruction is not simulated but will be recorded on the List printer.
5. A wait (WAIT) instruction will be recorded on the list printer.
6. A storage protect setting instruction (STS with both the F-bit and the 9th bit equal to zero) will result in a termination.
7. An attempt to store into a skeleton area other than the INSKEL COMMON and work level areas will result in a termination.
8. Operation codes of 00, 38, 58, 78, and FF are invalid and will result in a termination.
9. A subroutine I/O area parameter pointing to the skeleton will result in a termination.

SUMMARY OF NONPROCESS MONITOR CONTROL CARDS

Tables 5-10 give a brief summary of all Nonprocess Monitor control cards. For details of card preparation and their functions, see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Table 5. Monitor Control Cards

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>// JOB</td>
<td>Initializes a nonprocess job</td>
</tr>
<tr>
<td>// DUP</td>
<td>Reads the disk utility program into core for execution</td>
</tr>
<tr>
<td>// XEQ</td>
<td>Reads the user's programs into core for execution</td>
</tr>
<tr>
<td>// ASM</td>
<td>Reads the Assembler into core for execution</td>
</tr>
<tr>
<td>// FOR</td>
<td>Reads the FORTRAN compiler into core for execution</td>
</tr>
<tr>
<td>// SIM</td>
<td>Reads the Simulator program into core allowing a nonprocess program to be simulated</td>
</tr>
<tr>
<td>// PAUS</td>
<td>Causes the system to WAIT</td>
</tr>
<tr>
<td>// END</td>
<td>Signals the Nonprocess Supervisor that all nonprocess work is complete</td>
</tr>
<tr>
<td>// END OF ALL JOBS</td>
<td></td>
</tr>
</tbody>
</table>

Table 6. Loader Control Cards

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>*INCLD</td>
<td>Causes a named program to be included in the skeleton or in a mainline core load</td>
</tr>
<tr>
<td>*RCORD</td>
<td>Records interrupts that occur during the execution of process core loads</td>
</tr>
<tr>
<td>*FILES</td>
<td>Provides for the designation of disk areas to be used by the FORTRAN program in which the files were defined</td>
</tr>
<tr>
<td>*LOCAL</td>
<td>Permits groups or blocks of subprograms to be loaded into core when they are called</td>
</tr>
<tr>
<td>*CCEND</td>
<td>Last loader control card, calls the Core Load Builder</td>
</tr>
</tbody>
</table>
### Table 7. DUP Control Cards

<table>
<thead>
<tr>
<th>Card</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCORE</td>
<td>Specify the size of object core</td>
</tr>
<tr>
<td>NDISK</td>
<td>Specify the number of disk drives on the system</td>
</tr>
<tr>
<td>DEFINE</td>
<td>Specify the system configuration with respect to disk areas</td>
</tr>
<tr>
<td>CONFG</td>
<td>Allow the user to delete FORTRAN or the Assembler from the monitor disk</td>
</tr>
<tr>
<td>REMOV</td>
<td>Pack relocatable programs into unused areas identified by *DELET</td>
</tr>
<tr>
<td>PAKDK</td>
<td>Labels a disk pack and, if not system pack, writes addresses</td>
</tr>
<tr>
<td>DLABL</td>
<td>Stores relocatable programs in the Relocatable Program Area (user or temporary) on disk</td>
</tr>
<tr>
<td>STORE</td>
<td>Stores blocks of data in Core Load (core image) Area on disk</td>
</tr>
<tr>
<td>STOREDATA</td>
<td>Causes a core load to be built and stored in the Core Load Area on disk</td>
</tr>
<tr>
<td>DUMP</td>
<td>Dumps programs from the disk to the system I/O device or list printer</td>
</tr>
<tr>
<td>DUMPDATA</td>
<td>Dumps blocks of data as indicated in *DUMP</td>
</tr>
<tr>
<td>DUMPLET</td>
<td>Dumps LET and/or FLET on the list printer</td>
</tr>
<tr>
<td>DELET</td>
<td>Replaces a program name in LET or FLET with the name DUMMY thus making the program area available to the store function</td>
</tr>
<tr>
<td>DWRAD</td>
<td>Allows the user to write addresses on a specified area of disk</td>
</tr>
<tr>
<td>STOREMD</td>
<td>Allows the user to modify existing nonprocess core loads and relocatable programs without previously deleting them</td>
</tr>
<tr>
<td>SEQCH</td>
<td>Used to change the sequence of existing core load linkages for process or nonprocess core loads</td>
</tr>
<tr>
<td>SIMULCI</td>
<td>Reads the Simulator program into core, allowing a process program to be simulated</td>
</tr>
<tr>
<td>DICLE</td>
<td>Allows the user to modify the interrupt core load table</td>
</tr>
</tbody>
</table>

### Table 8. FORTRAN Control Cards

<table>
<thead>
<tr>
<th>Card</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*IOCS (CARD, TYPEWRITER, KEYBOARD, 1443 PRINTER, PAPER TAPE, MAGNETIC TAPE, DISK, PLOTTER)</td>
<td>Delete any not used (Store integer variables in one word) This function is automatic in process programs.</td>
</tr>
<tr>
<td>**Header information to be printed on each compiler output page</td>
<td>(Store floating point variables and constants in 3 words instead of 2)</td>
</tr>
<tr>
<td>*ONE WORD INTEGERS</td>
<td>(Switch 15 ON to print result of each assignment statement)</td>
</tr>
<tr>
<td>*EXTENDED PRECISION</td>
<td>(Switch 15 ON to print value of IF or Computed GO TO)</td>
</tr>
<tr>
<td>*ARITHMETIC TRACE</td>
<td>(List source program as it is read in)</td>
</tr>
<tr>
<td>*TRANSFER TRACE</td>
<td>(List subprograms called directly by compiled program)</td>
</tr>
<tr>
<td>*LIST SOURCE PROGRAM</td>
<td>(List symbols, statement numbers, constants)</td>
</tr>
<tr>
<td>*LIST SUBPROGRAM NAMES</td>
<td>(List source program, subprogram names, symbol table)</td>
</tr>
<tr>
<td>*LIST SYMBOL TABLE</td>
<td>(Identifies this compilation as a nonprocess program)</td>
</tr>
<tr>
<td>*LIST ALL</td>
<td>(Causes DUP to punch an object deck after successful compilation)</td>
</tr>
<tr>
<td>*NONPROCESS PROGRAM</td>
<td>*PUNCH</td>
</tr>
</tbody>
</table>
**Table 9. Assembler Control Cards**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*TWO PASS MODE</td>
<td>Read source deck twice; must be specified when *LIST DECK or *LIST DECK E is specified, or when intermediate output fills working storage</td>
</tr>
<tr>
<td>*LIST</td>
<td>Print a listing on the principal printing device</td>
</tr>
<tr>
<td>*LIST DECK</td>
<td>Punch a list deck on the principal I/O device (requires *TWO PASS MODE)</td>
</tr>
<tr>
<td>*LIST DECK E</td>
<td>Punch only error codes (cc 18-19) into program list deck (requires *TWO PASS MODE)</td>
</tr>
<tr>
<td>*PRINT SYMBOL TABLE</td>
<td>Print a listing of the symbol table on the principal printing device</td>
</tr>
<tr>
<td>*PUNCH SYMBOL TABLE</td>
<td>Punch a list deck of the symbol table on the principal I/O device</td>
</tr>
<tr>
<td>*SAVE SYMBOL TABLE</td>
<td>Save symbol table on disk as a system symbol table</td>
</tr>
<tr>
<td>*SYSTEM SYMBOL TABLE</td>
<td>Use system symbol table to initialize symbol table for this assembly</td>
</tr>
<tr>
<td>*PUNCH</td>
<td>A relocatable binary deck will be punched by DUP following this assembly</td>
</tr>
<tr>
<td>*OVERFLOW SECTORS n</td>
<td>n = number of sectors of nonprocess working storage allowed for symbol table overflow</td>
</tr>
<tr>
<td>*COMMON n</td>
<td>n = length of COMMON in words (decimal)</td>
</tr>
</tbody>
</table>

**Table 10. Simulator Control Cards**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*SNAP</td>
<td>Displays up to 10 locations following execution of an instruction</td>
</tr>
<tr>
<td>*TRACE</td>
<td>Traces or displays same information as for *SNAP</td>
</tr>
<tr>
<td>*DUMP</td>
<td>Dumps simulated core storage</td>
</tr>
<tr>
<td>*SAVE COMMON</td>
<td>Punches out binary deck of process and variable COMMON</td>
</tr>
<tr>
<td>*LOAD COMMON</td>
<td>DEFINES and analyzes COMMON from *SAVE COMMON OUTPUT deck</td>
</tr>
<tr>
<td>*XIO</td>
<td>Suppresses printing of IOCC words referenced by XIO instruction</td>
</tr>
<tr>
<td>*WAIT</td>
<td>Suppresses printing of WAIT instructions</td>
</tr>
<tr>
<td>*START SIMULATION</td>
<td>Signals that all Simulator control cards have been read</td>
</tr>
<tr>
<td>*END DATA</td>
<td>Terminates Simulator run</td>
</tr>
</tbody>
</table>

**EXAMPLES OF NONPROCESS MONITOR USAGE**

The prime purpose of this section is to illustrate a few of the many possible uses of 1800 TSX features, and to accentuate the many more possibilities based upon the ability of the user to apply the basic concepts and techniques. Numerous sample programs and coding examples are presented as demonstration of good programming practice and technique. These examples conform strictly to standard TSX coding conventions.

The **JOB**

When a programmer is given a problem, he analyzes that problem and defines a precise problem-solving procedure: that is, he writes a program or a series of programs. To the monitor system, executing a mainline program (and any subroutines and subprograms that it calls) is a job step. A job consists of executing one or more job steps.

At its simplest, a job consists of one solitary job step. For example, assembling or compiling a program is a job consisting of one job step. Similarly, executing a FORTRAN mainline program to invert a matrix is a job consisting of a single job step.

If the problem is complex, one job may consist of a series of job steps. Such a job may include multiple assemblies, compilations, disk utility functions, and executions. A job always begins with a // JOB control card which is the first statement in the sequence of control statements that describes a job.

The **JOB Deck**

The input to the Nonprocess Monitor may consist of one or more job decks. Each job deck is preceded by a // JOB. The processing of each job deck is
controlled by the Supervisor program as specified in the monitor control cards. As an example, consider the following stacked input arrangement (see Figure 30).

The above sequences will compile, store and execute both program PROG1 and program PROG2 provided that:

1. There are no source program errors, and
2. There is sufficient room in the Nonprocess Work Storage area.

A source program error will cause the DUP Store Operation to be bypassed for that program, and all following // XEQ requests preceding the next // JOB card will be disregarded. This feature (that is // XEQ -- request disregard) can prove very useful when the successful execution of one program depends upon the successful completion of the previous program. A combination such as this should be considered as one job. The // XEQ control cards should not be separated by a // JOB card. Note from Figure 30 that it would not be necessary to store the two programs if they were executed on a one-shot basis.

Assembling/Compiling Programs

Programs are of two types: process and nonprocess. A process program is one that continuously monitors a control process. All application programs are, by definition, process programs. A nonprocess program, on the other hand, is not directly related to the control process itself. An assembler program is an example of a nonprocess program; other examples include compilers, data reduction, payroll, bookkeeping, simulation of new and existing programs, and linear programming.

Process and nonprocess programs may be further classified as main programs or subroutines. Subroutines can be subdivided into the following: LIBF (library functions), CALL, Interrupt, IBM-supplied, and LOCAL subroutines.

In the off-line or time-sharing mode of operation, the user may exercise any of four options in assembling/compiling and executing a nonprocess program. Figure 31 illustrates these approaches in simplified form. A distinction should be drawn between process and nonprocess programs. The initial process program can only be executed through a cold start procedure for an on-line TSX system. If the process, mainline, or combination core load is already disk-resident (in the Core Load Area) it is called by a CALL CHAIN or CALL QUEUE.

EXAMPLE 1. ASSEMBLE AND EXECUTE A NONPROCESS PROGRAM FROM THE TEMPORARY AREA (see Figure 32).

The Assembler is unable to differentiate between process and nonprocess programs -- these are treated alike. Following assembly, the object program in relocatable format is moved to the temporary area on disk, and its entry (name, word count, and sector address) made in LET.

If the user desires to perform only an initial check on his program, and not execute it, // XEQ and *CCEND are not required. If he plans to verify the program logic and results (if any), he will execute it. The presence of the // XEQ and *CCEND control cards calls in the Core Load Builder, and a core load is built and executed. In addition, a listing of source statements as well as the corresponding object program, and a directory of all valid labels used in the program can be obtained by specifying these options with the appropriate Assembler control cards. The order in which programs are assembled is important when the *SAVE SYMBOL TABLE control card is used in assembling related programs.

Note that the relocatable program will reside in the temporary disk area until it is deleted by the next // JOB card. An *CCEND control card must always follow an // XEQ card if a relocatable program is referenced in the // XEQ card.
EXAMPLE 2. COMPILe AND STORE A NON­
PROCESS PROGRAM IN THE RELOCATABLE PRO­
GRAM (OR USER) AREA ON DISK (see Figure 33).

Unlike the Assembler, the Fortran Compiler dis­
tinguishes between the two types of programs by the
absence or presence of the *NONPROCESS PROGRAM
control card. In a process program, each integer
variable automatically occupies one word of storage.
In a nonprocess program, however, the *ONE WORD
INTEGERS control card forces the compiler to allo­
cate one word of storage to each integer variable; in
the absence of this card, the same allocation (that is,
two words) for real variables is made. In the case of
a large array, this could be prohibitive.

All FORTRAN programs are compiled in relocatable
format. Following compilation, the relocatable
object program is moved to the temporary disk area,
and an entry made in LET. It can now be called for
execution or loaded to the Relocatable Program (that
is, the User) Area on disk.

In Example 2, the relocatable program MAIN2 is
stored in the Relocatable Program (User) Area. The
actual storing of the program consists of physically
moving the program to its destination area (the User
Area) from the temporary area of Nonprocess Working
Figure 33. Compile and Store a Nonprocess Program in the Relocatable Program Area (User Area) on Disk

Storage. When the storing operation is completed, LET is updated and the communications and fixed area parameters reset to reflect these changes.

Note that a store from the temporary (TEMP) area to the permanent Relocatable Program Area causes TEMP to be packed to reflect that program moved. An exception exists when the program is the last entry in TEMP or when there is only one TEMP program initially.

EXAMPLE 3. COMPILE AND EXECUTE A NONPROCESS PROGRAM FROM THE CORE LOAD AREA (see Figure 34).

This example illustrates the third and fourth options which may be taken to assemble/compile and execute a nonprocess program from the Core Load Area. Note that subroutines TIMSB, ERROR, and PRINT are compiled and stored in the User Area as these subroutines are frequently referenced by this and other nonprocess programs. In building process mainline and combination core loads, it may not be necessary to store these subroutines. The store core image routine is used to store a program in core image form (as a core load) in the core load area and to assign the core load a name. By making column 9 of the *STORECI control card non-blank, a map of the locations and names of subroutines and subprograms loaded with the core load is obtained. When the nonprocess core load is correctly built, DUP will search through its program name table, find the name of the core load just built, and add its disk address and word count to the table. In addition, any programs referenced in this core load name table are looked up in FLET and their disk addresses and word counts added to the table. The core load is then executed from the Core Load Area. FX in columns 16 and 17 of the // XEQ monitor control card signifies that the input program is in core image format and that FLET is to be searched for this program name.

Deleting and Replacing Relocatable Programs, Core Loads, and Data Files

The *DELETE function allows the user to delete any named Relocatable program
Mainline core load
Combination core load
Interrupt core load
Nonprocess (or link) core load
Data file

from the disk. An entry of a program in LET or a core load/data file in FLET takes the normal form

<table>
<thead>
<tr>
<th>LET:</th>
<th>NAME</th>
<th>DISK BLOCK COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLET:</td>
<td>NAME</td>
<td>WORD COUNT</td>
</tr>
</tbody>
</table>

where each LET and FLET entry occupies three and four words of disk space respectively. Whenever a program or a core load is deleted, its NAME in LET or FLET is replaced by the symbolic 9DUMY and henceforth the system is no longer cognizant of this program or core load. Furthermore, the area on disk previously occupied by a program or core load is now available for the storage of other programs, core loads, or data files. These areas are available, but only used after all previously available areas have been used.

A core load may be deleted and, in addition, have its reference replaced by another core load's word count and sector address. The replacement core load must be of the same type. That is, a mainline
<table>
<thead>
<tr>
<th>SAMPLE CODING FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
core load may be replaced by another mainline core load, an interrupt core load by an interrupt replacement core load, a combination core load by a replacement core load, and a nonprocess core load by a nonprocess replacement core load. Replacement of the four types of core loads is governed by certain rules which are summarized as follows. Note that the replacement function within an *DELETE operation does not alter the core load name, but only its word count and sector address.

If a logical drive is specified in column 19 of the *DELETE card, only that drive is searched for the core load to be deleted. When the replacement function is used and a logical drive is specified, also the replacement core load must reside on that drive. If no logical drive is specified all drives are searched, starting with the temporary drive.

Combination and Interrupt Core Loads. In deleting a combination or interrupt core load, all references to this core load in the Program Name Table (PNT) and/or Interrupt Core Load Table (ICLT) must be replaced by a replacement core load name. Absence of this specification in the control card invalidates the deleting function. Furthermore, if an interrupt core load or combination is used to service multiple interrupts, all interrupt core load entries in the PN Tables, Queue Table, and ICLT are automatically replaced with a single delete operation by specifying 9999 for the interrupt level and bit positions on the control card (columns 39-42).

The rule is never to allow a previous serviceable level and its bit indicator to remain unserviceable.

Mainline and Nonprocess (or Link) Core Loads. In general, a mainline or nonprocess core load that is not currently being called by other core loads does not require replacement. If, however, it is still being referenced in the Queue Table, the PNT within the System Skeleton or some other PNT, deletion is restricted because it is still necessary to maintain this core load identity in the system. Note that a nonprocess core load may be deleted without a replacement core load even though it is still referenced. A negative value is then placed in the word count position of the PNT entry in those core loads referencing the deleted nonprocess core load. A nonprocess core load is also referred to as a link.

Data Files. By definition, a data file is an area in the Core Load Area established by an *STOREDATA function with a D in column 11. Data files can be deleted but not replaced. In deleting a data file from the disk, the user should be aware that the system does not check to see whether this data file is still being referenced by currently executing core loads. This means that if he wishes to delete a data file, he has to ensure by some programming means that there is no reference to this file: that is, no reading from or writing to this file. If there is a reference, there is a distinct possibility that core loads writing to or reading from this file might destroy one or more core loads stored in the same location the data file was located.

**EXAMPLE 4. DELETE A PROCESS MAINLINE, COMBINATION OR INTERRUPT CORE LOAD FROM THE CORE LOAD AREA (see Figure 35).**

In deleting a process mainline core load, the user should ensure that this core load is not being referenced or called by any other core load that may in turn reference further core loads. If such a situation exists, up to 14 names of calling core loads will be listed; if the number of calling core loads exceeds 14, any excess will not be indicated in the error message. The solution here is to eradicate the excess core loads from the Fixed Area, either by a sequence change or a deletion.

The delete operation is merely one of removing or eliminating an entry from the FLET table with a system mnemonic name 9DUMY, indicating an unused area on disk. Note that in a fresh (that is, new) disk pack, the Core Load or Core Image Area is initially represented in FLET by a 9DUMY entry thus:

```
NAME (= 9DUMY)
SECTOR COUNT
SECTOR Address
```

Subsequent *STOREC* operations will move this entry. A delete simply replaces a core load with a 9DUMY. In practice, a delete is normally followed by a replacement unless the core load being deleted is considered "dead," thus making its replacement unnecessary.

Example 4 also demonstrates the use of *DUMPLET as an effective programming tool. A dumplet following a delete operation is good programming practice; it shows conclusively that a program

```
<table>
<thead>
<tr>
<th>SAMPLE CODING FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
</tr>
<tr>
<td>11-20</td>
</tr>
<tr>
<td>21-30</td>
</tr>
<tr>
<td>31-40</td>
</tr>
<tr>
<td>41-50</td>
</tr>
<tr>
<td>9DUMY</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
<tr>
<td>PROBE</td>
</tr>
</tbody>
</table>
```

Figure 35. Delete a Process Mainline, Combination, or Interrupt Core Load from the Core Load Area
or core load is in fact removed from the FLET table. For an understanding of LET/FLET tables, the user is referred to the Systems Reference Library: IBM 1800 Time-Sharing Executive System, Operating Procedures, Appendix F, Form C26-3764.

In all three cases, the FLET table is searched for the core load name to be deleted, and its replacement name. Any references to the old program in the Program Name Table of all core loads are then replaced with the word count and sector address of the replacement core load. The old program name is finally deleted from the FLET table.

In the case of combination and interrupt core loads, the interrupt level and PISW bit position indicators are obtained from the card buffer, converted, and stored in the nonprocess communications area. The ICL Table is then updated.

Note also that in all cases, except for the deletion or replacement of nonprocess programs, a check of the queue in the skeleton is made to see if the program to be deleted or replaced is in the queue. If it is, the queue is updated.

EXAMPLE 5. REPLACING A NONPROCESS CORE LOAD IN THE CORE LOAD AREA (see Figure 36).

Like process mainline, combination and interrupt core loads, a nonprocess core load can also be deleted and replaced by an *DELETE operation (see Figure 35).

A nonprocess core load can also be replaced by storing a replacement core load to the Fixed Area, as illustrated in Figure 36. The user can thus modify existing nonprocess core loads without previously deleting them.

This is achieved by an *STOREMD operation. An *STOREMD with a Fixed Area destination is exactly equivalent to an *STORECI of a nonprocess core load provided that

1. The replaced entry must be in FLET for a Core Load Area
2. If the function is to modify the Core Load Area, the existing FLET entry must be for a nonprocess core load.

A search through FLET is first made to see if the replacement core load name is already an entry. A further search is then made for a large enough 9DUMY entry to contain the core load. On a find, the sector count of the 9DUMY is checked against the required sector count. The check is successfully terminated by locating a large enough entry on a specified drive which can also take an additional FLET entry. A successful find supplies a destination sector address, and, if previously unknown, the logical drive. Once it is determined that there is space to store the core load, the core load Program Name Table is updated.

Note that the replacement program can either be in the temporary area (of Nonprocess Working Storage) or in the Relocatable Program (that is, User) Area on disk. The name assigned to this program must not be the same as that of the program to be replaced. In Example 5, NAME1 and NAME2 designate two different names. NAME1 (which was previously resident in the Core Load Area) is deleted from the Fixed Area and its entry in FLET removed. The replacement core load NAME2 is stored in the Core Load Area and its name, size in words, and starting sector address then entered into the FLET table.

EXAMPLE 6. REPLACE A RELOCATABLE PROGRAM IN THE USER AREA (RELOCATABLE PROGRAM AREA) (see Figure 37).

Figure 37 illustrates a sequence of control cards that might be used to accomplish this. NAME1 is the name of the replacement program being stored. It must be compiled or assembled with the identical name of the relocatable program being replaced (that is, also NAME1), and it must be the prime entry point. This name must be in the temporary area of Nonprocess Working Storage.

Note that the control card name for the existing program to be replaced must have a LET entry of the same name for a User Area replacement. The replacement program will not overlay the current program, but only cause it to be deleted from the LET table. Thus, the size of a replacement program and the number of entry points in a relocatable-
Changing Core Load Linkages. Through the *SEQCH function, the user is given a powerful programming tool to alter the sequence or order of existing core load linkages for either process or nonprocess core loads. This means that he can now modify a core load Program Name Table such that all references to a core load originally specified will subsequently reference a replacement core load. Note that no deletion of core loads takes place as in an *DELETE with-replacement operation.

This is known as selective replacement, since the existing referenced core load, the replacement core load, and all other core loads in which changes are to be implemented are all specified.

Note also that the replacement and existing core loads must be type-compatible. That is, a mainline or combination core load may replace either a mainline or combination core load, but a nonprocess or link core load may only be replaced and called by a link. Process calls may emanate from any type of core load.

EXAMPLE 7. CONSIDER THE FOLLOWING SITUATION. In a typical operation, core loads ALPHA, BETA1, and DELTA will call or reference core load NAME1 by a CALL QUEUE statement, thus:

\[
\text{CALL QUEUE (NAME1, 1, 0)}
\]

The user now elects to replace NAME1 by NAME2 such that all further references to NAME1 by ALPHA, BETA1, and DELTA will be changed to NAME2. NAME1 can either be a combination or mainline core load resident in the Core Load Area; by definition, NAME2 must either be a combination or mainline core load -- assume that it is also stored (by an *STORECL) in the Core Load Area. The following sequence of control cards may be used to effect this change.

This will modify the Program Name Table of each of the core loads ALPHA, BETA1, and DELTA so that whenever they call NAME1, the call will refer to NAME2.

At this point in the operation, the user may have no further use for this sequence change, and may well delete core load NAME1, thus:

\[
\text{*DELETE M NAME1}
\]

In practice, however, he will probably not delete NAME1 but prefer to return to his original sequence, thus:

\[
\text{*SEQCH NAME2 NAME1, ALPHA, BETA1, DELTA}
\]

Note that because of the type-compatibility between existing and replacement core loads (mentioned earlier), a restriction exists in the case of nonprocess core loads. If, for example, NAME1 were a nonprocess core load, then NAME2, ALPHA, BETA1, and DELTA must, by definition, be also nonprocess core loads.

EXAMPLE 7A. AN ALTERNATIVE METHOD (TO THE *STOREMD FUNCTION) OF ON-LINE RE-
BUILDING OF PROCESS CORE LOADS. Figure 38 illustrates the technique employed, where

- CLA1 is the core load name to be modified; assume the core load is on disk. RELPR is the relocatable program which has been modified.

- CLA2 is a temporary core load name used to achieve proper deletion and replacement of the new version of CLA1.

Debugging Core Loads using the Simulator

Several options are available to the user for the debugging of process and nonprocess programs. These are summarized below:

Nonprocess Programs

1. Using TASK (with TASK EAC) in an off-line system only.
2. Using the Simulator in an (a) off-line, or (b) on-line system.

Process Programs

1. Using TASK (with TASK EAC) in an off-line system only. To do this, the process program must first be written as a nonprocess program; when fully tested, it is reconstituted into a process program for execution (in an on-line environment).
2. Using the Simulator in either the on-line or off-line environment.

The advantages of the Simulator as a debugging tool lie mainly in

1. The powerful diagnostic messages printed by the Simulator, which allow the user to determine the logic flow of the program by noting the subroutines called, and,
2. in the fact that a process or nonprocess core load may be fully tested without taking the system off-line.

The following examples illustrate the simulation of assembly language process programs.

EXAMPLE 8 (PROGRAM LISTING NO. 2). This program is written for the purpose of debugging the Simulator. If the Simulator erred in the reading of analog input cards, error messages would be printed.

Actual simulation is initiated after the core load build function has been completed. The first thing done by the Simulator is to read the Simulator control cards *XIO and *START SIMULATION. Note that any other Simulator control cards, such as *DUMP, *SNAP, etc., should precede the *START SIMULATION card; data cards should follow the *START SIMULATION card. Since no control card is used to describe the source of analog data, it is assumed that this data will emanate from card input.

After reading the control cards, the Simulator will proceed to interpret the instructions in the user's program, exactly as in execution. The first instruction being a LIBF AISQN, the Simulator prints the S50 message, giving the name of the routine and the absolute address of the LIBF. The S20 message is printed by the Simulator AISQN routine.
and consists of a description of the calling sequence. Since it is an analog input, a data card is read. However, due to the fact that column 5 is blank, the Simulator is not aware of the format of data on this card and informs the user accordingly with the S12 message. This is likewise the case with the next two cards. Note that the Simulator is still in the process of simulating the first AISQN call. It will continue reading cards until it completes this call. The next card read has a D in column 5, implying digital data input.

However, an absence of the E parameter in column 72 signifies an end of data and thus the S15 message. Only the number +00123 is read into the buffer since the word count is 2, one word of which is the analog address.

Upon completion of the first LIBF, the busy test is encountered. The Simulator will always take the busy exit the first time through a busy test. The second time through, it will exit at the not-busy instruction. Thus, if the busy exit contains an MDX back to the busy test, the Simulator output will show two "goes" through the busy test, one after another, as in the printout of the two S21 messages.

Next, in sequence, another LIBF AISQN is encountered. Again, the S50 message identifies the subroutine and the absolute address of the LIBF. The S20 message gives the calling sequence. A card is read with correct format and an E in column 72. However, only the numbers 1234 and FF12 are read, the first blank terminating the data. Since the word count for AREA2 is four, there is insufficient data on the cards to fill the buffer. Hence the S16 message.

Note that an E in column 72 terminates the call to the subroutine. Therefore, if there had been no E in the last card read, the Simulator would have tried to fill the buffer with data from the next card. The busy test following this is then simulated.

The last call recognized is that to VIAQ, and this terminates the job. The following S99 message is a snapshot of the instruction which caused termination.

Anytime a job is terminated, a snapshot is given to allow the user to determine why the job was terminated.

If the user had wished to see the status of registers at some point of the program, a *SNAP or a *TRACE card could have been added giving the relative address (obtained from the assembly) of the instruction. Note also that the WAIT instruction can be used as a trace aid since the Simulator automatically gives a snapshot of registers upon encountering a WAIT.
This test checks ability of the simulator to:

- reject data cards with incorrect format;
- skip excess data cards;
- sense insufficient data is supplied;

Printer messages upon successful test should be:

- S12 IDENTIFIABLE PROCESS INPUT DATA CARD READ;
- S15 TOO MUCH DATA, READ CARDS UNTIL E IN COL 72;
- S16 INSUFFICIENT DATA TO FILL I/O AREA, JOB

---Continued---

```
0000 20 01262615 START LIBF AISQN
0001 0 1000 DC /1000
0002 0 0020 DC AREA1
0003 0 0000 WAIT
0004 20 01262615 NEXT LIBF AISQN
0005 0 0000 DC /0000
0006 10 0000 DC AREA2
0007 0 0016 LD AREA1+3 CHECK THAT NO MORE
0008 1 04C200017 BSC L ERR1,2 THAN 1 DATA CARD WAS
0009 20 01262615 NEXT1 LIBF AISQN USED FOR DATA.
000A 0 0000 DC /1000
000B 0 0024 DC AREA2
000C 0 0000 WAIT
000D 20 01262615 NEXT2 LIBF AISQN
000E 0 0000 DC /0000
0010 0 0000 DC AREA2
0011 0 0016 LD AREA2+4 CHECK THAT I/O AREA
0012 0 0000 DC /0000
0013 01 42C00018 BSC L ERR2,2 ABOVE DATA CARDS WAS
0014 30 25241600 CALL VIAO
0015 30 16162097 ERR1 CALL MESSP
0016 0 0029 DC ERR1P
0017 0 000E DC AREA1
0018 0 0004 DC AREA2
0019 0 0010 DC /1001
001A 0 0001 DC /1001
001B 0 0000 DC /0000
001C 0 0000 DC /0000
001D 0 0000 DC /0000
001E 0 0020 DC /AAAA
001F 0 0020 ERR1P EBC * NOT SUCCESSFUL; EITHER CARD.
0020 0 0019 EBC * WITH BAD FORMAT WAS NOT.
0021 0 001C EBC * RECOGNIZED OR TOO MANY DATA.
0022 0 0012 EBC * CARDS WERE READ IN.
0023 0 0002 EBC * ERR.
0024 0 0010 ERR2P EBC * NOT SUCCESSFUL, I/O AREA;
0025 0 0010 EBC * ZERODE ABOVE AREA FOR WHICH.
0026 0 0011 EBC * DATA WAS SUPPLIED.
0027 0 0002 EBC *
0028 0 0000 CONST DC /AAAA
0029 0 0000 END START
```

No errors in above assembly.

DUP FUNCTION COMPLETED

```
CLB. BUILDB OB638
*SIMULCIL M OB638 OB638 OB638
*CEND

CREASE LOAD MAP
TYPE NAME ARG1 ARG2

CDW TABLE 3ER2 000C
*1BT TABLE 3ERE 001D
*P01 TABLE 3EBA 0010
*ETV TABLE 3EBB 0054
*1ST TABLE 3G6F 0036
*PNT TABLE 3FAA 0000
MAIN OB638 3F4E
PNT OB638 3F4B
PNT OB638 3F4C
CALL VIAO 3FD6
CALL MESSP 4OF6
CORE 40BF 3F42
```

Functions of Executive Programs 89
•X10
*START SIMULATION
$ 50 3F5E AISON
$ 20 CONTROL WORD 1000, IO AREA 13F6E, SPECIAL ENTRY 13000
$ 14 INPUT CARD I #AI 011120-1113402561-1082531562
$ 12 UNIDENTIFIABLE PROCESS INPUT DATA CARD READ
$ 14 INPUT CARD I #AI 12340210FF12FEDC
$ 12 UNIDENTIFIABLE PROCESS INPUT DATA CARD READ
$ 14 INPUT CARD I #AI 000010100001111000000001111111111
$ 12 UNIDENTIFIABLE PROCESS INPUT DATA CARD READ
$ 14 INPUT CARD I #AI D 000123-0001000127-02047
$ 16 TOO MUCH DATA, READ CARDS UNTIL E IN COL 72
$ 14 INPUT CARD I #AI H 12340210FF12FEDC
$ 50 3F52 AISON
$ 21 BUSY TEST
$ 50 3F52 AISON
$ 21 BUSY TEST
$ 50 3F58 AISON
$ 20 CONTROL WORD 1000, IO AREA 13F72, SPECIAL ENTRY 13000
$ 14 INPUT CARD I #AI H 1234FF12 FEDC
$ 16 INSUFFICIENT DATA TO FILL I/O AREA
$ 50 3F5C AISON
$ 21 BUSY TEST
$ 50 3F5C AISON
$ 21 BUSY TEST
$ 56 TSX VIAO NOT EXECUTED
$ 99 0015 3F63 3F65 5400 3F06 0000 0000 0000 0000 0000 3F3B
NOB ILLEG LDR CO
*END DATA
// *
DB638 END
EXAMPLE 9 (PROGRAM LISTING NO. 3). The Simulator control cards are read in and initializing processes are begun. The three analog input cards signify card input for data. Note that one or all of these cards could have specified the random number generator as an input source. Note also that in a program such as this, extreme care must be taken in setting up the data cards, remembering that an E parameter in column 72 terminates any attempt to fill an analog buffer.

The S56 message reflects the call in the FORTRAN program. Note that the parameters are also printed. If either of these parameters had been improper, a message would be printed accordingly.

The next block of four PRNTN calls refer to the WRITE (M, 10) statement. A FORTRAN call to output or input, generally, translates to several calls to the I/O subroutine, including busy tests, actual I/O, and special functions as in the carriage control shown here. The line after S33 gives the message which the user would see if the program were executed.

The three CARDN calls listed next are the result of the FORTRAN statement 20 READ (N, 30). The card image in hexadecimal is printed and the busy tests performed.

Next, in sequence, are four print calls: the result of WRITE (M, 40). Again, the line after S33 gives the users actual printout.

Note that the FORTRAN statement at 30+1 is an IF statement. The Simulation output gives no indication of this because no major subroutines are called; that is, the Simulator does not show when some arithmetic function or subroutine is called. However, all instructions of this statement have been simulated.

The first DAOP call in the Simulator output is the result of the FORTRAN CALL DAC statement. Again, a rundown of the calling sequence is given in S20. The S10 message describes the type of output, that is, random, sequential, etc. The four words of analog output are given just ahead of the next S50 message. The DAOP busy tests are a result of the second DAC call.

Next, in sequence, is a series of analog input calls, each one reading some point from cards. These are a direct result of the analog input calls in the FORTRAN program.

Again, there are four calls to the PRNTN routine: the result of WRITE (M, 65).

At this point, the IF following statement 65 and the IF at statement 90 force a return to statement 20 and a second "go" through the loop is simulated. The message "GOT THIS FAR" is printed out a second time, and for a third time, statement 20 is executed and a card is read. At this point, the IF at 30+1 forces a transfer to statement 120.

The CALL QUEUE, CALL SHARE, and CALL VIAQ are then simulated by the S56 messages. Note that in the case of CALL QUEUE, the name of the called program TC152 is also printed out. A snapshot of the terminating instruction is then printed.

Since the user included a *DUMP control card, the program is dumped. The addresses are absolute. The address in statement S98 gives the absolute address of the first word of the user's program. The XXXX at the end of the dump refers to undefined core.

A dump of the transfer vector may be obtained by using a negative number as the lower limit.

One method of tracing through a FORTRAN program is by strategic PAUSE (I) statements. When the Simulator encounters such a statement, it will print out the S56 PAUSE message together with the appropriate parameter. Thus, some idea of program flow may be obtained.
PROGRAM LISTING NO. 3 -- EXAMPLE 9

// JOB A
// =
// FOR TC152
*LIST ALL
*ICCS(I143 PRINTER )
*INE WORD INTEGERS

SIMULATOR TEST CASE 152

EXTERNAL TC152
DIMENSION NOUTA(10),IN1(9),IN2(10),ID1(10)
CALL UNMK(-1,-1)
INIT=
N = 2
M = 3
ITOL = 20
WRITE (M,10)
00000000
00000010
00000030

10 FORMAT(24H INTERLEAVED AIP,AIS,AIR)
20 FORMAT(10,6I)
30 FORMAT(617,HI)
40 FORMAT(10,6I)
50 FORMAT(10,10I)
60 FORMAT(14H GOT THIS FAR)
70 FORMAT(6H JP)
80 FORMAT(10,6I)
90 FORMAT(10,6I)
100 FORMAT(10,6I)
110 FORMAT(10,6I)
120 FORMAT(10,6I)
130 FORMAT(10,6I)

WRITE(M,68)
65 FORMAT(13H OUT OF TOLERANCE)

IF (IABS(ID1(1)-NOUTA(3)) - ITOL) 90,90,70
70 WRITE (M,80) ID1(1)
80 FORMAT (10M ID1(1) = .17/17M OUT OF TOLERANCE)
90 IF (IABS(NOUTA(1)-JP) - ITOL) 20,20,100
100 WRITE (M,110) JP
110 FORMAT (6H JP)
GO TO 20
120 CALL QUEUE(TC152,1.5)
130 CALL SHARE(300)
CALL VIAQ
GO TO 130

VARIABLE ALLOCATIONS
NOUTA=00000000
IN1 =0016
IN2 =0020
ID1 =002A
INIT #002B
N #002C
M #002D
ITOL #002E
I =002F
J =0030

STATEMENT ALLOCATIONS
10 =0042
20 =0043
30 =0044
40 =0045
50 =0046
60 =0047
70 =0048
80 =0049
90 =004A
100 =004B

FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS
TC152 UNMK DAC AIP AIS AIR IABS QUEUE SHARE VIAQ COMGO MRED MHRT MCOMP MDIX
MIOIX

INTEGER CONSTANTS
1=0042
2=0043
3=0044
4=0045
5=0046
6=0047
7=0048
8=0049
9=004A
10=004B

CORE REQUIREMENTS FOR TC152
COMMON 0 INSKEI COMMON 0 VARIABLES 66 PROGRAM 422

END OF COMPILATION
Functions of Executive Programs  93
GOT THIS FAR

10 0 20 1 4096 4097

SHARE NOT EXECUTED

DUMP OF SIMULATED CORE 3F58
Functions of Executive Programs 95
Data Manipulation

EXAMPLE 10. RESERVING A FILE AREA IN THE CORE LOAD AREA. The only method of reserving a file area in the Core Load Area on disk is through an *STOREDATA function. Depending on the intent and purpose of the user at the time this function is performed, two options are open to him:

1. Create a file in the Core Load Area and store directly into that file data input via the Card Reader (D in column 11, RD in columns 13-14, and FX in columns 17-18).

2. Create a file in the Core Load Area and store into that file data from Nonprocess Working Storage (D in column 11 and FX in columns 17-18). In this situation, information in Nonprocess Working Storage is physically moved to a data block that starts at the beginning of the next available sector in the Core Load Area.

3. Create a file area in the Core Load Area for future use, such as an output file for a core load program. The method of accomplishing this is analogous to option 2 except that the data being stored may be superfluous to the storing function.

To reserve a data file, therefore, implies that there is movement of information or data which may be valid or invalid. Note that the *STOREDATA function transfers this information to disk without any change of format.

To accomplish this, certain parameters must be known to DUP and the Core Load Builder. These are:

- Type of source input
- Logical drive number
- Destination area
- Name of data file
- Sector count and/or word count

Figure 39 gives the sequence of control cards that might be used in a typical situation, where the source input is data cards.

A data file, FILE1, one sector long, is reserved in the Core Load Area, and given a FLET entry. Since this is a data FLET entry, the control card sector count, or the sector equivalent of the word count, is contained in the entry.

The DEFINE FILE statement (see IBM 1130/1800 Basic FORTRAN IV Language, Form C26-3715) specifies to the FORTRAN Compiler the size and quantity of disk data records within a file (or files) that will be used for processing with a particular program and its associated programs; in this example, one data block of 320 words is used. Since we are storing data to be used by a FORTRAN program from cards, the associated *FILES control record must contain the identical name (that is, FILE1) of the data block established by the *STOREDATA function. Note that it is from the *FILES card that the Core Load Builder obtains the necessary data (name, file number and drive code) to construct the DEFINE FILE TABLE within a core load. This is a three-word table which equates program defined file numbers to symbolic disk data files specified on the *FILES card.

We have seen that in moving data from the Nonprocess Working Storage to the Core Load Area, the direction of movement is from the start of Nonprocess Working Storage on the disk drive specified.

<table>
<thead>
<tr>
<th>SAMPLE CODING FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
</tr>
<tr>
<td>11 JOB</td>
</tr>
<tr>
<td>11 DUP</td>
</tr>
<tr>
<td>*STOREDATA RD FX FILE1 091 38328</td>
</tr>
<tr>
<td>11 JOB</td>
</tr>
<tr>
<td>11 JOP</td>
</tr>
<tr>
<td>11 JOPK ALPHA</td>
</tr>
<tr>
<td>J053 (TELETYPEI1,1648 PRINTER)</td>
</tr>
<tr>
<td>M530 ALL</td>
</tr>
<tr>
<td>11 PUNCH</td>
</tr>
<tr>
<td>11 DEFINE FILE IS(1,320,3,NEC)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>WRITE (13,4) 17</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
However, in the case of data files referenced by FORTRAN I/O in Nonprocess Working Storage, the direction of movement starts from the end of Nonprocess Working Storage, and therefore the *STOREDATA function cannot move a temporary data file established by the execution of a FORTRAN-written program.

**EXAMPLE 11. DUMPING A PROGRAM OR DATA FILE FROM THE CORE LOAD AREA.** A program or a block of data in the Core Load Area may be

1. moved to the Nonprocess Working Storage,
2. punched into cards, or
3. printed on the List Printer.

Figure 40 illustrates these three cases.

The primary difference between the two dump functions, *DUMP and *DUMPDATA, lies in the handling of relocatable programs. *DUMP converts relocatable programs from disk system format either to card system format when dumping to cards, or to printer format when the List Printer is selected as the I/O media. *DUMPDATA performs no conversion and outputs a relocatable program as data identical to its original format.

Programs and/or data in the Core Load Area are assumed to be in disk core image format. The name of the program or data file must always be given.

Note that when dumping from the Core Load Area to punched cards, an *CCEND control card is punched out as the last card in the output deck. This card also contains the word and sector count needed in a *STOREDATA operation. (Note also that, although the *STOREDATA function requires that these counts be contained in the control card, the punched-out *CCEND card may be used in a subsequent store of the dumped core load.) PN specifies the Card Punch as the principal system output device, while PR specifies the List Printer.

In the same fashion, it is also possible to dump a mainline, combination, or interrupt core load to Nonprocess Working Storage or to any available I/O media.

**EXAMPLE 12. LOADING A PROGRAM OR DATA BACK INTO THE CORE LOAD AREA.** One of the features of the Disk Utility Program (DUP) is the ability to load (that is, store) a previously built core load, which has been dumped to cards by the *DUMPDATA function, back to the Core Load Area. One significant use of this ability is the reordering (that is, rearrangement) of the position of core loads within the Core Load Area.

Consider the following example. Core loads ALPHA, BETA1, DELTA, and GAMMA reside in this order—sequence in the Core Load Area. It is desired, for chaining purposes, to use them at on-line time in some other arrangement; say, GAMMA, BETA1, ALPHA, and DELTA. The four core loads are first dumped to cards by a series of *DUMPDATA operations, and then deleted from the Core Load Area. A reload of the new sequence of core loads is now performed in the order desired. The result is a greater efficiency in the usage of the disk by the reduction of disk seek time.

Figure 41 illustrates a possible card deck arrangement for this situation.

**SAMPLE CODING FORM**

<table>
<thead>
<tr>
<th>1-10</th>
<th>11-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Figure 40. Illustrating Various Card Arrangements in Dumping a Program/Data to Nonprocess Working Storage, Punched Cards, and the List Printer
Figure 41. Reloading Core Loads to User Sequence

Since the source specified is the User Area, a LET search is performed. In dumping a program to NPWS, a check is made to see if there is sufficient space in the designated area; if so, a physical move operation takes place. As mentioned earlier (see Example 11) the *DUMP function converts relocatable programs from disk system format to a format of the I/O media selected. (Note that the print format to list printer is identical for both *DUMPDATA and *DUMP).

Figure 42 gives a typical sequence of control cards used.

Note that a relocatable program may also be dumped from the Temporary Area to any I/O media.

A relocatable program dumped from the User Area to punched cards may be later reloaded, if desired, to the User Area by an *STORE function.

EXAMPLE 14. MOVING A DATA FILE (OR FILES) WITHIN THE CORE LOAD AREA. This is equivalent to dumping a data file (or files) from one disk to another -- that is, the copying of process data.

Figure 43 illustrates one possibility.

EXAMPLE 15. LOAD A PROGRAM/DATA BACK INTO THE NONPROCESS WORKING STORAGE (see Figure 44). A reloading operation implies that the program or data to be reloaded is the product of an *DUMP or *DUMPDATA function. That is, they must be in binary format (54 words per card).

Data card input decks have seventy-two columns of data, and a seven-column sequence number. Sequence columns 78-80 are assumed to be numeric (the first being 001) as punched during a *DUMPDATA function. In the reload operation, the card deck is read and stored, and a check made for consecutive sequence, modulo 1000. A sequence break is interpreted as a potential end of the deck. If the card generating the break is a *CCEND card, the *STOREDATA proceeds to store the card data directly into its destination area.

Implementation of LOCALs

An introduction to the term "LOCALs" has already been made in the introductory section: Overview of the IBM 1800 Time-Sharing Executive System. A local is classified in TSX as a subprogram or subroutine that is associated with a given core load, but not initially loaded with that core load. When a call for a local is encountered during the execution of the core load, the local is read in from the disk, overlaying the area between the end of the core load and the beginning of COMMON, unless the local is already in core. Control is then passed to the local routine.

Locals may be employed as individual subprograms or groups of subprograms. In the latter case, whenever a call for a given local is encountered, the entire group of which it is a member is loaded. Subsequent calls for other locals within the same group may then be made without necessitating
Figure 42. Dumping a Relocatable Program from the User Area

Figure 43. Moving a Data File within the Core Load Area

Figure 44. Reloading a Program to Nonprocess Working Storage
reloading from disk. Since all local groups occupy the same area in core, a call for a local in another group will involve a load from disk of the new group, overlaying the first group. At this point, another call to a local in the first group will require reloading that group from disk. Local groups or blocks are important for the following reasons: (1) specification requirements, (2) disk space utilization, and (3) disk efficiency. The area of core used is directly proportional to the size of the largest block. Data is passed to or from a local through its parameters, COMMON, or working storage.

The only difference between a local and a normally called subroutine is that a localized subroutine is not assembled as part of a core load. After the associated mainline program and all of its in-core subprograms (and their in-core subprograms) are relocated, each local subprogram block is converted to core image format and stored, sectorized, immediately following the core image core load, as shown in the schematic diagram below (Figure 45). See also Figure 3.

One advantage of the local feature is that logical subroutines can now be broken off from a large program. This means a virtual extension of core. There is no theoretical limit to the number of localized subprograms that can be implemented: the user can specify any number of locals within a group as long as the sum total of all assembled relocatable programs does not exceed the size of the Local Subroutine Area.

Communications Linkages

At object time, locals are located between the end of the main core load and COMMON. Linkage to and from locals is accomplished via a loader called FLIP (a miscellaneous subroutine within the TSX Subroutine Library) as follows.

**CALLS.** A call to a local consists of a BSI L X, where X is the location of a six-word entry in the Local Parameter Table (LPT) which is built by the Core Load Builder as part of each core load in which locals are specified. The table provides the linkage between the core load and the localized subprograms via the FLIP relocatable subroutine. There is one entry in the LPT for each entry point in the specified local subroutines that are referenced either on an *LOCAL control card, in the mainline program, or in one of the other subroutines localized in that group. Each LPT entry has the following format:

<table>
<thead>
<tr>
<th>WORD</th>
<th>CONTENTS</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>DC 0</td>
<td>A linkage word</td>
</tr>
<tr>
<td>X+1</td>
<td>BSI L</td>
<td>Long BSI to FLIP routine</td>
</tr>
<tr>
<td>X+2</td>
<td>FLIP</td>
<td></td>
</tr>
<tr>
<td>X+3</td>
<td>WC</td>
<td>Word count of the local group with which this routine is loaded</td>
</tr>
<tr>
<td>X+4</td>
<td>SA</td>
<td>Sector address of the first sector for the local group. This address is relative to the first local sector for the core load</td>
</tr>
<tr>
<td>X+5</td>
<td>EP</td>
<td>Absolute address of the entry point when the local group is loaded</td>
</tr>
</tbody>
</table>

The word at X is used for the return linkage; X+1 and X+2 are executed to link to the FLIP routine which uses the word count and sector address at X+3 and X+4 to load the proper local group from disk, if required. The necessity for loading is based upon a comparison of the WC and SA words with those of the local group currently in core. The word at X+5 is the address of the entry point of the local that is called. The FLIP routine stores the return address, at X, into the entry point of the local and then executes a branch to the instruction following the entry point (BSCL EP+1), thus simulating a BSI into the localized subroutine.

**LIBFs.** A LIBF to a local consists of a short BSI to an entry in the Variable Transfer Vector (VTV) associated with each core load. The VTV logic then executes a BSI L Y where Y is the first word of a LPT block, similar to that just described.
Restrictions on the Use of LOCALs

Certain rules apply with respect to the constitution of locals, calling locals, and to calls made by locals. These are summarized below under legal and illegal uses.

Legal Uses

A mainline can call a local. Note that a mainline (which can be a process, interrupt, combination, or nonprocess core load) can, by definition, include any subroutines loaded with the core load. Although routines in the main core load can call locals, all such calls must be completed (that is, corresponding returns to the calling routine made) before any call on a local in a second local group can be made.

A local can call a mainline.

A local can call a skeleton subroutine.

A local can call a local provided both locals are contained in the same local group.

Illegal Uses

A local cannot call another local in a different local group.

Due to the transient nature of local routines, I/O routines cannot be designated as locals.

Conversion routines (e.g., HOLL, EBPA, PRT) cannot be designated as locals.

Interrupt servicing subroutines cannot be designated as locals.

In-skeleton subroutines cannot be used as locals.

Other Considerations

One other restriction in specifying subroutines as locals is that if a subroutine has more than one entry, i.e., EDBR, EDBRX, EDIV, and EDIVX, and more than one entry point is called, then all called entry points must be indicated on the *LOCAL control card.

The user should also beware of hidden locals. If, for example, A, B, and C are subroutines, and A calls B, and B calls C, A and C should not be made locals because C would be hidden from the relocatable loader when A was prepared for loading, and on execution, local C would destroy local A. To overcome this problem, A and B, or B and C, or A, B, and C could be named as locals.

If the Local Subroutine Area includes a device I/O buffer area, no local should exit to a non-blocked (that is, non-local) subprogram until it has tested for a device routine not-busy status.

EXAMPLE 16. In certain application situations, portions of a problem program may not lend themselves to segmentation into individual core loads. In order to overcome this difficulty, by being able to contain such a program in the available machine core size, the local concept is immensely useful. The implementation of the load-on-call facility means that subroutines within the main body of a program can be called into core on demand.

The following example has been devised to illustrate this type of situation. It should not be construed as a model.

Assume a 32K system with a 16K skeleton. If all of FORTRAN I/O were used for all devices called by the nonprocess program, NCATE, core size limitations require that FORTRAN I/O be localized (see Figure 46). Since all FORTRAN I/O taken together
would comprise approximately 3500 words, by localizing them, the largest local block will be only approximately 1600 words, and thus small enough to be accommodated within the 1830 words available. 

Note that subroutines common to FORTRAN I/O (that is, called by FORTRAN I/O and not by the mainline) are automatically included in the mainline such that they may be shared. MAGT, being included in the mainline, can be referenced by either MFIO or UFIO.

Program Listing No. 4 also indicates the order of control cards acceptable to the Core Load Builder. For nonprocess programs, these must all be placed between the *STORECI (or // XEQ) control card and a *CCEND control card. Only the *RCORD control is not allowed.
PROGRAM LISTING NO. 4: EXAMPLE 16

// JOB
// DUAL
*STOREDATAD WSO FXO FILE2 2
DUP FUNCTION COMPLETED

// JOB
// FOR NCATE
*LIST ALL
*NONPROCESS PROGRAM
*IQS(CARD,1643 PRINT,DISK,TAPE)
*ONE WORD INTEGERS
C
C
C NONPROCESS MAINLINE--FORTRAN I/O IS LOCALIZED
C, I/O SEGMENTATION IS ALWAYS DESIRABLE
C
C LARGE DIMENSION IMPLIES SIMULATION OF EXTRA CODE
C FOUND IN LARGE PROGRAM
C
C DIMENSION SPACE(2000),ROOM(500)
COMMON ARRAY(2000),POINT(997),A,B,C
C
DEFINE FILE 1(320,1,1,IFIL1)
DEFINE FILE 2(320,1,1,IFIL2)
C
CALL INOUT1(1)
C
DO HIGHLY SOPHISTICATED PROGRAMMING
C
SPACE(1) = A*B/C+A**B*ATAN(C)-B**2
ROOM(1) = ABS(A)*ALOG(B)*EXP(C/A)/(A+COS(C)+TANH(B))
C
USER-WRITTEN NON I/O DUMMY PROCESSING ROUTINES
C
C WHICH MIGHT BE LOCALIZED
C
C CALL COMPT
C CALL SURCH
C CALL SORT
C CALL CAMPH
C
CALL INOUT2(1)
C
CALL EXIT
END

VARIABLE ALLOCATIONS
ARRAY(RC)=FFE-FO60 POINT(RC)=FOSE-EO96 A(RC)=EO94
B(RC)=E992 C(RC)=E890 SPACE(R)=0FAA-000C
ROOM(R)=1392-0FAC IFIL1(1)=139E IFIL2(1)=139F

FEATURES SUPPORTED
NONPROCESS
ONE WORD INTEGERS
CALLED SUBPROGRAMS
INOUT FATAN FABS FAOLOG FEXP FCOS FTANH FAXB FADD FMAY FDIV FLD FSTO FSTOX FSBR
FDVR FXI

INTEGER CONSTANTS
1=13A2
2=13A3

CORE REQUIREMENTS FOR NCATE
COMMON 6000 INSKEL COMMON 0 VARIABLES 5026 PROGRAM 88

END OF COMPILATION

NCATE
DUP FUNCTION COMPLETED
// FOR
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
*LIST ALL
C
C DO NOTHING I/O STATEMENTS FOR ILLUSTRATIVE PURPOSES ONLY
C
SUBROUTINE INOUT(I)
COMMON ARRAY(2000),POINT(997),A,B,C
C
GO TO (1,2),1

104
C
1 READ(2,3) A,R,C
    FINN(2)  
RETURN
C
2 READ(2,1) ARRAY
    WRITE(1,1) ARRAY
    READ(5) A,R,C
    END FILE 6  
    RACKSPACE 5  
    RETURN
C
3 FORMAT(3F10.3)
END

VARIABLE ALLOCATIONS
ARRAY(RC)=FFFE-F060
POINT(RC)=F05F-F.A9~

STATEMENT ALLOCATIONS
3 =0004  1 =0012  2 =0021

FEATURES SUPPORTED
NONPROCESS
ONE WORD INTEGERS

CALLED SUBPROGRAMS
COMMON 6000 INSSEL COMMON 0 VARIABLES 0 PROGRAM 74

END OF Compilation

INPUT
DUP FUNCTION COMPLETEFD
#STORECIL NCATE NCATE
#FILES(2,FILE2,03)
#LOCAL(MDFIO,MDFI,MDFD,MDAI,MDCOM,MDFX,MOIF,MOIF,MDF,MOIF,MOIF,MOIF)
#LOCAL(UFIO,UFIO,UFIO,UFIO,UFIO,UFIO,UFIO,UFIO,UFIO,UFIO,UFIO,UFIO)
#END

CLN, BUILD NCATE

CORE LOAD MAP
TYPE NAME ARG1 ARG2
#CITH TABLE 4002 000C
#INT TABLE 4002 0028
#FLO TABLE 4031 0010
#ETV TABLE 4041 000C
#VTV TABLE 4044 00AE
#PNT TABLE 40FC 0004
#LPT TABLE 410E 000A
#DFT TABLE 418A 000C
MAIN NCATE 5556
PNT NCATE 40FE
CALL FLIP 5584
LOCN MDFIO 4106 4040
LOCN MDAF 4106 4050
LOCN MDAI 410C 4033
LOCN MDCOM 4112 4056
LOCN MDRE 4118 4059
LOCN MDI 411E 405C
LOCN MDFX 4124 405F
LOCN MDIX 412A 4062
LOCN MDRED 4130 4065
LOCN MDFRT 4136 4068
LOCN MDFND 413C 4068
LOCN MFIO 4142 406F
LOCN MFED 4148 4071
LOCN MURE 414E 407A
LOCN MCOM 4154 4077
LOCN MOIF 415A 407A
LOCN MDAI 4160 407D
LOCN MDI 4166 4080
LOCN MDF 416C 4083
LOCN MDIX 4172 4086
LOCN MDFX 4178 4089
LOCN UFIO 417E 409C

Functions of Executive Programs 105
OCL URFD 414A 40AF
OCL UHR 414A 4092
OCL UINF 4190 4065
OCL UINF 4196 4098
OCL UIAI 419C 4098
OCL UIAF 41A2 409E
OCL UIFX 41A8 40A1
OCL UIIX 41AF 40A4
OCL UCOMP 41B4 40A7
CALL INPUT 5615
LIBF FLN 56A8 40AA
LIBF FMPLY 56B5 40AF
LIBF FDIV 5717 4080
LIBF FSTO 56A2 4083
CALL FAXR 5781
CALL FATAK 57C2
LIBF FADD 588C 4086
LIBF FAXI 590F 4099
LIBF FSER 590C 409C
LIBF FSTNX 5659 409F
CALL FALS 594E
CALL FALOC 594A
CALL FFXP 59FE
CALL FONS 5A7A
CALL FTMW 5B0E
LIBF FDVR 5750 40C2
LIBF ADACK 5862 40C5
CALL BT2BT 5BC6
CALL SAVE 58E2
CALL IOFIX 5C46
LIBF IDII 5C76 40C8
CALL BT1BT 5CC2
LIBF FLOAT 5D32 40CF
LIBF IFIX 5D4E 40CF
LIBF MAGT 5D7A 4091
LIBF SUMN 5F09 4094
LIBF COMGO 5FD2 4097
LIBF EDF 5F20 40A0
LIBF BCXSP 6064 40BD
LIBF RENN 602A 40E0
LIBF FARC 6080 40E3
CALL FTNTR 60E4
CALL FTNRN 60FE
CALL FLN 597A
LIBF FMPLY 5600 40E6
CALL FXSN 5A0E
LIBF XMOS 610E 40E9
LIBF FADIX 5866 40FC
LIBF FSURX 587B 40EF
LIBF FDIVX 5712 40F2
LIBF FLX 5687 40F5
LIBF MDRN 6136 40FA
CORE 6164 072C
CDIIH 6800 1770

CLA, NCCATE LO X0
DUP FUNCTION COMPLETED
General Utility Functions

EXAMPLE 17. PACKING THE USER (RELOCATABLE PROGRAM) AREA. It has been mentioned that during a delete operation, the LET table is searched for the name of the program to be deleted and that entry replaced by a 9DUMY. The space (that is, area) previously occupied by the deleted program remains unused until an *DEFINE PAKDK operation has been performed; it then becomes available for the storage of other programs (through the *STORE function). The user is therefore advised to repack relocatable programs for optimum disk utilization at convenient intervals.

When repacking is performed, the user should ensure that a current record of disk storage exists as a safeguard against any errors which might occur while packing is in progress. The amount of time involved in this operation is directly proportional to the quantity of data moved. The sequence of control cards for a typical packing operation is given in Figure 47. Note that the *DEFINE PAKDK function serves only to pack relocatable programs on disk.

Figure 48 illustrates how various portions of the TSX subroutine library can be deleted or removed from the disk if they are not needed for a given user system. The Relocatable Subroutine Area is then packed to conserve disk space.

EXAMPLE 18. HOW TO REPRODUCE CARDS. When the input to the *STOREDAT function is in card form, this function requires the card deck to be sequenced, modulo 1000, i.e., 1001, 1002, 1003, etc. Any form of input may exist from columns 1-72, as no conversion takes place. Fifty-four words are stored from each source card. The last card of the source input must be *CCEND.

<table>
<thead>
<tr>
<th>SAMPLE CODING FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
</tr>
<tr>
<td>11-20</td>
</tr>
<tr>
<td>21-30</td>
</tr>
<tr>
<td>31-40</td>
</tr>
<tr>
<td>41-50</td>
</tr>
<tr>
<td>1/1 JOK</td>
</tr>
<tr>
<td>1/1 DUM</td>
</tr>
<tr>
<td>*DEFINE PAKDK</td>
</tr>
<tr>
<td>*DUMPLET</td>
</tr>
<tr>
<td>1/1 ENV</td>
</tr>
</tbody>
</table>

Figure 47. Repacking User Area on Disk Drive 1

This function may be used to reproduce source decks prior to assembly (e.g., the TASK source deck), or compilation. An example of Card reproduction is shown in Figure 49.

EXAMPLE 19. DUMPING A LET/FLET TABLE. The *DUMPLET function is used to dump to the List Printer the contents of the LET or FLET or both tables for one or all drives specified during a particular job. The control card sequence for a LET/FLET dump is shown in Figure 50.

The format of a LET/FLET entry is summarized in Figures 51 and 52. A detailed explanation of the contents of both tables is given in the IBM 1800-Time-Sharing Executive System, Operating Procedures, Form C26-3754.

EXAMPLE 20. HOW TO CALL FOR A PROCESS CORE LOAD EXTERNALLY. Once an on-line TSX system has been built, the question remains of initializing or starting system operation. This is only possible through a cold start procedure -- Figure 53 illustrates a typical sequence of control cards for a three-drive system.

The cold start program is supplied with the IBM System and is normally resident on disk. It is read into high-addressed core storage by a two-card bootstrap (COLD START LOADER CARDS 1 and 2), and control passed to its first executable instruction. The Skeleton is then loaded to core storage, and certain mask registers in the Fixed Area are set to /FFFF thus forcing the Skeleton I/O routines to operate in a masked mode. Note that it is the user's responsibility to unmask his system, according to his configuration determined at system generation time, through his initial (that is, first) process core load.

The third control card in the sequence, the COLD START name card, specifies

1. Whether or not storage protection is required (1 or 0 in column 14).
2. Whether or not a request is made for the manual entry of the time of day (1 or 0 in Column 16).
3. The logical assignments of physical disk drives on the system.
4. The name of the initial process core load.

If the storage protection option is elected, the Skeleton I/O, the System Director, the Executive Branch Table, and certain words in the Fixed Area are protected against any user violation. When the
THE FOLLOWING SET OF TSX MONITOR CONTROL CARDS IS USED TO DELETE VARIOUS PARTS OF THE TSX SUBROUTINE LIBRARY IF THEY ARE NOT NEEDED FOR A GIVEN USER-CONFIGURATED SYSTEM.

THE FIRST DELETE ELIMINATES CARON IF THE USER HAS ASSEMBLED THE TASK WITH CARON INCLUDED—NOTE THAT FOR OFF-LINE SYSTEMS CARON SHOULD BE INCLUDED IN TASK SINCE THIS SAVES EXECUTION TIME.

* CORE I.E. CORES EQUATED TO 1

*DUP
*DELET CARON
*IF THE USER DOES NOT HAVE MAGNETIC TAPE ON HIS SYSTEM THE FOLLOWING DELETS APPLY
*DUP
*DELET MAGT
*DELET REWDO
*DELET UFIO
*IF THE USER DOES NOT HAVE PAPER TAPE ON HIS SYSTEM THE FOLLOWING DELETS APPLY
*DUP
*DELET PAPTN
*DELET PAPER
*DELET PAPHL
*DELET PAPPR
*IF THE USER DOES NOT HAVE A PLOTTER ON HIS SYSTEM THE FOLLOWING DELETS APPLY
*DUP
*DELET FCHAR
*DELET SCALF
*DELET PGRID
*DELET FPLOT
*DELET ECHAR
*DELET SCALE
*DELET EGRID
*DELET EPLTX
*DELET POINT
*DELET FCARR
*DELET FRULE
*DELET ECARR
*DELET ERULE
*DELET XVPLT
*DELET PLOTI
*DELET PLOTX
*IF THE USER IS BUILDING AN OFF-LINE SYSTEM THE FOLLOWING DELETS APPLY—NOTE, DO NOT ASSEMBLE AND STORE THE SYSTEM DIRECTOR
*DUP
*DELET CLEAR
*DELET CLOCK
*DELET COUNT
*DELET DRDRT
*DELET ENGTS
*DELET LEVEL
*DELET MASK
*DELET QMRTN
*DELET GIFON
*DELET QUEUE
*DELET REDMK
*DELET SAVMK
*DELET SETCL
*DELET TIMER
*DELET UNMK
*DELET UNG
*DELET VIAQ
*DELET CONMK
*DELET TRNRT
*IF THE USER HAS NO PROCESS I/O ON HIS SYSTEM THE FOLLOWING DELETS APPLY
*DUP
*DELET AIPTN
*DELET AISON
*DELET AIRN
*DELET ANINT
*DELET DINP
*DELET DIEXT
*DELET DINC
*DELET DAOP
*DELET IOPE
*DELET XSAVE
*DELET GAGED
*DELET AIP
*DELET AIS
*DELET AIR
*DELET CS
*DELET CSC
*DELET CX
*DELET DAC
*DELET OZERO
*DELET QZ210
*SUBROUTINE AREA WILL NOW BE PACKED TO CONSERVE DISK SPACE
*DUP
*DEFINE PAKDX 0
*END OF EXTRA SUBROUTINE DELETS

Figure 48. Repacking the Relocatable Subroutine Area Following a Removal of Various Portions of the TSX Subroutine Library
Figure 49. Reproduction of Cards

Note to Figure 52: In a FLET entry, the first two bits in NAME are used to indicate the type of entry:

<table>
<thead>
<tr>
<th>Bit Values</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>data file</td>
</tr>
<tr>
<td>01</td>
<td>nonprocess core load</td>
</tr>
<tr>
<td>10</td>
<td>mainline core load</td>
</tr>
<tr>
<td>11</td>
<td>interrupt or combination core load</td>
</tr>
</tbody>
</table>

Functions of Executive Programs 109
clock option is selected, the user manually enters the time of day in decimal hours and minutes (switches 0-7 and 8-15 respectively): when CONSOLE START is depressed, this is converted into hexadecimal hours and thousands of hours (see also System Design Considerations: System Director).

The assignment of physical disk drive units in a multi-disk system is based on a logical scheme to give maximum flexibility, as shown in Figure 54.

Note that the physical arrangement of the disk drive units (up to three) in a 2310 Disk Storage Unit is fixed in the sequence: disk drive 2, disk drive 0, and disk drive 1. Columns 18, 20, and 22 on the COLD START name card always designate a logical number sequence: 0, 1, and 2 in that order. These columns are used at cold start time to establish a relationship between a physical disk drive (either 2, 0, or 1) and its equivalent logical reference. For example, a 1 punched in column 18 means that a program that references logical 0 will refer to the physical drive (disk drive 1) which was assigned at cold start time to that logical number (0). In Figure 53, physical disk drives 0, 1, and 2 have been assigned to logical 0, 1, and 2 respectively. One of the advantages of this flexibility in assigning physical disk drives in a multi-disk system is backup capability.

### EXAMPLE 21. HOW TO INITIATE A NONPROCESS MONITOR OPERATION.

In an on-line system:

- CALL SHARE from a mainline program only, or
- CALL VIAQ (when the queue is empty). This forces a CALL SHARE.

CALL SHARE is deliberately used when time-sharing is desired at specific times and for specific durations. The amount of time is specified by the I parameter, and is variable depending upon the length of time the user wishes to be away from his process on the mainline level. This time is set in the programmed timer run under Timer C. Time-sharing is terminated when the timer returns to zero or is, alternatively, set to zero by a CALL ENDTS statement (see also System Design Considerations: System Director).

A CALL VIAQ when the Queue Table is empty forces a CALL SHARE statement: the time used in the CALL SHARE is the value set by TISHA (see System Design Considerations: System Director; also Use of Time-Sharing).

In an off-line system, Nonprocess Monitor operation may be initiated by:

1. A COLD START TASK procedure, or
2. Loading a TASK object deck to core with a four-card High Core TASK Loader.

The COLD START TASK procedure is identical to the on-line COLD START PROCEDURE (see Example 20), except that the TASK operating system is now
read into core storage (instead of the System Skeleton). A typical sequence of control cards is shown below (Figure 55).

An alternative method of starting an off-line system is to load a TASK object deck (previously assembled to user specification) to core-storage with a four-card bootstrap loader (High Core TASK Loader). The procedure is summarized below (see also IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754).

- Clear core. The 16 data switches may be set off, or to some predetermined value. Depress CLEAR CORE and START buttons simultaneously.

- Depress STOP button

- Reset registers to zero. Depress RESET button.

- Ready Card Reader. Depress PROGRAM LOAD on reader.

- Set Sense Switch 7 up. Depress CONSOLE INTERRUPT.

- Depress START button (on response to sense switches).

**EXAMPLE 22. HOW TO TERMINATE A NONPROCCESS MONITOR OPERATION (OFF-LINE SYSTEM UNDER TASK CONTROL).** Two methods are possible:

1. Set Sense Switch 7 up
   - Depress CONSOLE INTERRUPT

This immediately aborts the current job being processed, and proceeds to next stacked job.

2. Whenever the Card Reader is empty, the Nonprocess Supervisor will indicate this situation by printing the following message:

**EXAMPLE 23. PREPARING A GUARD (DUMMY) INTERRUPT CORE LOAD.** If an interrupt occurs on a level designated as "out-of-core" and there is no interrupt or combination core load associated with it, the interrupt will be recorded automatically. To prevent this, it is good practice to provide a guard or dummy interrupt core load to service all interrupts on all assigned out-of-core interrupt levels until each interrupt has its final interrupt core load built and stored on disk. The substitute core load should give some indication (such as a message) that the interrupt has occurred.

In the example (Figure 56), levels 8, 9, 10, and 11 were defined as "out-of-core" interrupt levels by the System Director equate cards ICLL1 and ICLL2. The relocatable main program is identified by GUARD located in the temporary portion of LET. Its entry address is 5. The interrupt core load is also identified by GUARD but is in FLET. The DICLE statement specifies that GUARD is entered in the ICL Table for each bit position on each level assigned. When the named program is later deleted and replaced by another program, all of the ICLT entries will be replaced.

**EXAMPLE 24. USE OF THE CONSOLE INTERRUPT.** The Console Interrupt is used by the system and may also be used by the user.

The system uses the Console Interrupt with sense switch 7 on either to abort a nonprocess job or to
commence Nonprocess Monitor action. This operation is required after a // JOB, // END OF ALL JOBS combination.

A block diagram of the generalized logic flow is given in Figure 57.

The user may have an interrupt program executed on a chosen level by depressing the Console Interrupt button with sense switch 7 off. The level is assigned by the user on the equate card CONTA at TASK assembly time.

The servicing routine or core load is written by the user and handled in the same way as a programmed interrupt servicing routine, with appropriate LLBB designations. It is through the programmed interrupt servicing routine that the Console sense/data switches are interrogated and which, in turn, direct this routine to the course of action desired.

One of the functions the servicing routine or core load must perform is to queue up a mainline core load which will notify the Customer Engineer when he can depress the C.E. Interrupt button for the removal or addition of I/O devices from the system; and also to print out error counters where hardware malfunctions have been recorded.

EXAMPLE 25. PREPARING A MAINLINE CORE LOAD TO PERMIT PROLONGED EXECUTION OF THE NONPROCESS MONITOR FOR THE DEBUGGING OF PROCESS PROGRAMS. The example (Figure 58) illustrates the use of the CALL SHARE statement which will continue to provide time for Nonprocess Monitor operation when the increment I for timesharing has expired and the mainline core load is reentered.

This mainline core load will be specified in the COLD START procedure when only nonprocess work is to be accomplished.
Figure 57. Illustrating Logic of Console Interrupt
Figure 58. Illustrating Perpetual Time-Shared Nonprocess Monitor Operation
In industrial control systems, individual user installation requirements may vary from installation to installation either in the hardware itself or in dissimilarities inherent in the application. These differences may take the form of:

- Different processes
- Special process I/O hardware
- Different input–output configurations
- Different core storage sizes
- Response time
- Throughput
- Priority considerations

This means that each installation must be defined or tailored to the specific system function requirements and input–output-configuration of that installation. The tailoring function is defined as system generation which provides the facilities for the creation and maintenance of a monitored system composed of IBM and user-written programs and subroutines. The end product of system generation is a disk-resident operating system which is custom built to provide an efficient Executive System for a specific machine environment.

In the IBM 1800 Time-Sharing Executive System, the builder or "tailor" is a stand-alone monitor program called the Temporary Assembled Skeleton (TASK). TASK permits a system to be constructed on one or more disk cartridges from absolute and relocatable program decks which contain the executable phases and the relocatable programs the installation elects to include in its system. Furthermore, the installation may modify the IBM-supplied configuration, delete functions not required by the installation and add installation-created functions and programs. The modular design and availability of many features and attachable units make possible numerous IBM 1800 configurations tailored to individual application requirements.

System Design Considerations

System Generation

As noted above, System Generation is the process of preparing a specially-tailored operating system to match the machine configuration and operating system options selected by the user. In general, two types of systems may be generated:

1. An on-line system,
2. An off-line system.

On-line System

An on-line system is one that responds continuously to the demands of the real-time world. For example, in industrial process control systems, a number of rapidly changing variables must be monitored, analyzed, and controlled at all times to produce an optimum result. A TSX on-line system implies a real-time operating system in which user-written programs continuously monitor and control a process operation under the command of an executive program (the System Director). The executive provides a means of supervising the use of input–output data and communications channels, evaluating and interpreting data, transmitting and storing information and programs, detecting and correcting errors, and interlacing time-sharing functions. It also controls the system's response to various optional requests, giving priority to emergency demands and postponing low-priority requests that may require considerable time to perform. Emergency actions can be scheduled at frequent intervals. This immediate response is secured through the medium of a powerful and flexible priority interrupt system.

In the on-line mode, the executive also permits the system to be time-shared (when free time is available) by the controlled process and unrelated nonprocess functions. This means that nonprocess programs may be assembled, compiled, simulated, and debugged without interfering with the on-line process. It is the rule rather than the exception that process control programs are subject to change, and it is a definite advantage to be able to implement changes at the installation without taking the system off-line.
Off-line System

An off-line system is completely unrelated to the real-time world, its main purpose being the handling of sequential job operations under the control of a monitor system. A TSX off-line system, by definition, constitutes a stack-job nonprocess monitor system which functions under the direction of TASK. In this mode, nonprocess operations such as assemblies, compilations, disk utility operations, and execution of user-written programs may be performed.

Since TASK core size is considerably less than System Skeleton core size, core storage requirements are less demanding for the off-line system. Also, as the various disk save areas are not now required, disk space is conserved. For those users who do not plan to utilize time-sharing, a nonprocess monitor system working under TASK gives the ability to build coreloads for an on-line system. It is from a non-process monitor system that an on-line system is ultimately constructed.

Figure 59 is a pictorial representation of the steps required for system generation.
For details of step-by-step system generation operational procedures, the user is referred to the System Reference Library, IBM 1800 Time-Sharing Executive System, Operational Procedures, Form C26-3754.

TEMPORARY ASSEMBLED SKELETON (TASK)

TASK EQUATE CARDS

Before TASK can be used to tailor a TSX system, like the System Director, TASK itself has to be assembled from a source deck. To do this, two groups (Groups 1 and 2) of equate cards must be physically placed in the TASK source deck to define the particular system. The relationship of the equate cards to the source deck is illustrated in Figure 60.

The size of the assembled TASK is directly proportional to the number of TASK functions the user elects to include in his system. For example, if he decides to include the complete TASK utility package which will assist him to debug his programs prior to a skeleton build, he equates TRORG to 1. If he decides to include CARDN in the Skeleton 1/O, he equates CDINS to 1. If he decides to make use of the 1053/1816 backup capability, he equates the BD1-BD8 cards accordingly to identify the backup printer(s) assigned.

Like the System Director, TASK can be assembled with extreme flexibility so that no core is wasted by selecting any of the numerous options available. Furthermore, portions of the package can be deleted. The user thus elects a configuration that best matches the functions required. This is illustrated by the example given in Figure 61.

![Figure 60. TASK Source Deck and TASK Equate Cards](image)

![Figure 61. A Set of TASK Equate Cards for the TSX Sample System (see Programming Techniques)](image)
which depicts a set of TASK equate cards chosen for the TSX Sample System described in Programming Techniques. The significance of each of the 60 cards is clearly denoted. The majority of these cards are self-explanatory; a few, however, call for some explanation. These include the following:

- NOBUF
- BZ1-8
- NOCYL
- NUMBE
- INTK1
- INTK2
- CONTA
- ECPT2
- CDINS
- PRICS
- ONLIN
- COMSZ

NOBUF. This label indicates whether or not the buffering of messages to disk is required. It should be equated to zero if the user

1. Has a 16K - 32K system where very few messages on all typewriters are printed.
2. System is restricted in skeleton core space.

NOBUF should be equated to 1 if the user has a 16K-32K system, and has adequate core space (about 300 words) for the buffering feature in TYPEN.

BZ1-8 (Message Unit Size). If the user has adequate core space, makes efficient use of disk space, or if he plans to print long messages, the message unit size should be large. If, however, core space is restricted, but there is sufficient disk space for a number of sectors for the buffering of messages, the message unit size should be small -- that is, of the order of 20 - 40 words.

In general, increasing the buffer size results in a more efficient use of disk space and a corresponding less effective utilization of core storage (see Buffering of Messages to Disk).

NOCYL. This should be basically equal to the largest possible message capacity in disk cylinders at any point in time.

For example, assume that all messages for a specific system are each less than 40 words long, and that the message unit size for all 1053 printers is 40 words. Then, if in any 10 minute period, the user calls for 80 messages to be printed, NOCYL should be equated to 10. That is, an 80-sector buffer is reserved. Note that 8 words of core storage are reserved for every increment of NOCYL (see Buffering of Messages to Disk).

NUMBE. This specifies a limit for the buffering of nonprocess messages. If at any moment, the number of message buffer sectors in use for nonprocess messages is equal to or greater than this number, no further buffering of nonprocess messages will occur until the nonprocess buffer utilization drops below NUMBE. NUMBE must not exceed NOCYLX8.

INTK1, INTK2. These equate cards specify the interrupt levels on which 1816 devices will be serviced on a keyboard request interrupt. If an out-of-core interrupt servicing program is to be used for this purpose, INTK1 and INTK2 must be equated to interrupt levels lower in priority than all other I/O interrupt levels.

CONTA. The user must specify the level to be program-interrupted for the servicing of the Console Interrupt.

The interrupt servicing routine would then interrogate the sense/data switches to determine the course of action required by the interrupt.

One of the uses of this routine is to queue up a mainline core load that will enable the Customer Engineer to utilize the C. E. Interrupt facility. (See INTKY; also Examples of Nonprocess Monitor Usage -- Example 24.)

ECPT2. If two 1053 Printers form part of the user's valid system, he should always define (at least) these two printers as EAC printers for backup purposes.

CDINS. For off-line systems, this should always be equated to 1. Note that this saves about 300 words of variable core. For on-line systems, this should be zero unless the user plans to include in the skeleton a subroutine which calls the 1442 card reader.

PRICS. The user must anticipate what type of arithmetic precision is required in his process programs. He should remember that once this is defined, subroutines used by process programs are assembled with this same precision.
ONLIN. If the user plans to operate (only) an off-line disk monitor system, this should be equated to zero. This gives the user 600 more words of variable core.

COMSZ. This equate card specifies the size of INSKEL COMMON. In an off-line system, this card has no effect since INSKEL COMMON is only present in an on-line system.

BUFFERING OF MESSAGES TO DISK

Efficient I/O handling is the most important single factor in the effective utilization of processor time. Input-output devices, being slow compared to the internal speed of the processor, must be programmed to overlap their operation with mainline computations whenever possible to

1. Greatly increase efficiency of I/O operations
2. Provide more throughput of data.

Consider the following situation. The incore 1053 Printer buffer (whose size is determined by the TASK equate cards BZ1-8) contained within the D.P. I/O subroutine TYPEN is full, and the printer is in the process of writing a message. If disk buffering were provided, the next message called would be temporarily stored on disk, and later returned to core when the current message is completed. This means that the processor-controller is not locked up and waiting for the input-output operation to be completed, and is thus able to continue with its processing.

The significance of disk buffering is that queueing of output messages or information can now be easily accomplished without putting excessive loads on core size or disk access capabilities of the system.

Without disk buffering, the system becomes printer-limited, and might deteriorate into a 15 character per second system.

We thus see that the buffering of messages to disk is important for two reasons:

1. It maximizes processor time. That is, it allows computing to continue after a call to the printer is given.
2. It frees the user from having to optimize his message requests, thus permitting more effective use of the device.

The interrelationship between disk message buffering and total skeleton core requirements can be shown by the following example.

Without buffering:

- Assume four 1053 Printers
- Minimum message unit size = 81 words
- Total core = 324 words

With buffering:

- Assume four 1053 Printers
- Minimum message unit size = 20 words
- Add additional portion of TYPEN = 300 words
- Total core = 380 words

It is seen, in this example, that the user obtains all the advantages of buffering at the small sacrifice of 56 words.

Message Unit Size

The user must define the message unit size for the 1053 Printer(s) attached to his 1800 TSX System at TASK assembly time. The printers may belong to Group 1 or Group 2 on the condition that the maximum number of 1053 Printers used does not exceed 8.

Message unit size is defined as somewhat larger than the average size of the message or information to be printed out. This may be within the range of 20-319 words which is dictated by the minimum and maximum core sizes that may be specified for a message buffer. In practice, an optimum size may fall between 40 and 80 words (80 to 160 characters).

Definition of the message unit size is also dependent on whether messages to the 1053 Printer are to be buffered.

If non-buffering is employed, the message must never be greater than that defined for a message unit. If the user plans to print out long messages or a large number of messages; has adequate core storage, and makes efficient use of disk space, the message unit size should be large. Assuming FORTRAN compilation is planned, the message unit size should be at least 81 words (162 characters).

If buffering is preferred (because the user is pressed for core storage, but has enough disk space for a number of sectors for the buffering of messages), the size of a message can be any length; that is, greater than the size of the message unit. The message unit size can now be defined as small as 20 words (40 characters).

In general, a large buffer size results in a more efficient use of disk space and a corresponding less effective utilization of core storage.
Determination of Disk Buffer Size

The following guide rules may be used for determining the size of the disk buffer:

Rule 1. For random message requests, if the user plans to print out less than 10,000 characters per hour on a single 1053 Printer, the device utilization will be less than 20%. A large percentage of applications falls into this category. In this situation, for a single 1053 Printer, the user will almost never require more than three disk message buffer spaces.

Rule 2. If the user plans to print out a large number of messages in a small interval of time (e.g., data logging at 50 messages), he will require a large number of disk message spaces. The length of the log determines how big the disk buffer shall be.

The following example illustrates a representative calculation. Assume:

1. A 10 message-unit log at the end of every 15 minutes.
2. An average of 50 operational information message units per hour.
3. An average of 10 alarm message units per hour.
4. Message unit size for 1053 Printer (i.e., BZ1) = 50 words.
5. Average length of messages = 30 words.

To handle a 10 message-unit log will probably require 9 sectors; that is, assuming that all 10 messages are called to be typed at the same instant of time. The reason for the 9 is because 1 message is moved directly to the output area, the remaining 9 being buffered on disk.

Let us further assume that the remaining 60 message units are randomly distributed across the hour (that is, 10 in one 10 minute period, and perhaps none in the next 10 minute period, etc.).

Then,

Number of characters typed = 60 x 30 x 2
Time to type these characters = \( \frac{60 \times 30 \times 2}{15 \times 60} \) = 4 minutes

The utilization of the 1053 Printer during the hour is

\[ \frac{4}{60} = 6.67\% \]

Therefore, the number of sectors required for the messages sent at random is 3 (From Rule 1). And the number of sectors required for the log is 9 (From Rule 2). Total number of sectors required is 9 + 3 = 12 sectors.

Now, assuming more than one typewriter is used, sum the number of sectors needed for each additional 1053 printer (computed as above). Let the total overall number of sectors = X.

Then \( \text{NOCYL} = \frac{X + 7}{8} \) complete cylinders (ignore remainder)

In this example, \( \text{NOCYL} = \frac{12 + 7}{8} = 2 \) complete cylinders. If three extra 1053 printers were included to handle random message requests, then from Rule 2, six additional sectors will be required.

\( \text{NOCYL} \) now becomes \( \frac{12 + 6 + 7}{8} = 3 \) complete cylinders.

The user may also use the above guide rules to compute nonprocess disk buffering. Assuming a random message distribution pattern, each 1053 printer will require three sectors. Unless he has excessive disk storage, nonprocess disk requirements should be kept to a bare minimum.

CALCULATING TASK CORE SIZE

1. For an off-line system, the size of TASK is calculated as follows:

\[
\text{TASK} = \text{FIXED AREA} + \text{Disk device tables} + \text{DISKN} + 1053 \text{ device tables} + 1816 \text{ device tables} + \text{TYPEN} + 1053-1443 \text{ Timing Response Routine} + 1443 \text{ device table} + \text{PRNTN} + \text{Constants, work areas, etc.} + \text{CARDN} \text{ (always included)} + \text{TASK Program Set}
\]

Where

\[
\text{TASK Program Set} = 1690 + 8XN + 653XONLIN + 200XTRORG + 221XTA01XTRORG + 358XTA02XTRORG
\]
TRORG, ONLIN, TA01-3, DORG1, and MKLEV are TASK equate cards.

The remaining parameters have already been given in the calculation for Skeleton I/O: see System Design Considerations: System Director.

Once the system is built, the starting address of variable core is found at word 66 hexadecimal (102 decimal) of the Fixed Area. The label of this location is $VCOR$. For an on-line system, the start address of variable core is equal to VCORE.

THE IBM NONPROCESS SYSTEM

The IBM Nonprocess System is a nonprocess system deck which constitutes the major portion of the TSX system. It is composed of control programs and a complete package of IBM relocatable subroutines necessary for the proper execution of the TSX system. A breakdown and brief description of each of its component parts in the order in which it is supplied and loaded to disk follows below (see Figure 62).

Cold Start Cards. The on-line or off-line system is brought into operation by three cold start cards (two Cold Start and one Name Card) which initiate the Cold Start program. A cold start requires that a
minimum of one core load be resident in the core load area on disk for execution. The name of the initial core load as well as the logical assignments of the physical disk drives are obtained from the Name Card.

System Generation (SYSGEN) TASK and Loader Cards. SYSGEN TASK is the "starter" system which contains the basic minimum components for initial system generation. It is loaded to memory by a four-card TASK High Core Loader.

System Loader. The System Loader performs three essential functions at system generation time: 1) loads the IBM Nonprocess System to disk drive zero and file-protects this disk drive from sector 0 to the start of Nonprocess Work Storage, 2) builds the Assignment (AT) and Input-Output Unit (IOUT) Tables and stores them on disk and 3) edits the disk and the Disk Communications Area with a standard layout as a base for TSX non-process programs. It is also used for reload purposes and to make partial modification, if any, to the TSX system.

Disk LET/FLET Tables. LET (Location Equivalence Table) serves as a disk map for system programs, subroutines, and relocatable programs. It contains the name of each function and its size (that is, disk block count, where 1 disk block = 20 words). Each entry point in a subroutine has an entry in the LET table. As the user stores his own relocatable programs on the disk, entries for these programs are also made in LET.

FLET (Fixed Location Equivalence Table) is a map of core loads and data stored in the Process Core Image Storage (or Core Load) Area, and the Save Areas on disk.

Disk Communications Area (DCOM). DCOM is used by all nonprocess system programs and is stored on logical disk drive zero at sector 00000. It is essentially a disk communications map of vital information needed by nonprocess system programs. Some words within DCOM are used by process programs such as Cold Start.

This area is brought into core each time a // JOB is read; certain words are then initialized to reflect the current status of the disk as depicted by the LET/FLET tables. Whenever a // END or // XEQ card is encountered, DCOM is written back to disk.

Bootstrap for Nonprocess Supervisor. This is a relocatable program that can be located anywhere in core for any one system. When VCORE (the start address of variable core) is established, its entry point in variable core is fixed. The bootstrap serves as a linkage between the System Director or TASK, and the Nonprocess Supervisor. It is updated during system generation by TASK, the System Loader, and the Skeleton Builder program. It always resides on sectors 1 and 2 of logical disk drive zero.

Nonprocess Supervisor (SUP). This program directs all nonprocess monitor operations. It decodes the monitor control records in the stacked input for non-process jobs, and calls the appropriate monitor program (Assembler, FORTRAN Compiler, Simulator, etc.) to perform the desired operation. The supervisor provides continuous processor-controller operation during a sequence of jobs that might otherwise involve several independent programming systems. It also supervises the transfer of control between monitor and user programs.

Core Load Builder (CLB). This program constructs mainline, nonprocess and interrupt core loads from user-written programs. Using data contained in control records and in the program itself, the Core Load Builder combines the mainline program, required subroutines, generated work area tables and transfer vectors into an executable core load.

Cold Start Program (CLST). This program initiates the TSX system into operation. In an on-line system, it loads the System Skeleton to core and transfers control to the System Director. In an off-line system, TASK is loaded to core, and control transferred to the first executable instruction within TASK.

Disk Utility Program (DUP). DUP is a set of routines designed to aid the user in performing the functions of disk maintenance. That is, it has the capabilities of storing, deleting, and outputting user programs, defining system and machine parameters, and also of maintaining communications areas. DUP also automatically updates the LET/FLET tables to reflect all changes to the disk. It is called into memory by the Nonprocess Supervisor.

Assembler (ASM). The Assembler is a disk-oriented symbolic assembly program that translates programs written in symbolic language into machine language. Basically, it is a one-for-one type assembly program. Provision is also included for the user to easily make use of input-output, conversion, and
arithmetic subroutines that form a part of the subroutine library.

FORTRAN Compiler (FOR). This is a disk-oriented program that translates programs written in the FORTRAN language into machine language, and automatically provides for the calling of the appropriate arithmetic, functional, conversion, and input-output subroutines.

Simulator (SIM). The Simulator provides the user with the means for testing and debugging programs without disruption to the on-line process.

Error Programs. This is a collection of error subroutines called by the TSX Error Alert Control (EAC) program. EAC is executed when an internal or TSX detected error occurs.

Subroutine Library. The Subroutine Library is a package of IBM TSX and user-written subroutines resident in the relocatable subroutine area of disk. IBM TSX subroutines include: Real-time subroutines, Arithmetic and Functional subroutines, Conversion subroutines, FORTRAN I/O subroutines, and DP I/O subroutines.

Skeleton Builder. The Skeleton Builder uses tables constructed by the System Loader, user-assigned control records, and user-specified programs and subroutines to build the System Skeleton. The System Skeleton constitutes that portion of the system that remains in core during the execution of a TSX on-line system.

Stand-alone Utilities. Five of these optional utility routines can be loaded and executed only under the control of TASK (in an OFF-line system) or the System Director (in an ON-line system). The five utilities are: TASK Card to Disk, TASK Disk to Card, TASK Disk Patch, TASK Disk Duplication, and TASK Disk Loader.

Two self-loading programs are included: Relocatable Dump to Cards (CRDMP) and Relocatable Dump (UT7A).

A dump analysis program (DMPAN) is included. It consists of two parts, DMPA1 and DMPA2. DMPA1 must be loaded for execution by the TASK absolute loader.

DMPA2 is loaded and executed as a non-process core load.

System Director. This is the nucleus of the System Skeleton. It maintains control over the on-line process application by servicing all interrupts, handling error conditions, providing timer control over the process, and process program sequencing. The System Director is supplied as a source deck.

TASK. TASK is a "builder" operating system which controls the system generation process, and provides for the definition of the TSX system according to user specifications. It is supplied in source format.

Note that control programs are supplied assembled in absolute format; subroutines, in relocatable format. The System Director and TASK are the only exceptions: they are supplied as source decks.

In its original form, the IBM Nonprocess System does not contain those parameters which define and differentiate a system currently under construction from another, and is therefore unsatisfactory for direct use by a customer installation. Variability of interconnection of input-output devices is, however, permitted at the hardware level, and it is these variations which need to be communicated to the I/O subroutines if correct and intended operation is to be realized.

This communication is accomplished through the medium of the System Loader which accepts as input a statement of the system configuration (including the correlation between external device and interrupt identification, and internal hardware-sensed codes) and the IBM Nonprocess System master deck. To ready the IBM Nonprocess System for system loading, data from assignment cards is integrated into the master deck.

SYSTEM LOADER OPERATION

The System Loader assumes at system load time that only one disk drive (logical disk drive 0) is present on the system. After the IBM Nonprocess System is loaded, the user has the option of relocating certain disk areas (such as the Core Load Area, Process Work Storage, etc.) to an auxiliary disk drive or drives. This and other aspects of disk organization are discussed in System Design Considerations: Disk System Configuration.

Three essential functions are accomplished during a system load operation. These are:

- Loading the IBM Nonprocess System, including the subroutine library, to disk
- Building various TSX operating tables
- Editing the disk layout

Loading the IBM Nonprocess System

A typical sequence in which the input programs are loaded by the System Loader is given in Figure 63.
Each program in the IBM Nonprocess System is preceded by an *LDDSK Control card which is read and analyzed by the System Loader. As a single sector at a time of a program is accepted, the appropriate sector address to which it is written on disk is determined by the first two words following a sector break record. A sector break record is a header record which serves two purposes:

- Enables the System Loader to establish a new disk sector either at a relative or absolute sector address
- Indicates if the phase of a program being read in involves either a principal I/O device or a principal print device, and, if any, which one.

Each phase within a program contains one sector break record. For example, since the FORTRAN Compiler is made up of 27 phases, it has 27 sector break cards. Sector break records are supplied in binary format (see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754. A separate discussion of sector break records is given at the conclusion of this section).

As each program is loaded to disk, an entry in the respective LET/FLET tables is updated accordingly. Note that the relocatable subroutine library may include user-written subroutines provided they are assembled/compiled by the TSX Assembler/FORTRAN Compiler. During the System Load stage, an error program within the System Loader ensures proper handling of error situations. Figure 64 reflects the layout of the IBM Nonprocess System on logical disk drive 0 after a system load operation.

Building the TSX Operating Tables

After the assignment cards have been read, two tables are built: 1) the Assignment Table, 2) the I/O Unit Table.

The input to the table-building phase are the assignment cards which are prepared by the user and merged with the IBM Nonprocess System.
The Assignment Table (AT) serves to inform the Skeleton Builder (at Skeleton build time) which I/O device or PISW is assigned to a specific ILSW bit on a specific interrupt level. A 16-bit (IAC) code entry is furnished for each ILSW bit, which the Skeleton Builder later replaces by a branch address to transform it into the Master Branch Table (MBT). The (AT) table is stored on disk in reverse sequence; that is, level zero in highest location, etc. The number of AT entries and I/O interrupts are counted during the table build process and stored in sector 1 of logical drive zero.

The I/O Unit Table is constructed from the logical unit number (LUN) and/or its associated interrupt assignment code (IAC). The table is 44 words in length and is built in descending sequence; a maximum of 19 entries is allowed. The IOU Table is stored in the last 87 words of sector 2.

Editing the Disk Layout

The disk editing phase is entered after all absolute (or core image) and relocatable programs have been stored on disk, and the *DEDIT control card has been read.

The editing function initiates the disk and disk communications area with a standard layout as a base for TSX nonprocess programs. It uses LET/FLET and DCOM as communications areas.

In order to fix the boundaries of the various disk areas, certain information is required:

- Size of core of the Object Machine. This should be specified on the *DEdit control card; otherwise, the source core size is construed as object core size.
- Size of Message Buffer. Note that the only area definition made by the user before the IBM Nonprocess System is loaded is the length of the message buffer. This must be specified on the *DEdit control card and should correspond to NOCYL (TASK equate card) at TASK assembly time. The calculation of message buffer size is discussed at some length in the section System Design Considerations: TASK.
- Size of IBM Nonprocess System areas. These are made known to the System Loader after the system is loaded to disk.

Note that the boundaries of the following areas:

- Nonprocess Save Area
- Process Save Area
- Special Save Area

depend on the estimated size of the System Skeleton (see System Design Considerations: Disk System Configuration).

LET/FLET Entries. Fixed entries, derived from control cards, exist in LET for the following:

- Disk Communications Area (DCOM)
- Master Branch Table/Assignment Table (MBT/AT)
- Skeleton Subroutine Map (SK-SUB)
- Nonprocess Supervisor (SUP)
- Core Load Builder (CLB)
- Disk Utility Program (DUP)
- Assembler (ASM)
- FORTRAN Compiler (FOR)
- Simulator (SIM)

An entry for each subroutine is made while it is being loaded.

FLET entries, on the other hand, are made from computed and assumed sizes for the following:

- Cold Start
- Error Programs
- Message Buffer

After these FLET entries have been made, the .E entry of LET is updated to reflect the boundaries of the Nonprocess Work Storage for the remaining disk space available.

DCOM Entries. The first sector address of each of the following areas are entered in DCOM:

- Nonprocess Supervisor (SUP)
- Disk Utility Program (DUP)
- Assembler (ASM)
- FORTRAN Compiler (FOR)
- Simulator (SIM)
- Location and Fixed Location Equivalence Tables (LET/FLET)
- Nonprocess Work Storage (NPWS)
FUNCTION OF THE *ASSIGNMENT CARDS

The assignment card serves to assign the various I/O devices and machine functions to a particular interrupt level and bit. Assignments are in the form of interrupt assignment codes (IAC) which are fixed for each device, and logical unit numbers (LUN) which are selected by the user for linkage to user-written FORTRAN programs.

Through the assignment card, the user

1. Assigns IAC codes to the various interrupt levels and ILSW bits (within the level used on the system).
2. Assigns LUN numbers as they are used in user-written FORTRAN programs, to certain data processing input-output (DP I/O) devices by equating them to corresponding IAC codes.

Interrupt assignment codes uniquely define all process interrupts, I/O devices, console interrupts, and interval timers. They are fixed and may not be changed by the user. Their values range from 00 through 63.

Logical unit numbers on the other hand are used to identify DP I/O devices in user-written FORTRAN programs, and are specified by the user at system load time. The LUN's are entered into the I/O Unit Table to permit communication of FORTRAN programs with FORTRAN I/O at object time. Once fixed, they cannot be changed without repeating the Assignment Table building phase of the System Loader and Skeleton rebuild under certain conditions, as well as the recompilation of every user-written FORTRAN program utilizing DP I/O devices affected.

A maximum of 19 different LUN's is possible on a TSX system with a full complement of I/O devices. LUN values range from 01 through 44. Note that no LUN may be assigned to more than one particular device. In a minimum (8K) TSX system, it is advisable, for purposes of space conservation, to use the lowest LUN numbers first, since the System Loader will build a table providing space for all LUN's up to the largest number assigned. Keeping LUN numbers small, therefore, conserves core storage. The reader should refer to IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754 for details of assignment card formats and operational procedures.

Examples of the Use of LUN Numbers/IAC Codes

Consider the following assignment cards:

EXAMPLE 1.

```
03 06 01 /1,2,157/01,05,55,06
```

Level 3 contains 6 ILSW bits. IAC 01 represents an 1816/1053 printer which has a LUN of 41 assigned to it, while IAC 02, representing a 1442 card/read punch, has the same LUN as its IAC; that is, it requires no LUN entry. The combination 37/01 represents another printer to which a LUN of 01 is assigned by the user; IAC 05 represents a 1627 plotter unit with the same LUN number as its IAC code (that is, 05). IAC 33 represents a process interrupt. IAC 08 represents a 2310 disk drive which has no assignable LUN number.

EXAMPLE 2.

```
99 01 43/01,44/09
```

The 1816 keyboard on group 2 has a LUN of 1 while the second magnetic tape drive has a LUN of 9.

EXAMPLE 3.

```
//SYSTEMLOADER
*ASSIGNMENT
00 02 33,00
01 04 33,04,08,09
02 02 33,14
03 05 33,01,36,37,38
04 03 33,34,35
05 05 33,10,16,11,12
06 02 33,06,03
07 02 33,02
08 03 33,32,05,07
09 01 33
10 01 33
11 03 33
12 01 33
13 05 33
*CCEND ASSIGNMENT
```

```
DEVICE LEV BIT IAC LUN
PISW 00 00 33
TIMERS 00 01 00
PISW 01 00 33
DISK-1 01 01 04
DISK-2 05 02 08
DISK-3 05 03 09
PISW 02 00 33
MAGT-1 02 01 14 14
PISW 03 00 33
TYPAG2 03 01 01 01
TYP2DG 03 02 36 36
TYP3G2 03 03 37 37
TYP4G2 03 04 38 38
PISW 04 00 33
COMP-1 04 01 34
COMP-2 04 02 35
PISW 05 00 33
ADC-1 05 01 10
ADC-2 05 02 16
DINP 05 03 11
DADP 05 04 12
PISW 06 00 33
PRNT-1 06 01 06 03
PISW 07 00 33
CARD-1 07 01 02 02
PISW 08 00 33
CONSDL 08 01 32
PLOT-1 08 02 05 07
PISW 09 00 33
PISW 10 00 33
PISW 11 00 33
```

YOU DEFINED 000018 I/O DEVICES
AND A TOTAL OF 000031 ILSW BITS
This illustrates an example of user assignment of I/O devices and process interrupts to 12 levels of interrupts defined in a sample machine configuration given in System Design Considerations: System Director.

Note that only two I/O devices have been assigned LUN numbers:

1627 Plotter (IAC = 05) = 07
1443 Printer (IAC = 06) = 03

The remaining devices use their IAC codes (a user option) as LUN's. Note also that process interrupts and certain DP I/O devices have no assignable LUN's. The map correlates each process interrupt or device with its level, bit, IAC code, and LUN (if any).

Note that IAC/LUN groups may contain either the IAC code alone or a combination of the IAC code and the LUN as assigned by the user to that IAC (and separated by a slash). When the LUN number is omitted, it means that either no LUN is defined (that is, not assignable) or that the System Loader considers the LUN to be identical to the IAC code. The user has the option of assigning the value of the corresponding IAC code to the LUN for a particular device.

**Devices with no Interrupt-entry on any Level**

The 1816 Keyboard units on printer groups 1 and 2 and the second Magnetic Tape drive have no separate defined interrupts, their interrupts being the same as that of the first 1816/1053 printer and first magnetic tape drive respectively. However, a LUN has to be assigned to them whenever they are used in connection with FORTRAN programs. In these three special cases, a dummy interrupt level number 99 is defined, followed by a standard format entry for bit count and IAC code. The dummy level 99 can be omitted should all three possible devices have a LUN identical to their IAC code.

**THE *DEDIT CONTROL CARD**

The *DEDIT* Control card starts the disk editing phase: that is, it starts the function of editing the layout of the disk during which time the System Loader uses LET/FLET and DCOM as communications areas. Some of the activities carried out during this phase include (see *Editing the Disk Layout*):

1. Initialization of the FLET area on disk
2. Calculation of the source core size
3. Entry of the object core size into the disk communications area (DCOM)
4. Entry of message buffer size in cylinders into DCOM
5. File protection of the IBM Nonprocess System

**Parameters**

Two important parameters must be specified by the user:

1. Size of core of the object machine
2. Size of message buffer size

The calculation of the core size of the source machine (that is, the machine on which the IBM Nonprocess System is loaded) is achieved by TASK and the result is stored in the Fixed Area in core. The System Loader then places this result in DCOM. The user may exercise the option to define a different core size for the object machine (that is, the machine on which the TSX system is executed). This will also be stored in DCOM. If the object core size is not specified on the *DEDIT* card, the source core size will serve as object core size.

As noted earlier, the only area definition made by the user before the IBM Nonprocess System is loaded is the definition of the length of the disk message buffer. This is specified in cylinders in the *DEDIT* card and must equal NOCYL (TASK equate card). The calculation of the size of the message buffer is discussed in detail in System Design Considerations: TASK.

An example of the use of the *DEDIT* card is given below:

```
*DEDIT 16K 011CYL
THE SOURCE CORE-SIZE IS 016384
THE OBJECT CORE-SIZE IS 016384
```

The *DEDIT* control card is the last card recognized by the System Loader.
Reentering the Disk Edit Phase

The disk editing function permits a reentry by the user after the IBM Nonprocess System is loaded and control returned to SYSTEM TASK. This may be needed for:

1. Rebuilding the FLET table.
2. Changing the Message Buffer Size.
3. Changing Object Core Size.
4. Changing the assignment of LUN numbers, such as, for example, if an error was made in the user-assignment of an IAC or a LUN.

SUMMARY OF ASSIGNMENT CARD RESTRICTIONS

Assignment designation is governed by the following rules:

1. A separate assignment card is used for each interrupt level. Assignment cards may be in any order of interrupt level number.
2. The number of IAC/LUN codes specified per level must be equal to the number of interrupt level status word (ILSW) bits used.
3. Only the IAC code 33 (for process interrupts) may be used more than once. In the case of LUN numbers, the same LUN cannot be assigned to more than one device, nor can a device have more than one LUN assigned to it.
4. For IAC codes 42, 43, and 44, a dummy interrupt level entry of 99 must be specified. These refer to the 1816 keyboards on printer groups 1 and 2 and the second magnetic tape drive.
5. For RPQ devices, IAC codes 20 - 31 and 45 - 63 may be used. In any TSX system, IAC codes 00, 02, 04, and 32 must be used; 01 or 06 must also be used.
6. If more than one group of process interrupts are assigned to a particular level, the second group must be treated as an RPQ device, given an RPQ IAC code and a user-written ISS subroutine to accommodate this device. The subroutine will indicate to the System Loader which IAC code it responds to; it will have to be core-resident at all times.

SECTOR BREAK RECORDS FOR ABSOLUTE PROGRAMS

Absolute programs are generated by an absolute 1800 assembly and are loaded by the System Loader, one record at a time (taking into account all data breaks and origin changes), to disk in true Core-Image Format. That is, each program resides on disk in exactly the same format in which it will reside in core storage. Core-Image Format is also called Data Format because a program thus stored on disk can be transferred to core by a single call (to DISKN) without any data manipulation. All IBM system programs (e.g., Assembler, FORTRAN Compiler, Simulator) are stored in this format.

However, it is from a header or sector break record that the absolute loader portion of the System Loader determines the sector address at which succeeding data is to be stored. The sector within which the data is to be stored is first read into core, one word of data at a time, until that sector in core is completed. When full, the 320-word sector buffer is written to disk and the next sector break record is read to locate the next sector to be written.

Four types of Sector Break Records are used by the System Loader:

Type 1
Type 2
Type 9
Type E

Note that Type F cards are "trailer" or "transfer" cards which occur at the end of a binary deck. The format of each card is in the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

From each type of sector break record, the System Loader interprets the sector address as follows:

Type 1: As an absolute address.
Type 2: As a displacement from the last sector loaded.
Type 9: As a displacement from the last absolutely defined sector address (that is, defined by a Type 1 sector break record).

Type E is a special sector break record type used only by the Simulator subroutine package. It is treated by the System Loader like a Type 1, except that it causes data to be streamed to the disk contiguously, ignoring data breaks brought about by BSS's; or by an ORG to the same location as the data card immediately following the Type E record.

The Type 1 sector break record is generated by an ABS statement in an absolute assembly; Type 9
or Type E records must be inserted manually in the object deck by the user. Type 2 sector break records are generated during 1) assembly of mainline-type programs without an ABS statement, and 2) FORTRAN compilation of mainline programs.

The sector address on disk to which the System Loader begins writing the program is defined in the second word of the program following the ABS and first ORG statements. The first word may contain any value; no word count is required. An example is given below:

<table>
<thead>
<tr>
<th>SAMPLE CODING FORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>21-30</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>DC</td>
</tr>
<tr>
<td>/0538</td>
</tr>
</tbody>
</table>

The first DC is at location /0538, the second at /0539. The System Loader will start to load at sector /0100. The first word of the sector is at /053A -- the content of /0538 is not loaded to disk since it does not constitute an integral part of the program. If this program was later called from the disk, the word count and sector address would be specified by the AREA (portion) of the disk call required at /0538 for proper execution.

If the first two words of the program are followed by another ORG statement, as shown below, the program will be placed in a location on disk reflecting a displacement from the address defined by the first ORG.

The program will be loaded starting at position 15 of sector /0100, leaving positions 0 through 14 at whatever value they previously had on disk. The displacement D could have any value -- thus skipping over several sectors.

Note, however, that D cannot have a negative value. This is a necessary requirement of the TSX system which is designed such that it is impossible to inadvertently destroy a program residing on preceding sectors by back-origining. Thus, the lowest origin in the program is required to be immediately after the ABS statement. As shown below, this constitutes, in no way, a system limitation. Note that normal back origins, as they occur in every program, are perfectly legal.
The program will be correctly located on disk reflecting exactly the layout in core.

The following example illustrates an ERROR STOP (System Loader error message - LO5) situation.

This is clearly illegal because whatever program that was residing on sector /00FF would be destroyed.

The final example below illustrates Type E sector break record functioning:

If this sequence is preceded by a Type 1 or Type 9 sector break record, the data up to the second ORG would not appear on disk, but would be overlaid by the second sequence. If, however, the sequence is preceded by a Type E card, all the data would appear on disk, and the data following the second ORG statement would immediately follow the first with no sector break being forced.

**SYSTEM DIRECTOR**

When the IBM Nonprocess System is loaded, assignment cards supply the System Loader with data which relate to the interrupt level allocation of I/O units, process interrupts, interval timers, etc. That is, they provide a statement of the system and interrupt configurations.

At System Director assembly time, the System Director must also be tailored to meet the exact requirements set by the user. These requirements include:

1. Definition of the size of the System Director
2. Definition of functions required
3. The allocation of internal and external interrupt levels
4. The number of CALL Count routines to be included by the user
Since the System Director must be assembled and stored on disk before the TSX System Skeleton can be built, some means must be employed to make the System Director aware of these requirements. To achieve this, a set of System Director EQUATE cards (provided pre-punched by IBM) is prepared by the user and physically placed in the System Director source deck. The resulting integrated deck is then assembled under the control of an off-line nonprocess monitor. Figure 65 depicts the physical relationship of the EQUATE cards to the System Director Source Deck.

Figure 66 illustrates an example of a complete set of System Director EQUATE cards. In terms of definition requirements, the set can be broken down into convenient subsets as follows:

- **a) Size of System Director**
  - VCORE, NUQUE; also a function of b), c), and d)
- **b) Functions required**
  - (Interval Timer Control, ITCUS, TBASE, CBASE, TIME1-2, TISHA; TIMES; OPMOI and DUMP1)
- **c) Allocation of internal and external interrupt levels**
  - NULEV, USE00-23, NB00-23, NIL00-23, NLWS1-2, and ICLL1-2
- **d) Number of CALL Count routines required by user**
  - NITP1-2

### SIZE OF SYSTEM DIRECTOR

Since the System Director is a component part of the System Skeleton, it must be core-resident at all times in an on-line system in order to respond to the real-time world. Its required core size will, however, vary according to the user's machine configuration, process requirements, and other options. For example, if the user specifies when the system is assembled that time-sharing is to be used, the Time-Sharing Control (TSC) program will be included in core. If he has no use for time-sharing, TSC may be eliminated.

Similarly, if the user specifies that interval timers are not used, the Interval Timer Control (ITC) program as well as TSC may be eliminated. It is a rule, however, that ITC must be in core if time-sharing is utilized. The Program Sequence Control (PSC), Error Alert Control (EAC), and Master Interrupt Control (MIC) programs must always be used, but each is variable in size according to the number of interrupt levels elected.

In addition, a work area is associated with each interrupt level; for example, if the user elects 12 levels of interrupts, 12 work areas are required; if he elects 24 levels, the System Director will require 24 work areas. Three other additional work areas are included: one each for Error, Mainline, and Nonprocess.
### Mainline Core Load Queue Table

Resident within MIC is a Queue Table made up of three-word entries used for the stacking of main­line core loads requested for execution, as shown in Figure 67.

Each time the QUEUE routine is called, an entry is made in the queue if there is not a like entry of equal priority and sector address already in the queue. Entries are removed from the Queue Table by the subroutines UNQ and VIAQ (see Program Scheduling).

The size of this table -- that is, its maximum number of entries -- is specified by the user on the NUQUE equate card. It should be large enough so that the Queue Table shall not overflow under normal operating conditions.

VCORE determines the starting address, which must always be even, of the variable core area. The appropriate value of VCORE can be arrived at by calculating the size of the System Director, Skeleton I/O and the user-written subroutines.

#### Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

```plaintext
Figure 66. Example of a Set of System Director Equate Cards

<table>
<thead>
<tr>
<th>ENTRY 1</th>
<th>ENTRY 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD COUNT</td>
<td>WORD COUNT</td>
</tr>
<tr>
<td>SECTOR ADDRESS</td>
<td>SECTOR ADDRESS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>
```

### Figure 67. Mainline Core Load Queue Table

```plaintext
Figure 67. Mainline Core Load Queue Table

Mainline Core Load Queue Table
```

```plaintext
Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

```
Figure 66. Example of a Set of System Director Equate Cards
```

```plaintext
<table>
<thead>
<tr>
<th>ENTRY 1</th>
<th>ENTRY 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD COUNT</td>
<td>WORD COUNT</td>
</tr>
<tr>
<td>SECTOR ADDRESS</td>
<td>SECTOR ADDRESS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>
```

```plaintext
Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

```
Figure 66. Example of a Set of System Director Equate Cards
```

```plaintext
<table>
<thead>
<tr>
<th>ENTRY 1</th>
<th>ENTRY 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD COUNT</td>
<td>WORD COUNT</td>
</tr>
<tr>
<td>SECTOR ADDRESS</td>
<td>SECTOR ADDRESS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>
```

```plaintext
Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

```
Figure 66. Example of a Set of System Director Equate Cards
```

```plaintext
<table>
<thead>
<tr>
<th>ENTRY 1</th>
<th>ENTRY 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD COUNT</td>
<td>WORD COUNT</td>
</tr>
<tr>
<td>SECTOR ADDRESS</td>
<td>SECTOR ADDRESS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>
```

```plaintext
Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

```
Figure 66. Example of a Set of System Director Equate Cards
```

```plaintext
<table>
<thead>
<tr>
<th>ENTRY 1</th>
<th>ENTRY 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD COUNT</td>
<td>WORD COUNT</td>
</tr>
<tr>
<td>SECTOR ADDRESS</td>
<td>SECTOR ADDRESS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>
```

```plaintext
Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

```
Figure 66. Example of a Set of System Director Equate Cards
```

```plaintext
<table>
<thead>
<tr>
<th>ENTRY 1</th>
<th>ENTRY 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD COUNT</td>
<td>WORD COUNT</td>
</tr>
<tr>
<td>SECTOR ADDRESS</td>
<td>SECTOR ADDRESS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>
```

```plaintext
Calculating System Director Core Size

As discussed above, core size is a function of several parameters which are in turn determined by the number of features the user elects to include in his TSX system. The computation of this value in 16-bit words can be simplified by using certain equate card entries as multiplication factors as shown below, where System Director Core Size is given as a summation of the following (these figures may change with modifications and versions of the system):

```
Figure 66. Example of a Set of System Director Equate Cards
```

```plaintext
<table>
<thead>
<tr>
<th>ENTRY 1</th>
<th>ENTRY 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD COUNT</td>
<td>WORD COUNT</td>
</tr>
<tr>
<td>SECTOR ADDRESS</td>
<td>SECTOR ADDRESS</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRIORITY</td>
</tr>
</tbody>
</table>
```
1116 (constant for MIC, PSC, and EAC and their work areas)
+ 220 (if ITC is included: that is, when ITCUS = 1)
+ 95 (if EAC dump is required: that is, when DUMP1 = 1)
+ 109 multiplied by the number of interrupt levels (that is 109 x NUL23)
+ 3 multiplied by the number of Queue entries (that is, 3 x NUQUE)
+ 2 multiplied by the number of process interrupts (that is, 2 x sum of NIL00 through NIL23)
+ 2 multiplied by the number of programmed interrupts on levels 0 through 13 (that is, 2 x NLWS1)
+ 2 multiplied by the number of programmed interrupts on levels 14 through 23 (that is, 2 x NLWS2)
+ 2 multiplied by the number of count subroutines 0-15 (that is, 2 x NIPT1)
+ 2 multiplied by the number of count subroutines 16-31 (that is, 2 x NIPT2)
+ 340 (334 + 6 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 10)

From the configuration set out in Figures 66 and 68, a typical calculation is deduced below.

System Director Core Size = 1116
+ 220
+ 95
(109 x 12) + 1308
(50 x 3) + 150
(2 x 57) + 114
(2 x 12) + 24
(2 x 6) + 0
(2 x 16) + 32
(2 x 8) + 16
TSC + 34
3409 words

DEFINITION OF FUNCTIONS REQUIRED

Interval Timer Control

When the ITCUS label is equated to 1, the ITC program is included within the System Director and serves to set up user-specified times and correct linkages to the user's subprograms. Once this is done, ITC will control the timers until one or more specified intervals have elapsed, at which point control is transferred to a user's subprogram.

Specifications for any timer may be set or changed in relation to the timer base at any time during an on-line process operation by the calling sequence.

It was mentioned in Functions of Executive Programs: The System Director, that a programmed real-time clock, a time-sharing control timer, and nine programmed interval timers are controlled (that is, updated) by the third machine interval timer C. It is, however, the user's responsibility at assembly time to establish:

1. A primary time base (TBASE) for the real-time clock; that is, how often the clock should be updated.
2. A secondary time base (CBASE) for the programmed timers and time-sharing control timer.

Primary Time Base

This is that interval of time used to update the real-time clock, and is called the Interrupt Time Base. It is the product of the wired-in hardware time base and a number chosen by the user (TBASE) at assembly time, expressed as follows:

\[ \text{INTERRUPT TIME BASE} = (\text{WIRED-IN HARDWARE TIME BASE}) \times (\text{USED-ASSIGNED NUMBER}) \]

For example, if the machine interval timer C is wired for a four millisecond time base and the real-time clock is to be updated every two seconds, the user-assigned number can be calculated to be 500. TBASE is thus equated to minus (-) 500. A negative number is used because the interval timer is incremented in the positive direction, causing an interrupt when zero is reached. The primary time base for the real-time clock in this example (that is, how often it is to be updated) is thus two seconds.

To enable ITC to keep track of elapsed time since the last or previous interrupt occurred, a double-word TIME1 and TIME2 is equated to the hexadecimal equivalent of the interrupt time base. This value is added to the real-time clock each time an interval timer C interrupt occurs.
In the above example, TIME1 and TIME2 are equated to /0000 and /07D0. The label TIME1 is always /0000 unless the calculated interrupt time base exceeds 65,535 milliseconds.

The NORSP equate card is used to specify the time period that elapses between no-response checks of the 1053 and 1443 printers. NORSP must be a positive decimal integer, ranging between 1 and 127, and is the number of timer C interrupts that occur before the no-response check is made. For example, if timer C is wired to a four millisecond time base, TBASE is equated to -500, and NORSP is equated to one, two seconds will elapse between no-response checks. NORSP should be adjusted to give a time value between two and three seconds.

**Secondary Time Base**

The programmed timer base for the nine programmed timers and time-sharing control timer is a user-assigned multiple of the interrupt time base established for the real-time clock, and expressed as follows:

$$\text{PROGRAMMED TIMER BASE} = (\text{INTERRUPT TIME BASE}) \times (\text{USER-ASSIGNED NUMBER})$$

For example, if the interrupt time base is fixed at two seconds, and the user wants the programmed timers to operate at 30-second intervals, the label CBASE is equated to 15.

This base is used specifically for the nine programmed timers and the time-sharing control timer, and is the smallest interval of time that can be specified for the programmed timers or for time-sharing operations.

**Time-Sharing**

The TIMES label specifies at assembly time whether or not time-sharing is to be used.

It was noted in the preceding section that the programmed timer base is the smallest interval of time that can be specified for programmed timing or time-sharing operations. When time-sharing is used, a user-assigned multiple of the programmed timer base is established.

For example, if the programmed timer base is fixed at 30 seconds and the user desires time-sharing operations of two minutes' duration whenever the queue is empty, the label TISHA is equated to 4. Thus, the time-shared operation is terminated whenever the time interval specified (in this case two minutes) has elapsed. TISHA is identical to the parameter I in the requesting CALL SHARE statement in the mainline program. If the user wishes to remain in time-sharing until some core load name is put into the queue by an interrupt program which uses CALL ENDTS, then TISHA may be specified for the longest possible numerical value, that is, 32767. The reason for this is to keep the time-sharing function from exchanging core unnecessarily at frequent intervals to check the Queue Table when no entries have been put in the queue. This is the recommended procedure.

**Operations Monitor**

The user may select an option in ITC to reset the Operations Monitor (a hardware feature) during nonprocess operations. He does this by equating the OPMOI card to 1 or 0: a 1 indicates that the monitor is to be reset by ITC; a 0 indicates that the monitor is to be reset by user program control. It should be noted that the Nonprocess Monitor does not incorporate the Operations Monitor reset instruction. ITC will only execute the reset if time-sharing is in progress.

**Error Alert Control (EAC) Dump**

The label DUMP1 gives the user the option of including the dump routine (dump core to disk) for subsequent user error analysis. The functions of EAC are explained in another section of this manual (see Functions of Executive Programs: The System Director).

**Allocation of Internal and External Interrupt Levels**

Interrupts can be generated by events which originate in the plant or the environment that is being controlled, or by conditions internal to the computer hardware itself. These may be classified as external (or process) interrupts and internal interrupts.

Internal interrupts may be caused by an error condition being detected, an input/output operation being completed, an interval timer interrupt, a computer operator setting a switch, etc.

External or process interrupts may be caused by the closing of an electrical contact, a rise in temperature above a set limit, etc.

Since the number of internal and external interrupts required by a particular system is decided by the user, the System Director must be provided with a labelled assignment of each interrupt used.
**Interrupt Levels**

A level of interrupt represents a degree of removal from the normal computer mode. The multi-interrupt feature of the IBM 1800 Data Acquisition and Control System is composed of a maximum of 24 levels, each level containing 16 request positions, thus making available 384 interrupt lines to signal the computer to halt the program being executed and branch to unique hardware memory locations.

The number of interrupt levels (NULEV) planned by a user is assigned contiguously to the 24 available levels, starting from zero to 23. If, for example, 16 interrupt levels are elected by the user, levels 0–15 are used. The numerical value to which the label NULEV is equated is always 1 plus the highest numbered interrupt level used.

Priority assignments are necessary in order that an order of precedence (that is, a level) can be established among the several interrupt conditions. In configuring a multi-interrupt system, the user should remember that certain I/O devices such as the disk, magnetic tape, and timers require high response capabilities. Other I/O devices such as the list printer, typewriter, and card-reader do not demand such a critical response.

In general, process interrupts (PISW's) are assigned lower priority levels than data processing and process I/O devices, except for process interrupts that do not require I/O and demand immediate response or initiate extended operations at lower levels through the programmed interrupt feature. The reason process interrupts are assigned lower priorities than I/O devices is that user-written subroutines for the servicing of these process interrupts can then utilize all I/O devices. I/O devices must receive an operations complete interrupt, which cannot occur if it is located on a lower priority level than the level from which the I/O device is called. Exceptions to this rule are the disk and the 1053 Printer where the I/O routine is so written that it will remain within itself until the operation is complete. These exceptions were allowed due to EAC requirements, but should not, in general, be considered as acceptable practice.

The amount of computer time required to service a particular interrupt can influence its priority assignment. If, for example, its servicing is relatively short, an interrupt may be accorded higher priority than one which entails more elaborate servicing procedures.

Those basic I/O devices that demand fast response include the disk, magnetic tape, and timers. Because the 1053 Printer uses the disk when it buffers messages, the analog interrupts should be at a higher level than the assignment of the 1053 Printers due to a possible loss of comparator interrupts. It should be pointed out that although fast response is not normally required by the 1053 Printer, this device should be assigned to a high enough interrupt level to allow it to run continuously at a maximum rate. Thus, typewriter messages will be serviced without overloading the message buffer.

It is recommended that the Analog Input Comparator feature be assigned to a higher priority level than the Analog Input. The remaining I/O devices do not possess any special characteristics for assignment at a high level, except that they must be at a level higher than the highest level from which they are called, and at a higher level than any assigned interrupt core load (see equate card ICLLL, Figure 66).

Figure 68 (in conjunction with Figure 66) illustrates how a multi-interrupt system configuration might look in the IBM 1800 Data Acquisition and Control System for a typical process control application. The example serves to convey some of the principles noted above: it should not be taken as a model.

The machine configuration chosen for this example includes:

1. IBM 1802 Processor - Controller
2. IBM 2310 Disk Storage Unit with three disk drives
3. IBM 2401 Magnetic Tape Unit
4. IBM 1053 Printer Units
5. IBM 1443 Printer Unit
6. IBM 1442 Card Read Punch Unit
7. IBM 1627 Plotter Unit
8. Analog Input Basic with Comparator
9. Analog Input Extended with Comparator
10. Digital Input
11. Digital and Analog Output
12. Interrupt levels

Other considerations are:

- 57 Process Interrupts (spread over 12 levels)
- 24 Count Servicing Subroutines
- 12 Programmed Interrupts
- 3 Timers
- Queue Table size = 50

A group of process interrupts is assigned to each of 12 levels, 0–11. Note that process interrupts are normally factory wired to terminals in
### Figure 68. Example of Interrupt Level Status Word Assignment

<table>
<thead>
<tr>
<th>INTERRUPT LEVEL</th>
<th>INTERRUPT LEVEL STATUS WORD</th>
</tr>
</thead>
</table>
| 0                | PISW 1  
|                  | TIMERS A,B,C |
| 1                | PISW 2  
|                  | 2310/1  
|                  | 2310/2  
|                  | 2310/3  |
| 2                | PISW 3  
|                  | 2401    |
| 3                | PISW 4  
|                  | AIBC    |
| 4                | PISW 5  
|                  | AIB     |
|                  | AIE     |
|                  | DI      |
|                  | DAO     |
| 5                | PISW 6  
|                  | 1053/1  |
|                  | 1053/2  |
|                  | 1053/3  |
|                  | 1053/4  |
| 6                | PISW 7  
|                  | 1443    |
| 7                | PISW 8  
|                  | 1442    |
| 8                | PISW 9  
|                  | C.I.    |
|                  | 1627    |
| 9                | PISW 10 |
| 10               | PISW 11 |
| 11               | PISW 12 |

**NOTE**

1. Interval Timers must be on a higher interrupt level than the 2310, 1816/1053, 1442 and 1443 devices.
2. The 1816/1053s must be on a lower interrupt level than the 2310s.
groups of 4 to corresponding bit positions of one or more PISW's. In this example, only 1 process interrupt is utilized on each of the levels 0-8, and 16 process interrupts on each of levels 9-11, giving a total of 57 process interrupts.

The three timers A, B, and C are assigned to the highest interrupt level (level 0) in order to give them high response. They are thus placed to interrupt any event or device in progress. With the timers at this level, the timer servicing routines should not be calling any I/O device, but should make use of the programmed interrupt or queueing a mainline technique for servicing requirements.

The reason for this is that it is not possible to call an I/O device from a level of higher priority than the I/O device being called (as already explained). In addition, it is not desirable to remain on the timer level for a period of time long enough to cause the system to miss a timed-out interval of higher priority than the one being processed.

Disk drives are assigned the next highest level (level 1) in order that the sector-gap can be made; that is, the disks can then operate at their maximum speed without incurring the penalty of a full revolution of disk time (40 ms.). The magnetic tape unit is placed at the next level (level 2) also for the same ability to service it at full capacity.

The Analog Input Basic with Comparator (AIBC) and Analog Input Extended with Comparator (AIEC) are recommended to be assigned a higher level (level 3) than their corresponding Analog Input Basic (AIB) and Analog Input Extended (AIE) devices (level 4). They must always be assigned to a higher level than the 1053 Printers.

The 1053 Printers are placed on level 5 because they are continually active, but do not require much execution time. Thus, assigning them above the process interrupts give these devices the ability to print while user's core loads are in execution.

The remaining devices present no real demand problems, and are assigned to lower levels as shown in Figure 68.

Summary of Interrupt Assignment Restrictions

For proper operation of the TSX system, the following interrupt assignment restrictions must be observed:

1. All I/O device interrupts must be assigned to a higher priority interrupt level than external interrupts, unless the external interrupt is serviced by a skeleton interrupt routine.

2. If external interrupts and I/O devices are both assigned to the same level, the external interrupts must be serviced by skeleton interrupt routines.

3. A skeleton interrupt routine cannot use an I/O device whose interrupt is assigned to the same or a lower priority level, except for the disk, 1053 Printer, and 1443 Printer; however, the 1053 test function cannot be used.

4. ILSW bits must be assigned contiguously, beginning with position 0.

Interrupts Per Level

It has been noted that a level of interrupt represents an order of precedence or priority, and that each level contains a total of 16 request positions.

When one or more lines are connected to any one priority level, it is necessary by programming means to identify the specific condition which caused that interrupt level to request service. To do this, a 16-bit word called the Interrupt Level Status Word (ILSW) is used. The programmer does not specify the ILSW in his instructions; this specification is fixed. That is, one ILSW is hardware assigned to each of 24 interrupt levels. Through the ILSW, the operational status or condition of an I/O device or process is revealed to the executive system.

The choice of interrupting I/O devices and/or process conditions on a specific interrupt level is specified by the user on the NB00-23 equate cards. If, for example, one process interrupt and four I/O devices are assigned contiguously (starting from bit zero) to level 10, the user equates NB10 to 5. The NB label is always equated to a numerical value equal to the rightmost bit (on the ILSW) plus 1 for a level. If no devices or process interrupts are assigned to a level, the label is equated to zero. Note that the NB00-23 equate cards must reflect exactly the number of bits on the System Loader assignment cards. Also, those levels that use programmed interrupts only do not contain ILSW bits; the NB cards for these levels are, therefore, equated to zero.

Level Work Areas

Whenever an interrupt caused by an I/O, a process interrupt, or a programmed interrupt occurs, an indirect branch takes place to a fixed word in core. This word contains the address of a communications area known as a Level Work Area. There is one
work area per level of interrupt specified by the user, and only those levels configurated by the user will be assembled and will be available when the System Skeleton is built. In addition, three additional work areas are always assigned to the system: one each for Nonprocess Core Loads (if time-sharing is used), Mainline (that is, Process) Core Loads, and Errors (Trace and C.I. Interrupt). If time-sharing is not used, the Nonprocess work area is originated out at System Director assembly time.

Briefly, the level work area serves as a means of communications whenever the computer transfers control from one level of interrupt to another. The address of the interrupted level is then saved and the address of the work area for the current level set up. When the level in progress has completed its processing, the address of the interrupted level is restored. This method of coding automatically saves all reentrant coded subroutine work areas.

Figure 69 illustrates the layout of a level work area which is 104 words in length, but this may be increased or decreased by the user (at System Director assembly time), depending on the functions related to each level of interrupt. Note that for proper testing of errors and time-sharing, the MIC work area portion should not be deleted.

Work levels are divided into two major sections: a fixed section and an overlay section. Each word in the fixed section is assigned to one specific program and may be core protected. The manner in which it is assigned is shown in Table 11.

Words in the overlay section may be assigned to several different programs providing these programs do not call one another. This section cannot be core protected.

The overlay section begins with the 58th word of a level work area (see Table 12). If this section is used, the user is advised to reference this "start" position with a label, thus eliminating extensive program modification in the event the fixed section has increased or decreased. If a modification is required, the first word of the overlay section (the 58th word) will always be an even core location to facilitate double load and store instructions, etc. Also, programs using the overlay section should always commence at the beginning, except those programs that are called by a program already using this area. For example, DP I/O programs use the first 25 words of the overlay section; therefore any program that requires storage locations in this section and also calls a DP I/O routine cannot use the first 25 words. The later program will thus start at the 26th or 27th word of the section.

---

Figure 69. Layout of a Level Work Area
Table 11. Fixed Section of a Level Work Area

<table>
<thead>
<tr>
<th>WORD POSITION</th>
<th>PROTECTION STATUS</th>
<th>CONTENTS</th>
<th>WORD POSITION</th>
<th>PROTECTION STATUS</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>N</td>
<td>Save interrupt exit</td>
<td>27</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>N</td>
<td>Not used</td>
<td>28</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>N</td>
<td>Busy indicator address for UFIO; indicates that unformatted I/O buffer has been saved to disk and must be restored by MIC.</td>
<td>29</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>N</td>
<td>UFIO restore indicator; non-zero if a restore is needed.</td>
<td>30</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Y</td>
<td>Address of word 5 of the ICIT entry for this level.</td>
<td>31</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Y</td>
<td>Address of word 1 (in-core-with-skeleton indicator word) of the ICIT entry for this level.</td>
<td>32</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Y</td>
<td>Address of word 2 (in-core-with-mainline indicator word) of the ICIT entry for this level.</td>
<td>33</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Y</td>
<td>Address of word 3 (record indicator word) of the ICIT entry for this level.</td>
<td>34</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Y</td>
<td>Address of word 4 (recorded indicator word) of the ICIT entry for this level.</td>
<td>35</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Y</td>
<td>Address of word 0 of this level work area</td>
<td>36</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>N</td>
<td>Status save location</td>
<td>37</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Y</td>
<td>Level number</td>
<td>38</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>N</td>
<td>Entry point to interrupt level coding</td>
<td>39</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Y</td>
<td>STX sets this level busy</td>
<td>40</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Y</td>
<td>STX saves XR3</td>
<td>41-43</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Y</td>
<td>Sets XR3 to work level</td>
<td>44-49</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Y</td>
<td>Saves XR2</td>
<td>N</td>
<td>Locations used by QZSAV/QZEXT to save and restore:</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Y</td>
<td>Saves XR1</td>
<td>A-register Q-register X1 X2 Carry and Overflow indicators X3</td>
<td>50-54</td>
<td>N</td>
</tr>
<tr>
<td>14</td>
<td>Y</td>
<td>BSC long to MIC</td>
<td>N</td>
<td>Locations used by TVSAV/TVEXT to save and restore:</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Y</td>
<td>BSC long indirect to interrupt via the Master Branch Table.</td>
<td>55</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Y</td>
<td></td>
<td>56</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Y</td>
<td></td>
<td>57</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>N</td>
<td>MFIO restore indicator; non-zero if a restore is needed</td>
<td>20</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>N</td>
<td>Busy indicator address for MFIO; also first word of PISW IOCC.</td>
<td>21</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>N</td>
<td>Second word of PISW IOCC. This is a standard PISW set-up by TSX for this level. If the user wishes to sense a PISW other than the standard for this level, Word 21 would have to be modified by him. See SYSTEM DESIGN CONSIDERATIONS, SYSTEM DIRECTOR; PISW Assignment Restrictions.</td>
<td>22</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>N</td>
<td>Save location for XR1</td>
<td>23</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>N</td>
<td>Save location for XR2</td>
<td>24</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>N</td>
<td>Save location for XR3</td>
<td>25</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>N</td>
<td>Level busy indicator. Positive, if the level is busy; zero if not.</td>
<td>26</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>N</td>
<td>Save location for PISW sense</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Save location for the address of the work level in use at the time the interrupt occurred; i.e., word 68:16 of the Fixed Area in core.

Save location for the A-register.

Save location for the Q-register.

Save location for WKS, word 37:16 of the Fixed Area in core.

CARDN indicator. If zero, detection of a // card causes an error.

Save location for MDFIO sector address on a save operation.

Busy indicator address for MDFIO

File protect indicator; this must be set non-zero prior to every write to a file-protected area.

First word of ICIT (in-core skeleton address)

Second word of ICIT (in-core mainline address)

Third word of ICIT (RECORD address)

Fourth word of ICIT (RECORDED address)

Fifth word of ICIT (first entry), Words 36-40 constitute the work area used by MIC. This area is loaded with addresses from the ICIT entry for this level to inform the ALLGO routine how to handle the interrupt and/or where to find the servicing routine or core load.

Save locations for FORTRAN FAC (floating accumulator)

Locations used by QZSAV/QZEXT to save and restore:

A-register Q-register X1 X2 Carry and Overflow indicators X3

Locations used by TVSAV/TVEXT to save and restore:

A-register Q-register X1 X2 Carry and Overflow indicators X3

FORTRAN functional error indicator

FORTRAN divide check indicator

FORTRAN overflow indicator
Table 12. TSX Reentrant Subroutine Work Level Requirements

<table>
<thead>
<tr>
<th>SUBROUTINES</th>
<th>FIXED</th>
<th>OVERLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FADD, FADDX, FSBR, FSBRX, FSUB, FSUBX</td>
<td>X</td>
<td>41-43, 52</td>
</tr>
<tr>
<td>FALOG, FLN</td>
<td>X</td>
<td>41-43, 55</td>
</tr>
<tr>
<td>FATAN, FATN</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>FAVL, FABS</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>FAXB, FAXBX</td>
<td>X</td>
<td>41-43, 46, 48, 49, 55</td>
</tr>
<tr>
<td>FAXI, FAXIX</td>
<td>X</td>
<td>42, 52, 55</td>
</tr>
<tr>
<td>FDIV, FDIVX, FDVR, FDVRX</td>
<td>X</td>
<td>41-43, 52, 56</td>
</tr>
<tr>
<td>FEXP, FXPN</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>FLD, FLDX, FSTO, FSTOX</td>
<td>X</td>
<td>41-43, 52</td>
</tr>
<tr>
<td>FMFY, FMPYX</td>
<td>X</td>
<td>41-44, 52</td>
</tr>
<tr>
<td>FSIN, FSINE, FCOS, FCOSN</td>
<td>X</td>
<td>41-43, 55</td>
</tr>
<tr>
<td>FSQRT, FSQR</td>
<td>X</td>
<td>41-43, 57</td>
</tr>
<tr>
<td>FTANH, FTNH</td>
<td>X</td>
<td>41-43, 46, 48, 49</td>
</tr>
<tr>
<td>FTRTN, FTNTR</td>
<td>X</td>
<td>41-43, 52, 56, 73-79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUBROUTINES</th>
<th>FIXED</th>
<th>OVERLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>EADD, EADDX, ESBR, ESBRX, ESUB, ESUBX</td>
<td>X</td>
<td>41-43, 52</td>
</tr>
<tr>
<td>EALOG, ELN</td>
<td>X</td>
<td>42, 52, 55</td>
</tr>
<tr>
<td>EATAN, EATN</td>
<td>X</td>
<td>41-43, 52, 56</td>
</tr>
<tr>
<td>EAVL, EABS</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>EAXB, EAXBX</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>EAXI, EAXIX</td>
<td>X</td>
<td>41-43, 52</td>
</tr>
<tr>
<td>EDIV, EDIVX, EDVR, EDVRX</td>
<td>X</td>
<td>41-43, 52, 56</td>
</tr>
<tr>
<td>EEKP, EXPN</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>ELD, ELDX, ESTO, ESTOX</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>EMFY, EMFYX</td>
<td>X</td>
<td>41-43, 52</td>
</tr>
<tr>
<td>ESIN, ESINE, ECOS, ECOSN</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>ESQRT, ESQR</td>
<td>X</td>
<td>41-43, 57</td>
</tr>
<tr>
<td>ETANH, FTNH</td>
<td>X</td>
<td>41-43, 46, 48, 49</td>
</tr>
<tr>
<td>ETRTN, ETNTR</td>
<td>X</td>
<td>52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUBROUTINES</th>
<th>FIXED</th>
<th>OVERLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADRCK</td>
<td>X</td>
<td>7</td>
</tr>
<tr>
<td>COMGGR, COMGI</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>DATSW</td>
<td>X</td>
<td>41-43, 52, 56</td>
</tr>
<tr>
<td>DVCHK</td>
<td>X</td>
<td>41-43, 70-75, 70-75</td>
</tr>
<tr>
<td>ESIGN (EXTENDED PRECISION)</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>FSIGN (STANDARD PRECISION)</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>FCSTST</td>
<td>X</td>
<td>55</td>
</tr>
<tr>
<td>IOU</td>
<td>X</td>
<td>SIXGN</td>
</tr>
<tr>
<td>ISTOX</td>
<td>X</td>
<td>50, 52</td>
</tr>
<tr>
<td>LDFA, STFA, SBFAC, DVFA, MDF</td>
<td>X</td>
<td>42</td>
</tr>
<tr>
<td>MDFBA, MDAF, MDIA, MDMA, MDFX, MDIX, MDIX, MRED, MDWRT</td>
<td>X</td>
<td>41-43</td>
</tr>
<tr>
<td>MDFND</td>
<td>X</td>
<td>41-43, 70-92</td>
</tr>
<tr>
<td>MFIO, MRED, MWR, MCOM, MIOA, MIOX, MIOA, MIOA, MIOA, MIOF</td>
<td>X</td>
<td>19-20, 41-43, 55</td>
</tr>
<tr>
<td>MGOTO, MIF, MIF, MIF</td>
<td>X</td>
<td>70-72, 74-78</td>
</tr>
<tr>
<td>MIAR, MIARX, MFAR, MFARX, MEAR, MEARX</td>
<td>X</td>
<td>70, 71, 74-85, 89</td>
</tr>
<tr>
<td>OVERF</td>
<td>X</td>
<td>57</td>
</tr>
<tr>
<td>PAUSE</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>SAVE, IOFIX</td>
<td>X</td>
<td>7, 11, 19, 33, 34</td>
</tr>
<tr>
<td>SLITE, STIIT</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>SSWTC</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>STOP</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>SUBROUTINES</td>
<td>SUBIN</td>
<td>SUBSC</td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVERLAY</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 12. TSX Reentrant Subroutine Work Level Requirements

<table>
<thead>
<tr>
<th>SUBROUTINES</th>
<th>SUBIN</th>
<th>SUBSC</th>
<th>TSTOP</th>
<th>TSTRT</th>
<th>TEST, TSET</th>
<th>UFIO, UFIOX, UIOX, UCOMP, UIOI, UIOF, UIOAI, UIOAF</th>
<th>TRAC (TRPNT)</th>
<th>TRACE (TRPNT)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>58-83</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OVERLAY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Continued)
Table 12. TSX Reentrant Subroutine Work Level Requirements

<table>
<thead>
<tr>
<th>SUBROUTINES</th>
<th>FIXED</th>
<th>OVERLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARDN</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PAPTN</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MAGT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PLOTX</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>REWIND/BCKSP/EOF</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DAOP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AIPTN, AIPN</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AISQN, AISN</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>AIRN</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DIEXP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DICMP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DINP</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ANINT, COMP1, AINT1, COMP2, AINT2</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>AIP</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>AIS</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>AIR</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CO, DO, PO, DAC</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CS, VX, PI, DI</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CSC, VSC, PIC, DIC</td>
<td>X</td>
<td>IS REENTRANT</td>
</tr>
<tr>
<td>CSX, VSX, PIX, DIX</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>IOPE, OUSLY, ETS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>XSAVE, XEXIT, XLOAD</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>GAGED, UNGAG</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>QZERQ</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>QZ010</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>BT1BT</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>BT2BT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>BINDC</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DCBIN</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>BINHX</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HXBIN</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HOLEB</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>HOLPR</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EBPRT</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PAPERB</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PAPHL</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PAPPB</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EBPA</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PRT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FCHAR</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SCALF</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FGRID</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FPLT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ECHAR</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SCALE</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EGRID</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EPLT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>POINT</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FCHR, FCHR1, WCHR1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FRUL, FMOVE, FINC</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ECHR, ECHR1, VCHR</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Table 12. TSX Reentrant Subroutine Work Level Requirements

<table>
<thead>
<tr>
<th>SUBROUTINES</th>
<th>FIXED</th>
<th>OVERLAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERULE, EMOVE, EINC</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>XYPLT</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>PLOTI, PLOTS</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>SKELETON I/O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISKN</td>
<td>X</td>
<td>35</td>
</tr>
<tr>
<td>PRNTN</td>
<td>X</td>
<td>58-69</td>
</tr>
<tr>
<td>TYPEN/WRTYN</td>
<td></td>
<td>70-80</td>
</tr>
</tbody>
</table>

(Cocontinued)

Table 12 illustrates the work level requirements of TSX reentrant subroutines. These may depend on various modification levels of the TSX system. If absolute information is required, the current listings should be referred to.

The five status columns (NOT REENTRANT, TVSAV, QZSAV, A-REG, and MASKED) indicate whether each subroutine is reentrant, and if it is, what modes of reentry are used. For example, FADD (Floating-point ADD) is reentrant since the first column is blank; it uses TVSAV, but note that it also masks all levels at one or more points within the subroutine.

Some subroutines are reentrant, but do not use any words in the level work area. ENDTS is such an example. VIAQ is not reentrant; it does, however, mask all levels to prevent the Queue Table from being modified by QUEUE, QIFON, and UNQ during several instructions.

Level work areas are defined by the USE labels USE00-23), the number of work areas being determined by NULEV. If a USE label is equated to 1, a work level is included on that level; if zero, no work level is included. For example, if NULEV = 7, USE00-06 are all equated to 1; the remaining USE cards being equated to zero.

See also: Programming Subroutines Using Reentrant Coding.

Process Interrupts Per Level

Like the ILSW, the Process Interrupt Status Word (PISW) is a 16-bit word associated with the use of process interrupts. Process interrupts are physically terminated on 16-position terminal blocks within the 1800 system. The PISW indicators are turned on or off by contact closures or voltage shifts in the process. A total of 24 PISWs are allowed in the system for normal usage. To provide the maximum number of interrupt levels for process interrupts, one PISW could be assigned to each ILSW. For multiple groups per level, see PISW Assignment Restrictions.

The System Director must also be aware of the number of process interrupt bits on each of the 24 hardware levels. This information is provided by the user on the NIL00-23 equate cards. If, for example, one process interrupt is assigned to interrupt level 10, the bit configuration for the PISW for that level could be bits 0 to 15. For example, if bits 0 to 7 are assigned, the NIL10 label will be equated to 8. The NIL label will always be equated to a numerical value equal to the rightmost PISW process bit position used plus one. If no process interrupts are assigned to a level, the label will be equated to zero.

With this information, an interrupt core load table (ICLT) is built which contains an entry for each interrupt level assigned by the user plus two entries for programmed interrupts and two entries for count routines. The user specifies how many process interrupts he has on a particular level and only those words that are necessary to contain his configuration are entered in the table.

Figure 70 illustrates a partial ICL table, for one level, say level seven.

IN SKELETON represents the PISW bits of the PISW associated with the level this entry serves. If a subroutine is loaded as part of the skeleton to serve a process interrupt, a bit is set up in this word which corresponds to the bit on the PISW. For example, if the user has a process interrupt on level 0, and this PISW is wired such that when the interrupt occurs the zero bit comes on, a bit is put into the corresponding zero bit of the IN SKELETON word of the level zero ICLT entry.

The start address of the servicing routine is then loaded into START ADDRESS.

IN MAINLINE CORELOAD and RECORD are set up each time a mainline core load is read into core. The former word specifies that the interrupt servicing
PISW Assignment Restrictions

PISW (Process Interrupt Status Word) groups can be assigned to interrupt levels either as a single group per level or in multiple groups per level. For proper operation of the TSX system, the following rules and restrictions must be observed.

One Group Per Level

Normal usage of process interrupts requires that only one group of process interrupts be assigned to each interrupt level. Process interrupts assigned in this way can each be serviced with separate interrupt routines. The servicing routines must reside in the skeleton area only if their associated interrupt level is equal to or higher than any I/O device interrupt level.

When only one PISW is connected to a level, the correlation of the interrupt level number to the PISW group number is as follows:

<table>
<thead>
<tr>
<th>Interrupt Level</th>
<th>PISW Group</th>
<th>Second Word of I/OCC for PISW Sensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>/5F02</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>/5F03</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>/5F04</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>/5F05</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>/5F18</td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>/5F19</td>
</tr>
</tbody>
</table>

Note that MIC performs the ILSW and PISW sensing, and transfers control to the proper interrupt servicing routine.

Multiple Groups Per Level

In special cases, such as (1) when fast response is desired, and (2) when each bit does not require a unique program to service it (such as when all the on-bits in a group might represent a particular code), it is desirable to have more than one PISW group assigned to an interrupt level. The PISW that is to be sensed by the user must be assigned (on the *ASSIGNMENT card at system generation time) as an RPQ device, and the following restrictions must be observed by the user's subroutine. It must:

1. Reside within the skeleton area
2. Service the interrupt as though it were an interrupt from an RPQ device. Thus the subroutine must be coded with an ISS (interrupt service subroutine) statement with the IAC (interrupt assignment code) number following the ISS statement matching the RPQ number given to the PISW at system generation time.

3. Sense all PISWs assigned to the level.

4. Upon completion, exit to MIC via the I/O exit (that is, BSC I 90).

When assigned in this way, there is no correlation restriction between the interrupt level number and the PISW group number.

Combination PISW Assignments

It is also possible to combine the two assignment methods and have some interrupt levels with only one PISW each, and other levels with more than one PISW. The same rules and restrictions for each type outlined above still apply. For example, to have two groups of four PISWs each assigned to interrupt levels 4 and 5, one valid combination is:

<table>
<thead>
<tr>
<th>Interrupt Level</th>
<th>PISW Group</th>
<th>Second Word of IOCC for PISW Sensing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>/5F02</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>/5F03</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>/5F04</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>/5F05</td>
</tr>
<tr>
<td>4</td>
<td>One or more groups of PISWs</td>
<td>User-Sensed</td>
</tr>
<tr>
<td>5</td>
<td>One or more groups of PISWs</td>
<td>User-Sensed</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>/5F08</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>/5F09</td>
</tr>
<tr>
<td>8</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>9</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>10</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>11</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>12</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>13</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>14</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>15</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>16</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>/5F13</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Not assignable;</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>21</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>22</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any combination can be used for the PISW assignments on levels 4 and 5.

Note that the user is not restricted in assigning multiple PISWs only to those levels which are not sensed by MIC: they can be assigned to any level. For example, if level 7 has the standard sense for group of assigned PISWs, the user could include on that level another group of PISWs which he desires to sense himself. User-written interrupt servicing routines must be coded as an I/O ISS routine.

For further information on the assignment of process interrupts whose IOCCs are to be sensed by user-written subroutines, refer to the following: System Design Considerations: The IBM Nonprocess System; Programming Techniques — Writing Assembler Language Subroutines; IBM 1800 Assembler Language, (Form C26-5882).

Programmed Interrupts

CALL LEVEL -- Programmed Interrupt

External interrupt levels can also be programmed. Programmed interrupts are initiated by the CALL LEVEL statement, and follow the same basic rules which pertain to process interrupts. The format of this statement is as follows:

CALL LEVEL (L)

where

L is an integer constant 0-23 that specifies the interrupt level desired.

There can only be one programmed interrupt routine per assignable interrupt level. The use of the servicing routine may, however, be expanded by setting a numeric value (integer) in Inskel COMMON prior to a CALL LEVEL (L). The LEVEL subroutine can then interrogate this value introduced by the calling program. In this way, as many subroutines per level as are desired can be provided.

CALL LEVEL, which can be used only in process mainlines or interrupt programs, causes a pseudo ILSW bit to be set in the programmed interrupt IOCC (locations 16010 and 16210) in the Fixed Area. The IOCC bits are interrogated by MIC when an interrupt occurs on a specific level. MIC always reactivates those levels that have been specified within the CALL statement, and which have not been serviced.

The programmed interrupt is recognized immediately when called from a lower level. When the servicing routine finally exits to MIC, program operation at the calling level is resumed with the statement following the CALL LEVEL statement.

A programmed interrupt called from a higher level is recognized after the calling program is completed, and after any intervening interrupts are serviced. If a level is called and any ILSW bit is on when this level is recognized, the programmed interrupt is recognized after the first ILSW "on" bit is serviced.
The equate card NLSW1 is used to specify the lowest priority level number (within the group 0-13) plus 1 assigned to a programmed interrupt. Similarly, NLSW2 applies to the group 14-23. This determines the number of programmed interrupts available to the user.

In the example given in Figure 66, the number 12 punched in columns 35-36 means that programmed interrupt space in the ICL table has been allocated for levels 0-11 (12 levels). NLSW2 is equated to 0.

Consider a further example. Ten (10) programmed interrupts are required on levels 14-23; none on levels 0-13. NLSW1 is equated to 0 and NLSW2 is equated to 10.

Note that there can only be one programmed interrupt routine per interrupt level. The user need not initially have programs on disk corresponding to all levels of programmed interrupt stated. These may be loaded later. Also note that all called levels must be specified as used on corresponding USE EQU cards.

Out-of-Core Interrupt Levels

The user has also to determine those levels on which interrupts will be serviced by out-of-core interrupt core loads. Other interrupts on levels that are serviced by out-of-core interrupt core loads are masked so that they cannot interrupt another interrupt being serviced by a corresponding out-of-core routine. Interrupts serviced by in-core routines are not masked. Only one level of exchange is maintained.

Interrupts that are in the skeleton should not be on the same or lower level as out-of-core interrupts, unless those interrupts can be masked for the period of time required to service the out-of-core interrupts.

The best practice for servicing interrupts is to group those interrupts that are serviced by in-skeleton routines on higher priority levels than interrupts which are serviced by interrupt core loads. This is not, however, a restriction on the system: in-skeleton routines may be intermixed with interrupt core loads as long as the skeleton routines can be masked for the period of time required to service the interrupt core loads.

The user specifies those interrupt levels he has elected to be masked for the servicing of interrupt core loads on two equate cards: ICLL1, ICLL2. He may elect a single interrupt level or consecutively numbered interrupt levels within the two groups of interrupt levels, 0-13 and 14-23. Even though the highest priority out-of-core interrupt level cannot be interrupted from another out-of-core interrupt level, it must nevertheless be specified in the ICLL1-2 equate cards. This is because whenever an exchange of variable core is to be carried out, the ICLL1-2 values are used, together with the user’s current mask pattern, to mask the core-exchange routine. This prevents the core-exchange routine from being interrupted by an out-of-core interrupt when it is already busy on the mainline or non-process level. Two examples illustrate how the ICLL1-2 cards are used.

In the first example, assume

\[
\text{NULEV} = 12
\]

Three interrupt core load interrupt levels are required (9, 10, and 11).

The equated representation is shown below:

<table>
<thead>
<tr>
<th>HEX EQUIV</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT LEVEL</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>ICLL1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that this corresponds to the machine configuration given in Figure 66.

In the second example, assume

\[
\text{NULEV} = 24
\]

Interrupt core load interrupt levels 7, 8, 9 and 19, 20 are required.

The example assumes that levels 10-18, 21-23 contain in-skeleton routines whose servicing can be delayed until the interrupt core loads have been serviced. Furthermore, if interrupt core loads on levels 7-9 are infrequently used, the normal servicing of in-skeleton interrupts (levels 10-18) would not be inhibited during the servicing of interrupt core loads on levels 19, 20.

As shown below, to obtain this, ICLL1 is equated to /01C3; ICLL2 to /063F.

<table>
<thead>
<tr>
<th>HEX EQUIV</th>
<th>C</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT LEVEL</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ICLL1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The equated representation is shown below:

<table>
<thead>
<tr>
<th>HEX EQUIV</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT LEVEL</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>ICLL2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that this corresponds to the machine configuration given in Figure 66.
NUMBER OF CALL COUNT SUBROUTINES REQUIRED BY USER

Call Count subroutines are user-written servicing routines which are assigned by *INCLD control cards to be included in the skeleton area when the skeleton is initially built. Their function is to service TSX CALL COUNT statements.

Examples of the use of count routines are:

1. Scanning variables at periodic intervals
2. Initiating control adjustments at appropriate times
3. Constructing a log

Count servicing may also be achieved by not including a subroutine. In such cases, the event is only recorded during the normal processing of core loads, and later serviced by a CALL QIFON or CALL CLEAR statement in the mainline program.

In practice, count subroutines should have short execution times and are normally only used to set an indicator, such as the setting of a programmed interrupt to some lower priority level, or queuing a mainline core load. For example, if a count routine is used to run a scan, the scan should be run from a lower priority level. This means that the count subroutine then sets a programmed interrupt: the scan routine itself could be in the skeleton, in the mainline or on the disk since programmed interrupts have the same capabilities as process interrupts.

If a log is required, the log core load should be queued from the count subroutine. This means that the count subroutine calls QUEUE specifying a mainline core load. The log core load would then be executed in sequence according to the priority assigned to it. If time-sharing is used, a CALL ENDS statement should be included within the count routine immediately following the CALL QUEUE.

The number of count routines planned by the user is punched in the NITP1 and NITP2 System Director equate cards. For example, if the user elects to use a total of 12 (Skeleton-included and recorded) count routines in his system, NITP1 and NITP2 are equated to 12 and 00, respectively. See also example in Figure 66.

During System Director assembly time, an allocation of two words per entry is set up in the Interrupt Core Load Table for the start address of each count routine, as illustrated in Figure 70.

If a count routine is included in the skeleton area, the Skeleton Builder, at skeleton build time, places its start address in the ICL table. On the other hand, if a routine is not included in the skeleton area, the numbered count only will be recorded during the normal processing of core loads. That is, a corresponding bit is "set on" by ITC in the ICL table and the servicing timer for that event is turned off. This means that the user may want to "wait" on one of the count indicators to "come on" (recorded) prior to servicing an event. This indicator is later reset by the CALL QIFON or CALL CLEAR statement when requested by the user.

The size of the ICL table is determined at assembly time from data given in the equate cards. Its entries are inserted by the Skeleton Builder and DUP. A maximum of 32 count routines is allowed.

DISK SYSTEM CONFIGURATION

Magnetic tape oriented systems are of value when applied to pure data acquisition tasks where very large storage requirements exist. It has been found, however, where control is involved, that core storage must be supplemented by a rotating memory such as a disk file to operate an efficient monitor program. The data transfer rate of 32K words to and from each disk, the shorter seek time and higher reliability of a disk file make it ideally suited for process control and data acquisition purposes.

In the IBM 1800 Time-Sharing Executive System, the software design concept is based on the use of a key unit -- a single, high-capacity disk storage unit (the IBM 2310 Disk Storage Unit) as a programming systems residence device.

Conceptually, the disk is treated in much the fashion as a reel of magnetic tape, and organized in a sequential manner. Any one segment of the disk is quickly available by moving the access mechanism directly to the starting point of a block of information, and serially transferring data. This approach also permits the user to easily store and/or retrieve blocks of data such as core loads, programs, matrices, and tables. The ability to read and write data on the same device allows the user to modify resident programs without the necessity of a second storage unit, such as in magnetic tape systems.

The disk cartridge is designed to permit rapid interchange of disks under conditions which afford adequate protection of recording surfaces. The cartridge design also ensures precision control of disk rotation at high speeds past the recording heads.
Up to three disk drives are available with the 1800 system. Typical disk uses include the following:

1. The disk file is well suited for the storage of executive programs, subroutine libraries, diagnostic routines, etc., as well as the implementation of the 1800 FORTRAN language.
2. As a storage device for recording raw data, the disk file is extremely inexpensive, and obviates the need for the more costly magnetic tape drives.
3. As a device for storing edited or corrected data, the removable disks can be transferred for permanent storage or for use on another 1800 machine for off-line computations.
4. It can be included to allow additional working storage for customer programs, or it can be used as a "spare file" for greater reliability and maintenance flexibility.

Furthermore, the disk cartridge allows each user at a system installation to "customize" standard programming packages, and to include his own subroutines and programs according to his particular needs. Tailoring resident-device characteristics to match the performance of the processor-controller makes it possible to balance the system for optimum performance.

Changing Disk Cartridges

Under certain conditions it is necessary to change disk cartridges containing data files.

The following procedure may be used to change disk packs:

1. Print a message to inform the operator to change the disk packs.
2. A user-written call to the PAUSE subroutine can be used to put the system in a wait condition. However, interrupts may still be serviced. Therefore, programs or files on the disk to be changed should not be called.
3. Change disk cartridges. The minimum time required is about 2-1/2 minutes.
4. Press START.
5. Initialize the new disk pack by using the SEEK HOME function. For details on the SEEK HOME function see IBM 1800 TSX manual; Subroutine Library; Form C26-3723 Chapter Disk Subroutine.

DISK ORGANIZATION

The 2315 Disk Cartridge is organized into 200 cylinders (plus three spare cylinders) of two tracks each: one track per disk surface. For ease of block handling, each track is further divided into four sectors, each sector having a fixed length of 320 sixteen-bit words. The sector is defined as the basic addressable unit of disk storage for reading and writing.

Information is written on or read from the disk by a pair of magnetic read/write heads, one head for each surface of the disk. The three spare cylinders ensure that the stated capacities are maintained for the life of the cartridge.

Figure 71 illustrates the relationships between bits, data words, disk blocks, sectors, tracks, and cylinders for a disk storage unit.

<table>
<thead>
<tr>
<th>No. Of</th>
<th>Word</th>
<th>Disk Block</th>
<th>Sector</th>
<th>Track</th>
<th>Cylinder</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>16</td>
<td>320</td>
<td>5,112</td>
<td>20,400</td>
<td>40,960</td>
<td>8,192,000</td>
</tr>
<tr>
<td>Data Words</td>
<td>20</td>
<td>320*</td>
<td>1,260</td>
<td>2,560</td>
<td>512,000</td>
<td></td>
</tr>
<tr>
<td>Disk Block</td>
<td></td>
<td>16</td>
<td>64</td>
<td>128</td>
<td>25,600</td>
<td></td>
</tr>
<tr>
<td>Sectors</td>
<td></td>
<td>4</td>
<td>8</td>
<td></td>
<td>1,600</td>
<td></td>
</tr>
<tr>
<td>Tracks</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>Cylinders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

*These follow the first actual word of each sector, which is used for the address.

Figure 71. Disk Storage Unit Conversion Factors

Sector Numbering and File Protection

In the interest of providing disk features permitting versatile and orderly control of disk operations, two important conventions have been adopted which govern sector-numbering and file protection. Successful use of the disk subroutine, DISKN, can be expected only if user programs are built within the framework of these conventions.

The primary concern of the conventions is the safety of data recorded on the disk. To this end, the file protection scheme plays a major role, but only in a manner that is dependent upon the sector-numbering technique. The latter contributes...
to data safety by allowing the disk subroutine to verify the correct positioning of the access arm before it actually performs a write operation. This verification requires that sector identification be pre-recorded on each sector, and that subsequent writing to the disk be accomplished in a manner that preserves the existing identification. DISKN has been written to comply with these requirements.

Sector Numbering

Each disk sector is assigned a logical address from 0, 1, ....... 1599 corresponding to the sector's position in ascending sequence of cylinder and sector numbers from cylinder zero (outermost), sector zero through cylinder 199 (innermost), sector 7. Since the disk cartridge is divided into 1600 sectors, DISKN can now address any one of these sectors.

This sector address is recorded by a standard TASK utility program (TDWA) in the sector's first word, and occupies the rightmost eleven bit positions. Of these eleven positions, the three low-order positions serve to identify the sector (0-7) within each cylinder. Utilization of this first word for identification purposes diminishes the per sector availability of data words to 320; transmission of full sectors of data is therefore performed in units of this amount.

File Protection

File protection is provided to guard against the inadvertent destruction of previously recorded data. This control can be achieved by having the normal writing functions uniformly test the file-protected status of cylinders they are about to write.

File protection is implemented in the TSX system by defining any cylinder as being file protected or not file protected. The DUP *DWRAD function is used to designate file protection. If a cylinder is file-protected, the sector address on that cylinder will contain a one-bit in bit position zero of the sector address word.
The two hundred cylinders of a disk cartridge for a single disk-drive TSX system are divided into two major logical sections for system operation: a nonprocess portion and a process portion as shown in Figure 72. The nonprocess portion, known as the IBM Systems Area, is used to store the integral component parts of the IBM Nonprocess System as discussed under System Design Considerations: The IBM Nonprocess System.

Areas of disk storage contained within the process portion will now be explained in some detail. These areas are, in general, automatically assigned by the system, but they vary in size depending upon the disk system configuration and customer definition. As will be shown later, some of these areas can be modified, removed, or relocated to another storage device.

**User Area.** The User Area is so called because it is used to store user-written as well as IBM-furnished subroutines. It is variable in length and is file-protected. It is divided into two component areas:

1. **Relocatable Subroutine Area** — an area where all relocatable IBM and user-written subroutines are stored (not shown in Figure 72).
2. **Relocatable Program Area** — an area where all user-written nonprocess relocatable programs are stored as a result of a DUP *STORE control record function. This is a user-defined area — that is, its boundary increases or decreases Nonprocess Work Storage as programs are deleted or added respectively. The number of sectors required for a program to be stored is subtracted from the Nonprocess Work Storage if they are both assigned to the same disk cartridge. When a program is deleted (by an
*DELETE function), a separate DUP *DEFINE PAKDK operation must be performed to repack the area. The reason for this is that the relocatable program has not been overlaid and packed -- the area it occupied is thus not available for program storage. The *DEFINE PAKDK operation will cause this area to be incorporated into Nonprocess Work Storage.

The packing operation could be executed each time an *DELETE function specified a nonprocess program, but since this operation need not be performed unless a shortage of nonprocess area develops, and since the packing takes some time, it is only carried out at periodic intervals when the *DEFINE PAKDK control record is processed. The user should, however, assure himself that he has a satisfactory audit trail in the event some information is lost during the running of the disk packing operation.

Note that the area available for user programs can be expanded by using the *DEFINE REMOV function to remove the Simulator, Assembler, or FORTRAN Compiler. As an example, if all three programs are removed, the nonprocess system will be reduced to about 11 cylinders. A Relocatable Program Area can be assigned to each of up to three drives.

Nonprocess Work Storage (NPWS). This area is always adjacent to the Relocatable Program Area, and is used for temporary storage during the execution of nonprocess programs. It is used extensively during the operation of the Nonprocess Monitor. For example, it is used by the Assembler during the assembly of a program and to store the successfully assembled program in relocatable format. In a FORTRAN nonprocess program, the DEFINE FILE statement can refer to this area. The Nonprocess Work Storage is variable in length, and increases or decreases as programs are added to or deleted from the Relocatable Program Area. A NPWS area can be assigned to each of up to three disk drives.

Error Dump Area (EPDM). When a machine error occurs, the TSX Error Alert Control (EAC) program optionally writes all of core to this area on disk. The number of sectors used is directly proportional to object core size: a 32K object core requires 103 sectors, 16K requires 52 sectors, and 8K requires 26 sectors. This is a user option specified at system generation time.

Error Save Area (EPSV). This area is used when EAC is executed to save the portion of variable core used by the error detection program. An Error Save Area of six sectors is always automatically allocated when the E parameter (see The DEFINE CONFG Operation) is specified. Only one such area is required.

Nonprocess Save Area (NPSV). An area on disk used to save a partially completed nonprocess (time-shared) program that must be saved when time-sharing is terminated. A NPSV equal to variable core is always automatically assigned to the disk. No such area is required if time-sharing is inoperative.

Message Buffer (MESS). An area used to buffer 1053 messages when a WRITE to a 1053 occurs while the 1053 is busy. Its length is established at TASK assembly time (see System Design Considerations: TASK).

Process Work Storage (PRWS). Area used for temporary data storage during the execution of process programs. It is user-defined and can be specified on each of up to three disks, but only one such area can be specified in any one *DEFINE CONFG control record.

FORTRAN I/O Save Area (FIOS). Area used to save the FORTRAN I/O buffer area when a higher level interrupt uses FORTRAN I/O. The length of this area is three sectors per interrupt level using FORTRAN I/O, plus three sectors if the system has time sharing.

Interrupt Save Area (INsY). An area of disk used to save a user-specified variable area of core when an out-of-core interrupt occurs. This is user-defined and must be equal in length to the largest interrupt core load used. The area should also be large enough to include the amount of COMMON used for communications between the mainline and the subprograms in the interrupt core load.

Core Load Area. An area of disk set aside for the storage of core loads as a result of a DUP *STORECLI operation. Data files are also stored.
here. The *FILES control record can refer to data files located in this area.

All programs and data files stored in the Core Load Area are assigned fixed disk locations. This permits the disk location of a process program to be kept with the calling program, and results in faster access to the program. A Core Load Area can be assigned to each disk.

Special Save Area (SPSV). An area of disk used to save the variable area of core when a CALL SPECL is executed and time-sharing is inoperative.

Process Save Area (PRSV). Area used to store the variable area of core when a CALL SHARE or a CALL VIAQ (with an empty queue) is entered.

Skeleton Area (SKEL). Used to store a core image copy of the TSX System Skeleton. See System Design Considerations: System Skeleton.

Error Programs (EDP). This area is always required, and is used to store the disk portion of the TSX error analysis program. This program is executed when a machine error occurs.

Cold Start (CLST). This area holds the TSX Cold Start program -- which is the program that starts the TSX system into operation by loading the System Skeleton to core and transferring control to the System Director.

THE DEFINE CONFIG OPERATION

At an appropriate stage in system generation, the establishment of the user's disk configuration and changes to that configuration are performed using the CONFG code in an *DEFINE operation. The definition or redefinition of some of the parameters consists of changing a stored value on disk.

In some cases, however, the results of the definition are interrelated with the user's routines, and, as a result, the Skeleton Builder must be executed after these definitions are made. This is most notably the case when the size of the System Skeleton is changed to accommodate additional programs or data areas and when system areas are moved as a result of an increase of the core-image area.

Since this operation establishes the lengths of several related disk areas, and since the user may perform the redefinition several times before a final determination of the contents of the System Skeleton area is made, the DEFINE operation is designed to establish all of these areas using a small number of configuration control records.

*DEFINE CONFIG Control Card

Figure 73 gives an overview of the DEFINE CONFIG operation for a single drive system. The *DEFINE CONFIG control card is used when the user's variable areas are established on disk. Several types of definitions may be necessary depending upon the information punched in the card.

The control card contains a field of consecutive columns which are punched with a string of one or two character codes that have special meaning in the CONFG routine. These characters are each followed by a disk drive code to specify the drive(s) where the areas are to be established.

The areas are normally taken from the highest numbered sectors of Nonprocess Work Storage, and in the same sequence as specified for that drive in the string the user punched in the CONFG card (except for M on a one-drive system).

Other fields in the card permit the user to specify lengths for areas that are not fixed by the system. These include the length estimated for the System Skeleton in core, the interrupt area utilized for out-of-core interrupts, the core load area on disk, the process work storage on disk, and the FORTRAN I/O save area on disk.

The user must execute the DEFINE CONFIG operation before he builds his skeleton. If areas established by this definition turn out to be incorrect due to miscalculation of user requirements, he will probably have to re-define them, and if LSKE is too small and cannot be increased, rebuild his skeleton since this will not be known until Skeleton build time.

Each CONFG control card can define up to 11 user-assigned disk areas. In practice, two or three control cards should be necessary to define even the most complex systems.

In referring to Figure 73, we note that the user may employ up to nine different alpha characters in any flexible order to specify a maximum of 11 user-assigned areas, each area being related to its specific disk drive. Since we are primarily con-
cerned with a one-disk system, disk drive zero will be indicated.

Consider the general case of an *DEFINE CONFG control card with all nine possible alpha character codes punched in the first field, as shown in Figure 73.

In executing the *DEFINE CONFG operation, the define routine first establishes the legality of the control card, then scans the field starting at column 15, and interprets each alpha character in turn from left to right. Simultaneously, it establishes areas on the system disk drive.
specified in the exact sequence of parameters punched in the card.

The disk is configurated from its highest address upwards, and definition of user-assigned areas proceeds step-by-step vertically until the scanning of the parameter columns is terminated by a blank (see Figure 74).

In initial system generation, that is, when a new disk system is first created, certain system areas are always established at the high address end of disk by specifying the S parameter in column 15. These areas constitute a basic set of programs and disk areas that are required by the system, and are file-protected at all times. They are shown below (Figure 75) in their correct sequence of allocation.

Each of these programs and areas except the Core Load Area must be defined for only one drive in the system.

Note that the Special Save Area is included within the specification of the S parameter, only if S is immediately followed by an X in column 16.

At this point, a word of explanation is given on the establishment of the Message Buffer Area (MESS). This buffer (together with the Error Programs and the Cold Start Program) is always defined at the time the IBM Nonprocesa System object deck is loaded to disk by the System Loader (which computes the MESS length from data punched in the *DEDIT and TASK equate card, NOCYL). Its proper location at the high end of disk at the end of the loading operation is shown in Figure 76.

At DEFINE CONFIG time, the location of MESS is redefined, that is, it is bodily moved to a different location on the disk. For example, if the F parameter immediately follows SXO in the control card, MESS will be reflected as in Figure 77.
The Message Buffer Area always precedes the Core Load Area on a one-drive system. Note, however, that if MESS is not required, that is, no provision is made for it in the *DEDIT control card, M should not be specified in the *DEFINE CONFIG control card, and no redefinition of this area takes place.

Examples of Disk Configurations

The following examples illustrate typical disk configuration operations.

EXAMPLE 1. Define a single drive disk system such that logical disk drive zero shall contain the following areas:

- Core Load Area: 30 cylinders
- Process Work Storage: 6 cylinders
- FORTRAN I/O Save Area: 4 sectors
- Interrupt Save Area: 2000 words
- Skeleton Area: 16,000 words
- Message Buffer: 6 cylinders
- Error Dump Area
- Error Save Area
- Nonprocess Save Area

A 32K system is assumed.

It is the user's responsibility to specify the lengths of the first six areas required: the DEFINE CONFIG function automatically computes and allocates the remaining three areas. Since a 32K machine is used, an Error Dump Area equal to object core size (32K) and a Nonprocess Save Area equal to variable core (16K) are assigned to the disk. An Error Save Area of six sectors is always automatically allocated when the E parameter is specified.

The control card sequence for this operation is:

```
*DEFINE CONFG EXCLUDES FOR I/O SAVE SPECIAL SAVE
M 32080 82320 82396 808
I 750 ...
I END OF D-1, JOB
```

Figure 78 reflects the disk layout for disk drive zero on completion of this function.
EXAMPLE 2. Define a three-drive TSX on-line system as follows:

**Drive Zero**
- Label this disk cartridge: 13579
- Core Load Area = 30 cylinders
- Process Work Storage = 5 cylinders
- FORTRAN I/O Save Area = 24 sectors
- Skeleton Area = 16384 words
- Nonprocess Save Area

**Drive 1**
- Label this disk cartridge: 12345
- Establish a LET/FLET area of 2 cylinders
- Core Load Area = 100 cylinders
- Process Work Storage = 10 cylinders
- Message Buffer = 6 cylinders

**Drive 2**
- Label this cartridge: 09876
- Establish a LET/FLET area of 1 cylinder
- Core Load Area = 125 cylinders
- Interrupt Save Area = 24 sectors
- Error Dump Area
- Error Save Area

A 32K system is assumed.

As this is a multi-drive system, the total number of disk drives available to the system must be defined. Remember that the System Loader assumes at system load time that only one disk drive (logical disk drive zero) is present on the system; DCOM thus indicates only one disk drive. The *DEFINE NDISK is therefore a redefinition of the system; this must always be performed before the skeleton is built.

Disk cartridges, other than that on logical drive zero, which are intended for system usage are then initialized by the DUP *DLABL function. Since disk drives 1 and 2 have not previously been identified as system drives, they are cleared (including file-protect bits), disk addresses are written, a LET/FLET area is established as specified, and a label written on each disk cartridge. In the case of disk drive 2, a one-cylinder LET/FLET area is defined by default. Logical drive zero is relabelled with 13759.

A separate *DEFINE CONFIG control card must be used for each disk cartridge if it is to contain a Core Load Area and/or Process Work Storage. Note that S is only used to specify a basic set of system programs/areas, which includes the Core Load Area, for the system drive (logical drive zero). The parameter C, indicating a Core Load Area, is only used for auxiliary disk drives (drives 1 and 2).

As pointed out in Example 1, it is the user's responsibility to specify the lengths of the following:

- Core Load Area
- Process Work Storage
- FORTRAN I/O Save Area
- Interrupt Save Area
- Skeleton Area
- Message Buffer for 1053 Printer

The *DEFINE CONFIG function automatically computes and allocates disk storage for:

- Error Dump Area
- Error Save Area
- Nonprocess Save Area
- Nonprocess Working Storage

In our example, since a 32K machine is implied, an Error Dump Area equal to object core size (32K), and a Nonprocess Save Area equal to variable core (16K) are allocated. An Error Save Area of six sectors is automatically set aside whenever E is specified. Note that Message Buffer Size is determined by the TASK equate card NOCYL at TASK assembly time.

Figure 79 reflects the layout of the three disks following configuration.

The control record sequence for this multiple operation is given below:

Note: If different system packs are used alternatively with the same non-system pack (for example, if one system pack is used to store a program onto a non-system pack and another to delete the same program), errors may arise if the definitions of LSKEL in the system packs are not equivalent.
EXAMPLE 3. Define a three-drive TSX off-line system as follows:

**Drive Zero**
- Core Load Area = 50 cylinders
- Task Skeleton Area = 8000 words
- Message Buffer Area = say 6 cylinders
- Process Save Area

**Drive 1**
- Label this disk pack: 12635
- Establish a LET/FLET area of 2 cylinders
- Core Load Area = 150 cylinders

**Drive 2**
- Label this disk pack: 23764
- Establish a LET/FLET area of 5 cylinders
- Core Load Area = 50 cylinders

A 16K system is assumed.

The number of disk drives available to the off-line system is first defined, as in Example 2. Disk drives 1 and 2 are then initialized, and LET/FLET areas established (see Figure 80). Since each disk cartridge is to contain a Core Load Area, a separate control card is used to define each cartridge. Note that the skeleton area on logical drive zero is configurated to hold the TASK monitor system.

Figure 80 reflects the disk layouts of the off-line system.

The control record sequence is given below:
DISK CARTRIDGE INITIALIZATION

There are three programs within TSX concerned with disk cartridge initialization.

- TASK DISK WRITE ADDRESS (TDWA, a TASK utility program)
- DWRAD (a DUP function)
- DLABL (a DUP function)

A comparison of their features is given in Table 13 at the end of this section.

Use of TDWA

A disk cartridge cannot, by definition, be used for processing functions unless it is first initialized by TDWA. Thus, whenever a disk cartridge is initially supplied, or is to be re-initialized, TDWA must be used. This operation is carried out in the off-line mode under TASK control (e.g., at system generation time).

TDWA performs two basic functions: it (1) checks the ability of the disk to record and reproduce information, and (2) writes addresses on the disk, flags defective cylinders, zeroes all storage words, and records the first sector of each defective cylinder on sector zero of the disk cartridge which is file-protected. TDWA does not label the disk cartridge.

A practical example of the initialization of a two-disk-drive system is given in Programming Techniques: TSX Sample System. For TASK DISK WRITE ADDRESS system procedures, see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

Use of DWRAD

DWRAD allows the user to perform in an on-line or off-line mode the following functions:

- Rewrite sector addresses in any specified cylinderized area on any disk cartridge
- Retain or save the contents of the sectors indicated, if desired, for analysis purposes
- Zero from one to 199 cylinders as specified (except cylinder zero)

- Enforce file protect or file unprotect on the entire area specified

DWRAD does not label the disk cartridge.

As disk sector addresses may be inadvertently modified or destroyed during the execution of user programs, such as in the transfer of data to core by a READ command, or by hardware failures, DWRAD provides the user with the ability to rewrite sector addresses of specified areas without recourse to a re-initialization process by TDWA. Whenever addresses in a certain area are destroyed, data cannot be retained or preserved in these sectors; the area must be readdressed and the data zeroed.

The file protect/file unprotect feature is useful in those situations where the user desires to enforce or remove file protection from a specified disk file or systems area. He might, for example, require to remove file protection from a certain portion of the Core Load Area for Assembler WRITE operations, and later restore file protection to that area.

The following examples depict typical operations.

EXAMPLE 4. Two cylinders beginning at sector 408 and ending at sector 417 are to be file protected. Assume that the disk cartridge is on disk drive zero. Information on these cylinders is not to be changed.

```
SAMPLE CODING FORM
```

<table>
<thead>
<tr>
<th></th>
<th>1-10</th>
<th>11-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 JOB 2</td>
<td>&lt;address&gt;</td>
<td>&lt;address&gt;</td>
<td>&lt;address&gt;</td>
<td>&lt;address&gt;</td>
<td>&lt;address&gt;</td>
</tr>
</tbody>
</table>

Note that file protection (or the removal of file protection) will only be effective if the disk drive indicated (in column 11) is defined to be on the system. That is, it is specified on the // JOB control card.

EXAMPLE 6. Zero cylinders 27,10 to 32,10 with file protection removed. Assume that disk cartridge is on drive 2.

```
SAMPLE CODING FORM
```

<table>
<thead>
<tr>
<th></th>
<th>1-10</th>
<th>11-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 JOB 2</td>
<td>X</td>
<td>&lt;address&gt;</td>
<td>&lt;address&gt;</td>
<td>&lt;address&gt;</td>
<td>&lt;address&gt;</td>
</tr>
</tbody>
</table>

Use of DLABL

Disk cartridges on disk drives other than logical drive zero, and intended for system usage, must be initialized by means of the DUP *DLABL function. DLABL serves three purposes:

- Places a label on the cartridge
- Establishes a LET/FLET table
- If certain conditions are met, it also writes addresses on an entire disk
DLABL places a numeric label, as specified by the user on the *DLABL control card, in word zero of sector zero on the disk cartridge. To prevent users from inadvertently destroying the system, only the label is written when the disk drive specified is drive zero, and when the drive is a system drive. All other data on these cartridges, including defective cylinder addresses in sector zero, remain unchanged.

For all other disk cartridges, DLABL assumes an unlabelled, pre-addressed disk cartridge. It then clears the cartridge, including the file-protected areas, but not including the defective cylinder addresses in sector zero of cylinder zero; writes a new label, if specified, in word one, sector zero of the disk, and establishes a file-protected LET/FLET area (LET in sector one, FLET in the last sector of the LET/FLET cylinder area). Note that unless the disk cartridge operated on is located on logical drive zero, or a system drive specified on the //JOB card, DLABL will erase all data as it writes addresses, and always establish one file-protected cylinder.

The size of the LET/FLET area is determined either by user specification on the DLABL control card, or, if unspecified, automatically made to be eight sectors.

Each DUP *DLABL function must be run as a separate job; that is, each *DLABL control card should be preceded by a //JOB, //DUP card combination, and should be followed by the next //JOB card. It can be performed in either the on-line or off-line mode.

For typical DLABL operations, see Examples 2 and 3; also, Programming Techniques: TSX Sample System.

### Table 13. Comparison of TDWA, DWRAD, and DLABL Features

<table>
<thead>
<tr>
<th>FEATURES</th>
<th>TDWA</th>
<th>DLABL</th>
<th>DWRAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARTRIDGE TEST</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>ON-LINE OR OFF-LINE</td>
<td>OFF-LINE</td>
<td>BOTH</td>
<td>BOTH</td>
</tr>
<tr>
<td>CLEARS CARTRIDGE</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>FILE PROT./FILE-UN PROT.</td>
<td>FILE PROT. CYL. ZERO ONLY</td>
<td>NO</td>
<td>FILE UNPROTECTS</td>
</tr>
<tr>
<td>WRITES SECTOR ADDRESSES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>WRITES LABEL</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ESTABLISHES LET/FLET AREA</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

### SUMMARY OF DISK STORAGE REQUIREMENTS AND ASSIGNMENT RESTRICTIONS

1. Disk areas that are fixed and equal for all systems, regardless of core size. These are illustrated below. Note that these areas constitute a basic nonprocess system.

| PARTS     | \hline
| DCOM      | 1 CYLINDER |
| MBT-AT    |            |
| S K SUB   | 11 SECTORS |
| . SUP     | 9 SECTORS  |
| . CLB     | 64 SECTORS |
| . DUP     | 40 SECTORS |
| . ASM     | 104 SECTORS|
| . FOR     | 100 SECTORS|
| . SIM     | 1 CYLINDER |
| LET-FLET  |            |
| / EPSV    | 6 SECTORS  |
| . EDP     | 30 SECTORS |
| / CLST    | 6 SECTORS  |

2. Disk areas that are not dependent upon core size, but which may vary in disk storage requirements

**Relocatable Program Area.** This will expand or contract in size as relocatable programs are added to or deleted from the system.

**Nonprocess Work Storage.** NPWS will increase or decrease as relocatable programs and subroutines are added to or deleted from the system.
Message Buffer Area. The size of the Message Buffer Area is computed by the user and specified during system generation. The factors which determine its size are discussed in System Design Considerations: TASK.

Core Load Area. The size of the Core Load Area is specified by the user at DEFINE CONFIG time.

System Skeleton Area. This is a copy of the skeleton area of core; its size is determined by the formula:

\[ \text{Skeleton Size} = \text{Total Core Size} - \text{Size of Variable Core} \]

FORTRAN I/O Save Area. The number of sectors required for this area is determined by the number of interrupt levels which use FORTRAN I/O. See Disk Organization.

3. Disk areas where storage requirements depend upon the size of variable core. These include:
   - Nonprocess Save Area
   - Interrupt Save Area
   - Special Save Area
   - Process Save Area

4. Disk areas that must be assigned to logical disk drive 0:
   - DCOM
   - MBT-AT
   - SK-SUB
   - SUP
   - CLB
   - DUP
   - ASM
   - FOR
   - SIM
   - IBM Subroutine Library

5. Disk areas that must be assigned to one disk drive, but need not be assigned to logical disk drive zero:
   - EPDV
   - EPSV
   - CLST
   - INSV
   - EIOS
   - PRSV
   - SPSV

6. Disk areas that may be assigned to one disk drive, but need not be assigned to logical disk drive zero:
   - EPDM
   - NPSV
   - MESS
   - FIOS
   - SPSV

7. Disk areas that will be assigned to the same disk drive, but need not be assigned to logical disk drive zero:
   - PRSV
   - SKEL
   - ERPG
   - CLST

8. Disk areas that can be assigned to more than one disk drive:
   - Relocatable Program Area
   - NPWS
   - PRWS
   - Core Load Area

9. Disk areas that must be assigned to every disk drive:
   - LET/FLET

10. Disk File Protection
    (a) Certain areas of the disk are file-protected. This means that the user cannot write into any of these areas at object time, although he can update file-protected files by using FORTRAN I/O only. System programs can, however, write into file-protected areas.
        These areas include the following:
        - DCOM
        - MBT-AT
        - SK-SUB
        - SUP
        - CLB
        - DUP
        - ASM
        - FOR
        - LET/FLET
        - IBM Subroutine Library
        - Relocatable Program Area
        - Core Load Area
        - SPSV
        - PRSV
        - SKEL
        - EDP
        - CLST

160
User-written programs are stored in the file-protected areas on disk. Programs are written into (that is, added to the system) file-protected areas by the DUP *STORE, *STOREMD, *STORECI, and *STOREDATA operations.

File-protected areas are fixed in size and cannot be altered by the user.

SYSTEM SKELETON

In an on-line TSX system, a nucleus of supervisory programs and their associated work areas and tables must be permanently core-resident to obtain efficient and continuous operation. At the center of this nucleus is the System Director which provides the essential communications between core loads and interrupt servicing routines. This framework of programs is referred to as the System Skeleton.

TASK is used in conjunction with the Skeleton Builder program to construct the in-core skeleton as required for TSX system operation within the limits prescribed by the user. The size and content of the skeleton is dependent on the size of the object machine, the size of user's process programs, and the size of process core loads which the user may plan to move in and out of core during system operation. The skeleton can be considered as the permanent part of all executable core loads.

For on-line processing to take place, the System Skeleton must be loaded to core memory; this is accomplished initially by the system cold start program.

CONSTITUTION OF THE SYSTEM SKELETON

Figure 81 illustrates the various component parts that make up the System Skeleton. Each of these parts and its function are explained below.

Fixed Area. This is effectively a systems communications area containing information used by all TSX system programs. It is initially assembled as part of TASK. At skeleton build time, various values in this area are initialized by the Skeleton Builder from System Director input. A disk image of the Fixed Area can be obtained by a disk dump of the first sector of the Skeleton. See Appendix C: Contents of the Fixed Area of Core.

Skeleton I/O. An identical set of input-output routines to that used by TASK forms the basis of Skeleton I/O. This permits the user to perform various disk, printer, and card utility functions (see System Design Considerations: TASK).

Skeleton Common. The maximum size of the Skeleton Common area for an object machine is defined and fixed by the user at TASK assembly time through the equate card COMSZ. COMSZ may be zero or any positive decimal value that will not cause the skeleton...
size to exceed the start address of variable core (that is, VCORE). TASK will determine the start address and word count of this common area and store them in words 156 and 157 in the Fixed Area. INSKEL COMMON is the only common area that is permanently core-resident. It provides communications between various core load types, and those subroutines included in the skeleton. When INSKEL COMMON is referenced in a FORTRAN program, listed variables are assigned addresses in Skeleton Common. All other attributes of the COMMON, as used in the FORTRAN language, are retained.

System Director. This forms the operating center of the TSX system. It has the responsibility of directing interrupt servicing, loading of user core loads, supervising time-sharing, and servicing of interval timers and error conditions. When the system is operating under control of the System Director, control is passed to it by TSX calls, interrupts, and error conditions. The System Director is that portion of TSX, other than the Skeleton I/O, which must be in core at all times in order to respond to a real-time environment. A detailed discussion of its functions is given in another section of this manual.

User Subroutines. The user has the option of including frequently called subroutines and high priority interrupt routines in the skeleton. These may include:

- Skeleton subroutines
- Interrupt subroutines
- Programmed Interrupt subroutines
- Count subroutines
- User-written trace and error subroutines
- Timer subroutines

These programs must have previously been compiled/assembled by the user and stored in relocatable format in the Relocatable Program Area on disk.

In addition, if FORTRAN I/O is utilized, then those conversion routines (e.g., HOLEB, EBPRT) necessary for its proper use must also be included in the skeleton by having them specified on *INCLD control cards. A detailed examination of some of the important considerations governing the inclusion of subroutines in the skeleton is made later in this section.

Patch Area. This is the portion of core storage that remains between the end of the subroutine area and the Skeleton Program Name Table (PNT) which is allowed (not explicitly defined) for the modification of IBM and user programs within the skeleton. Its size is determined by user requirements, but should, in practice, be at least 100 words in length to allow for future IBM modifications.

Program Name Table. This is the part of the skeleton table area that maps (name, word count, and sector address) all core loads referenced to Program Sequence Control (PSC) by calls made by in-skeleton subroutines. The symbolic name (SYDIR) in truncated EBCDIC code for the System Director forms the first entry in this table. See Figure 82.

Executive Transfer Vector (ETV). The Executive Transfer Vector serves as a linkage between LIBF-type calls of a core load and corresponding routines in the skeleton. It is originally constructed by TASK; for each LIBF routine put into Skeleton I/O by TASK, an entry is made into the ETV. At skeleton build time, the Skeleton Builder inserts an entry into the ETV for each entry point of each LIBF routine placed in the skeleton, by extending the size of the original ETV to reflect the entries for the included subroutines.
Figure 82. A Partial Dump Following a Skeleton Build to Illustrate the Program Name Table and the Executive Transfer Vector

Each entry in the ETV is three words long, corresponding to the format of the VTV table associated with each core load. When the skeleton build process is completed, word 3 will contain the address of the corresponding entry in the Executive Branch Table (EBT).

The Executive Transfer Vector is, in reality, a copy or duplicate of the ETV in all core loads, and is effective only during disk-to-core transfers.

Figure 82 is given to provide a better insight of the Program Name Table (PNT) and the Executive Transfer Vector (ETV). It is a partial dump snapshot of the skeleton built for the 1800 TSX-Sample System (see Programming Techniques), and should be studied in conjunction with the Skeleton Core Map.

Each entry in the PNT is four words in length. Since the System Director is the first (and constitutes the minimum) entry in a PNT, let us examine this entry. From the dump we see

220C | Name (SYDIR) of System Director in truncated EBCDIC form
2259 | Name (SCAN2) of this core load in truncated EBCDIC form
1572 | Core load word count
0C2E | Disk drive and sector address where core load resides
139C | Core load I/O Table

In turning to the ETV, we see that each entry is three words long. Consider the entry DISKN. From the dump we see

0000 | Entry point to the subroutine ETV entry
4480 | BSI indirect branch
00B8 | Address where subroutine branches through

Consider another entry, SCAN2. The dump shows

220C | Name (SCAN2) of this core load in truncated EBCDIC form
1572 | Core load word count
0C2E | Core load word count
139C | Disk drive and sector address where core load resides

Figure 82 also illustrates the Core Load Data Words (CDW), the Core Load IBT, the Core Load ETV, and the FORTRAN I/O table.
Executive Branch Table (EBT). A map of all LIBF
and call subroutines in the user subroutine and Sys­
tem Director areas of the skeleton. Each entry is
one word long. The EBT is employed as a transfer
vector: an indirect branch through the EBT is used
to enter the referenced subroutine.

Skeleton Interrupt Branch Table (SKIBT). A map of
all interrupt servicing subroutines in the skeleton.
It is used in conjunction with the Master Branch
Table (core load header words in sectors 1 and 2 on
disk) to guide the interrupt to its proper routine in
the skeleton. Each entry consists of a two-word
BSC instruction.

The table is built during skeleton build time by
entries put into the MBT. Word 2 of each entry is
filled by a word from the corresponding MBT entry,
while the location of the SKIBT entry itself replaces
the MBT entry.

SKELETON CORE SIZE

The length of the skeleton in words is defined by the
user at DEFINE CONFIG time, and given by the param­
eter LSKEI (see System Design Considerations: Disk System Configuration). LSKEI must be an even value
and is equal to the address of the first word in variable
core (VCORE).

In general, skeleton size is estimated by the user
after making allowances for the System Director,
Skeleton I/O, Skeleton Common, user-written sub­
routines, and the PATCH area. Skeleton Common
and the PATCH area dimensions are determined by
exact user requirements. Some of the important fac­
tors influencing skeleton size will now be considered.

Core Storage

The amount of core storage available determines the
number of features which can be included in a TSX
system (see System Design Considerations: System
Director). For example, is the system an off-line
or an on-line system? Is time-sharing required? Is
the Interval Timer Control feature used?

As a rule, in an on-line system using time-sharing,
variable core must be a minimum of 3692 words. For
example, in an 8K system! the maximum skeleton size
would then be 4500 words (see Figure 83).

If the time-sharing feature is not required in an on­
line system, the in-core skeleton may occupy all of
core less 2500 words. This 2500-word restriction is
necessary because of space requirements for the Cold
Start and Error programs at the high end of core
storage (see Figure 84). Under this system, non­
process work would have to be accomplished off-line
under TASK control (see Figure 85).

The maximum size of the skeleton is always dic­
tated by the balance of core storage above the skele­
ton: 3692 words minimum for Nonprocess Monitor
use, or for non time-sharing users, a minimum of
2500 words minimum for the Cold Start and Error
programs.

It should be noted that too small a skeleton may
demand frequent disk exchanges requiring excessive
time, while too large a skeleton may reduce the var­
iable space available for core loads and thus cause
over-segmentation.

Figures 83, 84, and 85 summarize these rules
for the on-line system (with time-sharing), on-line
system (non-time-sharing) and the off-line system.

Inclusion of Subroutines in the Skeleton

The user may elect to include interrupt and other
subroutines permanently in the skeleton for more
rapid system response. The criteria governing these
inclusions and the advantages gained thereby are dis­
cussed below.

Figure 83. On-Line (Time-Sharing) System
**Response Time.** The shortest response time (that is, the minimum time before an interrupt servicing routine is entered after a process or program interrupt is recognized) is obtained by placing the servicing routine in core with the skeleton. The main advantage of having as many routines as possible permanently resident in the skeleton is faster response time.

**Commonly-used Subroutines.** Subroutines which are commonly enough referenced in core loads warrant inclusion in the skeleton. The advantages derived are:

1. Better utilization of disk user area and core load area. Every subroutine that is included in the skeleton is commonly shared by the several core loads referencing that subroutine. This means that each core load no longer contains the called subroutine, thus reducing core load length, and hence disk space.
2. Since the length of individual core loads has been shortened, less disk time is required to load the core load.

3. The effect of having a larger skeleton produces a disk space advantage. A larger Core Load Area is now possible because save areas, that is Special Save (SPSV), Process Save (PBSV) and Nonprocess Save (NPSV) can be smaller. Of necessity, the Interrupt Save (INSV) Area (as defined by the DEFINE CONFG parameter (LINS) can be no longer than the above-mentioned save areas. It can now be smaller. Furthermore, the user can now delete those subroutines that are included in the skeleton from the user area. This extra space on disk thus gained may be utilized for other purposes such as work areas and relocatable programs.

Subroutines Specified in the CALL TIMER Statement. Timer servicing subroutines should, as a rule, always be included in the skeleton. These subroutines can perform some execution, but it is preferred, if any I/O device is required, that they simply set a programmed interrupt (by a CALL LEVEL (I)) or queue a mainline core load and return control to ITC.
Consider for example where a mainline calls a timer as follows:

```
EXTERNAL SUB1
CALL TIMER (SUB1, 1, INCRE)
```

The user is responsible to ensure that the mainline that requested the CALL TIMER statement remain in core until the end of the elapsed specified time (that is, until the timer times out). One way of achieving this is for him to mask out all out-of-core interrupt levels and not change core loads until the timer interrupts. He thereby ensures that the core load containing the subroutine SUB1 remains in core.

If, however, he does not wish to remain in a masked state, a second approach is to have previously included SUB1 in the skeleton. In this way, he does not incur the penalty of waiting for the timer to interrupt, and also gains the advantage of not tying up variable core.

**Count Subroutines.** The Count subroutine is simply another method of servicing an event. Unlike timer interrupts, count interrupts may run off a different time base, and utilize larger time intervals. If the user plans to use the CALL COUNT statement, he should remember that for immediate servicing of an event, it is preferred that he set up these routines as part of his skeleton. If he does not, the event can be recorded and subsequently serviced by a CALL QIFON.

**I/O Devices and their Associated Conversion Subroutines Required for FORTRAN DP I/O.** On an on-line TSX system, the normal DP I/O utility functions are carried out by a package of skeleton I/O routines which also forms the basis of input-output operations in TASK. That is, the same set of I/O routines exist in TASK and the System Skeleton. Since the TSX System is a disk operating system, DISKN will be in core (that is, in the skeleton) at all times. If a 1053 printer or a 1816 keyboard/printer has been specified in the assignment stage, TYPEN/WRTYN must also be permanently in core. Similarly, if a 1443 printer has been assigned, PRNTN must be resident in the skeleton I/O area. These routines are automatically included in the System Skeleton at skeleton build time.

Although CARDN is always resident in core during TASK execution, this routine is an optional part of the Skeleton I/O, depending on user requirements. The user must, however, define at TASK assembly time (by equating CDINS to 1 or 0) whether or not CARDN is to be in the skeleton. If it is not included, it will be loaded as a part of those core loads which address the card reader. Note that in this event, the non-process components (such as the Assembler, FORTRAN Compiler, etc.) use their own card I/O routine.

If the user intends to do FORTRAN DP I/O from the skeleton, he should ensure that the conversion subroutine associated with any DP I/O device used by skeleton subroutines be included in the skeleton. He does this by means of an *INCLD control card at skeleton build time which loads the appointed conversion routine from the IBM Subroutine Library.

Figure 86 shows the relationship between each DP I/O device and its associated pair of function and conversion subroutines. For example, the 1442 card/read punch unit is associated with CARDN (its function routine) and HOLEB (its conversion routine). Note that there is no conversion routine for the 2401 magnetic tape drive: the conversion is by-passed. Conversion routines should be consistent with the precision required (that is, whether standard or extended) as specified in the TASK equate card PRICS.

**Inclusion of Explicit and Implicit Subroutines.** In the compilation of a FORTRAN problem program, the compiler-generated machine language coding includes a large number of branch instructions which transfer control to subroutines during execution of that program. It is, in fact, the subroutines that perform the majority of operations in any given problems. These subroutines can be classified into two distinct types: explicit, and implicit.

Explicit subroutines are those subroutines that are clearly formulated or externally visible in a main program. Implicit subroutines, on the other hand, are those subroutines which are involved in the solution of a problem program, but not externally revealed.

<table>
<thead>
<tr>
<th>ASSOCIATED SUBROUTINES</th>
<th>DP I/O DEVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPEN</td>
<td>1053/1816 Keyboard Printer</td>
</tr>
<tr>
<td>EBPRT</td>
<td></td>
</tr>
<tr>
<td>CARON</td>
<td>1442 Card/Read Punch</td>
</tr>
<tr>
<td>HOLEB</td>
<td></td>
</tr>
<tr>
<td>PRNTN</td>
<td>1443 Printer</td>
</tr>
<tr>
<td>EBPRT</td>
<td></td>
</tr>
<tr>
<td>PAPTN</td>
<td>1054/1055 Paper Tape Reader &amp; Punch</td>
</tr>
<tr>
<td>PAPEB</td>
<td></td>
</tr>
<tr>
<td>MAGT</td>
<td>2401 Magnetic Tape Unit</td>
</tr>
<tr>
<td>TYPEN</td>
<td>1816 Keyboard Unit</td>
</tr>
<tr>
<td>HOLEB</td>
<td></td>
</tr>
<tr>
<td>PLOTX</td>
<td>1627 Plotter Unit</td>
</tr>
<tr>
<td>ECHRI or FCHRI</td>
<td></td>
</tr>
</tbody>
</table>

Figure 86. Illustrating Relationship of DP I/O Devices to Associated Function and Conversion Subroutines
If the user plans to include FORTRAN subroutines in the Skeleton, he should make adequate core space allowance both for explicitly named and implicitly called subroutines. The explicitly named user-written subroutine is included in the Skeleton by specifying its name in an *INCLD control card at skeleton build time, while any implicitly referenced subroutines will be automatically loaded at the same time.

The following two examples examine skeleton core requirements for typical situations involving explicit and implicit subroutines.

**EXAMPLE 1 - FORTRAN CASE.** Consider a main program which is required:

1. To set up variables of a 10 by 10 matrix and
2. To call a user-written subroutine MSQRT which is to compute the square root of each element in the array.

Program Listing No. 5 shows the compilation run, from which it can be seen that if the user intends to include the subroutine MSQRT in the skeleton, he should make appropriate skeleton space allowances for the following:

- (Explicit) \( \text{MSQRT} = (\text{program + variable}) \)
  
  \[
  \begin{array}{l}
  \text{(Explicit)} \quad \text{MSQRT} = 72 \text{ words} \\
  \text{(Implicit)} \quad \text{FSQRT} = 86 \text{ words} \\
  \quad \text{FSTOX} = 102 \text{ words} \\
  \quad \text{SUBSC} = 44 \text{ words} \\
  \quad \text{SUBIN} = 36 \text{ words} \\
  \end{array}
  \]

- (Subroutines req. by FSQRT):
  
  \[
  \begin{array}{l}
  \text{FMPI = 65 words} \\
  \text{FDIV = 106 words} \\
  \text{FADD = 158 words} \\
  \end{array}
  \]

- (Subroutines req. by FMTY, FDIV, FADD):
  
  \[
  \begin{array}{l}
  \text{FARC = 52 words} \\
  \text{NORM = 44 words} \\
  \text{FTRTN = 40 words} \\
  \text{XMDS = 42 words} \\
  \text{847 words} \\
  \end{array}
  \]
PROGRAM LISTING NO. 5: EXAMPLE 1 -- FORTRAN CASE

// JOB
// FOR MATRX
#NONPROCESS PROGRAM
#LIST ALL
C SAMPLE MAIN PROGRAM TO CALL A MATRIX SQUARE ROOT SUBROUTINE
DIMENSION VALUE(10,10)
N = 10
SUM = 0.0
DO 5 J = 1,N
DO 5 J = 1,N
SUM = SUM + 1.0
5 VALUE(I,J) = SUM
CALL MSQRT(VALUE,N)
CALL EXIT
END

VARIABLE ALLOCATIONS
VALUE(1) = 00C6-0000 SUM(R) = 00CB N(I) = 00CA I(I) = 00CC J(I) = 00CE

STATEMENT ALLOCATIONS
5 = 00F0

FEATURES SUPPORTED
NONPROCESS
CALLED SUBPROGRAMS
MSQRT FADD FLD FSTO FSTOX SUBSC

REAL CONSTANTS
.0000000E 00=0002 .1000000E 01=0004

INTEGER CONSTANTS
10=0006 1=0007

CORE REQUIREMENTS FOR MATRX
COMMON 0 INSKEL COMMON 0 VARIABLES 210 PROGRAM 66

END OF COMPILATION

MATRX
DUP FUNCTION COMPLETED
// FOR MSQRT
#LIST ALL
#NONPROCESS PROGRAM
SUBROUTINE MSQRT(A,N)
C USER WRITTEN MATRIX SQUARE ROOT SUBROUTINE
DIMENSION A(10,10)
DO 1 I = 1,N
DO 1 J = 1,N
1 A(I,J) = SQRT(A(I,J))
RETURN
END

VARIABLE ALLOCATIONS
I(I) = 0000 J(I) = 0002

STATEMENT ALLOCATIONS
1 = 001A

FEATURES SUPPORTED
NONPROCESS
CALLED SUBPROGRAMS
FSQRT FSTOX SUBSC SUBIN

INTEGER CONSTANTS
1=0006

CORE REQUIREMENTS FOR MSQRT
COMMON 0 INSKEL COMMON 0 VARIABLES 6 PROGRAM 66

END OF COMPILATION

MSQRT
DUP FUNCTION COMPLETED
*STORE MSQRT
MSQRT
DUP FUNCTION COMPLETED
*STORECIL MATRX MATRX
*CCEND
To call a user-written subroutine QUAD to solve a quadratic equation

\[ AX^2 + BX + C = 0 \]

using the positive square root, and assuming the quantity under the square root sign is greater than zero.

From Program Listing No. 6, it can be seen that if the user intends to include QUAD in the skeleton, adequate space allowances should be made for the following:

(Explicit) \( \text{QUAD} = (\text{program + variables}) \)

- FLD = 102 words
- FMPY = 65 words
- FSQR = 86 words
- FADD = 158 words
- FDIV = 106 words

(Implicit)

- FARC = 52 words
- FTNTR = 40 words
- XMDS = 42 words
- NORM = 44 words

777 words

EXAMPLE 2 -- ASSEMBLER CASE. Consider a main program which is required:
PROGRAM LISTING NO. 6:  EXAMPLE 2 -- ASSEMBLER CASE

// JOB
// ASM
// LIST

* SAMPLE SUBROUTINE TO SHOW THE USE OF
  IMPLICIT AND EXPLICIT SUBROUTINE
  CALLS

* THIS SUBROUTINE SOLVES A QUADRATIC
  EQUATION
  \[ A(x^2) + Bx + C = 0 \]
  USING THE POSITIVE SQUARE ROOT, AND
  ASSUMING THE QUANTITY UNDER THE
  SQUARE ROOT SIGN IS GREATER THAN ZERO

0012 18901100  ENT QUAD
0000 00000000  BSS E 0
0000 00000000  A DEC 0.0  ARG A WILL BE STORED HERE
0002 00000000  B DEC 0.0  ARG B WILL BE STORED HERE
0004 00000000  C DEC 0.0  ARG C WILL BE STORED HERE
0006 00000000  X DEC 0.0  RESULT WILL BE STORED HERE
0008 00000000  TEMPI DEC 0.0
000A 00000000  TEMPI DEC 0.0
000C 004000083  FOUR DEC 4.0
000E 004000082  TWO DEC 2.0
0010 0002  TEMPI BSS 2

0012 0000  QUAD DC 0
0013 01C400012  LD I QUAD VALUES COME FROM MAIN PROG
0015 0DBF   STD TEMP
0016 0C800010  LDD I TEMP
0018 0087  STD A
0019 017401012  MDX L QUAD,1
001B 01C400012  LD I QUAD
001D 0087  STD A
001E 0C800010  LDD I TEMP
0020 0081  STD B
0021 017401012  MDX L QUAD,1
0023 01C400012  LD I QUAD
0025 0DBF   STD TEMP
0026 0C800010  LDD I TEMP
0028 0DBF   STD C

0029 20064C4000  LIBF FLD TO CALCULATE DIVISOR 2*A
002A 1000E  DC TWO
002B 2006517A00  LIBF FMPY
002C 10000  DC A
002D 20068A3580  LIBF FSTO
002E 10008  DC TEMPI

002F 20064C4000  LIBF FLD
0030 10000  DC A
0031 2006517A00  LIBF FMPY
0032 10000  DC C
0033 2006517A00  LIBF FMPY
0034 10000  DC FOUR
0035 20068A3580  LIBF FSTO
0036 10000  DC TEMP2
0037 20064C4000  LIBF FLD
0038 10002  DC B
0039 2006517A00  LIBF FMPY
003A 10000  DC B
003B 20068A4080  LIBF FSB
003C 10000  DC TEMPI
003D 20068A9640  CALL FSQR
003F 20068A4080  LIBF FSB
0040 10002  DC B
0041 2006109940  LIBF FDIV
0042 10008  DC TEMPI

0043 20068A3580  LIBF FSTO
0044 10006  DC X
0045 017401012  MDX L QUAD,1
0046 01C400012  LD I QUAD
0049 00DC6  STD TEMP
004A 0C88  LDD X
004C 01DC00010  STD I TEMP
004D 017401012  MDX L QUAD,1
004F 014C800012  BSC I QUAD

0052

NO ERRORS IN ABOVE ASSEMBLY.
QUAD
DUP FUNCTION COMPLETED
// ASM MAIN
// LIST

170
* MAIN PROGRAM TO CALL SUBROUTINE QUAD *

; BSS E 0
; AA DEC 1.0
; BB DEC 3.0
; CC DEC 2.0

; TO BE FILLED IN BY SUBR

; START CALL QUAD

; 0000 0000 DC AA
; 0000 0001 DC BB
; 0000 0002 DC CC
; 0000 0006 DC XX
; 0000 0000 WAIT
; 0000 0008 END START

; NO ERRORS IN ABOVE ASSEMBLY.

; MAIN DUP FUNCTION COMPLETED
; // SEQ MAIN L
+CCEND

; CLB, BUILD MAIN

; CORE LOAD MAP
; TYPE NAME ARG1 ARG2

; *CDW TABLE 4002 000C
; *FIO TABLE 4031 0010
; *ETV TABLE 4041 000C
; *FIO TABLE 404D 0021
; *PNT TABLE 406E 0004
; MAIN MAIN 407A
; PNT MAIN 4070
; CALL QUAD 4096
; LIBF FLDP 413A 404D
; LIBF FMPY 4153 4050
; LIBF FSTD 4120 4053
; LIBF FSTD 4120 4053
; LIBF FSUB 4144 4056
; CALL FSOR 4246
; LIBF FADD 4180 4059
; LIBF DIV 4289 405C
; LIBF FARC 42EE 405F
; LIBF FSTD 40D6 4062
; LIBF FLOG 4135 4065
; LIBF FDIV 4284 4068
; LIBF FADD 41A4 406B
; CALL FTNTR 4322
; CALL FTNTR 433C
; CORE 434C 3CB4

; CLB, MAIN LD XQ

; // JOB
; // END OF ALL JOBS
CALCULATING SKELETON CORE SIZE

Skeleton Core Size can be arrived at by computing the value of the start address of variable core (VCORE), an even number. VCORE is equal to a summation of:

\[
\text{SKELETON I/O} + \text{INSKEL COMMON} + \text{SYSTEM DIRECTOR} + \text{USER'S PROGRAMS} + \text{SKELETON TABLES} + \text{PATCH AREA}
\]

The manner in which the SYSTEM DIRECTOR is calculated has already been discussed (see System Design Considerations: System Director). INSKEL COMMON, the PATCH AREA, and the USER'S PROGRAMS are defined by the user.

SKELETON I/O = Fixed Area = 200 words
+ Disk Device Tables = 66 words per drive
+ DISKN = 900 + (12 + (9XDORG2)) X DORG1 + 12 X MKLEV
+ 1053 Device Tables = 14 + message unit size (add 1 if the message unit size is odd) per 1053 and 1816
+ 1816 Device Tables = 14 words per keyboard
+ TYPEN = 256 + (3XN) + (341 + (8XNOCYL)) X NOBUF + (20 + (2XTORG))
  X(1 Nobuf) + (418 + (8X(M-1))) X TORG + 16XMKLEV
  where \( N \) = number of 1053/1816s
  and \( M \) = number of 1816s
+ 1053/1443 Timing Response Routine = 2 + (78XTORGN) + 22XPORG
+ 1443 Device Table = 14 words
+ PRNTN = 323 words + 16XMKLEV
+ Constants & Work Areas = 200 + 42XECPT1 + 27 (1-ECPT1)
  where ECPT1 = 0 or 1 if EAC printer
  is a 1053 or 1443 respectively
+ CARDN (if included) = 328 words + 23XCRDNO + 10XMKLEV

Note that NOBUF, TORG, TORGN, MKLEV, PORG, NOCYL, and ECPT1 are TASK equate cards.

SKELETON TABLES = Program Name Table = 4 + 4 X the number of called mainline and combination core loads from the skeleton.
+ Executive Transfer Vector = 3 X the number of LIBF functions in the skeleton.
+ Executive Branch Table = 1 word per entry for each entry point for LIBF and CALL Subroutines contained in the skeleton (excepting Skeleton I/O).
+ Skeleton Interrupt Branch Table = 2 X the number of bits in all labels (that is, ILSW words) NB00-23.
USE OF *INCLD CONTROL CARDS

Before the System Skeleton can be built, it is a condition that user-written subroutines and programs intended for inclusion in the skeleton must be assembled and/or compiled and stored on disk in relocatable format. These are assigned to the skeleton area by *INCLD control cards at skeleton build time. The various types of subroutines that are suitable candidates for inclusion have already been mentioned elsewhere in this section.

Subroutines planned for tracing and/or error options may also be included in the skeleton, but it should be noted that they are not, as such, automatically functional, because these routines can only be entered through linkages provided in the individual core loads. In order to use them, therefore, their names must appear in appropriate *INCLD cards which must be present when building those core loads where tracing or error options are desired. Furthermore, in the case of error subroutines, the entry is made only if the core load is a process mainline, combination or interrupt core load. If any nonprocess core load or monitor function is in progress, the error routine will not be included.

The following example is given to illustrate the general use of *INCLD control cards in a typical skeleton build operation. The sequence of control cards is shown below:

<table>
<thead>
<tr>
<th></th>
<th>1-10</th>
<th>11-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>018</td>
<td>018</td>
<td>018</td>
<td>018</td>
<td>018</td>
<td>018</td>
</tr>
<tr>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
<td>118</td>
</tr>
<tr>
<td>1C8</td>
<td>1C8</td>
<td>1C8</td>
<td>1C8</td>
<td>1C8</td>
<td>1C8</td>
</tr>
<tr>
<td>1D8</td>
<td>1D8</td>
<td>1D8</td>
<td>1D8</td>
<td>1D8</td>
<td>1D8</td>
</tr>
<tr>
<td>1E8</td>
<td>1E8</td>
<td>1E8</td>
<td>1E8</td>
<td>1E8</td>
<td>1E8</td>
</tr>
<tr>
<td>1F8</td>
<td>1F8</td>
<td>1F8</td>
<td>1F8</td>
<td>1F8</td>
<td>1F8</td>
</tr>
<tr>
<td>2A8</td>
<td>2A8</td>
<td>2A8</td>
<td>2A8</td>
<td>2A8</td>
<td>2A8</td>
</tr>
<tr>
<td>2B8</td>
<td>2B8</td>
<td>2B8</td>
<td>2B8</td>
<td>2B8</td>
<td>2B8</td>
</tr>
<tr>
<td>2C8</td>
<td>2C8</td>
<td>2C8</td>
<td>2C8</td>
<td>2C8</td>
<td>2C8</td>
</tr>
<tr>
<td>2D8</td>
<td>2D8</td>
<td>2D8</td>
<td>2D8</td>
<td>2D8</td>
<td>2D8</td>
</tr>
<tr>
<td>2E8</td>
<td>2E8</td>
<td>2E8</td>
<td>2E8</td>
<td>2E8</td>
<td>2E8</td>
</tr>
<tr>
<td>2F8</td>
<td>2F8</td>
<td>2F8</td>
<td>2F8</td>
<td>2F8</td>
<td>2F8</td>
</tr>
<tr>
<td>308</td>
<td>308</td>
<td>308</td>
<td>308</td>
<td>308</td>
<td>308</td>
</tr>
<tr>
<td>318</td>
<td>318</td>
<td>318</td>
<td>318</td>
<td>318</td>
<td>318</td>
</tr>
<tr>
<td>328</td>
<td>328</td>
<td>328</td>
<td>328</td>
<td>328</td>
<td>328</td>
</tr>
<tr>
<td>338</td>
<td>338</td>
<td>338</td>
<td>338</td>
<td>338</td>
<td>338</td>
</tr>
<tr>
<td>348</td>
<td>348</td>
<td>348</td>
<td>348</td>
<td>348</td>
<td>348</td>
</tr>
<tr>
<td>358</td>
<td>358</td>
<td>358</td>
<td>358</td>
<td>358</td>
<td>358</td>
</tr>
<tr>
<td>368</td>
<td>368</td>
<td>368</td>
<td>368</td>
<td>368</td>
<td>368</td>
</tr>
<tr>
<td>378</td>
<td>378</td>
<td>378</td>
<td>378</td>
<td>378</td>
<td>378</td>
</tr>
<tr>
<td>388</td>
<td>388</td>
<td>388</td>
<td>388</td>
<td>388</td>
<td>388</td>
</tr>
<tr>
<td>398</td>
<td>398</td>
<td>398</td>
<td>398</td>
<td>398</td>
<td>398</td>
</tr>
<tr>
<td>3A8</td>
<td>3A8</td>
<td>3A8</td>
<td>3A8</td>
<td>3A8</td>
<td>3A8</td>
</tr>
<tr>
<td>3B8</td>
<td>3B8</td>
<td>3B8</td>
<td>3B8</td>
<td>3B8</td>
<td>3B8</td>
</tr>
<tr>
<td>3C8</td>
<td>3C8</td>
<td>3C8</td>
<td>3C8</td>
<td>3C8</td>
<td>3C8</td>
</tr>
<tr>
<td>3D8</td>
<td>3D8</td>
<td>3D8</td>
<td>3D8</td>
<td>3D8</td>
<td>3D8</td>
</tr>
<tr>
<td>3E8</td>
<td>3E8</td>
<td>3E8</td>
<td>3E8</td>
<td>3E8</td>
<td>3E8</td>
</tr>
<tr>
<td>3F8</td>
<td>3F8</td>
<td>3F8</td>
<td>3F8</td>
<td>3F8</td>
<td>3F8</td>
</tr>
</tbody>
</table>

Five types of subroutines are considered:

- EBPRT and HOLEB are IBM Library Conversion Subroutines. JOHN is a special user-written arithmetic subroutine.
- I0402/0402 serves as an interrupt servicing subroutine for a process interrupt assigned to level 04, bit 02.
- PGI05/2405 and PGI15/2501 are interrupt servicing routines which service programmed interrupts on levels 5 and 15 respectively.
- CNT04/2604 and CNT17/2701 are user-written count routines #4 and #17 respectively.
- TRAC2, EROR1 and EROR2 are special user-written trace and error subroutines. Note that at core load build time, these three subroutines must be named by *INCLD cards in those core loads selected for the trace and/or error options in order to establish their linkages in the skeleton. (If this is not done, no tracing or error checking takes place.)

The flexibility in system usage and design (in permitting a core load to be traced by a special user-written trace routine other than the trace routine normally included in the skeleton) is a definite advantage as it is simpler to modify a core load than to modify the skeleton.

SUMMARY OF THE SKELETON BUILD PROCESS

Before the System Skeleton can be built, several prerequisite conditions must have been met. These include:

- The IBM Nonprocess System has been loaded.
- User-written subroutines required for residence in the skeleton have been assembled/compiled and stored on disk in relocatable format.
- The System Director has been assembled and stored on disk in relocatable format.
- Disk file configuration has been defined.
- An Operating TASK (that is a user-configurated TASK) has been assembled and punched into cards.

Briefly, the skeleton build function can be broken down (in simplified form) into three separate steps:

1. TASK Initialization
2. Relocation of component parts
3. Building the Skeleton Tables

Figure 87 illustrates the skeleton build operation as each sector of the skeleton is constructed and relocated to the Skeleton Area on disk.
Figure 87. Layout of the System Skeleton as it would appear at Skeleton Build Time in NPWS and the Skeleton Area.
**TASK Initialization.** TASK begins the skeleton build function by initializing certain communications areas which reside in sectors 1 - 4 on disk. These areas include:

- Master Communications Words (MCW)
- Master Branch Table (MBT)
- Skeleton FORTRAN I/O Table (SFIO)
- Executive Transfer Vector (ETV)
- Skeleton Subroutine Name Table (SKSUB)

This step determines the amount of core storage available between the high addressed end of TASK and the first word of the Skeleton Builder: the available core storage is cleared and subsequently allocated to the various loading tables. The Skeleton Subroutine Table (SKSUB) is then read from disk into its allocated area in core. The Control Record Entry Table (CRENT), which was constructed by the Skeleton Build Supervisor phase (SKA), is read, one entry at a time. These are analyzed and inserted into the appropriate tables.

**Relocation of TSX Component Parts.** The main function of the skeleton build operation is the construction in core-image format in the Nonprocess Working Storage on disk of the permanent part of all core loads and to establish their linkages to the System Director and various other IBM and user-written programs. As each sector of the skeleton is built it is written to the Nonprocess Working Storage. The disk image of the completed skeleton is finally physically moved to the Skeleton Area of the process portion of disk and file protected.

The Skeleton I/O is first written to a predefined area in the Nonprocess Working Storage on the temporary disk drive for later reference by the Skeleton Builder.

The length and initial address of Skeleton Common are now obtained from the Master Communications Area, and the desired size of the Common Area in the skeleton image cleared and reserved. The upper boundary of this area is later set to correspond with the starting address of the System Director.

It was noted in System Design Considerations: System Director that the System Director was assembled and stored on disk prior to a skeleton build. At this same time, space was allocated for an Interrupt Core Load Table (ICLT), its size being determined by user-specified NIL equate cards.

Using the System Director as the main program, the Skeleton Builder constructs the skeleton in much the same fashion that the Core Load Builder builds a core load. Certain words in the ICL table (the first header word and the first and second entry words) associated with bits corresponding to those interrupt subroutines resident in the skeleton are now filled in. Later on, DUP makes the required entries into the ICL table whenever it stores an interrupt core load on disk.

User-written and other subroutines are now included in the skeleton image on disk following the System Director.

**Building the Skeleton Tables.** During the final stage of the skeleton build process, the four tables shown in Figure 81 are modified and/or built from data provided in the Master Communications Area and user-specified *INCLD control cards. Note that the tables are built and used by the Skeleton Builder during the construction of the skeleton core load.

All of the loading tables are assembled in a descending chain; that is, the first entry occupies higher core locations than the second, the second higher than the third, and so on. The details and function of each table are given elsewhere in this section (see Constitution of the System Skeleton).

For detailed step-by-step operational procedures of the skeleton build process, the user should refer to the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.

**Rebuilding the System Skeleton**

While relocatable programs can be deleted and replaced on-line by the Nonprocess Monitor, it is not possible to modify any features of the System Skeleton on-line. Changes in the skeleton area (including skeleton interrupt routines) will thus require an off-line Skeleton rebuild.

The TSX Skeleton may be rebuilt at any time by following the detailed procedures specified for an initial Skeleton build in the IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754.
Since INSKEL COMMON is not open-ended, the user may face the difficulty of adding to it once it is defined. It is recommended that an extra area should be reserved in INSKEL COMMON to allow for programming contingencies. See Core Load Rebuild Conditions.

The Executive Branch Table (EBT) and the Skeleton Interrupt Branch Table (SKIBT) have already been described. These tables are provided to allow the user to rebuild the System Skeleton if he were modifying subroutines, changing the logic flow or adding patches to the System Director and TASK such that addresses in core loads will still reference a fixed address in variable core. An ability to shift the entry points of subroutines within the Skeleton is thus available without the necessity of rebuilding the referencing core loads.

If, however, the entry points within these tables no longer pointed to the same subroutine, all core loads must be rebuilt.

Core Load Rebuild Conditions

When the Skeleton is initially built, the entry points to the in-core-with-skeleton interrupt routines are placed in the ICL Table, and all other table entries are cleared.

When this skeleton is later rebuilt, the in-core-with-skeleton entry points are substituted into the proper areas; the word count and sector address of out-of-core interrupt routines are then recovered from the old skeleton and placed in their respective table locations, provided that there is not already an entry point there.

In a skeleton rebuild, it may not be necessary to rebuild those core loads built under the previous (old) Skeleton if the following conditions are met:

1. No previously included CALL or LIBF type subroutines may be removed from the Skeleton. An in-core interrupt (ICI) or a count subroutine may be substituted, provided it has the same number of entry points and occupies the same relative position within the Skeleton as the deleted subroutine. Any core loads referencing the deleted subroutine must be rebuilt.

2. No additional LIBF or ISS subroutine (disk format types 3, 5, and 6) may be included in the new Skeleton. In-core interrupt (ICI) and CALL (type 4) subroutines may be added provided the patch area is large enough to contain the additions.

3. The number of entry points for included subroutines may not be modified between inclusions.

4. If TASK or the System Director is reassembled, the number of interrupt levels used, the length of INSKEL COMMON, and the length of the Skeleton should not be altered.

Note that a change in the size of INSKEL COMMON implies a reassembly of TASK. Also, if the location of INSKEL COMMON changes, all core loads using INSKEL COMMON must be rebuilt. If, for example, Skeleton I/O changed in size, an adjustment in INSKEL COMMON size equal to the change in the Skeleton I/O could eliminate the necessity for rebuilding all core loads that use INSKEL COMMON.

5. If there is to be any variation in the number or order of *INCLD control cards, new *INCLD control cards must be prepared using the previous core map. They must specify (in order of occurrence in the previous map) all ICI, CALL, and LIBF names beginning with the first map entry following PNT SYDIR.

An alternative is to use the original *INCLD control cards in their original order. Additional ICI and CALL subroutines may be specified on following *INCLD control cards.

6. Following the rebuild process, a comparison of the previous Skeleton and new Skeleton core maps must show identical entry points (that is, ARG2 -- see Figure 88) for those LIBF and CALL map entries common to both skeletons.

Example of Initial Skeleton Build and Skeleton Rebuild

Figure 88 illustrates the general sequence of control cards, the Skeleton Core map, and the Interrupt Core Load (ICLT) map for a typical skeleton build/rebuild situation. A separate interpretation of these maps is given at the end of this section.

In the initial skeleton:

- INT01/0000 is an interrupt servicing routine for a process interrupt assigned to level 00, bit 00.
• INT02/2405 is an interrupt servicing routine which services a programmed interrupt on level 05.

• INT03/2500 services a programmed interrupt on level 14.

• SUB01 is a special user-written arithmetic subroutine.

• INT04/2600 is a user-written count routine #00.

• MASK and EBPRT are IBM Library Subroutines.

The four K13 level 1 Skeleton Builder error messages, following the ICL Table Map, are warnings to the user that core load names PML01, PML02, CCL01, and CCL02 referenced by calls in routines (e.g., CHAIN, SPECL, QUEUE) contained in the initial skeleton have not yet been built, and, therefore, are not entered in FLET (see IBM 1800 Time-Sharing Executive System, Operating Procedures, Form C26-3754).

In the rebuilt skeleton:

• No major modifications are implemented. Neither the Skeleton I/O nor INSKEL COMMON are altered.

• Four implicit routines (and those explicit routines referenced by these routines) are added. These are:
  1. SUB02 -- a user-written arithmetic routine.
  2. INT05/2701 -- a user-written count routine #17.

3. DPART and UNMK are IBM Library Subroutines.

4. These additions constitute entries to the ICL Table.

Note that SUB02 and INT05/2701 cannot reference or call any LIBF function subroutines as this violates the conditions stated (see Core Load Rebuild Conditions).

The user should be aware that in rebuilding a skeleton, the control cards must be in exact order-correspondence with the previous skeleton so that routines will be loaded in the same order-sequence and entry points in the transfer vector remain valid.

Interpretation of the Skeleton Core Map and the Interrupt Core Load Table (ICLT) Map

Skeleton Core Map

The Skeleton Builder always prints a map of the assembled skeleton formatted as follows:

<table>
<thead>
<tr>
<th>SKEL</th>
<th>CORE</th>
<th>MAP</th>
<th>(Page Heading)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>NAME</td>
<td>ARG1</td>
<td>ARG2</td>
</tr>
</tbody>
</table>

Type indicates the map entry type (e.g., LIBF, CALL, PNT). Up to five alphameric characters are allowed under NAME to describe a subroutine, control program, etc. (e.g., DISKN, SYDIR).
Control Records For Initial Skeleton Build
//JOB 1 X
//XEO 5KBLD
*INCLUDINT0/0000, MASK, EBPRT, INT02/2405
*INCLUDSUB01, INT03/2500, INT04/2600
*CCEND

Skeleton Core Map

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NAME</th>
<th>ARG1</th>
<th>ARG2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBF</td>
<td>DISKN</td>
<td>021B</td>
<td>3EBB</td>
</tr>
<tr>
<td>LIBF</td>
<td>TYPEN</td>
<td>0562</td>
<td>3EBE</td>
</tr>
<tr>
<td>LIBF</td>
<td>WRTYN</td>
<td>0582</td>
<td>3EBE</td>
</tr>
<tr>
<td>LIBF</td>
<td>PRNTN</td>
<td>0A07</td>
<td>3EC1</td>
</tr>
<tr>
<td>LIBF</td>
<td>CARDN</td>
<td>0C71</td>
<td>3EC4</td>
</tr>
<tr>
<td>CALL</td>
<td>EXIT</td>
<td>1C9F</td>
<td>0086</td>
</tr>
<tr>
<td>CALL</td>
<td>LINK</td>
<td>1CA1</td>
<td>008E</td>
</tr>
<tr>
<td>INSK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYDIR</td>
<td>10C2</td>
<td>3EBB</td>
</tr>
<tr>
<td></td>
<td>INT01</td>
<td>1F80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT02</td>
<td>1F91</td>
<td>1005</td>
</tr>
<tr>
<td></td>
<td>INT03</td>
<td>1F9E</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td>INT04</td>
<td>1FAF</td>
<td>1200</td>
</tr>
<tr>
<td>LIBF</td>
<td>EBPRT</td>
<td>1FDC</td>
<td>3EC7</td>
</tr>
<tr>
<td>CALL</td>
<td>SUB01</td>
<td>207D</td>
<td>3E43</td>
</tr>
<tr>
<td>CALL</td>
<td>OUTTR</td>
<td>19DA</td>
<td>3E42</td>
</tr>
<tr>
<td>CALL</td>
<td>CHAIN</td>
<td>1BA7</td>
<td>3E41</td>
</tr>
<tr>
<td>CALL</td>
<td>INTEX</td>
<td>1AE1</td>
<td>3E40</td>
</tr>
<tr>
<td>CALL</td>
<td>SHARE</td>
<td>1C4F</td>
<td>3E3F</td>
</tr>
<tr>
<td>CALL</td>
<td>SPECL</td>
<td>1BEC</td>
<td>3E3E</td>
</tr>
<tr>
<td>CALL</td>
<td>BACK</td>
<td>1C03</td>
<td>3E3D</td>
</tr>
<tr>
<td>CALL</td>
<td>EACLK</td>
<td>1ED4</td>
<td>3E3C</td>
</tr>
<tr>
<td>CALL</td>
<td>QUEUE</td>
<td>2084</td>
<td>3E3B</td>
</tr>
<tr>
<td>PNT</td>
<td>PML01</td>
<td>2146</td>
<td>3E3A</td>
</tr>
<tr>
<td>PNT</td>
<td>PML02</td>
<td>2146</td>
<td>3E3A</td>
</tr>
<tr>
<td>PNT</td>
<td>CCL01</td>
<td>3E32</td>
<td></td>
</tr>
<tr>
<td>PNT</td>
<td>CCL02</td>
<td>3E36</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>PRT</td>
<td>218C</td>
<td>3E39</td>
</tr>
</tbody>
</table>

ICL TABLE MAP
LLBB WC/EP SA ICLT
0000 1F80 11CC
1005 1F91 1296
1100 1F9E 12AC
1200 1FAF 1284

K13 PML01 LEV.1
K13 PML02 LEV.1
K13 CCL01 LEV.1
K13 CCL02 LEV.1
SK6, SYDIR LD NX

Figure 88. Core Map for Initial and Rebuilt Skeleton

Control Records For Skeleton Rebuild
//JOB 1 X
//XEO 5KBLD
*INCLUDINT0/0000, MASK, EBPRT, INT02/2405
*INCLUDSUB01, INT03/2500, INT04/2600
*INCLUDSUB02/DPART, INT05/2701, UNMK
*CCEND

Rebuilt Skeleton Core Map

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NAME</th>
<th>ARG1</th>
<th>ARG2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBF</td>
<td>DISKN</td>
<td>021B</td>
<td>3EBB</td>
</tr>
<tr>
<td>LIBF</td>
<td>TYPEN</td>
<td>0562</td>
<td>3EBE</td>
</tr>
<tr>
<td>LIBF</td>
<td>WRTYN</td>
<td>0582</td>
<td>3EBE</td>
</tr>
<tr>
<td>LIBF</td>
<td>PRNTN</td>
<td>0A07</td>
<td>3EC1</td>
</tr>
<tr>
<td>LIBF</td>
<td>CARDN</td>
<td>0C71</td>
<td>3EC4</td>
</tr>
<tr>
<td>CALL</td>
<td>EXIT</td>
<td>1C9F</td>
<td>0086</td>
</tr>
<tr>
<td>CALL</td>
<td>LINK</td>
<td>1CA1</td>
<td>008E</td>
</tr>
<tr>
<td>INSK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYDIR</td>
<td>10C2</td>
<td>3EBB</td>
</tr>
<tr>
<td></td>
<td>INT01</td>
<td>1F80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT02</td>
<td>1F91</td>
<td>1005</td>
</tr>
<tr>
<td></td>
<td>INT03</td>
<td>1F9E</td>
<td>1100</td>
</tr>
<tr>
<td></td>
<td>INT04</td>
<td>1FAF</td>
<td>1200</td>
</tr>
<tr>
<td></td>
<td>INT05</td>
<td>1FB0</td>
<td>1301</td>
</tr>
<tr>
<td>LIBF</td>
<td>EBPRT</td>
<td>1FDC</td>
<td>3EC7</td>
</tr>
<tr>
<td>CALL</td>
<td>SUB01</td>
<td>2091</td>
<td>3E43</td>
</tr>
<tr>
<td>CALL</td>
<td>OUTTR</td>
<td>19DA</td>
<td>3E42</td>
</tr>
<tr>
<td>CALL</td>
<td>CHAIN</td>
<td>1BA7</td>
<td>3E41</td>
</tr>
<tr>
<td>CALL</td>
<td>INTEX</td>
<td>1AE1</td>
<td>3E40</td>
</tr>
<tr>
<td>CALL</td>
<td>SHARE</td>
<td>1C4F</td>
<td>3E3F</td>
</tr>
<tr>
<td>CALL</td>
<td>SPECL</td>
<td>1BEC</td>
<td>3E3E</td>
</tr>
<tr>
<td>CALL</td>
<td>BACK</td>
<td>1C03</td>
<td>3E3D</td>
</tr>
<tr>
<td>CALL</td>
<td>EACLK</td>
<td>1ED4</td>
<td>3E3C</td>
</tr>
<tr>
<td>CALL</td>
<td>QUEUE</td>
<td>209B</td>
<td>3E3B</td>
</tr>
<tr>
<td>CALL</td>
<td>PRT</td>
<td>21A0</td>
<td>3E39</td>
</tr>
<tr>
<td>CALL</td>
<td>UNQ</td>
<td>215A</td>
<td>3E3A</td>
</tr>
<tr>
<td>CALL</td>
<td>SHAR4</td>
<td>1E24</td>
<td>3E3A</td>
</tr>
<tr>
<td>CALL</td>
<td>EACLK</td>
<td>1ED4</td>
<td>3E3C</td>
</tr>
<tr>
<td>CALL</td>
<td>QUEUE</td>
<td>209B</td>
<td>3E3B</td>
</tr>
<tr>
<td>CALL</td>
<td>UNQ</td>
<td>215A</td>
<td>3E3A</td>
</tr>
<tr>
<td>CALL</td>
<td>PRT</td>
<td>21A0</td>
<td>3E39</td>
</tr>
<tr>
<td>CALL</td>
<td>SHAR4</td>
<td>1E24</td>
<td>3E3A</td>
</tr>
<tr>
<td>CALL</td>
<td>EACLK</td>
<td>1ED4</td>
<td>3E3C</td>
</tr>
<tr>
<td>CALL</td>
<td>QUEUE</td>
<td>209B</td>
<td>3E3B</td>
</tr>
<tr>
<td>CALL</td>
<td>UNQ</td>
<td>215A</td>
<td>3E3A</td>
</tr>
<tr>
<td>CALL</td>
<td>PRT</td>
<td>21A0</td>
<td>3E39</td>
</tr>
<tr>
<td>CALL</td>
<td>SHAR4</td>
<td>1E24</td>
<td>3E3A</td>
</tr>
<tr>
<td>CALL</td>
<td>EACLK</td>
<td>1ED4</td>
<td>3E3C</td>
</tr>
</tbody>
</table>

ICL TABLE MAP
LLBB WC/EP SA ICLT
0000 1F80 11CC
0501 017A 04C6 11FA
0A02 017A 04C8 1230
1005 1F91 1296
1100 1F9E 12AC
1200 1FAF 1284
1301 1F8D 12DA

SKB, SYDIR LD XQ
ARG1 and ARG2 may contain either a four-digit hexadecimal number or a blank field.

**Program Name Table (PNT).**

PNT NNNNN XXXX YYYY

The word count and disk address of the core load named NNNNN which is referenced within the skeleton are assigned to locations YYYY and YYYY+1 of the skeleton PN Table. The first PNT entry is always the System Director where NNNNN = SYDIR and XXXX is the initial core location of the ICL Table pointer block which is identified in the System Director listing by the symbolic name -- COMA. The XXXX field is blank for all remaining PNT entries.

**In-core-with-Skeleton Interrupt (ICI).**

ICI NNNNN XXXX LLBB

The entry point to the in-core-with-Skeleton interrupt servicing routine named NNNNN is at absolute location XXXX. LLBB designate the interrupt level and bit position within the ILSW for that associated level.

If LL is less than the number of interrupt levels (K) defined for the system, NNNNN is assigned to service the process interrupt on PISW bit position BB of level LL.

If LL = K or LL = K+1, NNNNN is assigned to service a programmed interrupt on level BB or BB+14 respectively. Note that BB is now used as a level designation.

If LL = K+2 or LL = K+3, NNNNN has been designated as count subroutine BB or BB+16 respectively. Note that BB is now used as a count subroutine number in the range 0-31.

For an ICI assigned to level 0 on PISW bit position 0, ARG2 will be printed as a blank field.

**Library Function Subroutines (LIBF).**

LIBF NNNNN XXXX YYYY

The LIBF-type subroutine entry point named NNNNN is at absolute location XXXX of the skeleton. The corresponding three-word transfer vector entry point will be at location YYYY in variable core.

**CALL-type Subroutines (CALL).**

CALL NNNNN XXXX YYYY

The CALL-type subroutine entry point named NNNNN is at absolute location XXXX of the skeleton. The indirect entry point is at location YYYY of the Skeleton Executive Branch Table (EBT).

**INSKEL COMMON (INSK).**

INSK XXXX YYYY

The low core storage boundary of INSKEL COMMON is at absolute location XXXX of the skeleton. The high boundary is at location YYYY.

**Patch Area (PTCH).**

PTCH XXXX YYYY

The patch area (that is, unused core locations) extends from the absolute location, XXXX, of the skeleton through location YYYY.

**COMMON (COMM).**

COMM XXXX YYYY

If any included subroutines contain references to COMMON, allocation is made between absolute locations XXXX and YYYY, using the standard method. It should be noted that these locations are in variable core; allowances must therefore be made in all core loads for overlapping results.

**Interrupt Core Load Table (ICLT) Map**

The ICL Table map is printed to reveal any interrupt assignments made in the skeleton ICLT. Its format is as follows:

ICL TABLE MAP
LLBB WC/EP SA ICLT (Column Heading)

The interrupt level and bit assignment are indicated by a four-digit hexadecimal number under LLBB. The two high-order digits contain the level; the two low-order digits represent the bit assignment.
If the entry is an in-core-with-Skeleton routine, the WC/EP column will contain the hexadecimal entry point to this routine. The SA field will be blank. The ICL Table absolute core location in which the entry point is placed is indicated in the ICLT column.

When rebuilding the skeleton with the SAVE ICL TABLE option, word counts and sector addresses of any interrupt core loads are retained from the old ICLT. Their interrupt assignments are indicated in the LLBB column. The WC/EP and SA columns will contain their word counts and disk addresses. The corresponding ICLT absolute core location is in the ICLT column.
To broaden the scope of this manual, and to facilitate its use by individuals of divergent backgrounds and experience, selected material emphasizing recommended practice and technique in the implementation of the IBM 1800 Time Sharing Executive System are presented in this section. This material directly supplements the concepts discussed so far in the text.

A separate chapter (Basic Concepts of Data Acquisition and Process Control Systems) is included for a two-fold purpose: (1) as an introduction to the TSX Sample System, and (2) to acquaint the inexperienced reader with the field of data acquisition and process control systems. It is not intended as an exhaustive study, and the reader is referred to further sources of information on this vast subject. The more experienced reader may prefer to scan this portion of the section as refresher material, or to skip it entirely.

The final chapter, TSX Sample System, is a comprehensive step-by-step example of a working TSX on-line system which touches on every aspect of TSX system concepts, design, and usage.

WRITING ASSEMBLER LANGUAGE SUBROUTINES

This chapter provides guidance to the user in the assembly and specification of user-written subroutines included either as additions or modifications to the TSX system. User-written assembler language subroutines must follow the writing specifications outlined below.

The subroutine source statements shown in the following examples should be preceded and followed by the following control cards for the assembly process.

```
// JOB
// ASM
*LIST
*PRINT SYMBOL TABLE
  :
Subroutine Source Deck
  :
// DUP
*STORE  NAME
```

Call Subroutines

At execution time, a CALL statement has become a long insert and branch instruction to the called subroutine.

If the called subroutine is in variable core, the generated instruction points directly to the subroutine.

If the called subroutine is in the skeleton the generated instruction is indirect, pointing to the subroutine via its entry in the Executive Branch Table (EBT). However, if the CALL is also from the skeleton, then a direct insert and branch instruction is generated.

For example, a FORTRAN source statement

```
call sub(I, J, K, 101)
```

or an Assembler language calling sequence

```
call sub
   dc addr1
   dc addrj
   dc addrk
   dc addcon
```

appears in core at execution time as

```
bsi l sub
   dc addr1
   dc addrj
   dc addrk
   dc addcon
```

if SUB is in variable core, and as

```
bsi i addrh
   dc addr1
   dc addrj
   dc addrk
   dc addcon
```

if SUB is in the skeleton, where ADDRH is the executive Branch Table entry for SUB, ADDRI, ADDRJ, and ADDRK are the core addresses at which the variable I, J, and K are stored, and ADCON is the core address where the constant, 101, is stored.
Note that most subroutines entered by an Assembler language calling sequence expect the constants themselves to appear in the calling sequence rather than the address of the constants. Therefore, not all subroutines entered by a CALL can be called from a FORTRAN program.

The following example illustrates how to define the entry point, save the contents of the registers, get the parameters, and return to the calling program. Three parameters are assumed to be in the parameter list, and return to the calling program. Therefore, not all subroutines entered by a CALL can be called from a FORTRAN program.

LIBF SUBROUTINES

The source statements for subroutines that are called by a LIBF statement must be preceded by a LIBR statement.

At execution time, the LIBF call has become an insert and branch instruction indexed by index register 3 (BSI 3 (DISP.,)). A displacement is generated that will point to the subroutines transfer vector entry during execution, when index register 3 will point to the table itself.

If the LIBF call is to a subroutine in the skeleton, the transfer vector entry is located in the Executive Transfer Vector Table (ETV) and contains an insert and branch direct instruction (BSI l,...) pointing to the subroutine via its entry in the Executive Branch Table (EBT).

If the LIBF call is to a subroutine in variable core, the transfer vector entry is located in the Variable Transfer Vector Table (VTV) and contains an insert and branch direct instruction (BSI l,...) pointing directly to the subroutine.

The following example illustrates a LIBF subroutine and shows how to define the entry point, save the machine status, get the address of the parameter list, and return to the calling program.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Contents &amp; Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>SUBROUTINE NAME</td>
<td>AT ENTRY POINT</td>
</tr>
<tr>
<td>J</td>
<td>SUBROUTINE NAME</td>
<td>AT ENTRY POINT</td>
</tr>
<tr>
<td>K</td>
<td>SUBROUTINE NAME</td>
<td>AT ENTRY POINT</td>
</tr>
<tr>
<td>L</td>
<td>SUBROUTINE NAME</td>
<td>AT ENTRY POINT</td>
</tr>
</tbody>
</table>

INPUT/OUTPUT SUBROUTINES

The procedures for writing input/output subroutines are similar to those for CALL or LIBF subroutines, except that an ISS statement is used to define the entry of the call section of the routine; also, the interrupt entry points must be defined.

The basic identification for the interrupt entry portion is the IAC code. There is a unique IAC code for each ILSW bit that is turned on by an I/O interrupt. At system generation time, the user defines the IAC codes and their corresponding ILSW bit. The same IAC code must be used when writing an I/O subroutine.

As stated previously, an ISS statement is used to define the call entry point (only one call entry point is permitted). If the subroutine is to be called by a LIBF statement, the ISS statement must be preceded by a LIBR statement. The LIBR statement is omitted if the subroutine is to be called by using a CALL statement (the CALL statement method must be used if the subroutine is to be called from a FORTRAN program). Following the ISS statement, there must be a pair of DC statements for each interrupt entry point. The first DC statement must define the IAC code for that entry, and the second DC must define the address of the interrupt entry point. This is followed by an ORG *-X where X is the number of DC statements.

The following is an example of how a typical ISS subroutine is written.
PROGRAM LISTING NO. 7: SAMPLE CARD I/O ROUTINE

HONG SAMPLE CARD I/O ROUTINE  CRD00000

******************************************************************************

CRD00010

******************************************************************************

CRD00030

* THIS SUBROUTINE IS A SAMPLE OF THE TSX I/O ROUTINES, IT IS CALLED VIA LIBF, THE CALLING

SEQUENCE IS

*  LIBF  CARD
*  DC /XOXY CONTROL PARAMETER
*  DC AREA I/O AREA ADDR

* CONTROL PARAMETER IS COMPRISED OF 4 HEX DIGITS

OF WHICH ONLY X AND Y ARE USED.

* X EQUAL 0 TEST FUNCTION—DO NOT

INCLUDE AREA PARA

* X EQUAL 1 READ FUNCTION

* X EQUAL 2 PUNCH FUNCTION

* Y EQUAL 0 USE FIRST 1442

* Y EQUAL 1 USE SECOND 1442

* X 173 ENTRY POINT TO TVEXT

* NOTE THAT ON A PUNCH FUNCTION IT IS ASSUMED

THAT THE USER HAS THE END BIT SET IN THE

LAST WORD TO BE PUNCHES.

* PROGRAMMING TECHNIQUES 183

***********************************************************************

0000 03059100

ISS 2 CARD

SIGNIFIES THIS IS A LIBF

IS THE NO. OF INT

ENTRY POINTS

AND CARD IS LIBF

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

FRST ATR

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT

ENTRY POINT
0020 01 4C04001F  BSC L READY,E  READY CONDITION
0022 0 C000  LD 0 READY CONDITION ON
0023 0 0204  *  STD X2 BUSY  TO INDICATE DEVICE BEING USED
0024 0 C100  *  LD X1 0  GET FUNCTION PARAMETER
0025 0 180C  *  SKA 12 TEST FOR READ OR PUNCH
0026 01 4C040030  BSC L READ,E  BRANCH IF FUNCTION IS READ
0028 0 C101  *  PUNCH LD X1 1  GET SECOND PARAMETER
0029 0 0202  STD X2 PH  PUT AREA PARAM IN PH IOCC
002A 0 0A02  *  XIO X2 PH  START PUNCHING A CARD
002B 00 74010007  MDX L 7+1 INCREMENT GEN 1/0 BUSY IND
002C 0 1000  NUP
002D 0 7101  MDX 1 1 INCREMENT RET ADDR
002F 0 7007  MDX OUT  GO TO EXIT
0030 0 C101  *  READ LD X1 1  GET SECOND PARAMETER
0031 0 D200  STD X2 RD  PUT AREA ADDR IN RD IOCC
0032 0 0A00  XIO X2 RD  START READING A CARD
0033 00 74010007  MDX L 7+1 INCREMENT GEN 1/0 BUSY IND
0035 0 1000  NUP
0036 0 7101  MDX 1 1 INCREMENT RET ADDR
0037 0 7101  *  SETUP TO RETURN TO USER
0039 00 60000337  *  STX L 55 55 NOW CONTAINS THE RETURN ADDRESS
003A 00 448000AD  *  BS1 I 173  RETURN TO USER VIA TVEXT
003C 06 600005A  INT1 LDX L2 C01  XR2#1442-1, THIS IS
003E 0 7002  MDX ++2  INTERRUPT ENTRY POINT FOR
003F 06 6000062  INT2 LDX L2 C02  XR2#1442-2, THIS IS
0041 0 0A06  XIO X2 SENS  SENSE DSW WITH RESET
0042 0 1002  STD L 2  TEST FOR ERROR
0043 01 4C100051  *  BSC L DONE,=  BRANCH IF NO ERROR
0045 0 3000  *  WAIT  WAIT FOR USER TO RELOAD CD
0046 0 0A04  REDOP X2 SENSE  L0OP UNTIL 1442 IS IN A
0047 06 4C040046  BSC L READX,E  READY CONDITION
0049 0 C200  LD X2 RD  TEST FOR LAST FUn A RD
004A 0 1C100046  BSC L REDOP,=  BRANCH IF NOT READ
004C 0 0400  XIO X2 RD  REDO READ
004D 0 7001  MDX +=1  EXIT TO MIC
004E 0 0A02  REDOP XIO X2 PH  REDO PUNCH
004F 00 4C80005A  BSC I 90  RETURN TO MIC
0051 0 1010  DONE SLA 16  CLEAR ALL INDICATORS
0052 0 D200  STD X2 RD  CLEAR READ IOCC
0053 0 D202  STD X2 PH  CLEAR PUNCH IOCC
0054 0 D204  STD X2 BUSY  CLEAR BUSY INDICATOR
0055 00 740F0007  MDX L 7+1 DECREMENT GEN 1/0 BUSY IND
0057 0 1000  NUP
0058 06 4C80005A  BSC I 90  RETURN TO MIC
005A 0000  *  DEVICE TABLES FOR 1442-1 AND 1442-2
005B 0000  *  NOTE THAT THE RD IN COLUMN 71 IS THE
005C 0000  *  DISPLACEMENT OF THAT WORD FROM THE START
005D 0000  *  OF THE DEVICE TABLE
005E 0000  *  BS5 E 0  DEVICE TABLE MUST BEGIN ON
005F 0000  *  AN EVEN ADDR BECAUSE OF
0060 0000  BSC 0  IOCC.
0061 0000  CD1 DC 0  DEVICE TABLE FOR 1442-1 0
0062 01600  DC /1600 READ IOCC
0063 0000  DC 0  1 CRD01520
0065 0 1600  DC 0  2 CRD01530
0067 0 1500  DC /1500 PUNCH IOCC
0069 0 0000  DC 0  3 CRD01540
006B 0 1700  DC /1700 SENSE IOCC
006D 01000  DC 0  4 CRD01550
006F 0 1701  DC /1701 SENSE/RESET IOCC
0071 0 1701  0  5 CRD01560
0073 0000  UC 0  6 CRD01570
0075 0000  UC 0  7 CRD01580
0077 0800  GC /8000 READ IOCC
0079 0000  GC 0  1 CRD01610
007B 0000  GC 0  2 CRD01620
007D 0 8000  GC /8000 PUNCH IOCC
007F 0 0000  GC 0  3 CRD01630
0081 0 8000  GC 0  4 CRD01640
0083 0800  GC /8000 SENSE IOCC
0085 0000  GC 0  5 CRD01650
0087 0800  GC /8F00 SENSE/RESET IOCC
0089 0000  GC 0  6 CRD01660
008B 0800  GC /8F01 SENSE/RESET IOCC
008D 0000  GC 0  7 CRD01670
008F 0000  GC 0
0091 0000  LEQ 0  READ IOCC
0093 0200  PH 2  PUNCH IOCC
0095 0400  BUSY EQ 4  BUSY INDICATOR
0097 0400  SENSE EQ 4  SENSE IOCC
0099 0600  SENS 6  SENSE/RESET IOCC
009B 0000  END

184
PROGRAMMING SUBROUTINES USING REENTRANT CODING

NEED FOR REENTRANT CODING

One of the basic problems that arises in multi-level programming is requirement of the same subroutine by different levels of operation.

For example, the computer is servicing a main-line program which is executing a square-root subroutine when an external interrupt occurs. The hardware interrupt will automatically branch to an address which will allow servicing of the interrupt.

The program that services the interrupt may also require use of a square-root subroutine. If a method of reentrant coding were not used, the identical square-root subroutine would have to be in core storage twice (once for each program that called it); otherwise, the intermediate results which are needed when the computer returns to complete the mainline program would be destroyed by the interrupt program.

CONCEPT OF LEVEL WORK AREAS

To allow one subroutine to be entered at any time and from any interrupt level, without loss of intermediate results, a method of reentrant coding using level work areas is used.

Reentrant coding is defined as coding which allows a program to be entered and executed from different levels without destroying the intermediate results.

The IBM 1800 TSX System provides features which facilitate the coding of reentrant subroutines.

Each interrupt level specified by the user is provided with a level work area of 104 locations, which are reserved for the exclusive use of programs operating on that priority level.

The first 62 of these locations are reserved for specific routines (MIC, QZSAV, etc.) while the remaining 42 locations are available to allow other subroutines (arithmetic, functional, etc.), to maintain their ability to reenter.

The start address of the level work area for any priority level always appears in location LWA (fixed location 10410 = 6816). If an index register is loaded with the contents of this location, and all references to temporary storage locations are indexed, 42 temporary storage locations are made available to the subroutine for each level it may be operating on.

If the subroutine is reentered, different effective addresses are generated for each such indexed operand, and the reentry problem is solved.

The following sequence of instructions illustrates how the contents of the A-register are saved in TEMP in the level work area and later restored by the instruction at LOAD:

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Code</th>
<th>Offset</th>
<th>Operand &amp; format</th>
</tr>
</thead>
<tbody>
<tr>
<td>LWA</td>
<td>EQU</td>
<td>104</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEMP</td>
<td>EQU</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LWA, WORK AREA ADDR TO LWA</td>
</tr>
<tr>
<td>STRE</td>
<td>STO</td>
<td>1</td>
<td></td>
<td>SAVE A-REGISTER IN TEMP</td>
</tr>
<tr>
<td>LOAD</td>
<td>STO</td>
<td>1</td>
<td>TEMP</td>
<td>RESTORE A-REGISTER</td>
</tr>
</tbody>
</table>

Should the subroutine be interrupted and re-entered, there will be no storage conflicts, since the contents of LWA changes with each interrupt level. Hence, the instructions at STRE and LOAD reference different effective addresses for each interrupt level.

MECHANISM FOR REENTRANT CONTROL

For each interrupt serviced, MIC (Master Interrupt Control program) saves and subsequently restores the contents of the A- and Q-registers, index registers, machine status, and locations WK4 (5410 = 3616) and WK5 (5510 = 3716). MIC also sets LWA to the correct level work area address for each interrupt level.

Since locations WK4 and WK5 are saved by MIC for each interrupt level, these locations may also be used for temporary storage by reentrant subroutines, e.g., loading and storing of index registers. Furthermore, these locations are also used for other purposes, as explained below.

Protecting Entry and Return Addresses

The first location of a callable subroutine is set by a BS1 instruction. As with all fixed locations upon re-entry, this location may be changed and the return address may be lost. The TSX System supplies two pairs of subroutines which provide a method of protecting the return address. They also perform several additional functions useful for subroutines.
Subroutines Referenced by a CALL Instruction

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Operands</th>
<th>Call to Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>QZSAV</td>
<td>EQU</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>QZEXT</td>
<td>EQU</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>SUBRT</td>
<td>BC</td>
<td>R</td>
<td>ENTRY TO CALL ROUTINE</td>
</tr>
<tr>
<td>EXIT</td>
<td>BST</td>
<td>R, R</td>
<td>EXIT FROM SUBROUTINE</td>
</tr>
</tbody>
</table>

The QZSAV subroutine saves the contents of index registers 1, 2, and 3, the A- and Q-registers and machine status and places the return address in location WK4 \( (5\times10 = 36_{16}) \). In addition, index registers 1 and 3 are set to the first location of the level work area, and index register 2 is set to \( 127_{10} = 7F_{16} \).

The QZEXT subroutine restores the index registers, machine status, and A- and Q-registers and returns control to the calling routine via a BSC I WK4. The address set in WK4 by QZSAV must, therefore, be incremented by 1 for every parameter following the CALL.

Subroutines Referenced by a LIBF Instruction

For subroutines referenced by a LIBF (1-word BSI) instruction through the transfer vector:

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Operands</th>
<th>Call to Subroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVSAV</td>
<td>EQU</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>TVEXT</td>
<td>EQU</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>SUBRT</td>
<td>BC</td>
<td>R</td>
<td>ENTRY TO LIBF ROUTINE</td>
</tr>
<tr>
<td>EXIT</td>
<td>BST</td>
<td>R, R</td>
<td>EXIT FROM SUBROUTINE</td>
</tr>
</tbody>
</table>

The TVSAV subroutine saves the contents of index registers 1 and 2, the A- and Q-registers, and machine status and places the return address in WK5 \( (55_{10} = 37_{16}) \). In addition, index registers 1 and 3 are set to the first location of the level work area, and index register 2 is set to \( 127_{10} = 7F_{16} \).

The TVEXT subroutine restores the contents of index registers 1 and 2, A- and Q-registers, and machine status. Index register 3 is set to the transfer vector location, and control is returned to the calling routine via a BSC I WK5. The address set in WK5 by TVSAV must, therefore, be incremented by 1 for every parameter following the LIBF.

Other Considerations

It should be understood that the use of QZSAV or TVSAV does not obviate careful logic control. If parameters follow the call to the subroutine, it is the responsibility of the subroutine to obtain these parameters and to adjust WK4 or WK5.

If subroutines are nested, that is, one subroutine calls another, care must be exercised to save and restore the storage locations used by QZSAV and TVSAV across the nested call, as well as the return address in WK4 or WK5. Furthermore, nested subroutines must be planned so that the same locations in the level work area are not used by more than one subroutine.

Note that TVSAV, TVEXT, QZSAV, and QZEXT are referenced by indirect BSI instructions and not by CALL statements. The call to TVSAV or QZSAV must be the first instruction executed in (and immediately following) the entry location, as illustrated.

MASKING OUT THE INTERRUPTS

Another method of providing reentrant coding is to prevent the interrupts from being recognized.

In Assembler language, it is possible to use the XIO command with the IOCC-masking words provided by the TSS system. To mask all interrupt levels completely, the following instructions may be executed:

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK1</td>
<td>EQU</td>
<td>5A</td>
</tr>
<tr>
<td>MSK2</td>
<td>EQU</td>
<td>5B</td>
</tr>
<tr>
<td>XIO</td>
<td>L</td>
<td>MSK1, MSK2</td>
</tr>
<tr>
<td>XIO</td>
<td>L</td>
<td>MSK1, MSK2, STATUS</td>
</tr>
</tbody>
</table>

To restore the interrupt mask status, the following instructions may be executed:

<table>
<thead>
<tr>
<th>Label</th>
<th>Operation</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSK3</td>
<td>EQU</td>
<td>5A</td>
</tr>
<tr>
<td>MSK4</td>
<td>EQU</td>
<td>4B</td>
</tr>
<tr>
<td>XIO</td>
<td>L</td>
<td>MSK3, MSK4</td>
</tr>
<tr>
<td>XIO</td>
<td>L</td>
<td>MSK3, R30</td>
</tr>
</tbody>
</table>
interrupt system is defeated. However, for short sequences of instructions, the method of masking out the interrupts may be the fastest means of obtaining reentrant coding.

PROGRAMMING NOTES

The following examples illustrate the different approaches that may be used to write reentrant subroutines, and to explain the need for WK4 and WK5 (words 54 and 55 of the Fixed Area in core storage). These words are saved by MIC in the same manner that the accumulator and certain index registers are preserved during the handling of an interrupt.

Both examples depict a method of storing what is contained at the effective address reached by an index register plus its displacement, loading that value into another index register, and to be reentrant.

<table>
<thead>
<tr>
<th>Label</th>
<th>Operator</th>
<th>Ope.</th>
<th>Mem.</th>
<th>Operands &amp; Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>M. NON-REENTRANT CSE.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONST EQU 53</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD 1 CONST</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE 12</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X+1</td>
<td></td>
<td>X+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X-1</td>
<td></td>
<td>X-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X+3</td>
<td></td>
<td>X+3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X+5</td>
<td></td>
<td>X+5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X-5</td>
<td></td>
<td>X-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X-7</td>
<td></td>
<td>X-7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X+7</td>
<td></td>
<td>X+7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X-9</td>
<td></td>
<td>X-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X+9</td>
<td></td>
<td>X+9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X-11</td>
<td></td>
<td>X-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOAD X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORE X+11</td>
<td></td>
<td>X+11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WRITING USER-PROGRAMS FOR EXECUTION UNDER THE TASK ABSOLUTE LOADER

The TASK Absolute Loader can be used to load programs from cards to core for execution under TASK or for the storing on disk of user-written programs or data. The TSX system must be in the off-line (nonprocess) mode. To call the absolute loader, load or restart TASK and set sense switch 0 on.

PROGRAM/DATA FORMAT

The User programs and data must be assembled absolute and orgined above the last address of TASK. The object deck must be of the relocatable format type, i.e., compressor output, not core image. The execution address on the end-of-program card must be the address of the first user instruction if the program is to be executed, and must be the address of the program word count if the program is to be stored on disk. No LIBFs or CALLs are allowed in the program, but any TASK subroutine (DISKN, TYPEN, etc.) can be called as defined in TASK I/O Subroutines. The TASK object program set is an example of programs designed to run under the absolute loader.

NOTE: When executing user-written absolute programs under TASK, it is best to use an off-line cartridge, or a cartridge that does not contain the TSX system, so that TSX system areas which are not file protected are not destroyed. For example, TASK uses sectors 05C0 and up for buffering of 1053 messages. The upper limit of this area was established by the user at TASK assembly time.

The source deck format for executable programs is

```
ABS ORG ABC
START LD ABC (see NOTE below)
.
.
User's Program
.
.
END START
```

The source deck format for data or programs stored on disk is

```
ABS ORG ABC
ABC is a core address above the last TASK address (see NOTE below)
```
SECAD SECAD is the first sector address where the program is to be stored.

NOTE: To insure that the program is always above the end of TASK, let ABC be greater than or equal to /F1F0. The highest address of the program must be less than or equal to /FFFF.

It is the user's responsibility to recall the data or program from the disk.

ABSOLUTE LOADER OPERATION

When TASK has been reloaded or restarted, the following message is printed.

TASK 1800 TSX
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER

1. Set Sense switch 0 on.
2. Place the program to be loaded in the card read punch hopper. A stacked input is allowable.
3. Press reader START.
4. The user now has the option of selecting manual or automatic mode before pressing Console START.

Manual Mode

If data switch 15 is off, the absolute loader operates in the manual mode. After the program has been loaded to core, the following message is printed.

DATA SW 0 ON LD DISK OFF EXECUTE

Set data switch 0 off and press console START to execute the program just loaded. Set data switch 0 on and press Console START to write the program or data to disk. If the program or data is written to disk, the absolute loader starts reading the next program in the card reader into core after performing the disk write function.

Automatic Mode

If data switch 15 is on, the absolute loader operates in the automatic mode. After loading the user's program, the absolute loader executes it unless the control card illustrated below has been placed in front of the user's program. The format of the control card is:

column 1 = 8
column 2 = +
column 4 = 9
column 13 = 1
all other columns = blank

The user can assemble this control card in his object deck by placing the two source cards shown below right after the ABS card in his source deck.

ORG 40
DC 1

If the program is loaded in automatic mode and the control card has been included, the program will be stored on disk and the next program in the reader will be loaded.

The following (Program Listing No. 8) is an example of the loading and execution of an absolute 80-80 program using the TASK Absolute Loader. Note that the program is assigned for the upper 4K of core. The ABS card is used to indicate that this is an absolute assembly. The start address is present in the END statement.
PROGRAM LISTING NO. 8: TASK ABSOLUTE PROGRAM FOR DOING AN 80-80 LIST OF CARDS ON THE
LIST PRINTER

// JOB 
// ASK LIST
/*LIST*/
/*PUNCH*/
/*PRINT SYMBOL TABLE*/

ABS

******************************************************************************
* TASK ABSOLUTE PROGRAM FOR DOING AN 80-80 
* LIST OF CARDS ON THE LIST 
* PRINTER. 
******************************************************************************

0000 ORG /FIFO PLACE IN UPPER 4K
0070 LSTPT EQU 125 LIST PRINTER ENTRY POINT
0030 CARDN EQU 61 CARDN ENTRY POINT
003A HOLEB EQU 58 HOLEB ENTRY POINT
0038 EBPRT EQU 59 EBPRT ENTRY POINT
00A6 SLOGR EQU 166 LIST PT DEVICE TYPE IND
0020 CIND EQU 32 CD IND ON INT LEV WK AREA
0068 $TVK EQU 104 INT LEV WK AREA LOC ADDR

/*
FIFO 00 6780006A START LOX I3 STVXK TELL CARON NOT TO CHECK
FIFA 00 0000 LD X2 /* FOR // CARDS
FIFA 00 0320 STO X3 CIND
FIFA 00 C0000A6 LD L SLOGR WHAT DEVICE IS LIST PRINTER
FIFA 00 4C08F1FB BSC L LOOP,+ BRANCH IF 1053
FIFA 00 4680007D BSI 1 LSTPT SKIP TO CHANNEL 1
FIFA 0 3100 DC /3100
FIFA 00 4680003D LOOP BSI 1 CARON READ A CARD
FIFA 0 1000 DC /1000
FIFA 0 F291 DC IAREA
FIFA 0 0000 DC 0
F200 00 4680003D BSI 1 CARON LOOP BUSY
F202 00 0000 DC 0
F203 00 70FC MUX =-4
F204 00 4680003A BSI 1 HOLEB CONVERT TO EBC CODE
F205 00 0000 DC 0
F207 0 F292 DC IAREA+1
F208 0 F292 DC IAREA+1
F209 0 0050 DC 80
F20A 00 C0000A6 LD L SLOGR
F20C 00 420F2F4F BSC L P1443.Z BRANCH IF 1443
F20E 00 4680007D P1053 BSI 1 LSTPT LOOP BUSY
F210 00 0000 DC 0
F211 00 70FC MUX =-4
F212 00 4680007D BSI 1 LSTPT RETURN CARRIER
F214 0 0000 DC /2000
F215 00 F28F DC RETUR
F216 00 0000 DC 0
F217 00 6114 LOX 1 20 SET UP TO SUPPRES TRAILING
F218 0 6947 STX 1 0ARE A SET UP WDC
F219 00 C50F2F25 LP1 LD L IARE+A20 BLANKS=TEST FOR BLANK
F21A 0 0004 EDM H4040
F21C 00 420F2A2A BSC L N1,Z BRANCH IF NOT BLANK
F21E 0 071F MUX 1 -1 DECREMENT WDC
F21F 0 0000 EDM 1 LP1 TEST NEXT CHARACTER
F220 0 6954 STX 1 IAREA+21 SET UP ZERO WDC
F221 00 6114 LOX 1 20 TEST FIRST HALF OF CARD
F222 00 C60F2F91 LP2 LD L IAREA TEST FOR BLANK
F224 00 F03A EDM H4040
F225 00 420F2F2C BSC L N2.Z BRANCH NOT BLANK
F227 00 71FF MUX 1 -1
F228 00 70F9 MUX LP2 TEST NEXT CHARACTER
F229 00 70D1 MUX LOOP ALL OF CARD IS BLANK
F22A 00 6944 NZ1 STX 1 IAREA+21
F22B 00 6933 NZ2 STX 1 IAREA
F22D 00 44800038 BSI 1 EBRPT SET UP FIRST HALF OF MESS TO BE PRINTED
F22F 00 0003 DC 1
F230 00 0000 DC 0
F231 0 F292 DC IAREA+1
F232 00 0028 DC =-4
F233 00 4680007D BSI 1 LSTPT LOOP BUSY
F235 00 0000 DC 0
F236 00 70FC MUX =-4
F237 00 4680007D BSI 1 LSTPT PRINT FIRST HALF OF CARD
F239 0 0000 DC /2000
F23A 00 0000 DC QAREA
F23B 00 0000 DC 0
F23C 00 420F2F9B BSC L LOOP,+ BRANCH IF BLANK
F23D 00 4680003B BSI 1 EBPRT SET UP SECOND HALF OF CARD
F241 00 0001 DC 1 TO BE PRINTED
F242 00 0001 DC IAREA+21
F243 00 0000 DC OAREA+22
F244 00 0028 DC =-4

Programming Techniques 189
F245 00 4480007D BSI I LSTPT LOOP BUSY
F247 0 0000 DC 0
F248 0 70FC MDX =-4
F249 00 4480007D BSI I LSTPT PRINT SECOND OF CARD
F24B 0 2000 DC /2000
F24C 0 F275 DC DAREA+21
F24D 0 0000 DC 0
F24E 0 70AC MDX LOOP GO READ NEXT CARD
F24F 00 4480007D BSI I LSTPT LOOP BUFFER BUSY
F251 0 0010 DC /0010
F252 0 70FC MDX =-4
F253 00 4480003B BSI I EBPRT SET UP CARD TO BE PRINTED
F255 0 0001 DC 1
F256 0 F292 DC IAREA+1
F257 0 F266 DC DAREA+6
F258 0 0050 DC 80
F259 00 4480007D BSI I LSTPT PRINT THE CARD
F25B 0 2100 DC /2100
F25C 0 F260 DC DAREA
F25D 0 0000 DC 0
F25E 0 70AC MDX LOOP GO READ NEXT CARD

* TABLE AREA

F25F 0 4040 DC /4040 DC BLANK
F260 0 0200 DAREA DC 45 OUTPUT BUFFER
F261 0 0000 DC 0
F262 0 0000 DC 0
F263 0 0000 DC 0
F264 0 0000 DC 0
F265 0 0000 DC 0
F266 000F BSS 15
F275 0 0014 DC 20
F276 0019 BSS 25
F28F 0 0001 RETUR DC 1 RETURN CARRIER MESSAGE
F290 0 8121 DC /8121
F291 0 0050 IAREA DC 80 INPUT BUFFER
F292 0 0050 BSS 80
F2E2 FIFO END START

* SYMBOL TABLE

<table>
<thead>
<tr>
<th>CARDN 0030</th>
<th>CIND 0020</th>
<th>EBPRT 003B</th>
<th>WOLEB 003A</th>
<th>4040 F25F</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAREA F291</td>
<td>LOOP F1FB</td>
<td>LPL F219</td>
<td>LP2 F222</td>
<td>LSTPT 007D</td>
</tr>
<tr>
<td>NZ1 F22A</td>
<td>NZ2 F22C</td>
<td>DAREA F260</td>
<td>P1053 F20E</td>
<td>P1443 F24F</td>
</tr>
<tr>
<td>RETUR F28F</td>
<td>$LORG 0046</td>
<td>$TVWK 0068</td>
<td>START FIFO</td>
<td></td>
</tr>
</tbody>
</table>

NO ERRORS IN ABOVE ASSEMBLY.
LIST
DUP FUNCTION COMPLETED
BASIC CONCEPTS OF DATA ACQUISITION AND PROCESS CONTROL SYSTEMS (DACS)

INTRODUCTION

Data Acquisition and Process Control Systems are by definition real-time systems. In real-time processing, inputs may arrive randomly from the process being monitored to the computer which rapidly responds to each input, usually by transmitting an output back to the process. This is in contrast to conventional batch processing where groups of inputs are processed by passes through the computer. The notion of real-time usually implies that a computer is responding to inputs as they occur in the physical world.

The principal functions of a real-time data acquisition and process control computer system are: data scanning, data logging, process calculations, process outputs, input-output control, operator-machine communication, and specialized system monitoring. Each of these functions is implemented by one or more programs permanently stored in core or in secondary storage.

In general, data logging/data acquisition applications are basically monitor systems where the data signal traffic is toward the computer process I/O. Control systems (supervisory, operator-guide, or direct digital control), on the other hand, are characterized by a two-way signal flow across the process/computer interface. For the most part, data acquisition implies that the process operates in its normal manner without being affected by the computer, whereas computer control systems imply that the process is directly controlled by actions and commands of the computer.

Implementation of data acquisition and process control systems in real-time requires certain process I/O hardware which significantly affects software requirements. Most important of these are the analog input and output hardware which require relay and possibly solid-state multiplexing devices together with a multi-level priority interrupt system.

Analog Process I/O

By the very nature of processes, many internal process signals are analog -- that is, continuous functions with respect to time. Many instruments have been developed to pick-off various signals of interest and to modify them (that is, amplify, filter, linearize, etc.) in order to provide data or control signals. Analog-to-digital converters (ADC) are used to convert the resultant analog signals for computer entry. The ADC and amplifiers in a DACS may also operate directly on the output of a signal transducer.

Transducers have been designed around physical laws to convert one form of energy to another. Transducers used with data acquisition and process control systems usually convert the various physical quantities to be measured to their voltage analogs. For example, transducers are available to convert pressure to voltage and temperature to voltage. The quality of the transducer is determined by the accuracy, repeatability, linearity, and proportional range of the parameter to voltage conversion. In almost every process, temperature is one of the variables of interest, and in many cases, thermocouples are used to measure temperature. Strain gauges may be used to measure pressure, deformation, and loading. Thus, depending upon the size and package, one could use a strain gauge to measure direct blood pressure or to measure stresses set up in a missile structural member when the missile engine is ignited. Similarly, a thermocouple could be used to measure a blast furnace process temperature or air temperature.

To convert the transducer-produced voltage to digital values, an analog-to-digital converter (ADC) is used. Since an ADC can operate at a high rate compared to the rate of change of the individual signal voltages and because ADC units are relatively expensive, it is customary to time-share the ADC among a number of input voltages through an analog multiplexer.

An analog multiplexer is a device which switches the various analog inputs to the ADC. Most multiplexers can be programmed for sequential scan where each input channel is in turn scanned and connected for conversion to the ADC. Random scan sequences can also be programmed. Random scanning, as the name implies, permits any arbitrary sequence of input channels to be scanned and connected in turn to the ADC.

In summary, the signal flow in the analog process I/O can be explained as follows. Multiple parameters are continuously converted to voltages by appropriate transducers and signal conditioning electronics. The resultant voltages form inputs to a multiplexer. Under command of the processor-controller, multiplex switches are closed to allow an analog input signal to be amplified (in most cases) by a time-shared amplifier. The ADC connected to the amplifier output does the actual voltage to digital conversion, with the resultant digital value being inputted to the computer. In this manner, the signals
of interest in the process are sampled, measured and entered into the computer for additional processing.

Data flow from the computer to the process follows a converse procedure. Digital data is entered into multiple registers by the computer. Each register is connected to a digital-to-analog converter (DAC). A DAC is basically a digitally-controlled voltage or current source whose output is the voltage analog of the digital value. The analog outputs in turn control various set-points and other control points in the system. In some systems, a single DAC is connected to analog memory devices via an analog multiplexer. In this way, a single DAC can be used to provide many output signals.

By using digital-to-analog converters and analog-to-digital converters, a digital computer can be made to communicate with process signals and control equipment. It should be noted that the digital process I/O and the analog process I/O are all under program control.

Digital Process I/O

In addition to the analog parameters which must be monitored and controlled, there are other process signals which are binary in nature. Thus, for example, relays can be open or closed, a voltage level may or may not be present, or a parameter may be represented as a sequence of pulses. The computer must be able to accept this data which is essentially binary form. Similarly, the computer must be capable of producing binary outputs to control devices such as relays. The subsystem in a DACS, which handles these types of signals, is generally referred to as digital process I/O.

Two of the process I/O functions provided on the IBM 1800 Data Acquisition and Control System are contact sense and voltage sense. The output of the contact sense circuit signifies when a contact closure has occurred on the input. The output of the voltage sense circuit indicates when an input signal has exceeded a preset level. The output of each circuit is connected to a single bit in a register. The processor-controller can scan these registers at rates from 100,000 up to 500,000 words (groups) per second.

Closely allied with these digital input features is process interrupt. Process interrupt is a vital feature for real-time control because the computer is basically a sequential machine. When a significant process event such as an alarm occurs, as indicated by a relay closure or voltage level, a signal is transmitted to the computer as an interrupt requiring a special subroutine to take appropriate action.

Interrupts are usually assigned in order of priority, so that if two occur at once, the more important is serviced first by the computer. In essence, process interrupt is merely a special form of voltage/contact sense. The feature of customer-assignable multi-priority interrupts clearly differentiates a process control computer from the normal data processing system.

Pulse inputs form another category of digital inputs. A typical process source is the turbine flow meter which generates pulses at varying rates from a few pulses per second up to several thousand pulses per second. Electronic counters are used to accumulate the pulses; the response to an interrupt signal on counter overflow or a periodic scanning of the counter register then accomplishes transfer of the accumulated count data into the computer. High speed pulse counters and scalers for megacycle rates are also adaptable for high-speed data acquisition applications.

Similarly, some control devices in a typical process environment require input data in the form of pulses. Several types of output are available, one form being pulse output. The primary purpose of this output is to provide for pulse trains to operate such devices as latches, set point indicators, and other stepping motor devices.

One form of output which is opposite to the contact sense feature is Electronic Contact Operate (ECO). Many of the required control and display operations in a typical process application involve binary action; that is, a piece of equipment is turned on or off, or a valve is open or shut completely. This capability is provided by ECO.

To increase the versatility of the data acquisition and process control system, an output feature is provided whereby a digital word can be transferred to an external piece of equipment. The register output feature provides this capability on the 1800. Examples of its use might include transfer of data to another computer or to a display device.

Process-input-data flow to the computer is illustrated in Figure 89, as accomplished on an IBM 1800 Data Acquisition and Control System, together with corresponding process output features. Digital data may be read in under direct program control, as the program executes read-in instructions, or additional channel controls can be employed to sequence data to previously assigned areas of core storage without disrupting normal program operation. This is known as cycle-steal. When data is ready for entry, a cycle-steal operation is initiated by the channel control circuitry. One memory cycle of computer operation is used to read-in the data word directly to the assigned core storage location.
DATA ACQUISITION SYSTEMS

Data acquisition/data logging systems are generally the simplest form of DACS. When used in a data acquisition role, the computer controls the real-time collection of data from the process. Many data acquisition systems operate on analog signals recorded on magnetic tape rather than in real-time. However, the problems involved in both situations are similar, except of course, in real-time only one opportunity to capture the data exists. To overcome this, an analog recording is sometimes made simultaneously for backup.

In data acquisition/data logging applications, the data flows from the process to the computer system (or analog magnetic tape). A minimum number of signals flow from the computer to the process. Therefore, the computer will normally have minimum control over the process. The data signals from the process are connected to the process input computer circuits and/or to the analog tape recorder. Thus, the data acquisition system monitors and records what goes on in the process without affecting the process functions in any manner.

When the analog tape is played back, the signals appear to the DACS computer exactly as real-time signals and can be handled by the computer in the same manner. By playing back the analog tape several times, a more selective retrieval of data is possible. If a DACS malfunction occurs in real-time, the data can still be retrieved by replaying the backup analog tape.

There are many reasons and applications for computer-assisted data acquisition. The following list is given to denote the almost universal applicability of data acquisition:

1. Record keeping: Many applications and industries must keep expensive records of processes. For example, airlines are required by law to record aircraft flight performance.
2. Telemetry is basically data acquisition and data logging from a remote or inaccessible process.
3. Data editing and data reduction applications involve processing large quantities of data which in many cases are redundant or non-significant. The computer program is called upon to reduce this data to manageable proportions.
Many processes have been designed on an empirical basis. Data logging and data acquisition is used to collect sufficient data to more fully understand the process. This data can be for process optimization and for model building.

By continuously monitoring parameters within a process, alarm conditions can be detected. Each value is continuously compared against preset conditions. Whenever a parameter is not within limits or the status does not check as expected, an appropriate alert can be given.

By carrying the limit violation check procedure further, trend predictions can be made. Thus, when rate of change of parameters exceeds prescribed limits, remedial action can be taken.

Important application areas are quality control and industrial testing. By using computer-aided test stands, it is possible to do 100% testing on production line components and produce printed records of the test results. For many military specification components, 100% testing is mandatory. Furthermore, the statistical data accumulated is necessary for various quality control work and product assurance. Computer-assisted testing makes the procedure economically practical. The degree of confidence that a customer has in a product is enhanced when he can obtain a printed record of the product's acceptance test results.

OPERATOR GUIDE/SUPERVISORY CONTROL

Almost all Operator Guide/Supervisory Control (OG/SC) systems embody data acquisition principles. In order to control a process, one must first measure and monitor it. The data acquisition feature is the monitor portion of OG/SC. Thus, data acquisition is the first step to control. When starting a control project, one must obtain sufficient data about the process through a planned data acquisition system. The data acquisition finally becomes the process I/O of the control system. OG/SC systems are characterized by a two-way data flow between the process and the computer. The computer monitors the process and based upon these measurements determines what control is necessary to make the system function in accordance with the program. To a first approximation, the difference between Operator Guide Control and Supervisory Control is one of how the computer interfaces with the process. Operator Guide Control usually implies the presence of a human operator who receives instructions from the control processor on an output display device, such as a typewriter. The operator makes adjustments and sets controls on set-point controllers based upon the control processor originated instructions.

Supervisory control situations allow for direct control of the process by the computer via hardware electrical connection to the set-point controllers. The computer can make adjustments and sets control without operator intervention. Both categories rely upon direct computer data acquisition.

Most conventional processes rely upon analog controllers to hold various process variables within set points. Processes are usually made up of a number of cascaded operations with each step controlled by an analog controller. To adjust or optimize the process for particular modes of operation, each controller is set through its set-point control. The set-point may be reset during the process to compensate for various process perturbations.

When a computer controls the process, various modes of control are feasible. For example, in start-up control, the computer will first issue instructions to set the controllers to various set-points depending upon the process. The computer continually monitors the set-points on the controllers and the various controlled variables. Subsequent steps in the starting sequence are not permitted to commence until all values are within their prescribed tolerances. This type of control assures a reliable repeatable checkout and start-up procedure. Full documentation of all controls and variables are an implied by-product. Furthermore, by extending this concept, a standard operating procedure for the process is feasible.

Process optimization generally requires a mathematical model which has been defined on the basis of theory and/or data obtained using data logging techniques, etc. Successful optimization results in maximizing or minimizing parameters deemed important by management.

An attractive feature of the OG/SC type of computer control is that the process equipment is least disturbed. The only effect on the analog controllers is that they must be provided with remote set-point capability. Reversion to manual control is easily accomplished because the analog controllers are still in place. This provides an important backup capability for keeping the process running during a computer shutdown. In addition, the minor changes to the process instrumentation is an economic advantage.

Computer control can thus be installed in most existing processes without the necessity for building from the ground up.
DIRECT DIGITAL CONTROL

Direct Digital Control (DDC) is another advanced form of computer process control. In this mode of operation, the process will probably not run without the computer. The computer controls the process directly without the intervention of analog controllers. Computer-generated signals are used to control the process directly.

A DDC application generally requires re-instrumentation on existing processes. The proponents of DDC have advanced many arguments for this method of control. The marriage of the process to the computer provides many potential control techniques, not otherwise available, because of the tight control that is feasible.

A major advantage of DDC is the ease in which a parameter may be instrumented and controlled. A minor hardware addition in the process coupled with another program loop is all that is needed to control an additional variable. On the other hand, to add an analog controller is a major construction task compared with its DDC equivalent. To a large extent, the number of process loops that can be controlled is limited only by processor-controller and process I/O capabilities.

Analog controllers may be manually adjusted for best control for any given set of plant operating conditions. If these conditions change, analog controllers must be manually readjusted to retain optimum control. Readjustment is so time-consuming that it is usually impractical. In DDC, to change a critical time constant or to add a derivative function entails the addition of a few punched cards rather than major hardware. The effective characteristics of a controller are thus changed by inserting new cards into the deck. The flexibility and ease with which DDC can be altered is not possible with analog controllers.

Further Reading

For the benefit of inexperienced readers, the following IBM publications are recommended for further reading:

Principles of Data Acquisition Systems,
Form E20-0090

Programming for Computer Control of a Cement
Klin, Form Z20-1753

High-Speed Data Acquisition in Low Energy Physics,
Form E20-0171

The IBM 1800 Data Acquisition and Control System
for Gas Turbine Engine Testing – Developmental,
Form H20-0121

The IBM 1800 Data Acquisition and Control System
for Gas Turbine Engine Testing – Production,
Form H20-0123

Computer Control of the Continuous Hot-Dip Galvanizing Process, Form E20-0178

1800 Traffic Control System – Application
Description, Form H20-0212

1800 Process Supervisory Program (PROSPRO/1800) – Application Description, Form H20-0261

TSX SAMPLE SYSTEM

The TSX Sample System was conceived and developed for two primary reasons. First, it demonstrates in a step-by-step fashion how to generate a TSX system from start to finish. Second, it illustrates some of the ways in which TSX can be used to actually control a process. The sample system is a working operating system which has proven itself in continuous service. For practical purposes, and in the interest of simplicity, an IBM 1800 Data Acquisition and Control System is linked to a Process Simulator to reproduce an actual continuous process which closely approximates a paper machine used in the manufacture of paper of diverse grades.

The general objectives of the system are to give better quality and quantity control. Its design basis included four specific criteria:

1. Provide periodic logs of all important variables within the system.
2. Establish closed-loop control over eight process variables.
3. Incorporate process operator control over forty process variables.
4. Display up-to-the minute management information about the process.

The sample system demonstrates how a number of functions may be achieved by using TSX, with relatively little effort on the user’s part. The features of special significance in this example are:

- the scheduling of periodic, semi-periodic, synchronous and asynchronous programs
- closed-loop control
• process job scheduling
• error recovery procedures
• system design to achieve maximum utilization of the time-sharing mode

SYSTEM DESIGN

The first thing that must be done in designing a control system is the definition of the process to be controlled; the second is the definition of the hardware (that is, the computer and its associated instrumentation); and thirdly, the design of the programming software. In this example, the process is referred to in terms of a paper machine, but those readers who are familiar with the paper industry will immediately realize that a paper manufacturing system in its full detail is quite complex, and for this reason, many of the paper machine control functions normally encountered are intentionally omitted. Among these are the handling of paper breaks and the use of instruments for on-line measurements of consistency, basis weight, and moisture content.

The paper process is designed to manufacture small amounts of paper of high quality grades. Orders arrive at the plant for varying quantities of paper of different paper grades. For example, orders may differ with respect to dimension, basis weight, strength properties, color, chemical additives, etc. The machine must be able under continuous operation to switch from one grade of paper (when the order quantity for that paper has been completed) to the next grade of paper. This means that grade change must be made as quickly and efficiently as possible to avoid undue and costly waste. Also, during the manufacture of a given grade of paper, very tight control over the process is necessary to ensure quality that is consistent with past orders for this paper. The control variables must be kept at setpoint, and any undue variation in other variables within the system must be recognized and the operator informed.

Input to the system must be provided for the updating of job files on the disk so that the computer knows continually what grades of paper are to be manufactured and in what sequence. Data on grade and quantity of paper arrives from the order department on data cards (see Figure 90). These cards are read and placed in files, on the disk, by the nonprocess time-shared portion of the operating system; the process portion then goes through this information sequentially.

Provision must be made for a process operator, through the medium of a console, to be able to obtain and/or update information about the entire process system at any time. This may include the changing of a setpoint on a setpoint station or to have instantly all available information on a given variable, such as its current value, its high and low limits, and conversion factors. The process operator...
must also be able to modify the run-time of a par­
ticular grade of paper, and to change the sequence of
process jobs that reside on the disk.

The 1800 hardware configuration for this system
consists of the following:

- A two-microsecond 1801 Processor-Controller,
  with 32,768 words of core storage
- A 2310 Disk Storage unit, with two disk drives
- A 1443 Printer
- A 1442 Card Read/Punch Unit
- Three 1053 Printers

The 1443 Printer and 1442 Card Read/Punch,
along with one of the disk drives are used for non­
process work. This means that the card reader, the
printer, and one disk drive constitute a basic stand­
alone monitor system, while the process portion
consists of the remaining disk drive, three 1053s,
and process I/O.

PERIODIC PROGRAM SCHEDULER

Certain periodic functions must be handled within the
system. First, every 20 seconds the closed-loop
control program must be entered for a scan of the
setpoint stations to see if they are on setpoint and,
if they are not, to set up for the return of the set­
point positioners to their correct operating points.
Second, every two minutes a scan of all operator
guide points must be performed for limit checking,
and any out-of-limit violations printed so that the
process operator may bring these operating points
within limits. Third, every quarter hour on the
hour, a log of all variables within the system must
be printed. Fourth, every hour, on the hour, a
summary log of the previous hour's production must
be printed for the attention of the process operator,
the supervisor, and the foreman. Fifth, at the end
of every shift (12:15 a.m., 8:15 a.m., and
4:15 p.m.) a shift-in log is printed which repre­
sents a summary of the past shift's production.
Sixth, every Monday morning at 8:30, a weekly sum­
mary log is printed for close scrutiny by the super­
visor and his foreman.

The first two of these functions are called periodic
asynchronous functions because, while they are per­
iodic in nature (that is, they occur every 20 seconds
and every 2 minutes, respectively), they do not have
to be in synchronization with real-time. The other
functions are periodic synchronous since they are
periodic and must be in synchronization with real­
time, i.e., every hour, on the hour. To schedule
the programs properly, a scheduler CALL COUNT
subroutine is written in FORTRAN. The time-base
method is used to determine when a periodic,
synchronous program is next executed. In this way,
a time is recorded for a given program that is
chosen for execution. When the real-time clock is
greater or equal to the time base specified, that
program is scheduled for execution, i.e., it is
queued, an end-time-sharing command is given, and
the base-time is incremented by the period of the
program.

SAMPLE SYSTEM ERROR DESIGN

Error procedures constitute one of the most impor­
tant parts of any system. In the sample system,
error procedures can be broken down into two
phases, (1) checkpoint operation and, (2) system
operation with certain I/O devices down or off-line.

Checkpoint Operation

The basic philosophy of checkpoint operation is that
all system data needed by the computer to control
the process must be stored in a location where it will
not be destroyed. Thus, if normal operating data is
destroyed, the data last saved can be retrieved and
utilized. In the sample system, this is handled by
having, on disk, two files that are used to record
the process variables and certain conversion factors.
One of these files is a static file, the other being
dynamic in nature. Whenever a variable is modi­
fied in the system, one of the two files is updated
with the new information. For example, if the oper­
ator at the process operator's console changes the
setpoint of one of the setpoint stations, this informa­
tion is recorded both in INSHEL COMMON (working
data) and in a file on disk. If, at a later date,
something should happen that would cause either a
load or a restart of the system, all operational
data, such as setpoint values, limits on operator
guide points, conversion factors, switches, can be
read from disk to reinitialize INSHEL COMMON to
the most updated valid values.

FILE1, the static file, contains all conversion
factors used in the system. This file is normally
read into core at cold start time to update INSHEL
COMMON with the necessary matrices containing
conversion factors for setpoint and operator guide
analog inputs. The only time INSHEL COMMON
and FILE1 are modified is when the calibration programs, which run in a time-sharing mode, are executed. At that time, the conversion factors of selected points are updated in INSKEL COMMON and the values are updated in FILE1.

The second file in the system for checkpoint operation, the dynamic file, is labelled FILE3 and contains all of the dynamic variables within the system. These comprise the job number, the day, the time when the present grade of paper is completed, the setpoint for all of the closed loops, and the limits for all points under operator guide control. This file is updated whenever one of the dynamic variables is modified, such as at the start of a new grade of paper or the entry of data into the system from the process operator's console.

The system contains both a restart core load and a reload core load. The restart core load is referenced by all but the C.E. interrupt mainline. Whenever EAC initiates a restart, this core load is brought into core storage. The system then reads the static and dynamic files from disk to initialize INSKEL COMMON, makes sure that both the scheduler and the end-of-grade call count subroutines are functioning correctly, and checks to see if an end-of-grade has occurred during the restart procedure. If so, it calls CHAIN to the mainline core load, GRADE. Otherwise, it calls VIAQ, and the system has restarted properly.

The reload core load is similar to the cold start core load, except there is a decision point within the core load to determine whether a reload or a cold start condition has occurred. Normally, sense switch 6 is turned on at cold start time. The cold start core load interrogates sense switch 6, and if the switch is on, the program goes through its normal cold start procedure. If sense switch 6 is off, the cold start routine assumes a reload has occurred and reads both static and dynamic files, starts the scheduler running again, and performs other functions that are similar in scope to those that occur in the restart core load. The normal cold start procedure, therefore, is to set sense switch 6 on and, when the system is up and running, turn that switch off, so that if a reload occurs, the system recovers correctly.

I/O Error Procedures

The system is designed so that it can function with only one of the two disk drives in operation. Thus, with a hardware failure on either disk drive, the system is able to run at full capacity on the process. This requires a two-disk system in which all process-oriented functions, such as cold start, the resident skeleton, system save areas, and all of the process core loads, are on logical drive 1, and all nonprocess functions, such as DUP, FORTRAN Compiler, Assembler, are on logical drive zero. Cold start is to logical drive 1; since all of the process is recorded on logical drive 1, there should never be a need to reference logical drive zero, unless time-sharing occurs.

In order to guarantee that the system does not initiate time-sharing unless an operator is present to specifically instruct it to do so, the cold start core load (COLDS) sets the CALL VIAQ time-sharing period to zero. This means that logical drive zero will be referenced only if the console interrupt button is depressed with sense switch 7 on. A cold start to logical drive 1 may, therefore, be performed without any fear of not having logical drive zero operational. Thus, if one of the two physical drives goes down, the system is still running. When the drive is serviced, it can be restored on-line with the C.E. interrupt routine and time-sharing started. This allows for the C.E. to work on the drive while the system is still controlling the process. The system also continues to operate if any two of the three 1053 Printers are down. This is achieved by setting the backup pattern for the 1053 Printers in cyclic order, such that

1053 Printer No. 2 backs-up 1053 Printer No. 1
1053 Printer No. 3 backs-up 1053 Printer No. 2
1053 Printer No. 1 backs-up 1053 Printer No. 3

This means that if one 1053 Printer goes down, it is backed-up, and if either of the other two printers goes down, the third printer backs-up the first two, so that as long as one 1053 Printer is operational, the system is still running.

Of the two other data processing I/O devices on the system, the 1443 Printer is used only on the time-sharing side of the system. If it therefore goes down, it will not affect the process side; in this event, there will be no printout of new job data loaded to the disk. The 1442 Card Read Punch is used by the process only at cold start time; if the system is up and running when the 1442 device goes down, the system is still able to control the process. The process can thus still be controlled by the 1800 hardware if, in the worst case, at cold start time, one of the two disk drives, two of the three 1053s, and the 1443 are all down.
To summarize, all that is needed for a cold start is a 1442 Card Read Punch, a 1053 Printer, and a 2310 Disk Storage Unit with two disk drives. After the system is placed on-line and becomes operational, only one 1053 Printer and one disk drive are required. The way the system is designed, it functions even if the final 1053 goes down, but with one restriction: no operator messages are printed. Closed-loop control is still in force, and the user is still able to enter information through the process operator's console, but he is not able to obtain information on the grade of paper being produced, and any out-of-limit violation that may occur.

The EAC printer for the system is defined as all three 1053s. This ensures that if an error condition does arise, the operator on at least one of the three 1053 printer stations is informed of the problem. To enable use of the C.E. interrupt routines (that form part of TSX) to place a disk drive, a 1443 Printer, and a 1053 Printer on-line or off-line, a C.E. core load is provided. This core load is called whenever it is necessary to use the C.E. interrupt routine. A special C.E. core load is needed within the system because the C.E. interrupt cannot be masked. This means that when the C.E. interrupt button is depressed, no matter what the masked status of the 1800 is, the interrupt will occur. It also signifies that the C.E. routine has no choice but to assume that the whole system is masked at the time of the interrupt, and it therefore returns the machine in this status. The interrupt mainline core load must therefore unmask back to the desired configuration at the time of the C.E. interrupt. In order to assure that this occurs, a special core load is called into core before the C.E. interrupt button is depressed. The core load types out a message indicating that the operator can now press the C.E. interrupt button; it then performs a pause. The C.E. interrupt causes the mainline program to drop through the pause. When the C.E. interrupt routine is complete, the mainline will then restore the user's mask and exit by a CALL VIAQ.

Since the system has been designed for error recovery, restarts and reloads can be performed without loss of process control. While the system is on-line, the C.E. is able to carry out both preventive and corrective maintenance on the 1443 Printer and on two of the 1053 Printers. Once these devices become serviceable, they are restored to on-line duty by use of the TSX C.E. interrupt routine, and the system continues operation in the normal mode. The C.E. Aux Storage routines are also used while the system is on-line.

CLOSED LOOP CONTROL

It is desired to have the sample system perform closed loop control on eight setpoint stations. The scan requirement demands that these eight stations be interrogated every 20 seconds to ensure they are operating at the right setpoint value. A decision is now made whether the closed loop control program is to be a core load or an INSKEL interrupt subroutine. If the closed loop control program is to be a core load, it would require core exchanges every 20 seconds in order for it to operate as specified. This would be very inefficient in that the large number of core exchanges would cut down the useful amount of processor-controller time. Also, time-sharing periods would be of such small magnitudes that a nonprocess job would take a great amount of time to execute. For these reasons, the closed loop control program is written in FORTRAN as an INSKEL CALL LEVEL interrupt subroutine; that is, it resides in permanent core.

The setpoint station couples the 1800 hardware to an analog control loop; the station receives information from the computer in the form of a pulse train, and transmits information back to the computer in the form of an electrical signal so that this may be read by analog input. This signal indicates the position of the setpoint. The output from the setpoint station to the process can be used to operate a pneumatic control valve directly or through a pneumatic relay. The setpoint station has a scale range of 0 to 100, divided into 2,000 increments. One pulse output to the setpoint station drives the setpoint positioner one increment. The pulses must be spaced at least 15 milliseconds apart in order to drive the positioner at an effective rate. The sample system is programmed to anticipate setpoint positioner drift. Thus, the control loop more closely resembles direct digital control (DDC) than normal setpoint station operation.

The closed loop control program is divided into two sections. The scan section scans all eight setpoint stations and computes the number of pulses needed for each one; the output section outputs the pulses. A CALL LEVEL (10) command is given to enter the program and, depending on a switch setting, one of the two sections is executed. Every 20 seconds the scheduler sets up an entry into the scan section. Once the scan section has completed its task, it initiates the output section.

The switch for the program is now set so that future entries will enter the output section of the
closed loop program before another 20-second period has elapsed. This is necessary because the program must not stay on the interrupt level of the closed loop control program during the output of the pulse chain needed to move the setpoint station, which would tie up the system too long. For example, if 200 increments are put out to the setpoint station, the necessary loop would take 3 seconds for execution. Therefore, once the output section has outputed a pulse to the setpoint station, it calls for timer B to set up reentry to itself in 15 milliseconds. It then exits from the interrupt level so that computation or execution on the mainline level can continue. Fifteen milliseconds later, the timer runs out and the subroutine associated with the timer executes Call LEVEL back to the closed loop control program. At this point, if another pulse is needed for any of the setpoint stations, it is outputted and a Call TIMER is again given. If all of the desired pulses have been given, no Call TIMER is made and the program exits. The closed loop control program will not be entered again until 20 seconds have elapsed from the last entry to the scan section.

This method of control provides background and foreground operations with the TSX system, i.e., multi-programming. The closed loop control program constitutes the foreground job, and the program resident in variable core, the background job.

OPERATOR GUIDE CONTROL

One of the system design criteria is to incorporate operator guide control over forty process variables. At the start of a grade, the operator is informed of the limits on the 40 variables he is responsible for controlling. Every two minutes these variables are scanned, and if an out-of-limit condition occurs on any of the variables, this information is printed for the operator. Also, the operator has the ability to perform a two-minute scan on demand so that he can quickly scan all points, when desired. At the same time, the operator can call for total information on any one point under control, i.e., its present value, its high and low limits, and conversion factors. The operator also has the option of changing the limits of a point or taking a point off operator guide control. To take a point off-line, he sets to their maximum and minimum values, all of the limits checked by the two-minute scan. In this manner, he is continuously informed of any erroneous out-of-limit condition.

SYSTEM DESIGN FOR OPTIMUM TIME-SHARING

Another desirable system requirement is to make optimum use of the nonprocess mode of operation on a time-shared basis. Among the activities that the user may wish to implement in a time-shared mode are instrument calibration, updating of process job files on disk, accounting, payroll, etc. As it is assumed that the user is going to make very heavy use of the time-shared mode, a 1443 Printer is included in the system. The system and list printers for the Nonprocess Monitor are both defined as the 1443 Printer so that all normal nonprocess output would be on that printer. With the design of the system such that only one disk is required for process work, logical drive zero can be switched among different nonprocess monitor disk cartridges. This allows each department within the company to have its own nonprocess monitor disk cartridge, with its own set of programs. When a department job is ready for processing, and computer time is available, all that is required to be done is to place the disk cartridge on disk drive zero and execute. With the 1442/1443 combination, the nonprocess user appears to have a small independent computer for his own, i.e., a computer with a card read punch, a printer, and one disk. A data processing computer is thus combined with a strictly control systems computer in one machine and under one operating system. In fact, if the user desired, he could well set up a closed shop, stack-job environment where nonprocess jobs submitted by the various departments within the company are run while the computer is controlling or monitoring the process.

PROCESS OPERATOR'S CONSOLE

One of the basic requirements of any process control system is a simple and efficient method to input data from a process operator. The operator must also be able to interrogate the process, with relative ease, to obtain any information required. There is no single standard approach to this problem, since the nature of the information needed by any one industry and, in fact, any plant within an industry, varies greatly. In the sample system, a portion of the 1800 Computer Process Simulator is used for the operator's console (see Figure 91). Also, one of the 1053s works in conjunction with the console. This means that the process operator has the console,
Setpoint Station Movement Indicators
Digital Data Display (nixi tubes)
Setpoint Stations for Loops 7 & 8
Function Buttons

Digital Input Switches
Operator Data Entry Dials
Analog Inputs Under Operator Guide (8 of the 40 points)

Figure 91. 1800 Computer Process Simulator
along with a 1053, at his work station. Standard process I/O is used to operate the process operator’s console; analog input is used to enter information from the operator’s data entry dials; process interrupts are used as function buttons; digital input is used to define whether a certain closed loop is under control; digital output is used to indicate the direction of movement of a setpoint positioner when it goes through a control movement; and pulse output is used to count the number of increments given to all setpoint stations at any time.

**Data Entry Dials**

In normal practice the process plant operator has a definite need to enter digital data into the system. This may take the form of a new set of limits for an operator guide point, a new value for a setpoint station, or a new time for a particular grade of paper to have its production terminated. In order to accomplish this on the sample system, sixteen data entry dials are provided. Each data entry dial consists of a linear potentiometer connected to an analog input point. To obtain fine adjustment, a vernier scale is incorporated with each potentiometer. A major dial is graduated in scale from 0 to 10, while a minor dial (the short scale of the vernier) gives increments of subdivisions of each division recorded. Each data dial enters one digital number into the system. The process operator sets the potentiometer at one of the major divisions, i.e., 1, 2, ... 9, and that digital integer is entered into the system. The digital integer entered is translated by the potentiometer to give an analog value between 0 and -32,000. This analog value is then converted by the data entry dial read routine to a floating point number between 0.0 and 9.9; this number is finally transformed and truncated to an integer value that lies within the range of 0 through 9. The data entry dial routine then transmits to the calling program a 16-element linear matrix containing the digital values for all 16 data entry dials. The calling routine can combine these integer values to produce the desired digit-coded information.

**Function Buttons**

Each function button is connected to one process interrupt. The operator depresses a function button depending upon the function required from the console. An example might be an operator call for demand scan of operator guide points; all that is required of the operator is to depress the proper button. This causes a process interrupt; the two minute normal scan program is then loaded to core and executed. Some of the process operator's console programs are also executed via the data entry dials. The operator sets the required data entry dials, and pushes the associated function button. The proper program is loaded; it interrogates the data dials; and then performs the operation requested.

**Digital Input Switches**

These 16 switches are connected to one digital input word. In the sample system, eight of these switches are used to define whether a given closed-loop is on-line or off-line. Before the system outputs a pulse to a setpoint station or scans the closed-loop control point, it first executes a read-and-expand function on this digital input word. It then interrogates each bit read to determine whether that loop is on-line or off-line. If the loop is off-line, no control function is performed.

**Digital Data Display**

The data output side of the process operator’s console is divided into three sections. The first section is a set of nixi tubes for displaying digital data. These are used as a counter of the number of pulses given to the setpoint stations. The counter is manually reset from the operator's console. In a normal operating system, there is very little need for such a device, but it was found desirable to have it for experimental purposes on this system.

**Setpoint Station Movement Indicators**

This section consists of 16 lights, two for each setpoint station. One of the pair of lights per loop indicates whether the setpoint positioner is moving up, while the other light indicates the setpoint positioner is moving down. The lights are operated by digital output.

**IBM 1053 Printer**

This is used for outputting data, such as periodic logs, that the user requests. The process operator's console is simple to use and provides an efficient solution for man/machine interface problems on the system.
SYSTEM DOCUMENTATION

One of the most important and the most often neglected aspects of control systems is proper system documentation. The following paragraphs explain the need for this vital part of planning and installation.

Aid in System Debugging

A control system application is not solely a conglomeration of multiple programs working independently of one another. It is, rather, one big program that has been highly segmented. In order to insure correct communications between all program segments, a condensed picture of the system is needed, to show the interconnections between all program segments, their interactions, what common variables they reference, and how these programs are sequentially or randomly executed.

Basic System Documentation Needed

Documentation that can be maintained is the key to understanding and success of the system. The first supporting procedural device is an overall system flow chart that illustrates the basic interactions between programs (see Figure 92). The second is the maintenance of a list of all tables and variables that are referenced by more than one program (see Tables 14 and 15). Information is required to show what programs reference these system variables and which programs modify them. The third item is a specification sheet describing each log in the system (see Table 16). The fourth item is a specification sheet on each program, that briefly tells what the program does, what variables are referenced, what files are referenced, what programs call this program, what other programs this program calls, and any other information that has to do with the integration of this program into the system (see Table 17).

DESCRIPTION OF SAMPLE SYSTEM FLOWCHART

The sample system flowchart (see Figure 92) illustrates overall system operation and points out sequential, random, and time-based operations in a single flowchart. The flowchart is concerned only with the relationship between various programs within the system, not with the happenings within a program. It describes how programs are initiated and how they exit. Note that some program blocks are self-contained; that is, they neither cause another program to be initiated nor are they initiated by another program. They are triggered only by a random event, such as a process interrupt, and, once executed, perform only an exit. On the sample system flowchart, both chaining and queueing from one program to another are represented in the same manner because they constitute, basically, the same function. The main difference between them is that a CALL QUEUE may allow other programs to "sneak in" between the sequential execution of the calling program and the called program. Note also that two programs, SCHED and TCONT, call themselves with a time delay and thus insure that they are periodically executed. The TCONT program may have its chain of execution broken by the program TEBRT. This means that TCONT is periodic only for a certain duration of time and that the operator may control that duration of time if he so desires (see Table 17: Program Data Sheets).

CODING TECHNIQUES

The sample system depicts the employment of several coding techniques which may be of value to the user. Three techniques are discussed in this section. The first two are examples of good programming coding practice and should help to eliminate the type of system problems a user may encounter at TSX system installation time. The third coding technique illustrates how one of the functions on analog input can be used to advantage.

Use of INSKEL COMMON

INSKEL COMMON is one of the principal means by which coreloads and subroutines within the system can communicate with one another. This unique labelled common area is mapped each time a new core load is built. This means that problems can arise if two different core loads have INSKEL COMMON mapped differently. In order to avoid such conflicts in its use, all core loads must map INSKEL COMMON in the same manner. Also, for good coding practice, they should use the same variable names for every variable in INSKEL COMMON. The only way to insure this common usage is for the user to keypunch only one COMMON card which may subsequently be duplicated and inserted into each core load source deck. If at some later time a change is made to INSKEL COMMON, the new COMMON card may be keypunched, copies made, and all old COMMON cards in source decks replaced. In this way, the user ensures that INSKEL COMMON is defined identically for all core loads.
Figure 92. TSX Sample System Flow Chart
Table 14. TSX Sample System Table of Variables

<table>
<thead>
<tr>
<th>System Variable</th>
<th>Used by</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW0</td>
<td>SCHED, SOUT, RSTAR, COLDP, GRADE</td>
<td>Close Loop Control Stop</td>
</tr>
<tr>
<td>SW1</td>
<td>SCHED, COLDP</td>
<td>Update day-of-week switch</td>
</tr>
<tr>
<td>SW2</td>
<td>SCHED</td>
<td>Output Monday morning log switch</td>
</tr>
<tr>
<td>SW3</td>
<td>SCHED, ENDGD, RSTAR, COLDP, GRADE, MGRTP, CPJSP</td>
<td>Process Control Stop Switch</td>
</tr>
<tr>
<td>SW4</td>
<td>SCHED, LEV10, SOUT</td>
<td>LEV10 program section switch</td>
</tr>
<tr>
<td>SW5</td>
<td>SCHED, COLDP</td>
<td>SCAN2 Execution switch</td>
</tr>
<tr>
<td>DAY</td>
<td>SCHED, RSTAR, COLDP, GRADE, SCAN2, LOG15, LOG60, SHIFT, COGLP, CCLSP, MGRTP, CPJSP, AIMON</td>
<td>Day of week</td>
</tr>
<tr>
<td>JOBN</td>
<td>RSTAR, COLDP, GRADE, MGRTP, CPJSP, SPECL</td>
<td>Next process job</td>
</tr>
<tr>
<td>VALUE</td>
<td>LEV10</td>
<td>LEV10 Data input area</td>
</tr>
<tr>
<td>RANGE</td>
<td>COLDN, RSTAR, COLDP, GRADE, LOG15, TREND, CCLSP, AIMON, SPECL, SCALB, RCALB</td>
<td>Range of analog input values for setpoint stations</td>
</tr>
<tr>
<td>LOW</td>
<td>COLDN, RSTAR, COLDP, GRADE, LOG15, TREND, CCLSP, AIMON, SPECL, SCALB, RCALB</td>
<td>Low analog input values of setpoint stations</td>
</tr>
<tr>
<td>SETPT</td>
<td>LEV10, RSTAR, COLDP, GRADE, CCLSP, AIMON, SPECL</td>
<td>Operating points for setpoint stations</td>
</tr>
<tr>
<td>COUNT</td>
<td>LEV10</td>
<td>Number of pulses to be given to setpoint stations</td>
</tr>
<tr>
<td>OFLNL</td>
<td>LEV10</td>
<td>Off-line indicator for setpoint stations</td>
</tr>
<tr>
<td>AHL</td>
<td>RSTAR, COLDP, GRADE, LIMIT, COGLP, SPECL</td>
<td>High limits for op-guide points</td>
</tr>
<tr>
<td>ALL</td>
<td>RSTAR, COLDP, GRADE, LIMIT, COGLP, SPECL</td>
<td>Low limits for op-guide points</td>
</tr>
<tr>
<td>A</td>
<td>COLDN, RSTAR, COLDP, LOG15, TREND, AIMON, SPECL, SCALB, RCALB</td>
<td>Conversion factor A for op-guide points</td>
</tr>
<tr>
<td>B</td>
<td>COLDN, RSTAR, COLDP, LOG15, TREND, AIMON, SPECL, SCALB, RCALB</td>
<td>Conversion factor B for op-guide points</td>
</tr>
<tr>
<td>IBASE</td>
<td>SCHED, COLDP</td>
<td>Base time for LOG15</td>
</tr>
<tr>
<td>IBASZ</td>
<td>SCHED, COLDP</td>
<td>Base time for LOG60</td>
</tr>
<tr>
<td>IBAZZ</td>
<td>SCHED, COLDP</td>
<td>Base time for SHIFT</td>
</tr>
<tr>
<td>G</td>
<td>CONVR, COLDN, COLDP, CMIPT</td>
<td>Conversion factor G for data entry dials</td>
</tr>
<tr>
<td>H</td>
<td>CONVR, COLDN, COLDP, CMIPT</td>
<td>Conversion factor H for data entry dials</td>
</tr>
<tr>
<td>IENDT</td>
<td>RSTAR, COLDP, GRADE, MGRTP, CPJSP</td>
<td>Grade termination time</td>
</tr>
<tr>
<td>IPERD</td>
<td>STRND, TCONT</td>
<td>TREND execution period</td>
</tr>
<tr>
<td>ITCNT</td>
<td>TCONT, TABRT, STRND</td>
<td>Number of times TREND is to be executed</td>
</tr>
<tr>
<td>IPONT</td>
<td>TREND, STRND</td>
<td>Point or Loop for TREND to log</td>
</tr>
</tbody>
</table>

Table 15. Disk File Organization

<table>
<thead>
<tr>
<th>FILE</th>
<th>RECORD</th>
<th>VARIABLES</th>
<th>REFERENCING PROGRAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>RANGE, LOW, A, B</td>
<td>COLDN, RSTAR, COLDP, SPECL, SCALB, RCALB</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>G, H</td>
<td>COLDN, COLDP</td>
</tr>
<tr>
<td>2</td>
<td>1-100</td>
<td>I, ITIME, SETPT, AHL, ALL</td>
<td>GRADE, SPECL, CMIPT</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>JOBN, DAY, IENDT, SW0</td>
<td>RSTAR, COLDP, GRADE, MGRTP, CPJSP</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>SETPT</td>
<td>RSTAR, COLDP, GRADE, CCLSP</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>AHL, ALL</td>
<td>RSTAR, COLDP, GRADE, COGLP</td>
</tr>
</tbody>
</table>
Table 16. Log Description

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>INFORMATION</th>
<th>RECIPIENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCANZ</td>
<td>Periodic/Asynchronous</td>
<td>Out of Limit Conditions</td>
<td>Process Operator</td>
</tr>
<tr>
<td></td>
<td>(2 minute period)</td>
<td>on operator guide points</td>
<td>Process Operator, Foreman</td>
</tr>
<tr>
<td>LOG15</td>
<td>Periodic/Synchronous</td>
<td>Present value of all analog input</td>
<td>Process Operator, Foreman,</td>
</tr>
<tr>
<td></td>
<td>(15 minute period on</td>
<td>points</td>
<td>Supervisor</td>
</tr>
<tr>
<td></td>
<td>the quarter hour)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOG60</td>
<td>Periodic/Synchronous</td>
<td>Summary of last hour's production</td>
<td>Process Operator, Foreman,</td>
</tr>
<tr>
<td></td>
<td>(1 hour period on the</td>
<td></td>
<td>Supervisor</td>
</tr>
<tr>
<td></td>
<td>hour)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHIFT</td>
<td>Periodic/Synchronous</td>
<td>Summary of last shift's production</td>
<td>Process Operator, Foreman,</td>
</tr>
<tr>
<td></td>
<td>(8 hour period at 0:15,</td>
<td></td>
<td>Supervisor</td>
</tr>
<tr>
<td></td>
<td>8:15, and 16:15)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WEEK</td>
<td>Periodic/Synchronous</td>
<td>Summary of last week's production</td>
<td>Foreman, Supervisor</td>
</tr>
<tr>
<td></td>
<td>(every week at 8:30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Monday morning)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use of FORTRAN Files

FORTRAN files form the second means of communications among programs in TSX. The use of FORTRAN files, when compared to INSKEL COMMON, introduces an added complication. Not only are the files mapped at core load build time according to the number of records and the size of a record, but, in addition, each time they are read or written, the definition of the file contents takes place. At read/write time, the relative locations of variables within the record are mapped. In order, therefore, to insure proper communications between all programs that use FORTRAN files, the user must insure that all defined file cards, asterisk file cards, and read/write statements to files and records are the same. This may be accomplished by punching only one card for each of the above file reference statements and duplicating this card where needed in source programs. In this way, the user insures that the variable list is the same for each record within the system.

Use of Analog Input Read-and-Transfer Function

The read-and-transfer function for the AISQN subroutine is the most versatile of all the analog input calls. With this call the user is able to handle each analog input point as it comes in, and to simultaneously insure that the rate of input of analog input points is a maximum. The read-and-transfer function operates as follows: after one analog input point has been read into core, and conversion has been started on the next sequential point, the analog input subroutine transfers control to a user's subroutine, which can manipulate this point in any desired way while the next analog input point is being converted. The value just read can be transformed into engineering units, checked for out-of-limit conditions, and used in setting-up a control function.

All of these manipulations can be performed within the user's subroutine. This allows the system to achieve maximum overlap of data input and computation. In the sample system, the read-and-transfer function is used only in the SCAN2 program; it could also have been effectively used in the LEV10 program and, in fact, wherever analog input is used.

SYSTEM GENERATION

The entire system generation procedure for the sample system can be viewed on a step-by-step basis from the system and list printer output listings (see Program Listings No. 9: System Generation). These cover the initial writing of addresses on the disk cartridges, to the calibration of instruments (before a cold start is performed). Two items are excluded from the listings: the assembly of TASK and the assembly of the System Director. These two source decks have already been assembled and are in available object format.

Step 1. The first step in generating the system is the writing of addresses on the disk. Step 1, in the listing, shows the initialization of disks on disk drives 0 and 1. Note that at the time addresses were written on drive 0, there were no defective cylinders. When addresses were written on drive 1, three cylinders proved defective. The operator message states that logical cylinders /0013 are defective, indicating that there are three consecutive
defective cylinders. Logical cylinder /0012 is operational, but there is a three-cylinder gap before logical cylinder /0013 is reached. Except for one restriction on its use, this disk is acceptable for TSX system operation. The restriction (given by an error message) is that a skeleton must not be built using this storage device; that is, the disk cartridge cannot be used for logical drive zero for an on-line system.

Step 2. The IBM Nonprocess System is loaded to the disk via the System Loader. The first item printed out is the Assignment Table, which lists all hardware devices defined for this configuration, the interrupt level of each device, and the bit position within the Interrupt Level Status Word (ILSW) where the interrupt bit for each device is located. After the Assignment Table is built, absolute programs are loaded to disk. These are followed by IBM-supplied subroutines.

The final item in the system load operation is the *DEDIT function which instructs the system to reserve 20 disk cylinders for the buffering of messages and that the object machine will contain 32K words of core storage.

Steps 3 through 8. These steps are concerned with the organization of the disks for this particular installation. Step 3 defines the sample system as a two-drive system; drives 0 and 1 are then labelled. Steps 6 and 7 define the configuration of these two drives. A differentiation is now made between the process and nonprocess drives. All control programs and components associated with the process, such as the skeleton and EAC, are located on logical drive 1 while all nonprocess functions such as the FORTRAN Compiler and the Assembler are located on logical drive 0. This is the initial stage in the organization of the sample system. Step 8 reserves disk space for data files used by the system; the values that will be stored in the files are not defined at this time.

Step 9. In step 9, the System Director, in object format, is stored from cards to a relocatable area on logical drive 1.

Steps 10 through 23. These steps set out the compilation and assembly of all system subroutines. These include INSKELE interrupt subroutines, COUNT subroutines, a solitary TIMER subroutine, and normally-called subroutines. As these are compiled or assembled, they are stored on disk in relocatable format so that they will be readily available at skeleton build time. Step 23 is a checkpoint along the system generation route; the *DUMPLET verifies that all subroutines necessary for the creation of the skeleton are properly stored on disk.

Step 24. The actual System Skeleton building phase is now carried out. TASK is reloaded to core from cards for the first time since the commencement of system generation. Note that the // JOB card for the skeleton build function contains an "A" in column 15, which defines a two-drive system; this card must be identical to the // JOB card which precedes the final // END OF ALL JOBS.

At the commencement of skeleton build, a KOC error message is printed, signifying that the ANINT subroutine (an I/O subroutine) has missed an interrupt branch table entry. The error message states that IAC code /0023 is missing from the Master Interrupt Branch Table. Since this EAC code concerns the comparator feature on the Analog Input Expander, and there is no comparator on the sample system, the error code can be ignored. After the skeleton map is printed, a series of K13 error messages is printed, indicating that certain named programs have not yet been built. Since these programs cannot be built until the skeleton itself has been built, these errors can be ignored. Note that a K13 error would be significant if an operator performed a skeleton rebuild and obtained a K13 error message for a program that should have been on disk. Once the skeleton is built, it is time to compile and build the mainline core loads.

Steps 25 through 45. All system process core loads are now built and stored in core image format on disk. At this point in time, TASK is still resident in core controlling the Nonprocess Monitor. Step 30 is of special interest; three dummy core loads are provided in the system to be referenced on a DUP *DELETE card, when it is desired to modify any of the core loads in the system. For example, if core load GRADE contains an error, and it is desirable to recompile GRADE and restore it on disk, the first thing that must be carried out before this can be done is to delete the old version of GRADE. If, however, the system is on-line at the time GRADE is deleted, a core load must be available on disk which can be loaded and executed when the core load name GRADE is called after its deletion, but before storage of its replacement. Since a straight delete of GRADE is not possible, it is preferable to replace GRADE with a dummy core load until such time as its new version becomes resident on disk.

Between steps 44 and 45, TASK is eliminated from core, and system operation is initiated by a cold start which specifies COLDN as the initial core load. The function of COLDN is to provide perpetual
time-sharing so that the Nonprocess Monitor may be used to execute user-written nonprocess core loads during system generation. It is necessary to execute these nonprocess core loads during system generation because there are certain data areas that must be initialized prior to a process cold start. There are also four nonprocess core loads that will be executed in the time-shared mode, at a later time, that affect process operation.

Step 45 is concerned with the compilation and execution of a one-time-only core load, named SPECL, which is used to place initial values in the process.

Steps 46 through 51. Once the job file is initialized, step 46 compiles and stores the first of the calibration core loads. As soon as this core load is stored on disk, it is executed at step 47 to obtain initial calibration factors for closed loops 7 and 8. Steps 48 through 51 cover the calibration of other analog input points in the system.

Steps 52 through 54. At step 52, the program which will be used to update the job files on disk, as new grades of paper are placed in the process job schedule, is compiled and stored. It is executed at step 53 to ensure that it functions correctly. Step 54 is the final step in the system generation procedure; an *DUMPLET is performed on both drives to ensure that all core loads and data areas are correctly defined on disk. At this point, the sample system is ready to be placed on-line to control the paper machine.

ON-LINE OUTPUT FROM THE SAMPLE SYSTEM

Samples of output listings obtained during on-line continuous operation of the sample system are given at the end of this section (see Program Listing No. 10: On-Line Process Output). The specimen listings illustrate process operator, supervisor, and time-shared output.

Output 1. This is generated at cold start time on the process operator’s 1053 Printer. It reveals the cold start procedure and continues with a typical sample output over the next few hours of processing on the same printer. Note that at cold start time, a process job sequence number of zero is supplied to the system; the sample system went, therefore, into a production stop state. Two minutes later, the first process job number is entered into the system from the process operator’s console. When the process operator receives a signal that the paper machine is ready to go on-line with this grade of paper, he depresses the abort-grade button which initiates the production of grade 12345; this grade had been entered into the queue sequence a minute earlier. At the start of grade 12345, the production time is printed as one hour and 29 minutes; the start time is recorded as 8:03 a.m., Friday. At this stage, the high and low limits for all operator guide points as well as set point values for the set point stations are printed. At 8:05 a.m., the operator at the process operator’s console changes the set point value for loop 7 to a value of 50. Note that the normal scan of the operator guide points is executed every two minutes. At 8:15 a.m., the first quarter hour log is printed. As can be seen from the listing, closed loop 7 is now on set point value. It should be noted that the set point range for acceptable values is within plus or minus one of the actual set point value. Following the 8:15 a.m. quarter hour log, the shift-end log is printed. At 9:00 a.m., the first of the periodic one-hour logs is printed.

Output 2. This output is on the process operator’s 1053 printer and shows three of the operator console functions being executed. The first function occurs at 11:28 a.m., Friday and consists of changing the high and low limits on operator guide point 5. Note that two minutes later point 5 is recorded as being out of limits, and calls for a demand scan, at which time no limit violation is recorded. Four minutes later, at 11:34 a.m., the operator calls for complete information on operator guide point 5. The value of point 5 is recorded; its high and low limits and the two conversion factors are printed.

Output 3. Output 3 shows two of the other functions which can be performed from the process operator’s console. The first is the calling of a trend log on operator guide point 6. The operator specifies to the system through the console that he wants a trend log on operator guide point 6 every ten seconds, and that he wants this trend log repeated 300 times. This is followed by the output from the trend log. Between the normal scan at 9:17 a.m. and the next normal scan at 9:19 a.m., eleven values of the trend log are printed. (In normal operations, within a two-minute period, twelve trend logs would be printed.) The reason for the eleven printed values is because the last trend log and the normal scan are both queued at the same time, and that the normal scan is queued with a higher value than the
trend log. The normal scan is therefore printed first with only eleven trend logs between it and the preceding normal scan. After only nineteen trend logs have been printed, the operator decides to abort the trend operation before it runs to completion, by depressing the abort trend log function button on his console. At 9:21 a.m., the sequence of process jobs is changed when the operator enters a new sequence number of zero into the system. This is done because the operator wants the computer to go into a suspended state, as far as the paper machine is concerned, when the current job runs to completion. It also means that the paper machine is being taken off computer control. At 9:23 a.m., the manufacturing process of this grade of paper comes to an end; the system then recognizes that the operator has changed job sequences to indicate that it is now time for a production stop. This occurs at 9:25 a.m.

Output 4. Output 4 shows how the operator changes the job sequence so that a different grade of paper can be made when a present grade under production is completed. At 12:26, the production on the preceding grade of paper being completed, the new grade, 4530 (which the operator has entered into the system), is initiated.

Output 5. Output 5 shows the printout on the process operator's 1053 Printer at the time of a reload condition. At this point, an intentional reload of the system is caused by bringing about an op-code violation in the Skeleton by depressing stop, reset, start. Following the normal scan at 9:06 a.m., the EAC reload message is printed. The cold start core load which determines that this is a reload operation then prints the message: Process Restart Checkpoint. At this time, all of the variables in INSKEL COMMON have been reinitialized, and the system is back in control of the process. Following the normal scan at 9:07 a.m., the quarter hour log, the one hour log and the shift-end log are printed. These are printed at this time to assure the operator that the system is running correctly and that all functions are operating.

Note that at 9:10 a.m. the operator performs a demand scan and is told that a limit violation has occurred on point 5.

Output 6. Output 6 demonstrates what happens when an error occurs in a nonprocess time-shared job. On Wednesday at 13:56, a time-shared nonprocess program is executed while the process is being monitored. This time-shared program which contains a FORTRAN I/O error thus causes the system to abort that job. Note that this has no effect on the control of the process.

Before the drive is brought on-line through the use of the TSX C.E. interrupt routine to bring logical drive 0 on-line. (Logical drive 0 had been taken off-line prior to 9:02 a.m. on Tuesday.) This means that the system is operating with only the process cartridge on-line. At 9:02 a.m., it was decided to run some non-process jobs; it therefore became necessary to place a nonprocess cartridge on logical drive zero and to bring that drive on-line.

Output 7. Output 7 demonstrates the use of the TSX C.E. interrupt routine to bring logical drive 0 on-line. (Logical drive 0 had been taken off-line prior to 9:02 a.m. on Tuesday.) This means that the system is operating with only the process cartridge on-line. At 9:02 a.m., it was decided to run some non-process jobs; it therefore became necessary to place a nonprocess cartridge on logical drive zero and to bring that drive on-line.

Output 8. Output 8 is on the foremen's 1053 Printer. This shows two things: first, that information is given to the foreman, and the extent to which it varies from that supplied to the process operator. It also shows at what time system operation is terminated after a period of continuous operation. Note too, that the Monday morning report is printed exactly on time at 8:30 a.m.

Output 9. Output 9 is from the nonprocess time-shared side of the sample system. It shows what is printed on the 1443 Printer at the time the job files on disk are updated with data on new grades of paper to be produced.
Table 17. Program Data Sheets

These are specifications sheets of each program used in the TSX Sample System (see Basic System Documentation Needed). Each encircled listing number in the table corresponds to its exact counterpart in Program Listing No. 9: System Generation.

<table>
<thead>
<tr>
<th>Listing No.</th>
<th>Program</th>
<th>Type</th>
<th>Description</th>
<th>System Variables Referenced</th>
<th>Files Referenced</th>
<th>Programs Called</th>
<th>Calling Programs</th>
<th>System Subroutines Called</th>
<th>Restart Core Load</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SCHED</td>
<td>INSKEL CALL COUNT Subroutine</td>
<td>This subroutine schedules the periodic execution of the programs listed in the external statement. This subroutine is entered every 20 seconds.</td>
<td>SW0, SW1, SW2, SW3, SW4, SW5, DAY, IBASE, IBASZ, IBAZ</td>
<td>None</td>
<td>SCHED, SCAN2, LOG15, LOG60, SHEFT, WEEK</td>
<td>RSTAR, COLDP, GRAGE</td>
<td>None</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>LEVI0</td>
<td>INSKEL INTERRUPT Subroutine</td>
<td>This subroutine does closed loop control of eight set point stations. Every twenty seconds, the scheduler subroutine gives a call level to this routine and sets SW4 to point to the scan section such that all eight points are scanned. When the scan is finished, the output to each online station is computed and the first pulse output is given. Timer B is then used to set-up fifteen millisecond entries into the subroutine so that all necessary pulses are given.</td>
<td>OFFLN (1-8), SW4, VALUE (6-13), ISETPT (1-8), COUMT (1-8)</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SOUT</td>
<td>INSKEL CALL TIMER Subroutine</td>
<td>This subroutine services timer 5 and is used for initiating entries into the level 10 subroutine for the outputting of pulses to the set point stations.</td>
<td>SW0, SW4</td>
<td>None</td>
<td>None</td>
<td>LEVI0</td>
<td>None</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>QUE15</td>
<td>INSKEL INTERRUPT Subroutine</td>
<td>This subroutine queues the fifteen minute log routine on demand.</td>
<td>None</td>
<td>None</td>
<td>LOG15</td>
<td>None</td>
<td>None</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
Listing No.
Program. TCONT
Type. INSKEL CALL COUNT Subroutine
Description. This subroutine periodically queues the trend log program the number of times specified.
System Variables Referenced. ITCNT
Files Referenced. None
Programs Called. TREN, TCONT
Calling Programs. TCONT
System Subroutines Called. None
Restart Core Load. N/A

Listing No.
Program. GETVL
Type. USER Subroutine
Description. This subroutine reads the analog input value for each of the sixteen data entry dials.
System Variables Referenced. None
Files Referenced. None
Programs Called. None
Calling Programs. CONVR, CMIPT
System Subroutines Called. None
Restart Core Load. N/A

Listing No.
Program. TABRT
Type. INSKEL INTERRUPT Subroutine
Description. This subroutine aborts the trend log on demand.
System Variables Referenced. ITCNT
Files Referenced. None
Programs Called. None
Calling Programs. None
System Subroutines Called. None
Restart Core Load. N/A

Listing No.
Program. CONVR
Type. USER Subroutine
Description. This subroutine scans the 16 data entry dials at the process operator's console and converts them to an integer value with range 0 to 9.
System Variables Referenced. G(1-16), H(1-16)
Files Referenced. None
Programs Called. None
Calling Programs. COGLP, CLLSP, MGRTP, CPJSP, STRND, AIMON
System Subroutines Called. GETVL
Restart Core Load. N/A
Listing No.
Program. PTIME
Type. USER Subroutine
Description. This subroutine reads the clock and converts the time to a floating point number with the decimal point separating hours and minutes.
System Variables Referenced: None
Files Referenced: None
Programs Called: None
Calling Programs: RSTAR, GRADE, SCAN2, LOG1S, LOG60, SHFT, WEEK, COGLP, CCLS, CPS, AILMON
System Subroutines Called: None
Restart Core Load: N/A

Listing No.
Program. IADDR
Type. USER Subroutine
Description. This subroutine gets the address of a FORTRAN variable.
System Variables Referenced: None
Files Referenced: None
Programs Called: None
Calling Programs: None
System Subroutines Called: None
Restart Core Load: N/A

Listing No.
Program. ISBAD
Type. USER Subroutine
Description. This subroutine gets the address of the entry point to a subroutine.
System Variables Referenced: None
Files Referenced: None
Programs Called: None
Calling Programs: None
System Subroutines Called: None
Restart Core Load: N/A

Listing No.
Program. CESET
Type. INSKEL INTERRUPT Subroutine
Description. This subroutine queues the CE unmask program so that devices may be taken off-line or put on line.
System Variables Referenced: None
Files Referenced: None
Programs Called. CEINT
Calling Programs: None
System Subroutines Called: None
Restart Core Load: N/A
### Program: ABORT

**Type:** INSKEL INTERRUPT Subroutine  
**Description:** This subroutine queues the grade change program causing the present grade to be aborted.

<table>
<thead>
<tr>
<th>System Variables Referenced</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files Referenced</td>
<td>None</td>
</tr>
<tr>
<td>Programs Called</td>
<td>GRADE</td>
</tr>
<tr>
<td>Calling Programs</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>None</td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Program: COLDN

**Type:** MAINLINE CORE LOAD  
**Description:** This cold start core load is used to give perpetual time sharing.

<table>
<thead>
<tr>
<th>System Variables Referenced</th>
<th>RANGE, LOW, A, B, G, H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files Referenced</td>
<td>File 1, Records 1 + 2</td>
</tr>
<tr>
<td>Programs Called</td>
<td>None</td>
</tr>
<tr>
<td>Calling Programs</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>None</td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>COLDN</td>
</tr>
</tbody>
</table>

### Program: ENGD

**Type:** INSKEL CALL COUNT Subroutine  
**Description:** This subroutine aborts the grade in progress when the run time for that grade has elapsed.

<table>
<thead>
<tr>
<th>System Variables Referenced</th>
<th>SW3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files Referenced</td>
<td>None</td>
</tr>
<tr>
<td>Programs Called</td>
<td>GRADE</td>
</tr>
<tr>
<td>Calling Programs</td>
<td>RSTAR, COLDP</td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>None</td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Program: COLDS

**Type:** MAINLINE CORE LOAD  
**Description:** This is the normal cold start core load. It sets time sharing time to zero so that console interrupt must have been pushed before logical drive zero is ever referenced. This core load chains to COLDP to actually cold start the process.

<table>
<thead>
<tr>
<th>System Variables Referenced</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files Referenced</td>
<td>None</td>
</tr>
<tr>
<td>Programs Called</td>
<td>COLDP</td>
</tr>
<tr>
<td>Calling Programs</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>None</td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>COLDS</td>
</tr>
</tbody>
</table>
Listing No.
Program. RSTAR
Type. MAINLINE CORE LOAD
Description. This is the system restart core load. Whenever a restart condition occurs, this routine is loaded to variable core to make sure system constants in INSKEL COMMON are valid.
System Variables Referenced. SWO, SW3, RANGE, LOW, A, B, JOBN, DAY, IENDT, AHL, ALL, SETPT
Files Referenced.
File 1 Record 1
File 3 Records 1, 2, 3
Programs Called. SCHED, GRADE, ENGD
Calling Programs. None
System Subroutines Called. None
Restart Core Load. COLDS

Listing No.
Program. COLDP
Type. MAINLINE CORE LOAD
Description. This is the system process cold start and reload core load. If sense switch 6 is on, it does a process cold start and if sense switch 6 is off it assumes a reload condition has occurred so that it initializes the system to the last check point.
System Variables Referenced. SWO, SW1, SW3, SW5, RANGE, LOW, DAY, JOBN, A, B, SETPT, AHL, ALL, IENDT, G, H
Files Referenced.
File 1 Records 1, 2
File 3 Records 1, 2, 3
Programs Called. SCHED, GRADE, ENGD
Calling Programs. COLD
System Subroutines Called. None
Restart Core Load. COLDS

Listing No.
Program. CEINT
Type. MAINLINE CORE LOAD
Description. This core load is for use with the CE interrupt. It makes sure that all levels are unmasked after use of the CE interrupt routine.
System Variables Referenced. None
Files Referenced. None
Programs Called. None
Calling Programs. CESET
System Subroutines Called. None
Restart Core Load. CEINT

Listing No.
Program. DUM
Type. MAINLINE CORE LOAD
Description. This is a dummy core load for use in replacing or deleting system core loads.
System Variables Referenced. None
Files Referenced. None
Programs Called. None
Calling Programs. None
System Subroutines Called. None
Restart Core Load. DUM
### Listing No. 30

Program: IDUM  
**Type:** INTERRUPT CORE LOAD  
**Description:** This is a dummy core load for use in replacing or deleting system core loads.

- **System Variables Referenced:** None
- **Files Referenced:** None
- **Programs Called:** None
- **Calling Programs:** None
- **System Subroutines Called:** None
- **Restart Core Load:** N/A

### Listing No. 31

Program: GRADE  
**Type:** MAINLINE CORE LOAD  
**Description:** This program starts the production of a new grade.

- **System Variables Referenced:** SW0, SW3, DAY, JOBN, RANGE, LOW AIL, ALL, SETPT, IENDT
- **Files Referenced:**  
  - File 2  
  - File 3, Records 1, 2, 3
- **Programs Called:** SCHED
- **Calling Programs:** ABORT, ENDGD, RSTAR, COLDP
- **System Subroutines Called:** PTIME
- **Restart Core Load:** RSTAR

### Listing No. 30

Program: CDUM  
**Type:** COMBINATION CORE LOAD  
**Description:** This is a dummy core load for use in replacing or deleting system core loads.

- **System Variables Referenced:** None
- **Files Referenced:** None
- **Programs Called:** None
- **Calling Programs:** None
- **System Subroutines Called:** None
- **Restart Core Load:** CDUM

### Listing No. 32

Program: SCAN2  
**Type:** COMBINATION CORE LOAD  
**Description:** This combination core load scans all of the Op-guide points on the system and notes any limit violation to the operator.

- **System Variables Referenced:** DAY
- **Files Referenced:** None
- **Programs Called:** None
- **Calling Programs:** SCHED
- **System Subroutines Called:** PTIME, LIMIT
- **Restart Core Load:** RSTAR
<table>
<thead>
<tr>
<th>Listing No.</th>
<th>Program</th>
<th>Type</th>
<th>Description</th>
<th>System Variables Referenced</th>
<th>Files Referenced</th>
<th>Programs Called</th>
<th>Calling Programs</th>
<th>System Subroutines Called</th>
<th>Restart Core Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>LIMIT</td>
<td>USER Subroutine</td>
<td>This subroutine is the subroutine to be used in the AIS read and transfer function of the mainline core load.</td>
<td>AHL, ALL</td>
<td>None</td>
<td>None</td>
<td>SCAN2</td>
<td>NONE</td>
<td>N/A</td>
</tr>
<tr>
<td>33</td>
<td>LOG15</td>
<td>MAINLINE CORE LOAD</td>
<td>This is the fifteen minute log program which logs the values of all process variables in the system.</td>
<td>DAY</td>
<td>None</td>
<td>None</td>
<td>SCHED, QUE15</td>
<td>PTIME</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>LOG60</td>
<td>MAINLINE CORE LOAD</td>
<td>This program puts out the hour log.</td>
<td>DAY</td>
<td>None</td>
<td>None</td>
<td>SCHED</td>
<td>PTIME</td>
<td>RSTAR</td>
</tr>
<tr>
<td>35</td>
<td>SHIFT</td>
<td>MAINLINE CORE LOAD</td>
<td>This program outputs the shift log.</td>
<td>DAY</td>
<td>None</td>
<td>None</td>
<td>SCHED</td>
<td>PTIME</td>
<td>RSTAR</td>
</tr>
</tbody>
</table>
Listing No. Program. WEEK
Type. MAINLINE CORE LOAD
Description. This program outputs the weekly Monday morning log.
System Variables Referenced. DAY
Files Referenced. NONE
Programs Called. None
Calling Programs. SCHED
System Subroutines Called. PTIME
Restart Core Load. RSTAR

Listing No. Program. COGLP
Type. INTERRUPT CORE LOAD
Description. This core load changes the limits on operator-guide points at operator request.
System Variables Referenced. AHL, ALL, DAY
Files Referenced. File 3, Record 3
Programs Called. None
Calling Programs. None
System Subroutines Called. PTIME, CONVR
Restart Core Load. N/A

Listing No. Program. TREND
Type. MAINLINE CORE LOAD
Description. This is the trend log core load. It reads the value that the operator has asked. It is queued periodically by the TCONT subroutine with the period specified by the operator.
System Variables Referenced. IPONT, LOW, RANGE, A, B
Files Referenced. None
Programs Called. None
Calling Programs. TCONT
System Subroutines Called. None
Restart Core Load. RSTAR

Listing No. Program. CCLSP
Type. INTERRUPT CORE LOAD
Description. This core load changes the set point value for a set point station upon operator request.
System Variables Referenced. RANGE, LOW, SETPT, DAY
Files Referenced. File 3, Record 2
Programs Called. None
Calling Programs. None
System Subroutines Called. CONVR, PTIME
Restart Core Load. N/A
<table>
<thead>
<tr>
<th>Listing No.</th>
<th>40</th>
<th>Listing No.</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program.</td>
<td>MGRTP</td>
<td>Program.</td>
<td>STRND</td>
</tr>
<tr>
<td>Type.</td>
<td>INTERRUPT CORE LOAD</td>
<td>Type.</td>
<td>INTERRUPT CORE LOAD</td>
</tr>
<tr>
<td>Description.</td>
<td>This core load changes the run time for a grade upon operator request.</td>
<td>Description.</td>
<td>This core load initiates a trend log of the point specified by the operator.</td>
</tr>
<tr>
<td>System Variables Referenced.</td>
<td>IENDT, JOBN, DAY, SW3</td>
<td>System Variables Referenced.</td>
<td>IENDT, JOBN, DAY, SW3</td>
</tr>
<tr>
<td>Files Referenced.</td>
<td>File 3, Record 1</td>
<td>Files Referenced.</td>
<td>None</td>
</tr>
<tr>
<td>Programs Called.</td>
<td>None</td>
<td>Programs Called.</td>
<td>None</td>
</tr>
<tr>
<td>Calling Programs.</td>
<td>None</td>
<td>Calling Programs.</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called.</td>
<td>CONVR</td>
<td>System Subroutines Called.</td>
<td>CONVR</td>
</tr>
<tr>
<td>Restart Core Load.</td>
<td>N/A</td>
<td>Restart Core Load.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing No.</th>
<th>41</th>
<th>Listing No.</th>
<th>43</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program.</td>
<td>CPJSP</td>
<td>Program.</td>
<td>AIMON</td>
</tr>
<tr>
<td>Type.</td>
<td>INTERRUPT CORE LOAD</td>
<td>Type.</td>
<td>INTERRUPT CORE LOAD</td>
</tr>
<tr>
<td>Description.</td>
<td>This core load changes the sequence of grades upon operator request.</td>
<td>Description.</td>
<td>This core load logs all information about any point in the system.</td>
</tr>
<tr>
<td>System Variables Referenced.</td>
<td>JOBN, DAY, IENDT, SW3</td>
<td>System Variables Referenced.</td>
<td>A, B, DAY, LOW, RANGE, SETPT</td>
</tr>
<tr>
<td>Files Referenced.</td>
<td>File 3, Record 1</td>
<td>Files Referenced.</td>
<td>None</td>
</tr>
<tr>
<td>Programs Called.</td>
<td>None</td>
<td>Programs Called.</td>
<td>None</td>
</tr>
<tr>
<td>Calling Programs.</td>
<td>None</td>
<td>Calling Programs.</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called.</td>
<td>CONVR, PTIME</td>
<td>System Subroutines Called.</td>
<td>PTIME, CONVR</td>
</tr>
<tr>
<td>Restart Core Load.</td>
<td>N/A</td>
<td>Restart Core Load.</td>
<td>N/A</td>
</tr>
<tr>
<td>Listing No.</td>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program.</td>
<td>SPECL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type.</td>
<td>NONPROCESS CORE LOAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>This is a special one-time-only core load to set up the job files on disk for test purposes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Variables Referenced</td>
<td>AHL, ALL, A, B, RANGE, LOW, SETPT (1-8) JOBN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Files Referenced</td>
<td>File 1,Record 1 - RANGE, LOW, A, B File 2,Record 1 - A, JOBN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programs Called</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calling Programs</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing No.</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program.</td>
<td>SCALB</td>
</tr>
<tr>
<td>Type.</td>
<td>NONPROCESS CORE LOAD</td>
</tr>
<tr>
<td>Description</td>
<td>This nonprocess core load is for calibrating the set point stations.</td>
</tr>
<tr>
<td>System Variables Referenced</td>
<td>RANGE, LOW, A, B</td>
</tr>
<tr>
<td>Files Referenced</td>
<td>File 1,Record 1</td>
</tr>
<tr>
<td>Programs Called</td>
<td>None</td>
</tr>
<tr>
<td>Calling Programs</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>None</td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing No.</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program.</td>
<td>RCALB</td>
</tr>
<tr>
<td>Type.</td>
<td>NONPROCESS CORE LOAD</td>
</tr>
<tr>
<td>Description</td>
<td>This nonprocess core load is for calibrating the analog input points for Op-guide.</td>
</tr>
<tr>
<td>System Variables Referenced</td>
<td>A, B, RANGE, LOW</td>
</tr>
<tr>
<td>Files Referenced</td>
<td>File 1, Record 1</td>
</tr>
<tr>
<td>Programs Called</td>
<td>None</td>
</tr>
<tr>
<td>Calling Programs</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>None</td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing No.</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program.</td>
<td>CMIPT</td>
</tr>
<tr>
<td>Type.</td>
<td>NONPROCESS CORE LOAD</td>
</tr>
<tr>
<td>Description</td>
<td>This nonprocess core load is for calibrating the data entry dials.</td>
</tr>
<tr>
<td>System Variables Referenced</td>
<td>G(1-16), H(1-16)</td>
</tr>
<tr>
<td>Files Referenced</td>
<td>File 1,Record 2</td>
</tr>
<tr>
<td>Programs Called</td>
<td>None</td>
</tr>
<tr>
<td>Calling Programs</td>
<td>None</td>
</tr>
<tr>
<td>System Subroutines Called</td>
<td>GETVL</td>
</tr>
<tr>
<td>Restart Core Load</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Listing No. 52

Program. LOADJ

Type. NONPROCESS CORE LOAD

Description. This program loads the process job files on disk with data read from cards.

System Variables Referenced. None

Files Referenced. None

Programs Called. None

Calling Programs. None

System Subroutines Called. None

Restart Core Load. N/A
PROGRAM LISTING NO. 9: SYSTEM GENERATION

1  TASK 1800 TSX-II-1 SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER
TASK DISK WRITE ADDRESSES PROGRAM
ENTER NO. TRIES ON DATA SW MAX001F
DATA SWITCHES EQUAL LOGICAL DRIVE
DRIVE CODES--HEX 0000 0001 0002
CYLINDERS 0013 0013 0013
ARE DEFECTIVE
DO NOT USE SKELETON WITH THIS PACK
SEN SW 0 ON GO TO TASK OFF REDO
TASK DISK WRITE ADDRESSES PROGRAM
ENTER NO. TRIES ON DATA SW MAX001F
DATA SWITCHES EQUAL LOGICAL DRIVE
DRIVE CODES--HEX 0000 0001 0002
CYLINDERS 0013 0013 0013

2  //lSYSTEMLOADER
//# IBM 1800 TSX-II SAMPLE SYSTEM
#ASSIGNMENT
00 01 33
01 02 04,33
02 02 00,08
03 02 02,33
04 03 01,01,36,04,37,05
05 04 33,06,03,11,12
06 03 10,34,33
07 03 32,20,16
08 03 33
09 01 33
10 01 33
11 01 33
12 01 33
13 01 33
14 01 33
15 01 33
99 01 42/06
DEVICE LEV BIT IAC LUN
PISW 00 00 33
DISK-1 01 00 04
PISW 01 01 33
TIMERS 02 00 00
DISK-2 02 01 08
CARD-1 03 00 02 02
TYP1G1 04 01 33
TYP2G1 04 01 33
TYP3G1 04 01 33
PRNT-1 05 01 06 03
DINP 05 02 11
DAOP 05 03 12
ADC-1 06 00 10
COMP-1 06 01 34
PISW 06 01 33
PAPTE 07 00 03 07
CONSOL 07 01 32
RPO-01 07 02 20
ADC-2 07 03 10
PISW 08 00 33
PISW 09 00 33
PISW 10 00 33
PISW 11 00 33
PISW 12 00 33
PISW 13 00 33
PISW 14 00 33
PISW 15 00 33
KEYB-1 42 06
YOU DEFINED 000016 I/O DEVICES 
AND A TOTAL OF 000029 ILSW BITS
*LDISK .LET
SECTOR 0155
*LDISK .DCCOM
SECTOR 0000
*LDISK .MBT
SECTOR 0002

2  Continued
*LDISK .SUP
SECTOR 0005
*LDISK .CLB
SECTOR 0010
*LDISK /CLT
SECTOR 063A
*LDISK .DUP
SECTOR 001A
*LDISK .ASM
SECTOR 0066
*LDISK .FOR
SECTOR 0095
*LDISK .SIM
SECTOR 00P6
*LDISK .ERG
SECTOR 0618
*LDISK .BRT
IAND
CLEAR
CLOCK
COUNT
DMP DMPHX DMPDC
DMPS DMPST
DPART
ENDTS
IEDK
LO
LEVEL
MASK
OPMON
IORD
QIFCN
QUEUE
RESMK
SAKMK
SETCL
TIMER
UNMK
UNO
VIAC
CONHX
TRPRX
FLIP
EADD ESUB EADXX ESUBX ESBR ESBRX
EATN ETAAS
EAVL EABS
EAEX EAXX
EAIX EAXX
EDXR EDXRX EDIV EDIVX
ELD ELDX ESTD ESTOX
ELN EALOG
EMPY EMUPY
ESINE ESIN ECOSN ECOS
ESQR ESQRT
ETNH ETANH
ETRN ETNTR
EXPX EEXP
FSBR FSBRX FADD FSUB FADD FSUB
FARC
FATN FATAN
FAVL FABS
FAXX FXABX
FAAX FAXIX
FBTO FOTB
FIVX FDIV FDIVX
FIXI FIXIX
FLOX FLDX FSTO FSTOX
FLNX FALOG
FLDAT
FMAY FMPYX
FSINE FSIN FCOSN FCOS
FSQR FSQRT
FTNH FTANH
FTNNFTNR
FXPN FEXP
IABS
IFIX
INRM
SNR
XDD
XMD
XMS
XQSR
BINDC
BINTX

Programing Techniques 221
THE SOURCE CORE-SIZE IS 032768
THE OBJECT CORE-SIZE IS 032768
END SYSTEM LOAD
Continued

TASK 1800 TSX-II-1 SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR
SEN SW 2 ON FOR SKELETON BUILDER

// JOB
// DEFINE THE SAMPLE SYSTEM TO BE A TWO DRIVE SYSTEM
// DUP
*DEFINE NOISK 2
DUP FUNCTION COMPLETED

// JOB
// LABEL DISK DRIVE ZERO WITH 00000
// DUP
*DLABL 0 00000
DUP FUNCTION COMPLETED

// JOB
// LABEL DISK DRIVE ONE WITH 11111
// DUP
*DLABL 1 11111
DUP FUNCTION COMPLETED

// JOB
// DEFINE DRIVE ZERO CONFIGURATION
// DUP
*DEFINE CONFG COPO 020 000
DUP FUNCTION COMPLETED

// JOB
// DEFINE DRIVE ONE CONFIGURATION
// DUP
*DEFINE CONFG SXIN11IDE1M1F1P1 16000 16767 080 010 05
DUP FUNCTION COMPLETED

// JOB
// SET UP FILES FILE1, FILE2, AND FILE3 ON DRIVE ONE
// DUP
*STOREDATAD 1 1 FILE1 002
DUP FUNCTION COMPLETED
*STOREDATAD 1 1 FILE2 100
DUP FUNCTION COMPLETED
*STOREDATAD 1 1 FILE3 003
DUP FUNCTION COMPLETED

// JOB
// STORE SYSTEM DIRECTOR FROM CARDS ON DISK
// DUP
*STORE RD 1 SYDIR
SYDIR OUTTR CHAIN INTEX SHARE SPEC1 BACK EACLK
DUP FUNCTION COMPLETED
**INSKEL CALL COUNT SUBROUTINE**

**FOR**

**LIST ALL**

**PERIODIC PROGRAM SCHEDULER**

**SUBROUTINE** SCHED

C

C THIS SUBROUTINE SCHEDULES THE PERIODIC EXECUTION OF THE PROGRAMS

C LISTED IN THE EXTERNAL STATEMENT. THIS SUBROUTINE IS ENTERED

C EVERY 20 SECONDS.

C

INTEGER SW0, SW1, SW2, SW3, SW4, SW5, DAY

INTEGER VALUEI151, SETPTISI, COUNTISI, OFFLNISI

EXTERNAL SCAN2, LOG15, LOG60, SHIFT, WEEK

DIMENSION RANGEISI, AHLI401, ALLI401, AI401, BI401, LOWISI, GI16I, HI16I

COMMON/INSKEL/SWO, SWl, SW2, SW3, SW4, SW5, DAY, JOBN, VALUE, RANGE, LOW, SET

C

C SET UP NEXT ENTRY INTO THE SCHEDULER.

C CALL COUNT(0,1,20)

C

C INITIATE CLOSED LOOP CONTROL PROGRAM ON LEVEL 10.

C

GO TO (97, 11, 53)

97 GO TO (99, 98), 50

99 CALL LEVEL(10)

SW4 = 1

C TEST FOR TWO MINUTE LIMIT SCAN

C

SW5 = SW5 + 1

GO TO (11, 1, 11, 2), SW5

2 CALL QUEUE(SCAN2, 9, 0)

CALL ENTS

SW5 = 0

1 CONTINUE

C

C READ CLOCK

C

CALL CLOCK(1)

C

C TEST FOR FIFTEEN MINUTE LOG

C

IF(IBASE)4, 4, 5

4 IF(I250-I110, 6, 6

5 IF(IBASE=116, 6, 10

6 GO TO (66, 67), SW3

66 CALL QUEUE(LOG15, 11, 0)

CALL ENTS

IBASE=IBASE+250

67 IFBASE-IBASE-23760110, 7, 7

7 IBASE = 0

10 CONTINUE

C

C TEST FOR HOUR LOG

C

IF(IBASZ)11, 11, 12

11 IF(I250-I120, 13, 13

12 IF(IBASZ=113, 13, 20

13 GO TO (113, 14), SW3

113 CALL QUEUE(LOG60, 12, 0)

CALL ENTS

114 IBASZ=IBASZ+1000

14 IBASZ = 0

20 CONTINUE

C

C TEST FOR SHIFT END LOG AT 8.15, 16.15, 00.15

C

IF(IBAZZ-250)21, 21, 22

21 IF(I21000)22, 22, 30

22 IF(IBAZZ=23, 23, 30

23 CALL QUEUE(SHIFT, 13, 0)

CALL ENTS

IBAZZ=IBAZZ+9000

24 I17000=IBAZZ24, 24, 30

25 CONTINUE

C

C UPDATE DAY OF WEEK

C

IF(1-100), 31, 35

31 IF(SW132, 32, 36

32 DAY=DAY+1

33 IF(DAY=133, 33, 34

224
Continued

33 DAY=1
34 SW1=1
GO TO 36
35 SW2=1
CONTINUE
36 SW2=1

CALL QUEUE(WEKT,14,0)
SW2=1
CALL ENDTS
100 RETURN
END

VARIABLE ALLOCATIONS
CWO(I#)=FFFF
CSW1(I#)=FFFE
CSW2(I#)=FFFD
CSW3(I#)=FFFC
CSW4(I#)=FFFB
CSW5(I#)=FFFF

STATEMENT ALLOCATIONS
97 =0021 99 =0027 98 =002A 2 =003E 1 =004A 4 =0051 5 =0059 6 =005F 66 =0065 67 =006D
7 =0079 10 =007D 11 =0081 12 =0089 13 =008F 113 =0095 114 =009D 14 =00A9 20 =00AD 21 =00B3
22 =0089 23 =008F 24 =00D3 30 =00D7 31 =000D 32 =00EE 34 =00F1 35 =00F7 36 =00FC
40 =0102 41 =0108 42 =010F 43 =0113 100 =011F

FEATURES SUPPORTED
ONE WORD INTEGERS
CALLED SUBPROGRAMS
SCANT LOGIS LOG60 SHIFT WEEK COUNT LEVEL QUEUE ENDTS CLOCK COMGO

INTEGER CONSTANTS
0=0000 1=0009 20=0004 10=0005 9=0006 250=0007 11=0008 23760=0009 12=000A 1000=000B
23100=000C 13=000D 8000=000E 17000=000F 100=0010 8=0011 2=0012 8500=0013 14=0014

CORE REQUIREMENTS FOR SCHED
COMM 0 0 INSKEL COMMON 464 VARIABLES 2 PROGRAM 288

END OF COMPIILATION

SCHED
DUP FUNCTION COMPLETED
*DELETE SCHED
SCHED
D25 NAME NOT IN L/F
*STORE 1 SCHED
SCHED
DUP FUNCTION COMPLETED

// JOB A
// * INSKEL INTERRUPT SUBROUTINE
// FOR
// *LIST ALL
** LEVEL 10 SUBROUTINE FOR CLOSED LOOP CONTROL
SUBROUTINE LEV10
C
C THIS SUBROUTINE DOES CLOSED LOOP CONTROL OF EIGHT SET POINT
STATIONS. EVERY TWENTY SECONDS THE SCHEDULER SUBROUTINE GIVES
C A CALL LEVEL TO THIS ROUTINE AND SETS SW4 TO POINT TO THE SCAN
C SECTION SUCH THAT ALL EIGHT POINTS ARE SCANNED. WHEN THE SCAN IS
C FINISHED THE OUTPUT TO EACH ON-LINE STATION IS COMPUTED AND THE
C FIRST PULSE OUTPUT IS GIVEN. TIMER B IS THEN USED TO SET UP
C FIFTEEN MILLISECOND ENTRIES INTO THE SUBROUTINE SO THAT ALL
C NECESSARY PULSES ARE GIVEN.
C
INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),COUNT(8),OFFLN(8)
EXTERNAL SOUT
DIMENSION IBIT(8),IBIZ(8),IBIA(8),IOUT(8)
DIMENSION INVIT(11)
DIMENSION RANGE(8),AHL(40),ALL(40),AI40(40),R(40),LOW(8),G(16),H(16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
IPT,COUNT,OFLN,AHL,ALL,AI,AIBASE,IBASE2,IBAZZ,G,N,ENDT,IPERD,ITCNT
2,PONT
DATA IBIT/Z4000,Z1000,Z0400,Z0100,Z0040,Z0010,Z0004,Z0001/
DATA IBIZ/Z8000,Z2000,Z0800,Z0200,Z0080,Z0020,Z0008,Z0002/
DATA IBIA/Z3FFF,ZCFFF,ZF3FF,ZFCFF,ZFF3F,ZFFCF,ZFFF3,ZFFFC/
DATA IBIT/Z4000,Z1000,Z0400,Z0100,Z0040,Z0010,Z0004,Z0001/
DATA IBIZ/Z8000,Z2000,Z0800,Z0200,Z0080,Z0020,Z0008,Z0002/
DATA IBIA/Z3FFF,ZCFFF,ZF3FF,ZFCFF,ZFF3F,ZFFCF,ZFFF3,ZFFFC/

Programming Techniques 225
READ DIGITAL INPUT SWITCHES TO DETERMINE WHICH STATIONS ARE
ON-LINE.

INV(17)=64
CALL CSX(1011,INV(1),INV(17))
INV(1)=0
DO 678 J=1,8
   K=J+8
   OFFLN(J)=INV(K)
678  INV(1)=INV(1)+OFFLN(J)
   IF(INV(1))699,699,679
699  IOUT(1)=0
   GO TO 12
679  CONTINUE

BRANCH ON SW4 TO EITHER SCAN SECTION OR OUTPUT SECTION.

GO TO (1000,3000),SW4

1000 CONTINUE

SCAN SECTION

SET PULSE OUTPUT WORD TO ZERO

IOUT(1)=0
CALL AIS0001,VALUE(6),VALUE(15),4096
99  CALL AIS(0,IV)
   GO TO (99,88),IV
88  CONTINUE
   DO 100 J=1,8
      IF(OFFLN(J))1,100,1
1  IF(SEPT(J))111,100,111
111  VAL=SEPT(J)+VALUE(J)+5
   VAL=VAL*2000./RANGE(J)
   COUNT(J)=VAL
   IF(COUNT(J))100,4
   2  COUNT(J)=COUNT(J)-1
      IOUT(J)=IOR(IOUT(1),IBIT(J))
   GO TO 100
4  IOUT(J)=IOR(IOUT(1),IBIZ(J))
100  CONTINUE
   IF(IOUT(1))101,12,101
101  SW4=2
3000 CONTINUE

OUTPUT SECTION

IOUT(2)=126

THE FOLLOWING BIT IS SET IN THE OUTPUT WORD FOR INCREMENTING A
PULSE COUNTER FOR DETERMINING THE NUMBER OF PULSES GIVEN AT
ANY TIME.

IOUT(1)=IOR(IOUT(1),IBIZ(3))
CALL POI0001,IOUT(1),IOUT(3)

SEE WHICH POINTS ARE TO GO OUT NEXT TIME

DO 10 J=1,8
   IF(OFFLN(J))3,9,3
3  IF(COUNT(J))9,9,20
20  COUNT(J)=COUNT(J)-1
   IF(COUNT(J))9,9,10
9  IOUT(J)=IAND(IOUT(1),IBIA(J))
10  CONTINUE

CALL TIMER B FOR RE-ENTRY TO THIS SECTION IN 15 MILLISECONDS IF
THERE ARE ANY MORE PULSES TO GO OUT.

IF(IOUT(1))11,12,11
11  CALL TIMER(SOUT,2,15)
12  CONTINUE

DISPLAY PULSE OUTPUT FOR VISUAL VERIFICATION OF DIRECTION OF
MOVEMENT.

IOUT(2)=127
CALL POI0001,IOUT(1),IOUT(3)
CALL INTEX

END
### VARIABLE ALLOCATIONS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSWO(*)</td>
<td>FFFE</td>
<td></td>
</tr>
<tr>
<td>CSW1(*)</td>
<td>FFFD</td>
<td></td>
</tr>
<tr>
<td>CSW2(*)</td>
<td>FFFC</td>
<td></td>
</tr>
<tr>
<td>CSW3(*)</td>
<td>FFFB</td>
<td></td>
</tr>
<tr>
<td>CSW4(*)</td>
<td>FFFA</td>
<td></td>
</tr>
<tr>
<td>CSW5(*)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAY(*)</td>
<td>FFF9</td>
<td></td>
</tr>
<tr>
<td>JOB(*)</td>
<td>FFF8</td>
<td></td>
</tr>
<tr>
<td>VALUE(*)</td>
<td>FFE7-FFE9</td>
<td></td>
</tr>
<tr>
<td>RANGE(*)</td>
<td>FFE6-FFD8</td>
<td></td>
</tr>
<tr>
<td>LOW(*)</td>
<td>FFD7-FFD0</td>
<td></td>
</tr>
<tr>
<td>SETPT(*)</td>
<td>FFFC-FFCB</td>
<td></td>
</tr>
<tr>
<td>COUT(*)</td>
<td>FFFC-FFCO</td>
<td></td>
</tr>
<tr>
<td>IBASEI</td>
<td>FE77</td>
<td></td>
</tr>
<tr>
<td>IPERO(*)</td>
<td>FE32</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>0019-0012</td>
<td></td>
</tr>
<tr>
<td>STATEMENT(*)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSW1(*)</td>
<td>FFFD</td>
<td></td>
</tr>
<tr>
<td>JOB(*)</td>
<td>FFF8</td>
<td></td>
</tr>
<tr>
<td>OFFLNI</td>
<td>FFFB-FFB8</td>
<td></td>
</tr>
<tr>
<td>IBASZII</td>
<td>FE76</td>
<td></td>
</tr>
<tr>
<td>VALUE(*)</td>
<td>FFE7-FFE9</td>
<td></td>
</tr>
<tr>
<td>RANGE(*)</td>
<td>FFE6-FFE8</td>
<td></td>
</tr>
<tr>
<td>LOW(*)</td>
<td>FFD7-FFD0</td>
<td></td>
</tr>
<tr>
<td>SETPT(*)</td>
<td>FFFC-FFCB</td>
<td></td>
</tr>
<tr>
<td>COUT(*)</td>
<td>FFFC-FFCO</td>
<td></td>
</tr>
<tr>
<td>IBASEI</td>
<td>FE77</td>
<td></td>
</tr>
<tr>
<td>IPERO(*)</td>
<td>FE32</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>0019-0012</td>
<td></td>
</tr>
<tr>
<td>STATEMENT(*)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSW1(*)</td>
<td>FFFD</td>
<td></td>
</tr>
<tr>
<td>JOB(*)</td>
<td>FFF8</td>
<td></td>
</tr>
<tr>
<td>OFFLNI</td>
<td>FFFB-FFB8</td>
<td></td>
</tr>
<tr>
<td>IBASZII</td>
<td>FE76</td>
<td></td>
</tr>
<tr>
<td>VALUE(*)</td>
<td>FFE7-FFE9</td>
<td></td>
</tr>
<tr>
<td>RANGE(*)</td>
<td>FFE6-FFE8</td>
<td></td>
</tr>
<tr>
<td>LOW(*)</td>
<td>FFD7-FFD0</td>
<td></td>
</tr>
<tr>
<td>SETPT(*)</td>
<td>FFFC-FFCB</td>
<td></td>
</tr>
<tr>
<td>COUT(*)</td>
<td>FFFC-FFCO</td>
<td></td>
</tr>
<tr>
<td>IBASEI</td>
<td>FE77</td>
<td></td>
</tr>
<tr>
<td>IPERO(*)</td>
<td>FE32</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>0019-0012</td>
<td></td>
</tr>
</tbody>
</table>

### STATEMENT ALLOCATIONS

<table>
<thead>
<tr>
<th>Statement Address</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>678</td>
<td>000D0</td>
<td>699</td>
<td>0098</td>
</tr>
<tr>
<td>679</td>
<td>00A9</td>
<td>90</td>
<td>00C3</td>
</tr>
<tr>
<td>100</td>
<td>00A8</td>
<td>3000</td>
<td>0169</td>
</tr>
<tr>
<td>101</td>
<td>0145</td>
<td>20</td>
<td>01AE</td>
</tr>
<tr>
<td>102</td>
<td>018F</td>
<td>10</td>
<td>01BD</td>
</tr>
</tbody>
</table>

### CORE REQUIREMENTS FOR LEVEL 1

<table>
<thead>
<tr>
<th>COMMON NAME</th>
<th>VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>/INSKEL</td>
<td>464 VARIABLES 56 PROGRAM 466</td>
</tr>
</tbody>
</table>

### END OF COMPILATION

LEV10
DUP FUNCTION COMPLETED
// DUP
*DELET
LEV10
LEV10
D25 NAME NOT IN L/F
*STORE
LEV10
LEV10
DUP FUNCTION COMPLETED

### FEATURES SUPPORTED

- **ONE WORD INTEGERS**
- **CALLED SUBPROGRAMS**
- **REAL CONSTANTS**
- **INTEGER CONSTANTS**
- **CORE REQUIREMENTS FOR LEVEL 10**
- **COMMON VARIABLES**

### PROGRAM END

**LEVEL 10 FUNCTION COMPLETED**

**LEVEL 10 FUNCTION COMPLETED**

### SUBROUTINE

**SUBROUTINE SERVICES**

**THIS SUBROUTINE IS USED FOR INITIATING**

**ENTITIES IN THE LEVEL 10 SUBROUTINE FOR THE OUTPUTING OF PULSES**

**TO THE SET POINT STATIONS.**

**INTERGERS**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW0, SW1, SW2, SW3, SW4, SW5, DAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VALUE(15), SETPT(8), COUT(8), OFFLN(8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dimension</td>
<td>RANGE(8), A(140), ALL(40), A(40), B(40), L(16), H(16)</td>
<td></td>
</tr>
<tr>
<td>COMMUN/INSKEL/SW0, SW1, SW2, SW3, SW4, SW5, DAY</td>
<td>JOB(15), VALUE, RANGE, LOW, SET</td>
<td></td>
</tr>
<tr>
<td>COUT, COUT, OFFLN, A(140), ALL(40), A(40), B(40), L(16), H(16)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, IENDT:</td>
<td>1, IPERD</td>
<td></td>
</tr>
<tr>
<td>2, IPONT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, SETPT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, CALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, RETURN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### VARIABLE ALLOCATIONS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSWO(*)</td>
<td>FFFE</td>
<td></td>
</tr>
<tr>
<td>CSW2(*)</td>
<td>FFFD</td>
<td></td>
</tr>
<tr>
<td>CSW3(*)</td>
<td>FFFC</td>
<td></td>
</tr>
<tr>
<td>CSW4(*)</td>
<td>FFFB</td>
<td></td>
</tr>
<tr>
<td>CSW5(*)</td>
<td>FFFA</td>
<td></td>
</tr>
<tr>
<td>COUT(*)</td>
<td>FFFC-FFFC</td>
<td></td>
</tr>
<tr>
<td>IBASEI</td>
<td>FE77</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>FE76</td>
<td></td>
</tr>
<tr>
<td>VALUE(*)</td>
<td>FFF7-FFE7</td>
<td></td>
</tr>
<tr>
<td>RANGE(*)</td>
<td>FFE6-FFE6</td>
<td></td>
</tr>
<tr>
<td>LOW(*)</td>
<td>FFD7-FFD7</td>
<td></td>
</tr>
<tr>
<td>SETPT(*)</td>
<td>FFFC-FFC8</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>FE76</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>FE75</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>FE54</td>
<td></td>
</tr>
<tr>
<td>IBIASII</td>
<td>FE33</td>
<td></td>
</tr>
</tbody>
</table>

### STATEMENT ALLOCATIONS

<table>
<thead>
<tr>
<th>Statement Address</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0008</td>
<td>1</td>
<td>000E</td>
</tr>
</tbody>
</table>

### FEATURES SUPPORTED

- **ONE WORD INTEGERS**

### CALLED SUBPROGRAMS

**LEVEL COMGO**
Continued

INTEGER CONSTANTS
10=0000

CORE REQUIREMENTS FOR SOUT
COMMON 0 INSKE COMMON 464 VARIABLES 0 PROGRAM 20

END OF COMPILATION

SOUT
DUP FUNCTION COMPLETED
*STORE 1 SOUT
SOUT
DUP FUNCTION COMPLETED

// JOB
A
/// INSKE INTERRUPT SUBROUTINE
// FOR
#LIST ALL
** QUEUE 15 MINUTE LOG ON DEMAND
SUBROUTINE QUE15
C
C THIS SUBROUTINE QUEUES THE FIFTEEN MINUTE LOG ROUTINE ON DEMAND.
C
EXTERNAL LOG15
CALL ENDS
CALL QUEUE(LOG15,7,0)
CALL INTEX
END

FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS
LOG15 ENDS QUEUE INTEX

INTEGER CONSTANTS
7=0000 0=0001

CORE REQUIREMENTS FOR QUE15
COMMON 0 INSKE COMMON 0 VARIABLES 0 PROGRAM 14

END OF COMPILATION

QUE15
DUP FUNCTION COMPLETED
*STORE 1 QUE15
QUE15
DUP FUNCTION COMPLETED

// JOB
A
/// INSKE CALL COUNT SUBROUTINE
// FOR
#LIST ALL
** PERIODIC QUEUE OF TREND LOG SUBROUTINE TCONT
SUBROUTINE TCONT
C
C THIS SUBROUTINE PERIODICALLY QUEUES THE TREND LOG PROGRAM THE
C NUMBER OF TIMES SPECIFIED.
C
INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),COUNT(8),OFFLN(8)
EXTERNAL TREND
DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
COMMON/INSKE/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
PT,COUNT,OFFLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,EENDT,IPONT
2,ITCNT
CALL ENDS
CALL QUEUE(TREND,2,0)
ITCNT=ITCNT-1
IF(ITCNT<1)

1 CALL COUNT(2,3,IPERD)
2 RETURN

VARIABLE ALLOCATIONS
CSWO(1*)=FFFF
COSW(1*)=FFFF
DAY(1*)=FFFF
JOBN(1*)=FFFF
VALUE(1*)=FFFF-FFE9
RANGE(1*)=FFFF6-FFFF8
COUNT(1*)=FFFFC
OFFLN(1*)=FFFF8
AHL(1*)=FFFF6-FFFF8
ALL(1*)=FFFF6-FFFF8
IBASE(1*)=FE77
IBAZZ(1*)=FE76
IBAZZ(1*)=FE75
G(R*)=FE72-FFFF4
H(R*)=FFFF2-FFFF4
IENDT(I*)=FFFF3
IPONT(I*)=FE30
ITCNT(I*)=FE31
IPERD(I*)=FE32

STATEMENT ALLOCATIONS
1 =0017 2 =001C

FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS
TREND ENTS QUEUE COUNT

INTEGER CONSTANTS
2=0000 0=0001 1=0002 3=0003

CORE REQUIREMENTS FOR TCONT
COMMON 0 INSKEL COMMON 464 VARIABLES 0 PROGRAM 30

END OF COMPILATION

TCONT
DUP FUNCTION COMPLETED
#DELETE TCONT
TCONT
D2S NAME NOT IN L/F
#STORE TCONT
TCONT
DUP FUNCTION COMPLETED

// JOB
// # INSKEL INTERRUPT SUBROUTINE
// FOR
#LIST ALL
** ABORT TREND LOG SUBROUTINE
** SUBROUTINE TABRT
C
C THIS SUBROUTINE ABORTS THE TREND LOG ON DEMAND.
C
INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPTI8),COUMTI8),OFFLN,I8)
DIMENSION RANGE(8),AHLI40),ALLI40),AI40),BI40),LOWI8),GI16),HI16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
IPT,COUNT,OFFLN,AHL,ALL,A,B,IBASE,IBAZZ,G,H,IENDT,IPERD,ITCNT
2,IPONT
ITCNT=0
CALL INTEX
END

VARIABLE ALLOCATIONS
CSWO(1*)=FFFF
CSW(1*)=FFFF
DAY(1*)=FFFF
JOBN(1*)=FFFF
VALUE(1*)=FFFF7-FFFF9
RANGE(1*)=FFFF6-FFFF8
COUNT(1*)=FFFFC
OFFLN(1*)=FFFF8
AHL(1*)=FFFF6-FFFF8
ALL(1*)=FFFF6-FFFF8
IBASE(1*)=FE77
IBAZZ(1*)=FE76
IBAZZ(1*)=FE75
G(R*)=FE72-FFFF4
H(R*)=FFFF2-FFFF4
IENDT(I*)=FFFF3
IPONT(I*)=FE30
ITCNT(I*)=FE31
IPERD(I*)=FE32

FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS
INTEX

INTEGER CONSTANTS
0=0000

CORE REQUIREMENTS FOR TABRT
COMMON 0 INSKEL COMMON 464 VARIABLES 0 PROGRAM 8

END OF COMPILATION
CONTINUED

**STORE 1 TABRT

DUP FUNCTION COMPLETED

// JOB  A
// * USER SUBROUTINE
// FOR
** LIST ALL
** SUBROUTINE FOR READING DATA ENTRY DIALS
** SUBROUTINE GETVL(INVAL)
C
C THIS SUBROUTINE READS THE ANALOG INPUT VALUE FOR EACH OF THE
C SIXTEEN DATA ENTRY DIALS.
C
DIMENSION INS(12), INR(12), INVAL(16)
CALL AISI02011, INR(1), INR(10), 0
CALL AISI02001, INS(1), INS(12), 4098
1 CALL AISI00010, IV)
GO TO (1, 2), IV
2 CALL AISI011, IV)
GO TO (3, 3), IV
3 DO 4 J=1, 8
K=9-J
4 INVAL(J+8)=INR(K)
DO 5 J=1, 4
K=11-J
5 INVAL(J+4)=INS(K-6)
RETURN
END

VARIABLE ALLOCATIONS
INS(1 )=000B-000C INR(1 )=0017-000C IV(1 )=0018 J(I )=0019 K(1 )=001A

STATEMENT ALLOCATIONS
1 =0059 2 =0063 3 =0060 4 =0077 5 =00AA

FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS
AIS COMGO ISTOX SUBSC SUBIN

INTEGER CONSTANTS
S=0026 9=0027 4=0028 11=0029

CORE REQUIREMENTS FOR GETVL
COMMON 0 INSKEL COMMON 0 VARIABLES 32 PROGRAM 166

END OF COMPI LATION

GETVL
DUP FUNCTION COMPLETED
*STORE 1 GETVL
GETVL
DUP FUNCTION COMPLETED

// JOB  A
// * USER SUBROUTINE
// FOR
** LIST ALL
** SUBROUTINE FOR READING AND CONVERTING DATA ENTRY DIALS
** SUBROUTINE CONVR(INVAL)
C
C THIS SUBROUTINE SCANS THE 16 DATA ENTRY DIALS AT THE PROCESS
C OPERATORS CONSOLE AND CONVERTS THEM TO AN INTEGER VALUE WITH
C RANGE 0 TO 9.
C
DIMENSION INVAL(16)
DIMENSION RANGE(8), AHL(40), ALL(40), A(40), B(40), LOW(8), G(16), H(16)
Continued

COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
1PT,COUNT,DPFLR,AH,ALL,4,IBASE,IBASZ,IBAZZ,0,0
CALL GETVL (INVAL)
DO 10 J=1,16
10 INVAL(I)=INVAL(I)*GIJ+HIJ
RETURN
END

VARIABLE ALLOCATIONS
CCWO I
CCSW1 II*=FFFF
CCW2 II*=FFFD
CCSW3 II*=FFFC
CCSW4 II*=FFFB
CCSW5 II*=FFFA
DAYII*=FFF9
COUMTII*=FFC7-FFC0
IBASEII*=FE77
IBASZII*=FE75
IBAZZII*=FE75
ICOUNTII*=FFC7-FFC0
IBASII*=FE75
GII*=FE72-FE74
HII*=FE52-FE54
JII*)=0014

STATEMENT ALLOCATIONS
10 =0014

FEATURES SUPPORTED
ONE WORD INTEGERS
CALLED SUBPROGRAMS
GETVL, FADDX, FMPYX, IFIX, FSTO

INTEGER CONSTANTS
1=0004 16=0005

CORE REQUIREMENTS FOR CONVR
COMMON 0 INSKEL COMMON 460 VARIABLES 4 PROGRAM 54

END OF COMPILATION

CONVR
DUP FUNCTION COMPLETED
STORE 1 CONVR
CONVR
DUP FUNCTION COMPLETED

RETURN
END

VARIABLE ALLOCATIONS
II I=0002

FEATURES SUPPORTED
ONE WORD INTEGERS
CALLED SUBPROGRAMS
CLOCK, FADD, FMPY, FDIV, FSTO

REAL CONSTANTS
6000000 02=0004 .100000 04=0006 .000000 03=0008

INTEGER CONSTANTS
1000=000A 10=000B

CORE REQUIREMENTS FOR PTIME
COMMON 0 INSKEL COMMON 0 VARIABLES 4 PROGRAM 56

END OF COMPILATION

PTIME
DUP FUNCTION COMPLETED

Programming Techniques 231
Continued

// DUP
*STORE 1 PTIME
PTIME
DUP FUNCTION COMPLETED

// JOB A
* USER SUBROUTINE
* THE FOLLOWING TWO SUBROUTINES ARE USED IN FORTRAN TO OBTAIN THE
* ADDRESS OF EITHER A VARIABLE OR A SUBROUTINE
// ASM
* LIST
* PRINT SYMBOL TABLE

*******************************************************************************
IADDR SUBROUTINE
*******************************************************************************
* 
* THIS SUBROUTINE GETS THE ADDRESS OF A
* FORTRAN VARIABLE.
* 
* FORTRAN CALL
* I=IADDR(ABC)
* 
* AFTER EXECUTION OF THE ABOVE STATEMENT
* I EQUALS THE ADDRESS OF THE VARIABLE ABC.
* 
* ASM GENERATED CODE
* CALL IADDR
* DC ADDR(ABC)
* 
* RESULT IS IN THE ACCUM UPON RETURN
* 
* THE SUBROUTINE IS RE-ENTRANT
* 
*******************************************************************************

0000 09044119 ENT IADDR DEFINE ENTRY POINT
0000 0 0000 IADDR DC 0 CALL ENTRY
0001 01 66800000 LOX 12 IADDR SAVE RET ADDR AND SET XR2
0003 0 C200 LD 2 0 TO PARAMETER--GET PARAMETER
0004 00 4E000001 BSC L2 1 RETURN
0006

SYMBOL TABLE

IADDR 0000
NO ERRORS IN ABOVE ASSEMBLY.
IADDR
DUP FUNCTION COMPLETED
// DUP
*STORE 1 IADDR
IADDR
DUP FUNCTION COMPLETED
// ASM
* LIST
* PRINT SYMBOL TABLE
Continued

*****************************************************************************
* ISBAD SUBROUTINE *
*****************************************************************************
* * * * * * * * * * * * * * * * * * * * * * * * *
THIS SUBROUTINE GETS THE ADDRESS OF
THE ENTRY POINT TO A SUBROUTINE.
FORTRAN CALL
EXTERNAL SUBR

I=ISBAD(SUBR)

AFTER EXECUTION OF THE ABOVE STATEMENT
I EQUALS THE ADDRESS OF THE ENTRY POINT
OF THE SUBROUTINE SUBR.

ASK GENERATED CODE
CALL ISBAD
CALL SUBR

ENTRY ADDR TO SUB IS IN ACCUM UPON RETURN
THE SUBROUTINE IS RE-ENTRANT

*****************************************************************************

0000 098802044 ISBAD DEFINE ENTRY POINT
0000 0 0000 ISBAD DC 0 CALL ENTRY POINT
0001 01 66800000 LDX 12 ISBAD XR2#RET ADDR AND CALL PARA
0003 0 C200 LD 2 0 IS IT BSI L OR BSI I
0004 0 1008 SLA 0 PUT INDIRECT BIT IN O POS
0005 0 4100000BSC L #+3,- BRANCH FOR DIRECT BSI
0007 00 66800001 LD I2 L GET SUB ENTRY POINT
0009 0 7001 MDX #+1
000A 0 C201 LD 2 1 GET SUB ENTRY POINT
000B 00 40000002 BSC L2 2 EXIT
000C END

SYMBOL TABLE
ISBAD 0000
NO ERRORS IN ABOVE ASSEMBLY.
ISBAD
DUP FUNCTION COMPLETED
// DUP
*STORE 1 ISBAD
ISBAD
DUP FUNCTION COMPLETED

// JOB
// INSKEL INTERRUPT SUBROUTINE
// FOR
*LIST ALL
** QUEUE OF CE UNMASK ROUTINE
SUBROUTINE CESET
C
C THIS SUBROUTINE QUEUES THE CE UMASK PROGRAM SO THAT DEVICES MAY
C BE TAKEN OFF-LINE OR PUT ON-LINE.
C
EXTERNAL CEINT
CALL ENOTS
CALL SUBQUE(CEINT,20,0)
CALL INTEX
END

FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS
CEINT ENOTS QUEUE INTEX

INTEGER CONSTANTS
20=0000 0=0001
Continued

CORE REQUIREMENTS FOR CESET
COMMON 0 INSKEL COMMON 0 VARIABLES 0 PROGRAM 14

END OF COMPILATION

CESET
DUP FUNCTION COMPLETED
// DUP
#STORE 1 CESET
CESET
DUP FUNCTION COMPLETED

// JOB A
// = INSKEL INTERRUPT SUBROUTINE
// FOR ALL
** ABORT GRADE PROCESS INTERRUPT SUBROUTINE
SUBROUTINE ABORT
C
This subroutine queues the grade change program causing the
C present grade to be aborted.
C
EXTERNAL GRADE
CALL ENDS
CALL QUEUE(GRADE,5,0)
CALL INTEX
END

FEATURES SUPPORTED
ONE WORD INTEGERS
CALLED SUBPROGRAMS
GRADE ENDS QUEUE INTEX
INTEGER CONSTANTS
5=0000 0=0001

CORE REQUIREMENTS FOR ABORT
COMMON 0 INSKEL COMMON 0 VARIABLES 0 PROGRAM 14

END OF COMPILATION

ABORT
DUP FUNCTION COMPLETED
// DUP
#STORE 1 ABORT
ABORT
DUP FUNCTION COMPLETED

// JOB A
// = INSKEL CALL COUNT SUBROUTINE
// FOR ALL
** END OF GRADE COUNT SUBROUTINE
SUBROUTINE ENDO
C
This subroutine aborts the grade in progress when the run time for
C that grade has elapsed.
C
INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(S15),SETPT(B1),COUNT(B1),OFFSET(B1)
EXTERNAL GRADE
DIMENSION RANGE(B1),AHL(40),ALL(40),A(40),B(40),LW(16),O(16),H(16)
COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LW,SET
COUNT,OFFSET,AHL,ALL,A,B,IRASE,IBAZZ,G,H,IENDT,IPERO,ITCNT
2,IPONT
GO TO (1,2),SW3
1 CONTINUE
CALL ENDS
CALL QUEUE(GRADE,5,0)
2 RETURN
END
STATEMENT ALLOCATIONS
1 = 0000 2 = 0011

FEATURES SUPPORTED
ONE WORD INTEGERS
CALLED SUBPROGRAMS
GRADE ENDTS QUEUE COMGO
INTEGER CONSTANTS
5 = 0000 0 = 0001

CORE REQUIREMENTS FOR ENDGD
COMMON 0 INSKEL
COMMON 464 VARIABLES 0 PROGRAM 20

END OF COMPILATION

ENDGD
DUP FUNCTION COMPLETED
// DUP
*STORE 1 ENDGD
ENDGD
DUP FUNCTION COMPLETED

// JOB A
// DUP
*DUMPLET

LET

PACK LABEL
00000

Programming Techniques 235
Continued

**UFIO 001C 1908**
**URED**
**UWRT**
**U101**
**U10F**
**UIDAI**

**PAPTN 0010 194A**
**MAGT**
**AIP**
**AIRN**
**AIS**

**AIPNTN 0009 195C**
**DINP**
**XSAVE**
**DIEP**
**XEXP**

**GAGED 0003 19F8**
**UNAG**
**AIS**
**AIR**
**CS**

**VS**
**PI**
**DO**
**DI**

**BT2BT 0003 1A49**
**FCHAR**
**SCALF**
**EGRD**
**FRGR**

**WCHR1**
**ERULE**
**FMGVE**
**FLLOT**
**FINC**

**VCHR1**
**ERULE**
**EMOVE**

**PLOTS**

**PLT**

**PACK LABEL**
**00000**

**9DUMY 0040 05AO**
**E 00AO 05AO**

**LET**

**PACK LABEL**

**FLET**

**PACK LABEL**

**LET 0080 0000**
**SYDIR 009E 0080**
**OUTTR**

**SPECL**
**BACK**
**EACLK**
**SCHED**

**QUE15 0002 0159**
**TCNT**
**TABRT**
**GETVL**

**1ADD 0002 0175**
**ISBAD**
**CSE**
**ABORT**

**.E 1180 0180**

**FLET**

**PACK LABEL**

**DUP FUNCTION COMPLETED**

**// JOB A**
**// END OF ALL JOBS**

**24**

**TASK 1800 TSX-11-1 SAMPLE SYSTEM**
**SEN SW 0 ON FOR ABSOLUTE LOADER**
**SEN SW 1 ON FOR NONPROCESS MONITOR**
**SEN SW 2 ON FOR SKELETON BUILDER**

**PLACE TASK DECK IN CARD HOPPER**
**PUT SKL BUILD PROG IN CARD HOPPER**

**// JOB A**
**// XEO SKBLD**
**<INCLCHED/2600,ENDGD/2601,QUE15/0001,CESET/2500,ABORT/0000,LEV10/2410**
**<INCLUDTCONT/2602,LABRT/0002**

**<CCEND**

**KOC ANINT 0023 LEV.1**

**SKEL CORE MAP**
**TYPE NAME ARG1 ARG2**

**LIBF DISKN 0249 3E88**
**LIBF TYPEN 0674 3E8E**
**LIBF WRTYN 0674 3E8E**

**LIBF PRNTN 0866 F3C1**
ICL TABLE MAP

Continued

TABLE MAP

0000 2400  1454
0001 24B4  1456
0002 2505  1588
1100 24C2  15C4
1200 2391  15C8
1201 24A0  15CE
1202 26EA  15D0

K13 SCAN2 LEV.1

K13 LOG15 LEV.1

K13 LOG60 LEV.1

K13 SHIFT LEV.1

K13 WEEK LEV.1

K13 GRADE LEV.1

K13 CEINT LEV.1

K13 TREND LEV.1

DATA SW 0 ON TO ABORT SKEL
SK8, SYDIR LD NX

TASK 1800 TSX-II-1 SAMPLE SYSTEM
SEN SW 0 ON FOR ABSOLUTE LOADER
SEN SW 1 ON FOR NONPROCESS MONITOR

// JOB A
// * MAINLINE CORE LOAD
// FOR COLDN
*LST ALL
** OFF-LINE COLD START FOR PERPETUAL TIME SHARING
*IODES(DISK)

C THIS COLD START CORE LOAD IS USED TO GIVE PERPETUAL TIME SHARING.

C INTEGER SMO,SWM,SW2,SW3,SW4,SW5,DAY
C INTEGER VALUE(I),SETPT(I),COUTN(I),OFFLN(I)
C DIMENSION RANGE(I),AHL(I),ALL(I),A(I),B(I),LOW(I),G(I),H(I)
C COMMON/INSKEL/SMO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
CPT,COUNT,OFFLN,AHL,ALL,A,B,IBASE,IBAZZ,G,H,IENDT,IPERD,ITCNT
Z,IPO0T
C DEFINE FILE 112,320,U,I

C UNMASK ALL LEVELS AND CALL VIAQ WITH QUEUE EMPTY
C CALL UNMK(1,-1)
C READ1,RANGE,LOW,A,B
C READ1210,H
C CALL VIAQ
C END

VARIABLE ALLOCATIONS

CSWO(I)=FFFF CSW1(I)=FFFF CSW2(I)=FFFF CSW3(I)=FFFF CSW4(I)=FFFF CSW5(I)=FFFF
DAY(I)=FFFF JOBN(I)=FFFF VALUE(I)=FFFF RANGE(I)=FFFF LOW(I)=FFFF SETPT(I)=FFFF
COUNT(I)=FFFF OFFLN(I)=FFFF AHL(I)=FFFF ALL(I)=FFFF A(I)=FFFF B(I)=FFFF
IBASE(I)=FE77 IBASZ(I)=FE76 IBAZZ(I)=FE75 IENDT(I)=FE33
IPERD(I)=FE32 ITCNT(I)=FE31 IPO0T(I)=FE30 H(M)=FFFF I(M)=FFFF

FEATURES SUPPORTED
ONE WORD INTEGERS
IODES

CALLED SUBPROGRAMS
UNMK VIAQ MORED MDCOM MDAI MDAF

MACRO 

238
Continued

integer constants
1=000C 2=000D

core requirements for coldn
common 0 inskel common 464 variables 12 program 44

end of compilation

coldn
dup function completed
// dup
#define M COLDN DUM
d25 name not in l/f
*storecic 1 coldn coldn coldn
*ccendo

clb, build coldn

core load map
type name arg1 arg2

#cow table 3e82 000C
#ist table 3e8e 0010
#fid table 3e8b 0010
#etv table 3e8b 0051
#vitv table 3f0c 000c
#ist table 3f18 0036
*pnt table 3f4e 0008
#dfi table 3f56 0006
main coldn 3f64
pnt coldn 3f50
pnt coldn 3f54
call unmk 3f8e
libf more 4040 3f0c
libf mda1 3f0f 3f0f
libf mcm 40a2 3f15
call viaq 434c
call bt2bt 43ac
call save 43cb
call ifix 442c
core 445e 3ba2

clb, coldn ld xo

dup function completed

// job
// * mainline core load
// * asm colds
*list
*print symbol table

********************************************
++
++ this is the normal cold
++ start core load,
++
++ it sets time sharing time to zero
++ so that console interrupt
++ must have been pushed before
++ logical drive zero is ever
++ referenced.
++
++ this core load chains to coldp
++ to actually cold start the process
++
++
********************************************

0000 0 1010 start sla 16 set time sharing time to 0
0001 00 04000029 std l 41
0003 30 03201255 call chain
0005 30 03593117 call coldp chain to coldp
0008 0000 end start

programming techniques 239
Continued

SYMBOL TABLE

START 0000
NO ERRORS IN ABOVE ASSEMBLY.
COLDs
DUP FUNCTION COMPLETED
// DUP
#define COLDs M 1 COLDs COLDs COLDs
#define COLDs L COLDs COLDs
CLb, BUILD COLDs

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 3E82 000C
*IBT TABLE 3E8E 0010
*PID TABLE 3EAB 0011
*ETV TABLE 3EBB 0051
*IST TABLE 3F00 0036
*pnt TABLE 3F42 000C
MAIN COLDs 3F4E
PNT COLDs 3F44
PNT COLDs 3F48
PNT COLDs 3F50
CORE 3F58 40A8

CLb, COLDs LD X0
D 45 CORELOADS NOT FOUND
COLDp
DUP FUNCTION COMPLETED

// JOB A
// * MAINLINE CORE LOAD
// FOR RSTAR
** RESTART CORE LOAD
*IOCS(DISK*,TYPEWRITER)
C
C THIS IS THE SYSTEM RESTART CORE LOAD. WHEN EVER A RESTART
C CONDITION OCCURS THIS ROUTINE IS LOADED TO VARIABLE CORE TO MAKE
C SURE SYSTEM CONSTANTS IN INSKEL COMMON ARE VALID.
C
INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE15,SETPT18,COU18,OFFL18
EXTERNAL GRADE
DIMENSION RANGE18,AHI40,ALLI40,AI40,BI40,LDWI8,G16,H16
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
PT,COUNT,OFPLN,AL,AB,IBASE,IBA2Z,IBAZZ,G,H,IENDT,IPERD,ITCNT
C
DEFINE FILE 112,320,U,II
DEFINE FILE 313,320,U,II
C
RESTORE INSKEL COMMON
SWO=2
CALL UNMK(-1,-1)
CALL PTIMEITIME
WRITE(1,1)TIME
FORMAT(' PROCESS RESTART ')
READ(1)RANGE,LOW,A,B
READ(3)JOBN,DAY,IENDT,SW3
READ(3)SETPT
READ(3)AHL,ALL
GO TO 96,95,SW0
96 CALL VIAQ
95 CONTINUE
SWO=1
CALL COUNT0(1,5)
CALL CLOCK(1)
IF (I-IENDT)101,102,103
101 IPER=IENDT-I
GO TO 104
102 CALL CHAIN(GRADE)
103 IPER=I24000-IENDT+I
104 AA=IPER+3.5
IF(32000,AA)102,105,105
Continued

105 IPER=AA
CALL COUNT(1,2,IPER)
CALL VIAQ
END

VARIABLE ALLOCATIONS
CSW1(I)*=FFFF
CSW2(I)*=FFFF
CSW3(I)*=FFFF
CSW4(I)*=FFFF
CSW5(I)*=FFFF
DAY1(I)*=FFFF
JOBN(I)*=FFFF
VALUE(I)*=FFFF
RANGE(R)*=FFFF
LON(I)*=FFFF
SETPT(I)*=FFFF8
COUNT(I)*=FFC7
DFLN(I)*=FFB8
AML(R)*=FFB8
ALL(R)*=FFB8
AIR(I)*=FFB8
B(R)*=FFB8
IBASE2(I)*=FE76
IBASE2(I)*=FE76
IPERD2(I)*=FE32
ITCNI(I)*=FE31
IPOINT(I)*=FE30
CTIME(I)*=000C
AAIR(I)=000E
II(I)=0014

STATEMENT ALLOCATIONS
1 =0022
9 =0080
95 =0082
101 =0096
102 =009E
103 =009E
104 =00AA
105 =0088

FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS

CALLED SUBPROGRAMS
GRADE UNMK PTIME VIAQ COUNT CLOCK CHAIN FSUB FMPY FLD FSTO IFIX FLOAT COMGO LDFAC
MWRT MCOMP MIOF MDAI MDFAC MDCOM MEND

REAL CONSTANTS
.360000E 01 .320000E .05 .001A

INTEGER CONSTANTS
2=001C
1=001D
3=001E
0=001F
5=0020
24000=0021

CORE REQUIREMENTS FOR RSTAR
COMMON 0 INSKEL COMMON 464 VARIABLES 24 PROGRAM 172

END OF COMPILATION

RSTAR
DUP FUNCTION COMPLETED
*STORCIL M 1 RSTAR RSTAR CLODS
*FILES(1,FILE1,1)
*FILES(3,FILE3,1)
*CCEND

CLB, BUILD RSTAR

CORE LOAD MAP
TYPE NAME ARG1 ARG2

#CDW TABLE 3E82 000C
#ET THE TABLE 3E82 0010
#FIO TABLE 3E82 0010
#ETF TABLE 3E82 0051
#ETV TABLE 3F8C 002A
#IST TABLE 3F56 0026
#PNT TABLE 3F6C 000C
#OPT TABLE 3F78 000C
MAIN RSTAR 3F6A
PNT RSTAR 3F6E
PNT CLODS 3F72
LIBF EBPRT 403C 3F0C
CALL UNMK 40DC
CALL PTIME 4132
LIBF MWRT 42EC 3F0F
LIBF MIOF 4390 3F12
LIBF MCOMP 4379 3F15
LIBF MDRED 4406 3F18
LIBF MDAI 47A5 3F1B
LIBF MDFAC 47B2 3F1E
LIBF MDAT 4782 3F21
LIBF MDAC 4848 3F21
LIBF MDF 47AA 3F24
CALL VIAQ 4912
PNT GRADE 3F76
LIBF FSUB 4B86 3F27
LIBF LDFAC 4C10 3F2A
CALL PRT 4C3E
LIBF SUBIN 4C86 3F2D
LIBF FAOD 4992 3F30
LIBF IOU 4CC2 3F33
CALL IOFIX 4D5C
CALL BT1BT 4B0C
CALL SAVE 4C8F
CALL BT2BT 4DFF
CORE 460E 31F2

Programming Techniques 241
Continue

CLB, RSTAR LD XQ
D 45 CORE LOADS NOT FOUND
GRADE
DUP FUNCTION COMPLETED

// JOB A
// * MAINLINE CORE LOAD
// FOR COLOP
*LIST ALL
** ON-LINE COLD START CORE LOAD
*IOCS (TYPEWRITER, KEYBOARD, DISK)
C
C THIS IS THE SYSTEM PROCESS COLD START AND RELAOD CORE LOAD.
C IF SENSE SWITCH 6 IS ON IT DOES A PROCESS COLD START AND IF
C SENSE SWITCH 6 IS OFF IT ASSUMES A RELAOD CONDITION HAS OCCURED
C SO THAT IT INITIALIZES THE SYSTEM TO THE LAST CHECK POINT.
C
INTEGER SWO, SW1, SW2, SW3, SW4, SW5, DAY
INTEGER VALUE(15), SETPT(8), COUNT(S), OFFLN(S)
EXTERNAL GRADE
DIMENSION RANGE(8), AHL(40), ALL(40), A(40), B(40), LOW(8), G(16), H(16)
COMMON /INSKEL/ SWO, SW1, SW2, SW3, SW4, SW5, DAY, JOBN, VALUE, RANGE, LOW, SET
PT, COUNT(S), OFFLN(S), ALL, A, B, BASE, IBAZG, G, H, IENDT, ITERD, ITCNT
2, IPOINT
DEFINE FILE 1(2,320,U,II)
DEFINE FILE 3(3,320,U,II)
C
TAKE ALL CLOSED LOOPS OFF-LINE UNTIL OPERATOR BRINGS THEM ON-LINE
C
SWO=2
C
UNMASK ALL LEVELS
C
CALL UNMK(-1,-1)
C
SET UP /NSKEL COMMON
READ(1*1) RANGE, LOW, A, B
READ(1*2) G, H
SW5=5
SM=1
C
SET UP BASE TIMES FOR SCHEDULER
C
CALL CLOCK(I)
J=I/1000*1000
K=I-J
IBASE=J+(K/250*250)
IBASZ=J
IBAZZ=J/S000*SO00+250
SW3=2
CALL COUNT(0,1,1)
C
TEST FOR RELAOD OR COLD START
C
CALL SSWTCH(6,11)
GO TO (200, 100), 11
200 CONTINUE
WRITE(1,3)
3 FORMAT(1V PROCESS COLD START')
C
READ(2,2) DAY, JOBN
2 FORMAT(1W15)
CALL CHAIN(GRADE)
100 READ(3*2) JOBN, DAY, IENDT, SW3
READ(3*2) SETPT
READ(3*3) AHL, ALL
SWO=1
GO TO (96, 95), SW3
96 CONTINUE
CALL CLOCK(I)
IF(I-IENDT)101, 102, 103
101 IPER=IENDT+1
GO TO 104
102 CALL CHAIN(GRADE)
103 IPER=(24000-IENDT)+1
104 AA=IPERM+3.6
IF(AA>105.1)102, 105
105 IPER=AA
CALL COUNT(I, 2, IPER)
95 WRITE(1, 106)

242
Continued

106 FORMAT(1X, 'PROCESS RESTART CHECK POINT')
CALL VIAQ
END

VARIABLE ALLOCATIONS
CSWO(I*I)=FFFF
CSW1(I*I)=FFFE
CSW2(I*I)=FFFD
CSW3(I*I)=FFFC
CSW4(I*I)=FFFB
CSW5(I*I)=FFFA
COUNT(I*I)=FFEC-FFFC
DAY(I*I)=FFF9
COUMT(I*I)=FFC7-FFC0
IBASE(I*I)=FE77
IPERD(I*I)=FE32
J(I)=0014
STATEMENTS
CSW1(I*I)=FFFE
JOBN(I*I)=FFF8
OFFLN(I*I)=FFB8-FFB7
IBASZ(I*I)=FE76
!TCNT(I*I)=FE31
K(I)=0015
STATEMENTS
3 =0026 2 =0032 106 =0035 200 =0088 100 =00C8 96 =00F2 101 =00F0 102 =0105 103 =0109 104 =0111
105 =011F 95 =0129

FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS

CALLED SUBPROGRAMS
GRADE UNMK
CLOCK COUNT SSWT C\A I
VIAQ FSUB FMPY FLDB ELD FSTO IFIX FLOAT COMGO LDFAC
MRED MWRT MDRT MDRI MDAP MOD MDI TYPEN HOLEB EBPRT

REAL CONSTANTS
.360000E 01=0018
.320000E 05=001A

INTEGER CONSTANTS
2=001C 1=0010 5=001E 1000=001F 250=0020 8000=0021 0=0022 6=0023 3=0024 24000=0025

CORE REQUIREMENTS FOR COLDP
COMMON 0 INSKEL COMMON 464 VARIABLES 24 PROGRAM 280

END OF COMPILATION

COLDP
DUP FUNCTION COMPLETED
*DELETE M COLDP DUM
DUM 025 NAME NOT IN L/F
*STORECIL M 1 COLDP COLDP COLDS
*FILES(1,FILE1,11
*FILES(3,FILE3,11
*CCEND

CLB, BUILD COLDP

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CW TABLE 3E82 000C
*SB TABLE 3E88 0010
*TV TABLE 3E8E 0011
*VTY TABLE 3D00 002A
*1ST TABLE 3F36 0036
*PNT TABLE 3F6C 000C
*OPT TABLE 3F7B 000C
MAIN COLDP 3FBD
PNT COLDP 3FAE
PNT COLDS 3F72
LIBF EBPT 40A8 3F0C
LIBF HOLED 4148 3F0F
CALL UNMK 426A
LIBF MRTED 421E 3F12
LIBF MDRT 422B 3F15
LIBF MDAP 422E 3F18
LIBF MDRT 4376 3F18
CALL SSWT 426B
LIBF MRT 426C 3F1E
LIBF MCOMP 4869 3F21
LIBF MRED 47C9 3F24
LIBF MIO 4922 3F27
PNT YRED 3F76
LIBF MDI 420C 3F2A
LIBF FSUB 44A2 3F2D
LIBF LDFAC 402C 3F30
CALL VIAQ 405A
CALL PRT 408A
CALL BT2BT 4E04
CALL SAVE 4E20
CALL IFIX 4EA
LIBF IOU 4EB4 3F33
CALL BT1BT 4EAA
CORE 4F50 30BO
// JOB A
// * MAINLINE CORE LOAD
// FOR CEINT
// LIST ALL
// CE UNMASK CORE LOAD
// IOCS(TYPEWRITER)
// C
// THIS CORE LOAD IS FOR USE WITH THE CE INTERRUPT. IT MAKES
// SURE THAT ALL LEVELS ARE UNMASKED AFTER USE OF THE CE INTERRUPT
// ROUTINES.
// WRITE(1,1)
// 1 FORMAT(1X,1HCE UNMASK CORE LOAD--PRESS START TO EXIT FROM CORE LOAD)
// PAUSE
// CALL UNMK(-1,-1)
// CALL VIAQ
// END

STATEMENT ALLOCATIONS
1 =0006

FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS

CALLED SUBPROGRAMS
UNMK VIAQ MWRT MCOMP PAUSE TYPIX EBPR

INTEGER CONSTANTS
I=0004 0=0005

CORE REQUIREMENTS FOR CEINT
COMMON 0 INKEL COMMON 0 VARIABLES 4 PROGRAM 54

END OF COMPILATION

CEINT
DUP FUNCTION COMPLETED
// DUP
*STORECIL M 1 CEINT CEINT CEINT
*CCEND

CLB, BUILD CEINT

CORE LOAD MAP
TYPE NAME ARG1 ARG2
#COW TABLE 3E82 0000
#B8 TABLE 3E8E 0010
#FIO TABLE 3E8E 0010
#FIO TABLE 3E8A 0010
#VTV TABLE 3E8B 0005
#IST TABLE 3F1B 0036
#PNT TABLE 3F52 0008
MAIN CEINT 3F7E
PNT CEINT 3F54
PNT CEINT 3F58
LIBF EBPR 3F94 3F0C
LIBF MWRT 418E 3F0F
LIBF MCOMP 424B 3F12
LIBF PAUSE 4670 3F15
CALL UNMK 4684
CALL VIAQ 46CE
CALL PJRT 472E
LIBF IOU 4778 3F18
CALL 10IP 4812
CALL BIBT 4842
CALL SAVE 47AE
CORE 4B8B 3758
CLB, CEINT LD XQ
DUP FUNCTION COMPLETED

// JOB A
// * THE FOLLOWING ARE THREE DUMMY CORE LOADS FOR USE IN REPLACING
// # OR DELETING SYSTEM CORE LOADS.
**Program Description**

This program starts the production of a new grade.

```plaintext
// * MAINLINE CORE LOAD
// FOR GRADE
// LIST ALL
// * GRAGE CHANGE PROGRAM
// LOC(S,DISK,TYPewriter)

INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPTT(8),COUN(T(8),OFFLN(8)
DIMENSION INPP(8)
DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
1PT,COUNT,OFFLN,AHL,ALL,A,B,IBase,IBAS2,IBAZ2
EQUIVALENCE (VALUE(2),IENDT)
DEFINE FILE 2(100,320,JOBN)
DEFINE FILE 3(3,320,JOBN)

C THIS PROGRAM STARTS THE PRODUCTION OF A NEW GRADE.
C
SWO=2
C
READ NEXT SEQUENTIAL GRADE FILE OFF OF DISK
C
IF(JOBN=999,999)
999
SW3=2
WRITE(3,1)JOBN,DAY,IENDT,SW3
CALL PTIMEITIME)
WRITE(1,998)TIME,DAY
WRITE(4,998)TIME,DAY
WRITE(5,998)TIME,DAY
```

*Programming Techniques 245*
Continued

998 FORMAT(* PRODUCTION STOP  TIME='F9.2,'  DAY='I12)
CALL VIAQ
9999 SW=1
1 F (101-JOBNI)2,3
2 JOBNI=1
3 READ(J*JOBNI,J,TIME,SETP,ahl,all)
   DO 10 J=1,8
   IF(SEPT(J)>SETPT(J)*RANGE(J)/100.+LOW(J))
10 CONTINUE
C
   CALL COUNT TO SET UP TERMINATION OF GRADE ITIME SECONDS FROM NOW.
C
   CALL COUNT (12,ITIME)
   IENDT=ITIME/3600
   CALL CLOCK(I)
   IENDT=IENDT+I
   IF(I24000-IENDT)<100,100,101
   IENDT=IENDT-24000
   100 CONTINUE
   WRITE(3,1)JOBN,DAY,IENDT,SW3
   WRITE(3,2)SETPT
   WRITE(3,3)IAL,ALL
   SWO=1
   11=ITIME/3600
   ITIME=ITIME-11*3600
   TIME=II+ITIME/100.
   CALL PTIMEITIZI
   WRITE(1,11),TIME,TIZ,DAY
   WRITE(4,11),TIME,TIZ,DAY
   WRITE(5,11),TIME,TIZ,DAY
   11 FORMAT(* START UP GRADE='I6,'  PRODUCTION TIME='F9.2,'  START TIME='F
   9.2,'  DAY='I12)
   WRITE(1,500)
   500 FORMAT(* UP-GUIDE LIMITS FOR NEW GRADE*)
   WRITE(1,501)
   501 FORMAT(* POINT HIGH LIMIT  LOW LIMIT*10X POINT HIGH LIMIT
   1 LOW LIMIT*)
   DO 50 J=1,39,2
      K=41-J
      J1=J
      J2=J+1
      J3=J+2
      J4=J+3
      IA=9-I
      IB=8-I
      IC=7-I
      ID=6-I
      50 WRITE(1,5)J1,INPPI(J1),IAL(J1),ALL(J1),J2,INPPI(J2),IAL(J2),ALL(J2),J3,INPPI(J3),IAL(J3),ALL(J3),J4,INPPI(J4),IAL(J4),ALL(J4)
      WRITE(1,6)J1,INPPI(J1),IAL(J1),ALL(J1),J2,INPPI(J2),IAL(J2),ALL(J2),J3,INPPI(J3),IAL(J3),ALL(J3),J4,INPPI(J4),IAL(J4)
      DO 30 I=1,9,2
         J=4-I
         30 WRITE(1,7)J,INPPI(J),IAL(J),ALL(J)
   6 FORMAT(I18,10,3I10,10,3I10)
   CALL VIAQ
END

VARIABLE DEGREES

CSWO(1)=FFFF
CSW1(1)=FFFF
CSW2(1)=FFFF
CSW3(1)=FFFF
CSW4(1)=FFFF
CSW5(1)=FFFF
DAY(1)=0003F
JOBNI(1)=00054
CSW(1)=FFFF
CFADD
CFMPY
CFADDX
CFMPYX
CFMULT
CFDIV
CFDIVX
CFSTO
CIFIX
CFLOAT
CISTOX
CSTFAC
CBSFAC
MWRT
MCOMP
MIOFX
MIOIX
MIOF
MDRED
MDWRT
MDCOM
MDAI
MDAF
MDI
MTYPEN
EBPRT

FEATURES SUPPORTED

ONE WORD INTEGERS
IDCS

CALLED SUBPROGRAMS

PTIME
VIAQ
COUNT
CLOCK
FAOO
FMPY
FMPYX
FDIV
FDIVX
FDST
IFIX
FLOAT
ISTOX
STFAC
SBFAC
MWR
REAL CONSTANTS
   .100000E 03=002C .360000E 01=002E
INTEGER CONSTANTS
   2=0030 3=0031 1=0032 4=0033 5=0034 101=0035 8=0036 24000=0037 3600=0038 60=0039
CORE REQUIREMENTS FOR GRADE
COMMON 0 INSKEL COMMON 396 VARIABLES 44 PROGRAM 696
END OF COMPILATION
GRADE
DUP FUNCTION COMPLETED
DUP M GRADE DUM
GRADE
O25 NAME NOT IN L1F
STORECIL M 1 GRADE RSTAR
*FILES(2,FILE2,1)
*FILES(3,FILE3,1)
*CCEND
CLB, BUILD GRADE
CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 3E82 000C
*I0T TABLE 3E8E 0010
*FIO TABLE 3E8A 0010
*EVV TABLE 3E8B 0051
*VTV TABLE 3E8C 0036
*IST TABLE 3F42 0036
*PNT TABLE 3F7E 000E
*OPT TABLE 3F8C 000C
MAIN GRADE 4079
PNT GRADE 3F7A
PNT RSTAR 3F7E
LIBF EBPT 4264 3FOC
LIBF MDWRT 4457 3F0F
LIBF MDI 4310 3F12
LIBF MDOM 43CE 3F19
CALL PTIME 4E84
LIBF MWRT 483E 3F18
LIBF MIOF 48FF 3F18
LIBF MDI 48F4 3F1E
LIBF MCOMP 48CB 3F21
CALL VIAD 4CF0
LIBF MDRED 436C 3F24
LIBF MDAI 4318 3F27
LIBF MDAM 4308 3F2A
LIBF FADO 4070 3F2D
LIBF STPAC 4E04 3F30
LIBF SBFAC 4E08 3F33
LIBF MIOFX 4BF8 3F36
LIBF MIDIX 4D00 3F39
CALL PRT 4E1C
CALL BT2BT 4FEC 3014
CLS, GRADE LD XQ
DUP FUNCTION COMPLETED

// JOB A
// * TWO MINUTE LIMIT SCAN ROUTINE--COMBINATION CORE LOAD
// * THIS PROGRAM IS INITIATED EVERY 2 MINUTES OR BY PROCESS INTERRUPT
// * FIRST PART IS THE CONVERSION AND LIMIT CHECK SUBROUTINE USED BY AI
// * USER SUBROUTINE
// FOR
#LIST ALL
** CONVERSION AND LIMIT CHECK SUBROUTINE
** SUBROUTINE LIMIT(I,J)

Programming Techniques  247
THIS SUBROUTINE IS THE SUBROUTINE TO BE USED IN THE AIX READ AND TRANSFER FUNCTION OF THE MAIN LINE CORE LOAD.

INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),COUNT(8),OFFLN(8)
DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
    ,COUNT,OFFLN,AHL,ALL,A,B,IBASE,IBAZZ,G,H,IENDT,IPERD,ITCNT
    ,IPONT

C CONVERT I PARAMETER TO ARRAY INDEX

K=40-I
M=I+1

C CONVERT AI VALUE TO ENGINEERING UNITS

VSC=AIK)*LDIJ)+BIK)

C TEST FOR HIGH LIMIT VIOLATION

IF(VSC-AHLIK>2,1,1)
1 WRITE(1,100)M,VSC
100 FORMAT('HIGH LIMIT VIOLATION POINT',13,'VALUE',F12.4)
GO TO 4

C TEST FOR LOW LIMIT VIOLATION

2 IF(VSC-ALLIK<3,3,4)
3 WRITE(1,300)M,VSC
300 FORMAT('LOW LIMIT VIOLATION POINT',13,'VALUE',F12.4)
4 RETURN

END

VARIABLE ALLOCATIONS
CSWO(*)=FFF8 CSW1(*)=FFFC CSW2(*)=FFFD CSW3(*)=FFFA CSW4(*)=FFFB CSW5(*)=FFFA

DAY(*)=FFE9 JOBN(*)=FFFB VALUE(*)=FFE7-FFE9 RANGE(RE)=FFE6-FFE8 LOW(RE)=FFE0-FFE2 SETH(RE)=FFE6-FFE8
COUNTER(*)=FFC7-FECD OFFLN(*)=FFE7-FFE8 AHL(*)=FFE6-FFE8 ALL(RE)=FFE6-FFE8 A(RE)=FFE6-FFE8 B(RE)=FFE6-FFE7
IBASE(*)=FFE7 IBAZZ(*)=FFE7 IBASE(*)=FFE7 IBAZZ(*)=FFE7 IBASE(*)=FFE7 IBAZZ(*)=FFE7

STATEMENT ALLOCATIONS
100 =0008 300 =0020 1 =006C 2 =0076 3 =0082 4 =008A
FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS
LD FADDX FSUBX FMPYX FLD FSTO FLOAT LDFAC MWRT MCOMP MIDF MIOI SUBSC SUBIN

INTEGER CONSTANTS
40=0006 1=0007

CORE REQUIREMENTS FOR LIMIT
COMMON 0 INSKEL COMMON 464 VARIABLES 6 PROGRAM 134

END OF COMPILATION

LIMIT
DUP FUNCTION COMPLETED
// * LEAVE THIS SUBROUTINE IN TEMP STORAGE SINCE IT IS ONLY USED
// * WITH THIS MAINLINE
// * NOW COMPILE THE MAINLINE
// * COMBINATION CORE LOAD
// FOR SCAN2
*LIST ALL
** TWO MINUTE SCAN FOR LIMIT VIOLATIONS OF THOSE POINTS ON OP-GUIDE
*IOCS (TYPewriter)

C THIS COMBINATION CORE LOAD SCANS ALL OF THE OP-GUIDE POINTS ON THE
SYSTEM AND NOTES ANY LIMIT VIOLATION TO THE OPERATOR.

INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),COUNT(8),OFFLN(8)
EXTERNAL LIMIT
DIMENSION INF(42)
DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
    ,COUNT,OFFLN,AHL,ALL,A,B,IBASE,IBAZZ,G,H,IENDT,IPERD,ITCNT
    ,IPONT
Continued

C GET TIME IN HOURS-MINUTES
C CALL PTIME(TIME)
C DETERMINE IF THIS IS A DEMAND SCAN OR THE NORMAL 2 MINUTE SCAN BY
C DETERMINING WHAT LEVEL IT IS XEQ-ING ON
C I=7+LD(104)
IF(LD(11)=23)1,1,2
C DEMAND SCAN

1 WRITE(1,3)I,DAY,TIME
3 FORMAT('I,3I5',I,DAY,'3,4X',TIME,'F12.2)
GO TO 5
C NORMAL 2 MINUTE SCAN

2 WRITE(1,4)I,DAY,TIME
4 FORMAT('I,3I5',I,DAY,'3,4X',TIME,'F12.2)
C READ 40 RELAY POINTS USING AIS READ AND TRANSFER FUNCTION
C 5 CALL AIS(15001,INP(1),INP(42),0,LIMIT)
6 CALL AIS(0,1)
GO TO (6,7),1
7 CALL DPART
END

VARIABLE ALLOCATIONS
CSW0(I*)=FFFF CSW1(I*)=FFE CSW2(I*)=FFD CSW3(I*)=FFC CSW4(I*)=FFB CSW5(I*)=FFA

STATEMENT ALLOCATIONS
3 0036 4 0049 1 0060 2 0077 5 007F 6 0095 7 009F

FEATURES SUPPORTED
ONE WORD INTEGERS
IDCS

CALLED SUBPROGRAMS
LIMIT PTIME LD AIS DPART COMGD MWRT MCOMP MI0F MIDI TNPE EBPRT

INTEGER CONSTANTS
7=0030 104=0031 23=0032 1=0033 10001=0034 0=0035

CORE REQUIREMENTS FOR SCAN2
COMMON 0 INSNEL COMMON 464 VARIABLES 48 PROGRAM 114

END OF COMPILATION

SCAN2
DUP FUNCTION COMPLETED
*DELETE C SCAN2 CDUM 9999
SCAN2
*STORECIL C 1 SCAN2 SCAN2 RSTAR 1515
*CCEND

CLB, BUILD SCAN2

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*COW TABLE 3EB2 0000
*IBT TABLE 3EBE 0010
*FIO TABLE 3EAB 0010
*PTV TABLE 3EBA 0051
*VTV TABLE 3F0C 0021
*IST TABLE 3F20 0036
*PNT TABLE 3FA4 0008
MAIN SCAN2 3FC8
PNT SCAN2 3F66
PNT RSTAR 3F6A
LIBF EBPRT 400E 3F0C
CALL PTIME 408A
CALL LD 40E4
LIBF MWT 4284 3F0F
LIBF MIDI 4333 3F12
LIBF MI0F 4335 3F15
LIBF MCOMP 4311 3F18
CALL LIMIT 476F
CALL DPART 47C2
CALL PRT 4700
LIBF SUBIN 481A 3F1B
LIBF FADD 4874 3F1E
LIBF IADD 48F2 3F21
CALL IOPIX 498C
CALL BTBT 498C
CALL SVE 4928
LIBF FADXX 486E 3F24
LIBF FADXX 4863 3F27
LIBF DVF 4A20 3F2A
CALL VIAQ 4A4E
CALL BTBT 49BC
CALL IOU 498C
CALL VITX 4928
LIBF FSUBX 4863 3F27
LIBF LDFAC 4A20

CORE 4AB0 354F

CLB, SCAN2 LO X0
DUP FUNCTION COMPLETED

// JOB A
// * MAINLINE CORE LOAD
// FOR LOG15
*LIST ALL
** 15 MINUTE LOG ROUTINE
*IOCSITYPEWRITER)
C
C THIS IS THE FIFTEEN MINUTE LOG PROGRAM WHICH LOGS THE VALUES OF
C ALL PROCESS VARIABLES IN THE SYSTEM
C
INTEGER SWO, SW1, SW2, SW3, SW4, SW5, DAY
INTEGER VALUE(15), SETPT(15), COUNT(15), OFFLN(15)
DIMENSION INP(42), VSCI(40), INPPI(10)
DIMENSION RANGE(15), AHL(40), ALL(40), A(40), B(40), LEN(16), H(16)
COMMON/INSKEL/SWO, SW1, SW2, SW3, SW4, SW5, DAY, JOBN, VALUE, RANGE, LOW, SET
COUNT, OFFLN, AHL, ALL, A, B, IBASE, IBASE2, IBAZE, H, IEND, IPERD, ITCNT,
2, IPONT
C
C START INPUT OF RELAY POINTS
C
CALL AISI(12001, INP(1), INPPI(42), 0)
C
C PRINT HEADER
OF LOG
C
CALL PTIME(TIME)
WRITE(1,1) DAY, TIME
WRITE(5,1) DAY, TIME
1 FORMAT(/' LOG15 DAY', I3, 4X, 'TIME', F9.2)
WRITE(100)
WRITE(100)
100 FORMAT(/' OP-GUIDE POINTS')
WRITE (5,2)
WRITE (100)
2 FORMAT(/' CLOSED LOOP POINTS')
WRITE (5,2)
WRITE (100)
3 FORMAT(/' SS VALUES')
WRITE (100)
WRITE (100)
101 FORMAT(/' CLOSED LOOP POINTS')
WRITE (5,2)
WRITE (100)
4 CALL AIS (0,1)
GO TO (4,5,1)
5 DO 6 J=1,8
Continued

```plaintext
6 INPPI(J)=INPPI(J)-LOW(J)*100./RANGE(J)
DO 30 I=1,5,4
   J1=1
   J2=1+1
   J3=1+2
   J4=1+3
   IA=9-1
   IB=8-1
   IC=7-1
   ID=6-1
WRITE(1,7)J1,INPPIIA),J2,INPPIIB),J3,INPPIIC),J4,INPPI10)
30 WRITE(1,7)J1,INPPIIA),J2,INPPIIB),J3,INPPIIC),J4,INPPI10)
    FORMAT(18,110,31110,110)
CALL VIAQ
END
```

VARIABLE ALLOCATIONS

```plaintext
CSW0(*)=FFFF
OSAYII(*)=FFFF
COUNTI(*)=FFC7-FFC0
IBASEI(*)=FE77
IPEROI(*)=FE32
INPP(*)=0087-007E
J31(*)=0080
STATEMENT ALLOCATIONS
```

```plaintext
CSW2(*)=FFFF
OSAYII(*)=FFFF
COUNTI(*)=FFC7-FFC0
IBASEI(*)=FE77
IPEROI(*)=FE32
INPP(*)=0087-007E
J31(*)=0080
```
Continued

LIBF FADD 494E 3P27
LIBF IOU 4A80 3F2A
CALL IOFIX 4B4A
CALL BT1BT 4B7A
CALL SAVE 4AE6
CORE 4BED 3420

CLE, LOG15 LD X0
DUP FUNCTION COMPLETED

// JOB
// * MAINLINE CORE LOAD
// FOR LOG60
# ONE HOUR LOG
#I0C(TYPEWRITER)
#LIST ALL
C
THIS PROGRAM PUTS OUT THE HOUR LOG.

C
INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(I5),SETPT(I8),COUNT(I8),OFLN(I8)
DIMENSION RANGE(I8),AHL(I40),ALL(I40),A(I40),B(I40),LOW(I8),G(I16),H(I16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
1PT,COUNT,OFILN,AHL,ALL,A,B,IBASE,IBAZZ,G,H,JENDT,IPERD,ITCNT
2,IPONT
C
PRINT HEADING
CALL PTIME,TIMETIM
WRITE(1,1)DAY,TIME
WRITE(1,1)DAY,TIME
WRITE(1,1)DAY,TIME
1 FORMAT((,13,5X,TIME',F9.2)
C
OUTPUT ONE HOUR LOG
C
C
EXIT FROM ROUTINE
C
CALL VIAQ
END
C
STATEMENT ALLOCATIONS
CSWO(I*)=FFFF
C
CSW1(I*)=FFFE
CSW2(I*)=FFFD
CSW3(I*)=FFFC
CSW4(I*)=FFFF
CSW5(I*)=FFFA
C
DAY(I*)=FFFF
JOBN(I*)=FFFF
VALUE(I*)=FFFF7-FFFF9
RANGE(I*)=FFFF6-FFFF8
C
COUNT(I*)=FFFF7-FFFFD
OFLN(I*)=FFFFB-FFFFD
AHL(I*)=FFFF6-FFFF8
ALL(I*)=FFFF6-FFFF8
A(I*)=FFFF16-FFFF18
B(I*)=FFFF16-FFFF18
C
IBASE(I*)=FFFF
IBAZZ(I*)=FFFF76
IBAZZ(I*)=FFFF75
G(I*)=FFFF2-FFFF6
H(I*)=FFFF2-FFFF34
JENDT(I*)=FFFF

STATEMENT ALLOCATIONS
1 =0005
FEATURES SUPPORTED
ONE WORD INTEGERS
I0S
C
CALLED SUBPROGRAMS
PTIME VIAQ MWR MCOMP MIDF MIDI TYPEN ERPT
C
INTEGER CONSTANTS
1=0002
4=0003
5=0004
C
CORE REQUIREMENTS FOR LOG60
COMMON 0 INSKEL COMMON 464 VARIABLES 2 PROGRAM 52
C
END OF COMPILATION
LOG60
DUP FUNCTION COMPLETED
// DUP
#STORECIL M 1 LOG60 LOG60 RSTAR
#CCEND
CLE, BUILD LOG60
CORE LOAD MAP
TYPE NAME ARG1 ARG2
**TABLE 3E8E**

**F10 TABLE 3EAB**

**ETV TABLE 3EBB**

**VTV TABLE 3FOC**

**IST TABLE 3F24**

**PNT TABLE 3F5A**

**MAIN LOG60 3F60**

**PNT RSTAR 3F5C**

**LIBF EBPR 3F98**

**CALL PTIME 4044**

**LIBF MWRT 41FE**

**LIBF MIOF 42AF**

**LIBF MCOMP 428B**

**CALL VIAQ 4680**

**CALL PRD 471D**

**LIBF SUBIN 475A**

**LIBF FADD 4784**

**LIBF IOU 4832**

**CALL IOFIX 48CC**

**CALL BTIBT 48FC**

**CALL SAVE 4868**

**CORE 4962**

**DUP FUNCTION COMPLETED**

---

// JOB A
// ** MAINLINE CORE LOAD
// ** FOR SHIFT
// ** LIST ALL
// ** SHIFT END LOG
// IOCS (TYPEWRITER): 

**DUP FUNCTION COMPLETED**

**CONTINUED**

```
*MAIN LOG60 3F60
PNT RSTAR 3F5C
LIBF EBPR 3F98 3F0C
CALL PTIME 4044
LIBF MWRT 41FE 3F0F
LIBF MIOF 42AF 3F12
LIBF MCOMP 428B 3F18
CALL VIAQ 4680
CALL PRD 471D
LIBF SUBIN 475A 3F10
LIBF FADD 4784 3F1E
LIBF IOU 4832 3F21
CALL IOFIX 48CC
CALL BTIBT 48FC
CALL SAVE 4868
CORE 4962 369E
CLB, LOG60 LD X0
```

---

```
This program outputs the shift log:

INTEGER SW0, SW1, SW2, SW3, SW4, SW5, DAY
INTEGER VALUE(I), SETPT(I), COUNT(I), OFFLN(I)
DIMENSION RANGE(I), AHL(I), ALL(I), A(I), B(I), BASE(I), IBASE(I), IBASEZ(I), G(I), H(I), IENDT(I), IPERD(I), ICTNT(I)

PRINT HEADER
CALL PTIME(TIME)
WRITE(1,1)DAY, TIME
WRITE(2,1)DAY, TIME
WRITE(3,1)DAY, TIME
WRITE(4,1)DAY, TIME, FORMAT(1X,'SHIFT END LOG, DAY', I3, 5X, 'TIME', F9.2)

OUTPUT SHIFT END LOG
EXIT FROM ROUTINE

CALL VIAQ

VARIABLE ALLOCATIONS

<table>
<thead>
<tr>
<th>CSDW(I)</th>
<th>CSW1(I)</th>
<th>CSW2(I)</th>
<th>CSW3(I)</th>
<th>CSW4(I)</th>
<th>CSW5(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0FFFFF</td>
<td>FFFE</td>
<td>FFFD</td>
<td>FFFC</td>
<td>FFF8</td>
<td>FFFA</td>
</tr>
</tbody>
</table>

STATEMENT ALLOCATIONS

1 = 0005

FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS

CALLED SUBPROGRAMS

PTIME VIAQ MWRT MCOMP MIOF MIOF TYPEWR MEBPR

INTEGER CONSTANTS
1 = 0002
4 = 0003
5 = 0004
```

---

The rest of the text continues with the program listing and comments, detailing the variables, calls, and functions used in the program.
CORE REQUIREMENTS FOR SHIFT
COMMON 0 INSKEL COMMON 464 VARIABLES 2 PROGRAM 52

END OF COMPILATION

SHIFT
DUP FUNCTION COMPLETED
// DUP
*STORECIL M 1 SHIFT SHIFT RSTAR
*CCEND

CLB, BUILD SHIFT

CCRE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 3EB2 000C
*IBT TABLE 3EAE 0010
*PID TABLE 3EAB 0010
*ETV TABLE 3EBB 0051
*TVT TABLE 3FOC 0018
*IST TABLE 3F24 0036
*PNT TABLE 3F5A 0008
MAIN SHIFT 3F78
PNT SHIFT 3F8C
PNT RSTAR 3F60
LIBF EBPR7 3F98 3FOC
CALL PTIME 4044
LIBF MWRT 41FE 3FOF
LIBF MIDI 4284 3F12
LIBF MIDF 42AF 3F15
LIBF MCOMP 42BA 3F18
CALL VIAG 4680
CALL PRT 4710
LIBF SUBIN 4754 3F18
LIBF PADD 4784 3F18
LIBF IOU 4832 3F21
CALL IOFIX 48CC
CALL BTIBT 48FC
CALL SAVE 486B
CORE 4962 369E

CLB, SHIFT LD XQ

DUP FUNCTION COMPLETED

// JOB
// * MAINLINE CORE LOAD
// FOR WEEK
** LIST ALL
** MONDAY MORNING LOG
*IOTC(TYPewriter)

THIS PROGRAM OUTPUTS THE WEEKLY MONDAY MORNING LOG

INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),CDUMT(8),DFPLMN(8)
DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),HIGH(8),H(16),G(16),X(16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
PT,CDUMT,DFPLM,AML,ALL,A,B,IBASE,IBASZ,IBAZZ,G,H,ENDT,IPERD,ITCNT
Z,IPONT

PRINT HEADER

CALL PTIMEITIMEI
WRITE(4,1IDAY,TIME
WRITE(5,1IDAY,TIME
FORMAT('MONDAY MORNING REPORT

EXIT FROM PROGRAM

CALL VIAQ
END
VARIBLE ALLOCATIONS
CSW1(I*)=FFFF CSW2(I*)=FFFF CSW3(I*)=FFFF CSW4(I*)=FFFF CSW5(I*)=FFFFC
DAY(I*)=FFFF JOBNI(I*)=FF8 VALUE(I*)=FFFFF-FFFF9 AML(I*)=FF6F-FFF6A ML(R*)=FF66-FFFF8 AML(I*)=FF66-FFFF8 RSTAR(I*)=FF65-FFFF4 M(BR)=FFE6-FF76 M(BR)=FFE6-FF76
COUNT(I*)=FFC7-FFCC M(I*)=FF77 IBASEI(I*)=FE77 IPERDII(I*)=FE32
STAT~MENT ALLOCATIONS
1  =0004
FEATURES SUPPORTED
ONE WORD INTEGERS IOCS
CALLED SUBPROGRAMS
PTIME VIAQ MWRT MCOMP MIOF MIOI TYPEN EBPR
INTEGER CONSTANTS
M=0002 5=0003
CORE REQUIREMENTS FOR WEEK
COMMON 0 INSKEL COMMON 464 VARIABLES 2 PROGRAM 48
END OF COMPILATION

WEIGHT
DUP FUNCTION COMPLETED
// DUP
#STOKECJ C M 1 WEEK WEEK RSTAR
#CCEND

CLB, BUILD WEEK
CORE LOAD MAP
TYPE NAME ARG1 ARG2
#CDW TABLE 3E82 000C
#TBT TABLE 3E8E 0010
#FIO TABLE 3EAB 0010
#ETV TABLE 3EB8 0051
#VTV TABLE 3F0C 0018
#STT TABLE 3F24 0036
#PNT TABLE 3F5A 0008
MAIN WEEK 3F7F
PNT WEEK 3F5C
PNT RSTAR 3F5D
LIBF EBPR 3F94 3F0C
CALL PTIME 4040
LIBF MWRT 41FA 3F0F
LIBF MID 4280 3F12
LIBF MID 4280 3F15
LIBF MCOMP 4287 3F18
CALL VIAQ 46AC
CALL PRT 47UC
LIBF SUBIN 475F 3F18
LIBF FADD 4780 3F1E
LIBF T0B 482E 3F21
CALL IDPX 48C8
CALL 8TBT 48F8
CALL SAVE 49B4
CORE 495E 36A2
CL4, WEEK LD XQ
DUP FUNCTION COMPLETED

// JOB
/* MAINLINE CORE LOAD
// FOR TREND
#LIST ALL
** TREND LOG
#IOCS(TYPWRIT) C
C THIS IS THE TREND LOG CORE LOAD. IT READS THE VALUE THAT THE
C OPERATOR HAS ASKED. IT IS QUEUED PERIODICALLY BY THE TCONT
C SUBROUTINE WITH THE PERIOD SPECIFIED BY THE OPERATOR.
C
INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(I*)SETPT(I*),COUNT(I*),OFFLN(I*)
DIMENSION RANGE(8), AHL(40), ALL(40), A(40), R(40), LDWW(8), GI(16), H(16)
COMMON/INSKEL/SWO, SW1, SW2, SW3, SW4, DAY, JOBN, VALUE, RANGE, LOW, SET
1P, COUNL, DELFLN, AHL, ALL, A, IBASE, IBAZ, I, H, END, IPERD, ITCNT
2, IPONI
CALL AIP(11000, INV, IPONI)
1 CALL AIP(10, ITEST)
GO TO (12), ITEST
2 CONTINUE
IF(4096-IPONI)4,100
4 =IPONI-4096+1
INV =INV-LOW*100./RANGE
5 FORMAT(' TRED LOOP',12, VALUE*)
CALL VIAQ
100 K =IPONI+1
J =41-K
VAL =AIK*INV+BIIK
WRITE(11, 101)K, VAL
101 FORMAT(' TRED OP=GUIDE',13, VALUE*F10.2)
CALL VIAQ
END
STATEMENT ALLOCATIONS
5 =0010 101 =001F 1 =0036 2 =0040 4 =0046 100 =0074
FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS
CALLED SUBPROGRAMS
AIP VIAQ FADDX FMFY FDIVX FSTO IFIX FLOAT COMGO MWRT MCOMP MIOF MI01 SUBSC
TYPEN EBPRT
REAL CONSTANTS
.100000E 03=0008
INTEGER CONSTANTS
11000=000A 0=000B 4096=000C 1=000D 9=000E 41=000F
CORE REQUIREMENTS FOR TREND
COMMON 0 0 INSKEL COMMON 464 VARIABLES 8 PROGRAM 144
END OF COMPIIATION

TREND
DUP FUNCTION COMPLETED
*DELET M TREND DUM
TREND
D25 NAME NOT IN L/F
*STORECIL M 1 TREND TREND RSTAR
*GEND
CLB, BUILD TREND

CORE LOAD MAP
TYPE NAME ARG0 ARG1 ARG2
=CDW TABLE 3E2 000C
*IST TABLE 3EB 001D
*PID TABLE 3EB 001D
*ETV TABLE 3EB 0051
*VTV TABLE 3FC 0018
*IST TABLE 3F2 0036
*PNT TABLE 3F5 000B
MAIN TREND 3F5
PNT TREND 3F5
PNT RSTAR 3F0
L18F EBPRT 3FA 3F0C CALL AIP 409A
L18F MWRT 4250 3F0F
L18F MIOF 4306 3F12
L18F MCOMP 420D 3F15
CALL VIAQ 4702
L18F FADDX 477C 3F18
L18F MIOF 4301 3F1B
CALL PRT 4800
LIBF AIPTH 484A 3F1E
LIBF 1DU 48CC 3F21
CALL 1DFIX 4966
CALL BTBT 4966
CALL SAVE 4902
CORE 49FC 3604
CLB, TREND LD X0
DUP FUNCTION COMPLETED

// JOB A
// = INTERRUPT CORE LOAD
// FOR COOLP
#LIST ALL
** CHANGE OP-GUIDE LIMITS PROGRAM
#IOCS(TYPEWRITER)
#IOCS(DISK)

C
  THIS CORE LOAD CHANGES THE LIMITS ON OPERATOR GUIDE POINTS AT
  OPERATOR REQUEST.

C
  INTEGER 5W0,SW1,SW2,SW3,SW4,SW5,DAY
  INTEGER VALUE(151,SETPT(81,COUMT(81,OFFLN(81
  DIMENSION INVAL(81
  DIMENSION RANGE(81,AHL(81,ALL(81,A(81,B(81,LOW(81,G(81,H(81
  COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET

1PT,COU,T,OLFLN,AHL,ALL,A,IAA,BASE,IBAZZ,G,H,IENT,Y,IPERD,ITC
2,INTON

DEFINE FILE
  WRITE(1,5551
      555 FORMAT(//1
      CALL eONVR(INVAL)
      WRITE(1,302IINVAL
      302 FORMAT(8I5)
      IPT=INVAL(11*10+INVAL(9)
      AAA=INVAL(21*1000.+INVAL(31*100.+INVAL(4)*100.+INVAL(5)*10.+INVAL
      60+INVAL(71*11+INVAL(81*0.1
      BB=INVAL(101*1000.+INVAL(111*100.+INVAL(121*10.+INVAL(131*9.1
      IVAL(141*INVAL(151*11+INVAL(161*0.1
      IF(IPT<300,300,1
      K=41-IPT
      IF(K<300,300,2
      AHL(K)=AA
      ALL(K)=BB
      WRITE(3,3IAHL,ALL
      CALL PTIME(TIME)
      WRITE(1,3)IPT,AA,BB,DAY,TIME
      3
      FORMAT(' OP-GUIDE PI' 13,
      CALL INTEX
      WRITE(1,3011
      301 FORMAT(' INVALID ENTRY OP-REQUEST PI')
      CALL INTEX

END

VARIABLE ALLOCATIONS
CSWO(I*)=FFFF
CSW1(I*)=FFFF
CSW2(I*)=FFFF
CSW3(I*)=FFFF
CSW4(I*)=FFFF
CSW5(I*)=FFFF
DAY(I*)=FFFF
JOBN(I*)=FFFF
VALUE(I*)=FFFF
RANGE(I*)=FFFF
BILL(81=FFFF
OFFLN(I*)=FFFF
AHL(R*)=FFFF
ALL(R*)=FFFF
A(I*)=FFFF
B(I*)=FFFF
L(I*)=FFFF
G(I*)=FFFF
H(I*)=FFFF
IEN(1*)=FFFF
IPERD(I*)=FFFF
ITC(T111=FFFF
INVAL(I*)=0027-0018
IPT(I*)=0028

STATEMENT ALLOCATIONS
555 =004A 302 =0040 3 =0050 301 =0071 1 =0132 2 =013C 300 =0168
FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS

CALLED SUBPROGRAMS
CONVR PTIME INTEX FADD FMPY FLD FSTO FSTOX FLOAT MWRT MCMP MIDAI MIOF MID1 SUBSC
MDWRT MDCOM MDAF TYPEN EBPR

REAL CONSTANTS
.100000E 05=003A .100000E 04=003C .100000E 03=003E .100000E 02=0040 .100000E 00=0042 .100000E-01=0044

INTEGER CONSTANTS
1=0046 10=0047 41=0048 3=0049
Continued

CORE REQUIREMENTS FOR COGLP
COMMON 0 INS KEL COMMON 464 VARIABLES 58 PROGRAM 308

END OF COMPI LATION

COGLP
DUP FUNCTION COMPLETED
*DELET 1 COGLP IDUM 9999
COGLP
025 NAME NOT IN L/F
*STORECIL 1 COGLP COGLP COGLP 1100
*CCEND

CLB, BUILD COGLP

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 3E82 000C
*IBT TABLE 3E8E 0010
*FID TABLE 3E8B 0010
*ETV TABLE 3E88 0001
*VIV TABLE 3FOC 0027
*PNT TABLE 3F34 0004
*DFT TABLE 3F38 0006
MAIN COGLP 3F89
PNT COGLP 3F36
LIB EBPR 3E82 3F0C
LIB MKRT 3E80 3F0F
LIB MKOM 3E80 3F12
CALL CONVR 3E88
LIB MKOA 3E87 3F15
LIB MDAR 3F0C 3F18
LIB MDAP 3E81 3F16
LIB MDOM 3E84 3F21
CALL PTIME 4F0A
LIB MIOT 3E86 3F24
LIB MIOM 3E81 3F27
CALL PRT 4F0A
LIB I0U 4E54 3F2A
CALL INI 4CEE
CALL BT1BT 4E1E
CALL SAV 4C8A
LIB SUBIN 4082 3F2D
CALL GETVL 4DE6
LIB PADOX 47DB 3F30
CALL BT2BT 4E82
CORE 4E80 315F

CLB, COGLP LD XQ

DUP FUNCTION COMPLETED

// JOB A
// = INTERRUPT CORE LOAD
// FOR CCCLSP
*LIST ALL
** CHANGE OF CLOSED LOOP CONTROL SET POINT
#IDCS(TYPEWRITER)

#IGCS(DIRISK)

C
C THIS CORE LOAD CHANGES THE SET POINT VALUE FOR A SET POINT
C STATION UPON OPERATOR REQUEST.
C
INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),COUNT(9),DFLN(8)
DIMENSION INVAL(16)
DIMENSION RANG(8),AH(40),AL(40),A(40),B(40),LOW(8),G(16),H(16)
COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
JPT,COUNT,DFLN,AH,AL,A,B,IBASE,IBASZ,IBAZZ,G,H,IENDT,IPERD,ITCNT
2,IFONT
DEFINE FILE 3(3,320,U,II)
WRITE(1,555)
Continued

555 FORMAT(1/)
CALL CONVR(INVAL)
WRITE(1,100)INVAL
100 FORMAT(8/)
CALL PTIME(TIME)
I=INVAL(I)
J=INVAL(I)*10+INVAL(J)
IF(I)300,300,1
1
K=9-1
IF(K)300,300,2
2
IF(J)300,300,3
3
SETPT(K)=J*RANGE(K)/100.+LOW(K)
WRITE(3)SETPT
WRITE(1,4)I,J,DAY,TIME
~
FORMAT('LOO,P13,'NEW SET POINT','14,'DAY'I2,'TIME'F1.2)
CALL INTEX
300 WRITE(1,301)
301 FORMAT('INVALID ENTRY OP-REQUEST P2')
CALL INTEX
END
VARIABLE ALLOCATIONS
CSWO(I*)=FFFF
DAY(I*)=FFF9
COUNT(I*)=FFD7
IBASE(I*)=FE77
IPERD(I*)=FE32
CWS(I*)=FFFE
JOBN(I*)=FFFE
VALUES(R*)=FFE7
IPRINT(I*)=FE31
CSW2(I*)=FFFD
JAVN(I*)=FFE7
JBASE(I*)=FFB7
CWS3(I*)=FFFC
JGNT(I*)=FFE8
JBASE(I*)=FFB8
CWS4(I*)=FFFF
LOW(R*)=FFD8
IPON(I*)=FE30
CSW5(I*)=FFFA

STATEMENT ALLOCATIONS
555 =0029 100 =002C 4 =002F 301 =0047 1 =007F 2 =0089 3 =0080 300 =008F
555 =0029 100 =002C 4 =002F 301 =0047 1 =007F 2 =0089 3 =0080 300 =008F

FEATURES SUPPORTED
ONE WORD INTEGERS
10C

CALLED SUBPROGRAMS
CONVR PTIME INTEX FADD FMPYX FDIV FSTO IFIX FLOAT ISTOX MWRT MCOMP MIAI MIF MIOI

REAL CONSTANTS
+100000E 03=0022

INTEGER CONSTANTS
1=0024 10=0025 9=0026 3=0027 2=0028

CORE REQUIREMENTS FOR CCLSP
COMMON 0 INSKEL COMMON 464 VARIABLES 34 PROGRAM 164

END OF COMPILATION

CCLSP
DUP FUNCTION COMPLETED
*DELETE I CCLSP IDUM 9999
CCLSP
D25 NAME NOT IN L/F
*STORECIL I 1 CCLSP CCLSP CCLSP 1101
*FILE'S(3,FILE3,1)
*CCEND

CLB, BUILD CCLSP

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 3E82 00C
*INT TABLE 3E8E 0010
*FID TABLE 3EAB 0010
*CTV TABLE 3E88 0051
*VIV TABLE 3E82 0027
*PNT TABLE 3F34 0004
*OPT TABLE 3F3F 0006
MAIN CCLSP 3F8F
PNT CCLSP 3F66
LIBF EBPRN 3F8F
LIBF MWRT 4228 3F0F
LIBF MCOMP 4265 3F12
CALL CONVR 46E0
LIBF MIAI 42FF 3F15
CALL PTIME 4720
LIBF PADD 4770 3F18
LIBF MWRT 4941 3F1B
LIBF MIAI 4B02 3F1E
LIBF MCOMP 4BB8 3F21

Programming Techniques 259
Continued
LIBF MIDI 420E 3F24
LIBF MIDF 420D 3F27
CALL PRT 4B62
LIBF IOU 4BAC 3F2A
CALL IOFIX 4C66
CALL BT1BT 4C76
CALL SAVE 4B62
LIBF SUBN 4GDA 3F2D
CALL GETV 4BE2
LIBF PAOX 4BAC 3F24
CALL BT2BT 4B62
CORE 4DFB 3207
CLB, CCLSP LD XG
DUP FUNCTION COMPLETED

// JOB
// * INTERRUPT CORE LOAD
// FOR MGRTP
/// LIST ALL
/// MODIFY GRADE RUN TIME PROGRAM
/// IDCS(TYPewriter)
/// IDCSI(DISK)

 ** THIS CORE LOAD CHANGES THE RUN TIME FOR A GRADE UPON OPERATOR REQUEST.**

INTEGER SW0, SW1, SW2, SW3, SW4, SW5, DAY
INTEGER VALUE(15), SETPT(8), COUTN(8), OFFLT(8)
DIMENSION INVALID(16)
DIMENSION RANGE(8), AHTN(40), ALLT(40), IBASE(8), IBASE(8), G1, IEND, IPER, ITCNT
2, IPMT
DEFINE FILE 313, 320, U, II)
WRITE(1, 555)
555 FORMAT(/
CALL CONVR(INVALID)
WRITE(1, 302) INVALID
302 FORMAT(8I5)
I=INVALID(5)*10+INVALID(6)
J=INVALID(7)*10+INVALID(8)
IF(I24=1M) 300, 300, 20
20 IF(I16=1M) 300, 300, 21
21 CALL CLOCK(3)
K=I*100+J/100
IF(K-I)3, 2, 2
IPER=K-I
GO TO 3
3 AA=IPER*3.6
4 WRITE(4, 40)
40 FORMAT(' TOO LONG OF A RUN TIME')
CALL INTEX
5 IEN=AA
WRITE(391) JOBN, DAY, IEND, SW3
TIME=I*100+M/100.
WRITE(1, 6) TIME
6 FORMAT(' JOB WILL NOW TERMINATE AT 7.2)
CALL COUNT(1, 2, IPER)
CALL INTEX
300 WRITE(1, 301)
301 FORMAT(' UNKNOWN ENTRY 1P-REQUEST P3')
CALL INTEX
END

VARIABLE ALLOCATIONS
CSWO(I*)=FFFF
CSW1(I*)=FFFF
CSW2(I*)=FFFF
CSW3(I*)=FFFF
CSW4(I*)=FFFF
CSW5(I*)=FFFF

STATEMENT ALLOCATIONS
555 =0036 302 =0039 40 =003C 6 =004A 301 =005A 20 =009B 21 =009E 1 =00B2 2 =00C4 3 =00CA
4 =00D8 5 =00E 300 =010C

FEATURES SUPPORTED
ONE WORD INTEGERS
IDCS

CALLED SUBPROGRAMS
CONVR CLOK INTEX COUNT FSUB FM" Y FOIV FLD FSTD IFIX FLOAT LDAC MWRT MCOMP MIOAI
MIOF MWRT MCUM MDI TYPEN EBPR
REAL CONSTANTS
.360000E 01=0026 .320000E 05=0028 .100000E 03=002A

INTEGER CONSTANTS
1=002C 10=002D 24=002E 60=002F

CORE REQUIREMENTS FOR MGRTP
COMMON 0 INSKEL COMMON 464 VARIABLES 38 PROGRAM 236

END OF COMPILATION

MGRTP
DUP FUNCTION COMPLETED
*STORECIL I 1 MGRTP MGRTP MGRTP 1102
*FILES(3,FILE3,1)
*CCEND

CLB, BUILD MGRTP

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 3E82 000C
*IBT TABLE 3E8E 001D
*ITV TABLE 3EB8 0051
*VTY TABLE 3F0C 0027
*PMT TABLE 3F34 0004
*DFT TABLE 3F3B 0006
MAIN MGRTP 3F32
PNT MGRTP 3F36
LIBF EBPR 404A 3FOC
LIBF MWR 4274 3F0F
LIBF MCOMP 4301 3F12
CALL CONVR 472C
LIBF MDIP 434B 3F15
LIBF FSUB 4774 3F18
LIBF LDFF 47FE 3F1B
LIBF MDWR 497F 3F21
LIBF MDCM 48F6 3F24
LIBF MIF 4325 3F27
CALL PRT 4BAO
LIBF IOU 4BEA 3F2A
CALL IOFIX 4CR4
CALL BTIBT 4CR4
CALL SAVE 4C70
LIBF SRRN 4018 3F20
CALL GETVL 4C7C
LIBF FADDX 477A
CALL BT2BT 4E18
CORE 4E36 31C9

CLB, MGRTP LD XQ

// JOB
// * INTERRUPT CORE LOAD
// FOR CPJSP
#LIST ALL
** CHANGE PROCESS JOB SEQUENCE
*IOCSI(TYPewriter)
*IOCSI(DISK)
C

THIS CORE LOAD CHANGES THE SEQUENCE OF GRADES UPON OPERATOR
C REQUEST.
C
INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),COUNT(8),DFLLN(8)
DIMENSION INVAL(16)
DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
IPR,COUNT,DFLLN,AHL,ALL,A,B,IBASE,IBASEZ,IBASEZ,Z,H,END1,IPERD,IPONT
2,IPONT
DEFINE FILE 3(13,320,0,11)
WRITE(1,555)
Continued

555 FORMAT(1//)
CALL CONVIVAL
WRITE(1,1)INVAL
1 FORMAT(15)
I=INVAL(7)*10+INVAL(8)
AA=INVAL(4)*1000.+INVAL(5)*100.+INVAL(6)*10.+I
IF(AA.<4,22
2 IF(I32000.,AA)300,300,2
3 =100
4 JOBN=1
WRITE(1,1)JOBN,DAY,INDEX,SW
CALL INTEX
300 WRITE(1,301)
301 FORMAT(1,1)INVALID ENTRY ID-REQUEST P4
CALL INTEX
VARIABLE ALLOCATIONS
CSWO(1*)=FFFF CSW1(1*)=FFFE
JOBN(1*)=FFFF VALUE(1*)=FFF7-FFFE
COUNT(1*)=FFC7-FFC0 DPLIN(1*)=FFBF-FFB8
IBASE(1*)=FF77 IBASII(1*)=FF76
INDEX(1*)=FF32 FIRST(1*)=FF31
INVAL(1*)=001F-0010
STATEMENT ALLOCATIONS
555 =0034 1 =0037 5 =003A 301 =0056 22 =00AC 2 =00B3 3 =00B7 4 =00B8 300 =00E1
FEATURES SUPPORTED
ONE WORD INTEGERS
IOCS
CALLED SUBPROGRAMS
CONVIVAL PTIME INTEX FADD FSUB FMPY FLD FSTD IFIX FLOAT LDFA0 MWRT MCOMP MIDAI MIOF
MID0 MDWT MDCOM MDI TYPEN EBPR
REAL CONSTANTS
.1000000E 05=0028 .1000000E 04=002A .1000000E 03=002C .3200000E 05=002E
INTEGER CONSTANTS
1=0030 10=0031 100=0032 3=0033
CORE REQUIREMENTS FOR CPJSP
COMMON 0 INSKEL COMMON 44 VARIABLES 40 PROGRAM 192
LIBF MOWTR 495B 3F21
LIBF MDF 481+ 3F24
LIBF MDCOM 4902 3F27
CALL PTIME 488B
LIBF MDFI 4306 3F2A
LIBF MI0F 4301 3F2D
CALL PRT 4888
LIBF IOU 4C02 3F30
CALL IOFIX 4C9C
CALL BT1BT 4CCC
CALL SAVE 4C38
LIBF SUGIN 4303 3F33
CALL GE1V 4994
LIBF PADDX 4766 3F36
CALL BT2ST 4E30
CORE 4E4E 31B1
CLB, CPJSP LD XQ

DUP FUNCTION COMPLETED

// JOB A
// * INTERRUPT CORE LOAD
// FOR STRND
/* LIST ALL */
** START TREND LOG ROUTINE
* IOCS(TYPewriter) */

C
 THIS CORE LOAD INITIATES A TREND LOG OF THE POINT SPECIFIED BY
 THE OPERATOR.
C
INTEGER SWO,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),COUNT(8),OFFLN(8)
DIMENSION RANGE(8),AHL(40),ALL(40),A(40),B(40),LOW(8),G(16),H(16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,OAY,JOBN,VALUE,RANGE,LOW,G,H,IENOT,IPERD,ITCNT
WRITE(1,555)
555 FORMAT(/)
CALL CONVR(INVAL)
WRITE(1,1)INVAL
1 FORMAT(4I5)
II=INVAL(6)*100+INVAL(7)*10+INVAL(5)
JJ=INVAL(14)*100+INVAL(15)*10+INVAL(16)
IF(INVAL(1)EQ0,100,101)
IF(INVAL(3)EQ0,300,301)
300 WRITE(1,301)
301 FORMAT(' INVALID ENTRY OP-REQUEST PS')
CALL INTEX
END

STATEMENT ALLOCATIONS
555 =0026 1 =0029 5 =002C 104 =0049 301 =0061 2 =0087 3 =00C8 4 =00D4 100 =00E4 101 =00EC
102 =00F2 103 =00FA 200 =010E 300 =0110

Programming Techniques 263
Continued

FEATURES SUPPORTED
ONE WORD INTEGERS

CALLED SUBPROGRAMS

CONVR COUNT INTEX MWRX MCDMP MIDAI MIDIX MID1 TYPEN EBPR1

INTEGER CONSTANTS
1=0001 10=001F 10=0020 10=0021 8=0022 4095=0023 2=0024 3=0025

CORE REQUIREMENTS FOR STRND
COMMON 0 INSKEL COMMON 464 VARIABLES 30 PROGRAM 262

END OF COMPILATION

STRND
DUP FUNCTION COMPLETED
*DELETE I STRND IDUM 9999
STRND
025 NAME NOT IN L/F
*STORECIL I STRND STRND 1104
*CEEND

CLB, BUIIL STRND

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*CDW TABLE 3E82 000C
*IBT TABLE 3E8E 0010
*FT0 TABLE 3E8B 0010
*ETV TABLE 3E8B 0091
*VTY TABLE 3FOC 0018
*PNT TABLE 3F2F 0004
MAIN STRND 3F9D
PNT STRND 3F2A
LIBF EBPRT 4050 3FOC
LIBF MWRX 427A 3FOF
LIBF MCDMP 4307 3F12
CALL CONVR 4732
LIBF MIDAI 4351 3F15
LIBF MID1 4330 3F18
LIBF MIDIX 433C 3F18
CALL PRT 4766
LIBF IDU 4780 3F1E
CALL IOFIX 484A
CALL BT1BT 487A
CALL SAVE 47E6
LIBF SUBIN 48DE 3F21
CALL GETVL 4942
LIBF FADDX 49FB 3F24
CORE 4A7E 3F91

CLB, STRND LD X0

DUP FUNCTION COMPLETED

// JOB A
// FOR AIMON
*LIST ALL
** ANALOG INPUT LOG ROUTINE
*IOCS(TYPETWRITER)
C
C THIS CORE LOAD LOGS ALL INFORMATION ABOUT ANY ANALOG INPUT POINT
C ON THE SYSTEM UPON OPERATOR REQUEST.
C
C POINTS ARE SELECTED AS FOLLOWS --
C OP-GUIDE POINTS - D1=0, D7=01-40
C CLOSED-LOOP POINTS - D1=1, D7=01-08
C
C INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
C INTEGER VALUE(15),SETPT(8),COUNT(8),OFFLNCB)
C DIMENSION INVAL(16)
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMMON/INSKEL/SW0,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
C COUNT,OFFLNC
C
C DIMENSION VALUE(15),SETPT(8),COUNT(8),OFFLNCB
C COMM
WRITE(1,555)
555 FORMAT(//)
C READ DATA ENTRY DIALS
C CALL CONVR(INVAL)
WRITE(1,900)INVAL
900 FORMAT(8I5)
C CONVERT POINT DESIGNATION, D7-8
C IX1 = INVAL(7)*10 + INVAL(8)
C MUST BE GREATER THAN 0
C IF(IX1)1800,800,10
C CHECK D1
C IF(IX1)1=120,500,800
C CHECK INPUT OP-GUIDE POINT DESIGNATION
C IF(IX1)40130,30,800
C READ OP-GUIDE POINT
C 30 IX2 = IX1-1
CALL AIP(11000,VALUE(1),IX2)
35 CALL AIP(0,IX2)
GO TO (35,40), IX2
C CONVERT VALUE SELECTED
C 40 IX2 = 41-IX1
VAL = A(IX2)*VALUE(1) + B(IX2)
C PRINT TIME AND HEADER
C CALL PTIME(TIME)
WRITE(1,920)DAY,TIME
920 FORMAT(II2,1 TIMEIF7.2)
WRITE(1,930)
930 FORMAT (I OP-GUIDE POINT VALUE HIGH LIMIT LOW LIMIT FAC
-TOR A FACTOR B)
WRITE(1,940)IX1,VAL,AHL(IX2),ALL(IX2),A(IX2),B(IX2)
CALL INTEX
940 FORMAT(C7X,12,4X,F10.2,lX,F10.2,2X,F10.2,5X,E13.6,3X,E13.6)
C CLOSED-LOOP POINT, CHECK LIMIT ON DESIGNATION
C 500 IF(IX1)8510,510,800
510 IX2 = IX1+1095
CALL AIP(01000,VALUE(1),IX2)
520 CALL AIP(0,IX2)
GO TO (520,530), IX2
C CONVERT SETPT FOR OUTPUT
C 530 IX2 = 9-IX1
VALUE(1) = (VALUE(1)-LOW(IX2))*100./RANGE(IX2)
IVAL = (SETPT(IX2)-LOW(IX2))*100./RANGE(IX2)
C PRINT DAY, TIME, AND HEADER
C CALL PTIME(TIME)
WRITE(1,950)DAY,TIME
950 FORMAT (I CLO-OPEN-LOOP POINT SETPT OFFLN
- RANGE LOW)
C WRITE VALUES
C WRITE(1,960)IX1,VALUE(1),IVAL,OFFLN(IX2),RANGE(IX2),LOW(IX2)
CALL INTEX
960 FORMAT(BX,I2,7X,18,5X,16,10X,I2,7X,F10.2,4X,I6)
800 WRITE(1,810)
CALL INTEX
810 FORMAT(- INVALID ENTRY OP-REQUEST P61)
END
VARIABLE ALLOCATIONS
CSW1(IX*)=FFFF
CSW2(IX*)=FFFF
CSW3(IX*)=FFFF
CSW4(IX*)=FFFF
CSW5(IX*)=FFFF
DAY(IX*)=FFFF
JOBN(IX*)=FFFF
VALUE(IX*)=FFFFFF
RANGE(IX*)=FFFF6
LOW(IX*)=FFFF7
SETPT(IX*)=FFFF8
COUNT(IX*)=FFFF9
OFFLN(IX*)=FFFFD
AHL(IX*)=FFFF8
ALL(IX*)=FFFF8
A(IX*)=FFFF7
B(IX*)=FFFF7
IBASE(IX*)=PET7
IBASE2(IX*)=PET7
IBASE22(I*)=FE76
IBASE22(I*)=FE75
IBASE22(I*)=FE74
IBASE22(I*)=FE33
Programming Techniques 265
STATEMENT ALLOCATIONS

<table>
<thead>
<tr>
<th>555</th>
<th>=0028</th>
<th>900</th>
<th>=002B</th>
<th>920</th>
<th>=0039</th>
<th>940</th>
<th>=0062</th>
<th>950</th>
<th>=006F</th>
<th>960</th>
<th>=0097</th>
<th>B10</th>
<th>=0044</th>
<th>10</th>
<th>=0003</th>
<th>20</th>
<th>=0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>=00E3</td>
<td>35</td>
<td>=00F5</td>
<td>40</td>
<td>=00FF</td>
<td>50</td>
<td>=013E</td>
<td>510</td>
<td>=0144</td>
<td>520</td>
<td>=0156</td>
<td>530</td>
<td>=0160</td>
<td>800</td>
<td>=0188</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FEATURES SUPPORTED
ONE WORD INTEGERS
IDCS
CALLED SUBPROGRAMS
CONVR AIP PTIME INTEX FADDX FMKY FDIVX FLOX FSTO IFIX FLOAT COMGO ISTDX MWRT MCOMP
MIOA1 MIOFX MIOIX MIOF MIDI SUBSC TYPEN EBPT
REAL CONSTANTS
1.00000E 03=001C
INTEGER CONSTANTS
1=001E 10=001F 40=0020 11000=0021 0=0022 4I=0023 8=0024 4095=0025 1000=0026 9=0027
CORE REQUIREMENTS FOR AIMON
COMMON 0 INSHEL COMMON 464 VARIABLES 28 PROGRAM 422

END OF COMPILATION

AIMON
DUP FUNCTION COMPLETED
#DELET 1 AIMON IDUM 9999
AIMON
DZS NAME NOT IN L/F
#STDHECIL 1 AIMON AIMON 1105
#CCEND

CLR, BUILD AIMON
CORE LOAD MAP
TYPE NAME ARG1 ARG2
=COW TABLE 3E82 000C
#IBT TABLE 3E8E 0010
#PIO TABLE 3E8B 0010
#ETV TABLE 3E8B 0051
#VTV TABLE 3FOC 0027
#PNT TABLE 3F34 0004
MAIN AIMON 3FEC
PNT AIMON 3F36
LIBF EBPT 40FA 3FOC
LIBF MWRT 4324 3FOF
LIBF MCOMP 4381 3F12
CALL CONVR 470C
LIBF MIOA1 43F8 3F15
CALL AIP 4810
LIBF FADDX 4A56 3F18
CALL PTIME 48E6
LIBF MIDI 430A 3F18
LIBF MIOF 4305 3F1E
LIBF MIOFX 43E1 3F21
LIBF MIDIX 43E6 3F24
CALL PRT 4916
LIBF IDU 4960 3F27
CALL IOFIX 49F9
CALL STIBT 442A
CALL SAV 4996
LIBF SUBIN 4A8E 3F2A
CALL GETVL 4AF2
LIBF AIPTN 4BBE 3F2D
LIBF FADD 4BC5 3F30
CORE 4C12 33ED
CLR, AIMON LO XQ
DUP FUNCTION COMPLETED
<table>
<thead>
<tr>
<th>LET 0000 0000</th>
<th>SDUR 0000 0000</th>
<th>IQTR 0002 0016</th>
<th>CHAIN 0016 0076</th>
<th>INTER 0028 0139</th>
<th>SHARE 0039 0156</th>
<th>TEMP 0176</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0 0040 0540</td>
<td>E0 0040 0540</td>
<td>E0 0040 0540</td>
<td>E0 0040 0540</td>
<td>E0 0040 0540</td>
<td>E0 0040 0540</td>
<td>E0 0040 0540</td>
</tr>
</tbody>
</table>

Programming Techniques
FLET
PACK LABEL
11111

*DPRS 0051 1118
*FDOS 0000 1169
*MESS 00A3 1178
*/EPDM 7FFF 121B
*/EPVS 0780 1202
*/INSV 40FF 128B
/NPSV 4180 12C3
FILE1 0002 12F8
FILE2 0064 12FA
FILE3 0003 135E
COLDN 05DC 1361
COLD 00D6 1366
GRADE 116A 138E
SCANZ 09E 1382
SCINT 0A26 1382
OGMD 012C 1389
OGUM 0094 138C
OGUM 013A 938D
WEEK 0A26 13C3
TRENO 087A 13CC
COLDP 10CE 13CD
FIEL2 0002 1374
FIEL2 0002 1389
FIEL2 0094 138C
FIEL2 013A 938D
PRWS 0005 1118
PACK LABEL

DUP FUNCTION COMPLETED

// JOB A
// END OF ALL JOBS

// JOB A
// = NONPROCESS CORE LOAD
// = SPECIAL JOB TO SET UP FILES ON DISK
// FOR SPECL
*LIST ALL
** SPECIAL PROGRAM TO SET UP FILES ON DISK
*IDCS(DISK,1443 PRINTER)
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
THIS IS A SPECIAL ONE TIME ONLY CORE LOAD TO SET UP THE JOB
FILES ON DISK FOR TEST PURPOSES.
C
INTEGER SWO, SW1, SW2, SW3, SW4, SW5, DAY
INTEGER VALUE(15), SETPT(8), COUNT(8), OFFLN(8)
DIMENSION RANGE(8), LOW(8), IOCS(40)
COMMON/INSKEL/SWO, SW1, SW2, SW3, SW4, SW5, DAY, JOB1, VALUE, RANGE, LOW, SET
PT, COUNT, OFFLN, ALL, A, B, BASE, IBASE, IBAZ, IBAZ, GM, IDUM, IPER, IPTN
2, IPONT
DEFINE FILE 1(2,320, U, II)
DEFINE FILE 2(100,320, U, JOBN)
DO 10 J=1,40
A(J)=32000.
ALL(J)=-32000.
A(J)=-I.
10 B(J)=0.
DO 11 J=1,8
RANGE(J)=-32000.
11 LOW(J)=0.
WRITE(1,11) RANGE, LOW, A, B
DO 12 J=4..11
K=40-J
A(J)=K
12 ALL(K)=4.5
K=0
X=0
IIME=20*60
IX=1
DO 13 J=4,8
X=1
DO 13 J=1,8
13 SETPT(J)=0
JOB1=1
DO 100 J=1,100
SETPT(1)=50+J*20
SETPT(1)=50-J*20
SETPT(1)=50-J*20
WRITE(2,100) J, IIME, SETPT, ALL
WRITE(3,14) J, K, K
14 FORMAT(8I5)
WRITE(3,14) SETPT(1), SETPT(7), SETPT(6), SETPT(5), SETPT(4), SETPT(3), SETPT(2)
WRITE(3,14) SETPT(1)
DO 20 1=1,39,2
IC=4-I
10 CONTINUE
20 CONTINUE
15 FORMAT(24II,2F10.2)
100 CONTINUE
CALL EXIT
END

268
VARIABLE ALLOCATIONS

CONTINUED

STATEMENT ALLOCATIONS

FEATURES SUPPORTED

NONPROCESS

ONE WORD INTEGERS

IOCS

CALLED SUBPROGRAMS

REAL CONSTANTS

INTEGER CONSTANTS

CORE REQUIREMENTS FOR SPECL

COMMON 0 INSKE1 COMMON 464 VARIABLES 30 PROGRAM 340

END OF COMPILED
Continued

1 0 20
0 0 0 0 0 0 0 70 30
2 0 20
0 0 0 0 0 0 0 30 70
3 0 20
0 0 0 0 0 0 0 70 30
4 0 20
0 0 0 0 0 0 0 70 30
5 0 20
0 0 0 0 0 0 0 70 30
6 0 20
0 0 0 0 0 0 0 70 30

95 0 20
0 0 0 0 0 0 0 70 30
96 0 20
0 0 0 0 0 0 0 30 70
97 0 20
0 0 0 0 0 0 0 70 30
98 0 20
0 0 0 0 0 0 0 30 70
99 0 20
0 0 0 0 0 0 0 70 30
100 0 20
0 0 0 0 0 0 0 30 70

// JOB A
// ** NONPROCESS CORE LOAD
// FOR SCALEB
#LIST ALL
** CALIBRATION PROGRAM FOR SET POINT STATIONS
#IOCS(DISK,1443 PRINTER,CARD)
#NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
THIS NONPROCESS CORE LOAD IS FOR CALIBRATING THE SET POINT
C STATIONS.
C
INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY
INTEGER VALUE(15),SETPT(8),CUMT(8),DFPLN(8)
DIMENSION RANGE(8),AHL(40),ALL(40),ACHG(8),BCHG(8),LOW(8),G(16),H(16)
COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET
IP1,COUNT,DFPLN,AHL,ALL,A,B,IBASE,IBASZ,IBAZZ,H,IENDT,IPERD,ITCNT
2,IPONT
DEFINE FILE 112,320,U,111
100 READI2,111
1 FORMAT(I1)
1 IF (I1)100,100,2
2 J=9=1
K=4095=1
1 IF(J1100,100,3
3 WRITE(3,41)
4 FORMAT(* SET LOOP*12,* TO LOW VALUE*)
PAUSE
CALL AIP(11000,IL,K)
5 CALL AIP(0,11)
6 GO TO (15,61)
6 WRITE(3,9991)
WRITE(3,9991)
999 FORMAT(* VALUE IS*110)
LOW(J)=IL
WRITE(111,RANGE,LOW,A,B)
GO TO 100
END
VARIABLE ALLOCATIONS
CSWO(*)=FFFF CSW1(*)=FFFE
CSW2(*)=FFFF CSW3(*)=FFFC CSW4(*)=FFFF CSW5(*)=FFFF
CUMT(*)=FFFC FFCC DFPLN(*)=FFBF DFBB
IBASE(*)=FE77 IBASZ(*)=FE76 IBAZZ(*)=FE75 G(*)=FE72-FE74
IPERD(*)=FE32 ITCNT(*)=FE31 IPONT(*)=FE30 IIF(*)=0006
*HI(*)=0008
KII=0009 IL(1)=000A
Continued

STATEMENT ALLOCATIONS
1 0017 4 0019 7 0029 999 0038 100 0040 2 004A 3 005A 5 0067 6 0071 8 0082 9 =008C

FEATURES SUPPORTED
NONPROCESS
ONE WORD INTEGERS
IOCS
CALLED SUBPROGRAMS
AIP ЕСТОК FLOAT COMGO ISTIX MRED MWRT MCOMP MIDI SUBSC PAUSE MDWRT MDCOM MDAI MDAF

INTEGER CONSTANTS
2=000F 9=000F 4095=0010 3=0011 11000=0012 0=0013 1=0014 0=0015 0=0016

CORE REQUIREMENTS FOR SCALB
COMMON 0 INSREL COMMON 464 VARIABLES 14 PROGRAM 172

END OF COMPIGATION

SCALB
DUP FUNCTION COMPLETED
// DUP
*STORECIL 1 SCALB SCALB
*FILE(l,FILE1,1) *
CCEND

CLB, BUILD SCALB

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*GAD TABLE 3E82 000C
*IST TABLE 3E8E 0010
*FOO TABLE 3EAB 0010
*ETV TABLE 3E88 0051
*FTP TABLE 3F34 0004
*DPF TABLE 3F3B 0006
MAIN SCALB 3F78
PNT SCALB 3F36
LIBF MRED 3F3B 3F0C
LIBF MDWRT 40B5 3F21
LIBF MDWRT 40B3 3F24
LIBF MDAI 40B4 3F27
LIBF MDCOM 40BA 3F2A
CALL AIP 4084
CALL MDWRT 4085 3F21
CALL MDWRT 40B3 3F24
CALL MDWRT 40B4 3F27
CALL MDWRT 40BA 3F2A
CALL MDWRT 4084
CALL MDWRT 40B5 3F21
CALL MDWRT 40B3 3F24
CALL MDWRT 40B4 3F27
CALL MDWRT 40BA 3F2A
CALL MDWRT 4084
CALL MDWRT 40B5 3F21
CALL MDWRT 40B3 3F24
CALL MDWRT 40B4 3F27
CALL MDWRT 40BA 3F2A

DUP FUNCTION COMPLETED
/* JOB A 
// FOR RCALB 
*LIST ALL 
** CALIBRATION PROGRAM FOR OP-GUIDE POINTS 
**VGCR FIPC,1443 PRINTER,CARD 
**NONPROCESS PROGRAM 
**ONE WORD INTEGERS 
** THIS NONPROCESS CORE LOAD IS FOR CALIBRATING THE ANALOG INPUT 
** POINTS FOR OP-GUIDE. 
** INTEGER SW0,SW1,SW2,SW3,SW4,SW5,DAY 
** VALUE(15),SETPT(8),COUNTFB),OFFLN(10) 
** DIMENSION RANGE(8),AHL(40),ALL(40),A(40),R(40),LOW(16),HIGH(16) 
** COMMON/INSKEL/SWO,SW1,SW2,SW3,SW4,SW5,DAY,JOBN,VALUE,RANGE,LOW,SET 
** 1PT,COUNT,OFFLN,AHL,ALL,A,R,BASE,BASE2,BASE3,G,H,IENDT,IPERD,ITCNT 
** DEFINE FILE 112,320,U,I 
100 READ112,1)I,ENGH,ENGL 
1 FORMAT(2,8XF10.2,F10.2) 
 IF(I)100,100,2 
2 L=41-I 
 M=I-1 
 WRITE(3,999)IL 
 WRITE(3,7) 
7 FORMAT(I , SET POINT'IT3,' TO LOW 'VALUE') 
 PAUSE 
 CALL AIPl1000,IL,M) 
 CALL AIO,II) 
 GO TO 18,9),11 
9 AI(I)=ENGH-ENGL)/IIH-IL) 
 BL(I)=(ENGH-IAI)*IH) 
 WRITE(3,999)IH 
999 FORMAT(I , VALUE IS'IT10) 
 WRITEIl'l)RANGE,LOW,A,B 
 GO TO 100 
END 

VARIABLE ALLOCATIONS 
CSW0(*)=FFFF 
CSW1(*)=FFFE 
CSW2(*)=FFFF 
CSW3(*)=FFFF 
CSW4(*)=FFFF 
CSW5(*)=FFFF 
JOBN(*)=FFFF 
VALUE(*)=FFF7-FFE9 
RANGE(*)=FFE6-FF08 
LOW(*)=FFD7-FFD0 
SETPT(*)=FFCF-FFC8 
DAY(*)=FFC7-FFC0 
OFFLN(*)=FFBF-FFB8 
AHL(*)=FF66-FF18 
ALL(*)=FF60-FF10 
A(*)=FF16-FFC8 
BASE(*)=FFB6-FFB8 
BASE2(*)=FF7B 
BASE3(*)=FF75 
G(*)=FE32-FE54 
H(*)=FE52-FE54 
IENDT(*)=FE33 
IPERD(*)=FE32 
ITCNT(*)=FE31 
IPDNT(*)=FE30 
ENGR(*)=0006 
ENGL(*)=0008 
IIT(*)=000C 
I1(*)=000D 
L1(*)=000E 
M1(*)=000F 
ILI(*)=0010 
IH1(*)=0011 
STATEMENT ALLOCATIONS 
1 =0014 4 =0021 7 =0031 999 =0040 100 =0048 2 =0056 3 =0068 5 =0073 6 =0070 8 =008E 
9 =009B 

FEATURES SUPPORTED 
NONPROCESS 
ONE WORD INTEGERS 
I0CS 

CALLED SUBPROGRAMS 
AI PI SUB FMPYX FLD F50 F50X F50R FDVR FLOAT COMGO MRED MWRT MCOMP MIOF MIDI 
SUBSC PAUSE MDWR MTOM MDAT MDAP MDAB MHEB PANTN EBPT CARDN 

INTEGER CONSTANTS 
2=0014 
41=0015 
1=0016 
3=0017 
11000=0018 
0=0019 
0=001A 
0=001B 

CORE REQUIREMENTS FOR RCALB 
COMMON 0 INSKEL COMMON 464 VARIABLES 20 PROGRAM 188 

END OF COMPILATION 

RCALB 
DUP FUNCTION COMPLETED 
// DUP 
#DELETE RCALB 
RCALB D29 NAME NOT IN L/F 
STORECIL 1 RCALB RCALB 
*FILES(1,FILE1) 
* CCEND
Continued
CLB, BUILD RCALB

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NAME</th>
<th>ARG1</th>
<th>ARG2</th>
</tr>
</thead>
<tbody>
<tr>
<td>*CDW</td>
<td>TABLE 3E82</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>*IBT</td>
<td>TABLE 3E8E</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>*PIO</td>
<td>TABLE 3EAB</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>*ETV</td>
<td>TABLE 3EBB</td>
<td>0031</td>
<td></td>
</tr>
<tr>
<td>*VTV</td>
<td>TABLE 3F0C</td>
<td>0030</td>
<td></td>
</tr>
<tr>
<td>*PNT</td>
<td>TABLE 3F3C</td>
<td>0004</td>
<td></td>
</tr>
<tr>
<td>*OTF</td>
<td>TABLE 3F4D</td>
<td>0066</td>
<td></td>
</tr>
</tbody>
</table>

MAIN RCALB 3F88
PNT RCALB 3F3E

LIBF MHEB 4010 3F0C
LIBF EBPR 4132 3F0F
LIBF MHEO 4349 3F12
LIBF MIOI 4412 3F15
LIBF MIOF 4400 3F18
LIBF MCOMP 46E9 3F1B
LIBF MVRT 46C5 3F1E
LIBF PAUSE 480E 3F21
CALL AIP 4822
LIBF PSUB 4862 3F24
LIBF PSBR 484E 3F27
LIBF MWR 44DF 3F2A
LIBF MDAF 48F3 3F2D
LIBF MDAI 4900 3F30
LIBF MDCOM 4986 3F33
CALL PRT 4660
LIBF IOU 4C8A 3F36
CALL IOFIX 4444
CALL BT1BT 4074
CALL SAVE 4CE0
LIBF AIPTN 4D0D 3F39
CALL BT2BT 4E5A
CORE 4E78 3188

CLB, RCALB LD XQ

DUP FUNCTION COMPLETED

---

// JOB A
// * DATA CARDS FOR CALIBRATING POINTS 5 THROUGH 12
// XEQ RCALB FA

SET POINT 5 TO LOW VALUE
VALUE IS -100
NOW SET IT TO HIGH VALUE
VALUE IS -32362
SET POINT 6 TO LOW VALUE
VALUE IS -148
NOW SET IT TO HIGH VALUE
VALUE IS -32462
SET POINT 7 TO LOW VALUE
VALUE IS -126
NOW SET IT TO HIGH VALUE
VALUE IS -32338
SET POINT 8 TO LOW VALUE
VALUE IS -248
NOW SET IT TO HIGH VALUE
VALUE IS -32562
SET POINT 9 TO LOW VALUE
VALUE IS -102
NOW SET IT TO HIGH VALUE
VALUE IS -32546
SET POINT 10 TO LOW VALUE
VALUE IS -78
NOW SET IT TO HIGH VALUE
VALUE IS -32458
SET POINT 11 TO LOW VALUE
VALUE IS -96
NOW SET IT TO HIGH VALUE
VALUE IS -32466
SET POINT 12 TO LOW VALUE
VALUE IS -100
NOW SET IT TO HIGH VALUE
VALUE IS -32516
// JOB A
// # NONPROCESS CORE LOAD
// # FOR CMPI
#LIST ALL
**
=== CALIBRATION PROGRAM FOR DATA ENTRY DIALS
*10CS (DISK)
*10CS (144 PRINTER)
*NONPROCESS PROGRAM
*ONE WORD INTEGERS
C
C THIS NONPROCESS CORE LOAD IS FOR CALIBRATING THE DATA ENTRY
C DIALS.
C
INTEGER SWO, SWI, SW2, SW3, SW4, SW5, DAY
INTEGER VALUE(15), SETPT(8), COUNT(18), OFFLN(8)
DIMENSION INA(16), INB(16)
DIMENSION RANGE(8), AHL(40), ALL(40), A(40), B(40), L(16), H(16)
COMMON/INSKEL/SWO, SWI, SW2, SW3, SW4, SW5, DAY, JOBN, VALUE, RANGE, LOW, SET
JPT, COUNT, OFFLN, AHL, ALL, A, B, IBASE, IBASZ, G, H, IENDT, IPERT, ITCNT
2, IPONT
DEFINE FILE 112, 320, U, 11
WRITE(3, 11)
1 FORMAT(* SET DATA ENTRY DIALS TO 0*)
PAUSE
CALL GETVLI(INA)
WRITE(3, 2)
2 FORMAT(* SET DATA ENTRY DIALS TO 10*)
PAUSE
CALL GETVLI(INB)
DO 10 J = 1, 16
G(J) = I(NA(J)) - INA(J)
10 H(J) = 10. - G(J)
WRITE(12, 3) G, H
CALL EXIT
END

VARIABLE ALLOCATIONS
C
CSWO(*)=FFFF CSWI(*)=FFE8 CSW2(*)=FFD0 CSW3(*)=FFCC CSW4(*)=FFB8 CSW5(*)=FFFA
DAY(*)=FF9 JOBN(*)=FFB VALUE(*)=FF7-FF9 RANGE(*)=FF6-FFB LOW(*)=FFD-FFD SETPT(*)=FFC-FFC
COMMON(14*)=FFC7-FFCD OFFLN(*)=FFB-FFB AHL(*)=FFB-FFB ALL(*)=FFB-FFB A(*)=FFB-FFB B(*)=FFB-FFB
IBASE(*)=FE77 IBASZ(*)=FE76 IBASSI(*)=FE75 GIR(*)=FE72-FF94 HIR(*)=FE52-FF34 IENDT(*)=FE33
IPERT(*)=FE32 ITCNT(*I)=FE31 IPGNT(*I)=FE30 INA(*I)=0017-0008 INB(*I)=0027-0018 III(*)=0029
STATEMENT ALLOCATIONS
1 =0034 2 =0043 10 =007E
FEATURES SUPPORTED
NONPROCESS
ONE WORD INTEGERS
10CS
CALLED SUBPROGRAMS
GETVL PMPY PLOX FSTO FSTOX FSBR FDVR FLOAT MWRT MCOMP SUBSC PAUSE MDWRT MDCOM MDAF
REAL CONSTANTS
10000000 02 =002C
INTEGER CONSTANTS
3 =002E 1 =002F 16 =0030 2 =0031 0 =0032 0 =0033
CORE REQUIREMENTS FOR CMPI
COMMON 0
INSKEL COMMON 464 VARIABLES 44 PROGRAM 122
END OF COMPILATION

CMPI
DUP FUNCTION COMPLETED
*STORECIL 1 CMPI CMPI
*FILES(1,FILE1,1)
*CCEND

CLB, BUILD CMPI

CORE LOAD MAP
TYPE NAME ARG1 ARG2
"COW TABLE 3E82 000C
"TDT TABLE 3E88 001D
"FID TABLE 3E88 0010
"ETV TABLE 3EB8 0051
"VTM TABLE 3EC8 001E
"FMT TABLE 3F2A 0004
"DFT TABLE 3F2E 0006
MAIN CMPI 3F01
PNT CMIPT 3F2C
LIBF ESPT 3F04 3F0C
LIBF MKRT 41PE 3F0E
LIBF MCMP 428B 3F12
LIBF PAUSE 4680 3F15
CALL GETVL 46EE
LIBF FSBR 47BA 3F18
LIBF MDWR 497B 3F18
LIBF MDAF 48F2 3F21
CALL PRT 490C
LIBF IOU 48E6 3F24
CALL IOFIX 4C80
CALL STBT 4C80
CALL STU 4C1C
LIBF SUBIN 4144 3F27
CALL ST2BT 4D0E
CORE 406C 3294
CLB, CMIPT LO XQ
DUP FUNCTION COMPLETED

// JOB A
// XEQ CMIPFX
SET DATA ENTRY DIALS TO 0
SET DATA ENTRY DIALS TO 10

// JOB A
// * NONPROCESS CORE LOAD
// FOR LOADJ
*LIST
**
PROGRAM TO LOAD JOB DATA FILES ON DISK FROM CARDS
*I0CS(DISK,1443 PRINTER,CARD)
*NONPROCESS PROGRAM
ONE WORD INTEGERS
C
THIS PROGRAM LOADS THE PROCESS JOB FILES ON DISK WITH DATA
READ FROM CARDS.
C
INTEGER SETPT(8)
DIMENSION AHL(40),ALL(40),INPP(8)
DEFINE FILE 21100,320,U,III)
CALL PTIME(TIME)
WRITE(3,2)TIME
2 FORMAT(*PROCESS JOB FILE LOAD TIME*F7.2)
JOBN=O
ITIME=O
DO 3 J=1,8
3 SETPT(J)=O
DO 4 J=1,40
AHL(J)=32000.
4 ALL(J)=-32000.
10 READ(2,11)J,IV,BB,CC
11 FORMAT(Il,4XI5,2F10.0)
GO TO (100,200,300,400,700),J
100 IF(IV)600,600,101
101 JOBN=IV
ITIME=ITIME+IV
I=ITIME/100*100
K=ITIME-1
I=I/100
IF(K<60)200,200,102
102 AB=I*3600.+K*60.
IF(32000.-AB)600,103,103
103 ITIME=AB
AB=BB
GO TO 10
200 IF(IV)600,600,201
201 I=I-IV
IF(I)600,600,202
202 AHL(I)=BB
ALL(I)=CC
IF(AHL(I)-ALL(I)>600,600,10
300 IF(IV)600,600,301
Continued

301 I=9-IV
IF(I):00,,600,302
302 SETPT(I):=0B
IF(SEP TPT(1):600,600,303
303 IF(100-SEPTPT(I):600,600,10
400 IF(JOBN:600,600,409
409 IN=JOBN:100*100
I=JOBN:IN
IF(I):00,,401,402
401 I=100
402 WRITE(2,J):JOBN,TIME,SEPT, AHL, ALL
WRITE(3,403):JOBN, AB
403 FORMAT(' GRADE NUMBER*16,5X PRODUCTION TIME*10.2)
WRITE(3,500)
500 FORMAT(' OP-GUIDE LIMITS FOR GRADE*)
WRITE(3,501)
501 FORMAT(' POINT HIGH LIMIT LOW LIMIT*,10X POINT HIGH LIMIT
1 LOW LIMIT*)
DO 503 J=1;9,2
K=1-J
J1=J
J2=J+1
J3=1+2
J4=J+3
IA=9-1
IB=8-1
IC=7-1
ID=6-1
502 FORMAT('16,2F13.2,10X'16,2F13.2)
WRITE(3,510)
510 FORMAT(' CLOSED LOOP SET POINTS FOR GRADE*)
WRITE(3,511)
511 FORMAT(' POINT',5X'SETPT'),5X'POINT',5X'SETPT'),5X'POINT',5X'SETPT'
1,5X'POINT',5X'SETPT')
DO 6 J=1,8
6 INPPIJI=SETPTIJ)
DO 30
1=1,5,4
J1=I
J2=I+1
J3=1+2
J4=I+3
IA=9-1
IB=8-1
IC=7-1
ID=6-1
30 WRITE(3,7):J1, , INPPIIA),J2, , INPPIIB),J3, , INPPIICI,J4, , INPPI10)
7 FORMAT('18,I10,31110,I10)
GO TO 1
600 WRITE(3,601):J,IV, BB, CC
601 FORMAT(' INVALID DATA CARD',/I5,4XI5,2F10.7)
GO TO 1
700 CALL EXIT
END

VARIABLE ALLOCATIONS
AHL I=0044-0056 CTIME I=0046 BB(R) I=0048
INPPI I=0087-0080 SEPTPT I=008F-0088 JOBN I=00C1
IV I=00C4 J I=00C5 J1 I=00C6 IV I=00C7 J1 I=00C8
J3 I=00CA J4 I=00CB IA I=00CC J3 I=00C9 J4 I=00CE
STATEMENT ALLOCATIONS
2 =00EB 3 =00FD 403 =0103 500 =011B 501 =0127 502 =014B 510 =0154 511 =0166 7 =018F 601 =0196
1 =0197 3 =01BC 4 =010B 10 =018E 100 =0200 101 =0204 102 =022C 103 =024A 200 =024E 201 =0252
202 =025C 300 =0276 301 =027A 302 =0284 303 =0294 400 =02A1 409 =02A5 401 =02B8 402 =02BF 503 =02F4
6 =0324 30 =0368 600 =039C 700 =03AA
FEATURES SUPPORTED
NONPROCESS ONE WORD INTEGERS 107S
CALLED SUBPROGRAMS
TIME PTIME FADD FSUB FSUBX FMRY FLD FLDX FSTO FGXT IFIX FLOAT COSMO ISTOIX LDFAC MRED
MWRT MCOMP MIOFX MIOIX MIOI SUBSC SNR MDWR MDCOM MDIAF MDAF MDI MIDEA MDEB PRTTN
REAL CONSTANTS
.320000E 05+004 .100000E 03+006 .360000E 04+0008 .600000E 02+0004
INTEGER CONSTANTS
1+0000 1+0000 1+0000 1+0000 1+0000 1+0000 1+0000 1+0000 2+0001 100+0002 60+0003 41+0004 9+0005
CORE REQUIREMENTS FOR LOADJ
COMMON 0 INSKEL COMMON 0 VARIABLES 212 PROGRAM 728

END OF COMPILATION
Continued

LOADJ
DUP FUNCTION COMPLETED
*DELET
LOADJ
D25 NAME NOT IN L/F
*STORECIL 1 LOADJ LOADJ
*FILES(2,FILE2,1)
*CEMND

CLB, BUILD LOADJ

CORE LOAD MAP
TYPE NAME ARG1 ARG2
*COW TABLE 3EB2 000C
*F1T TABLE 3EBA 0010
*F1O TABLE 3EAB 0010
*ETV TABLE 3EBB 0051
*VTV TABLE 3FC0 003F
*PNT TABLE 3FC4 0004
*DPT TABLE 3F50 0006
MAIN LOADJ 40F7
PNT LOADJ 3F4E
L1BF HDLER 42FC 3F0C
L1BF EBPRT 4416 3F0F
CALL PTIME 44CA
L1BF MWT 4864 3F12
L1BF MDIP 4735 3F15
L1BF MCOMP 4711 3F18
L1BF SNR 483A 3F18
L1BF MXEO 4671 3F1E
L1BF MI01 473A 3F21
L1BF FADD 486E 3F24
L1BF FSUB 48B2 3F27
L1BF LDFAC 4BEC 3F2A
L1BF FADD 4850 3F2D
L1BF MDWRT 4060 3F30
L1BF MIOFX 4741 3F3F
L1BF MIOIX 4746 3F42
CALL PRT 4F8E
L1BF SUBIN 4F4E 3F45
L1BF IOU 5012 3F48
CALL IOFIX 50AC
CALL BTST 50DC
CALL SAVE 5048
CALL B2BT 5140
CORE 515E 2EA2

CLB, LOADJ LD XO

DUP FUNCTION COMPLETED

// JOB A
// * UPDATE A PROCESS JOB FILE ON DISK
// * EQ LOADJ FX
PROCESS JOB FILE LOAD TIME 14.58
GRADE NUMBER 12345 PRODUCTION TIME 1.30

DP-GUIDE LIMITS FOR GRADE

<table>
<thead>
<tr>
<th>POINT</th>
<th>HIGH LIMIT</th>
<th>LOW LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>2</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>3</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>4</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>5</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>6</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>7</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>8</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>9</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>10</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>11</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>12</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>13</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>14</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>15</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>16</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>17</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>18</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>19</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>20</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>21</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>22</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>23</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>24</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>25</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>26</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>27</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>28</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>29</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>30</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>31</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>32</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>33</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>34</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>35</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>36</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
</tbody>
</table>

Programming Techniques 277
Continued

LET

PACK LABEL
1111

.LET 0080 0000 SYDIR 009E 0080 OUTTR CHANNEL INT EX SHARE
SPEC1 BACK EACLK SCHED 0014 011E LEV10 0024 0132 SOUT 0003 0156
QUEUE 0002 0159 TCONT 0003 0158 TABAT 0002 015E GETVL 0008 0160 CONVR 0005 0168 PTIME 0005 0170
IADDR 0002 0175 ISBAD 0002 0177 CESET 0002 0179 ABORT 0002 017B ENGO 0002 017E TEMP 017F 0180
.E 1180 0180

FLET

PACK LABEL
1111

.PWWS 0051 1118 .FIOS 000F 1159 .MESS 00A3 1118 /EPDM 7FFF 121B /EPSV 0780 1282 /INSV 48FF 1288
/SPSV 4150 12C3 FILE1 0002 12F8 FILE2 0064 12F4 FILE3 0003 135E COLON 050C 1361 COLOS 0006 136A
/STASK 0F8C 1367 COLOP 10CE 1374 CEINT 02A6 1382 DUM 012C 1386 IDUM 1094 138C CDUM 013A 938D
GRADE 116A 138E SCAN2 0C2E 939C LOG15 005E 13A6 LOG60 0AE0 13B1 SHIFT 0AE0 13B6 WEEK 0ABD 13C3
/CRENS 097A 13CC COGLP 101E 13C6 CCLSP 0F76 13E3 MORTP 0F8A 13F0 CPJSP 0FCC 13FD STRND 08FC 140A
/AMON 0090 1416 SCALF OF3A 141F RCLAB 0FF6 142C CMUPT 0EEA 1439 LOAUD 12DC 1445 9DUMY 0123 1455
/SPSV 4180 1578 /PRSV 4180 15AD .SKEI 0036 15E2 .EPRG 0022 1618 /CLST 0780 163A .E 0280 12F8

DUP FUNCTION COMPLETED

// JOB A
// ENO
TURN OFF WRITE STORAGE PROTECT SWITCH
ENTER TIME THROUGH DATA SWITCHES
TIME ENTERED WAS 08.016 HOURS
PROCESS COLD START
PRODUCTION STOP TIME 8.00 DAY 6

0 0 0 1 2 3 4 5
0 0 0 0 0 0 0 0
NEXT JOB 12345 QUEUE SEQUENCE 45 DAY 6 TIME 8.02

START OF GRADE 12345 PRODUCTION TIME 1.29 START TIME 8.03 DAY 6

OP-GUIDE LIMITS FOR NEW GRADE

<table>
<thead>
<tr>
<th>POINT</th>
<th>HIGH LIMIT</th>
<th>LOW LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>2</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>3</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>4</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>5</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>6</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>7</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>8</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>9</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>10</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>11</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>12</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>13</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>14</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>15</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>16</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>17</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>18</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>19</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>20</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>21</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>22</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>23</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>24</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>25</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>26</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>27</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>28</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>29</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>30</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>31</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>32</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>33</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>34</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>35</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>36</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>37</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>38</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>39</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>40</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
</tbody>
</table>

CLOSED LOOP SET POINTS FOR NEW GRADE

<table>
<thead>
<tr>
<th>POINT</th>
<th>SETPT POINT</th>
<th>SETPT POINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

NORMAL SCAN DAY 6 TIME 8.03

7 0 0 0 0 0 0 5 0
0 0 0 0 0 0 0 0 0
LOOP 7 NEW SET POINT 50 DAY 6 TIME 8.05

NORMAL SCAN DAY 6 TIME 8.06

NORMAL SCAN DAY 6 TIME 8.07

NORMAL SCAN DAY 6 TIME 8.09

NORMAL SCAN DAY 6 TIME 8.12

NORMAL SCAN DAY 6 TIME 8.13

LOG15 DAY 6 TIME 8.15

OP-GUIDE POINTS

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1431.00</td>
</tr>
<tr>
<td>2</td>
<td>973.00</td>
</tr>
<tr>
<td>3</td>
<td>26.00</td>
</tr>
<tr>
<td>4</td>
<td>32.00</td>
</tr>
<tr>
<td>5</td>
<td>5.37</td>
</tr>
<tr>
<td>6</td>
<td>5.16</td>
</tr>
<tr>
<td>7</td>
<td>5.01</td>
</tr>
<tr>
<td>8</td>
<td>5.10</td>
</tr>
<tr>
<td>9</td>
<td>5.14</td>
</tr>
<tr>
<td>10</td>
<td>5.28</td>
</tr>
<tr>
<td>11</td>
<td>5.00</td>
</tr>
<tr>
<td>12</td>
<td>4.98</td>
</tr>
<tr>
<td>13</td>
<td>22.00</td>
</tr>
<tr>
<td>14</td>
<td>20.00</td>
</tr>
<tr>
<td>15</td>
<td>22.00</td>
</tr>
<tr>
<td>16</td>
<td>42.00</td>
</tr>
<tr>
<td>17</td>
<td>714.00</td>
</tr>
<tr>
<td>18</td>
<td>704.00</td>
</tr>
<tr>
<td>19</td>
<td>716.00</td>
</tr>
<tr>
<td>20</td>
<td>706.00</td>
</tr>
<tr>
<td>21</td>
<td>706.00</td>
</tr>
<tr>
<td>22</td>
<td>698.00</td>
</tr>
<tr>
<td>23</td>
<td>718.00</td>
</tr>
<tr>
<td>24</td>
<td>706.00</td>
</tr>
<tr>
<td>25</td>
<td>727.00</td>
</tr>
<tr>
<td>26</td>
<td>708.00</td>
</tr>
<tr>
<td>27</td>
<td>704.00</td>
</tr>
<tr>
<td>28</td>
<td>714.00</td>
</tr>
<tr>
<td>29</td>
<td>704.00</td>
</tr>
<tr>
<td>30</td>
<td>718.00</td>
</tr>
<tr>
<td>31</td>
<td>706.00</td>
</tr>
<tr>
<td>32</td>
<td>704.00</td>
</tr>
<tr>
<td>33</td>
<td>714.00</td>
</tr>
<tr>
<td>34</td>
<td>710.00</td>
</tr>
<tr>
<td>35</td>
<td>710.00</td>
</tr>
<tr>
<td>36</td>
<td>718.00</td>
</tr>
<tr>
<td>37</td>
<td>710.00</td>
</tr>
<tr>
<td>38</td>
<td>704.00</td>
</tr>
<tr>
<td>39</td>
<td>716.00</td>
</tr>
<tr>
<td>40</td>
<td>706.00</td>
</tr>
</tbody>
</table>

CLOSED LOOP POINTS

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>85</td>
</tr>
<tr>
<td>POINT</td>
<td>VALUE POINT VALUE POINT VALUE POINT VALUE POINT VALUE</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>14158.00 2 9770.00 3 28.00 4 34.00</td>
</tr>
<tr>
<td>2</td>
<td>5.37 6 5.16 7 5.01 8 5.10</td>
</tr>
<tr>
<td>3</td>
<td>14.00 10 5.27 11 5.00 12 4.99</td>
</tr>
<tr>
<td>4</td>
<td>18.00 14 28.00 15 18.00 16 22.00</td>
</tr>
<tr>
<td>5</td>
<td>706.00 18 706.00 19 706.00 20 710.00</td>
</tr>
<tr>
<td>6</td>
<td>704.00 22 704.00 23 714.00 24 706.00</td>
</tr>
<tr>
<td>7</td>
<td>702.00 26 702.00 27 706.00 28 706.00</td>
</tr>
<tr>
<td>8</td>
<td>700.00 30 702.00 31 706.00 32 710.00</td>
</tr>
<tr>
<td>9</td>
<td>708.00 34 712.00 35 708.00 36 712.00</td>
</tr>
<tr>
<td>10</td>
<td>704.00 38 714.00 39 710.00 40 714.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE POINT VALUE POINT VALUE POINT VALUE POINT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 2 0 3 75 4 0</td>
</tr>
<tr>
<td>2</td>
<td>0 6 5 7 40 8 85</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE POINT VALUE POINT VALUE POINT VALUE POINT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14158.00 2 9770.00 3 28.00 4 34.00</td>
</tr>
<tr>
<td>2</td>
<td>5.37 6 5.16 7 5.01 8 5.10</td>
</tr>
<tr>
<td>3</td>
<td>14.00 10 5.27 11 5.00 12 4.99</td>
</tr>
<tr>
<td>4</td>
<td>18.00 14 28.00 15 18.00 16 22.00</td>
</tr>
<tr>
<td>5</td>
<td>706.00 18 706.00 19 706.00 20 710.00</td>
</tr>
<tr>
<td>6</td>
<td>704.00 22 704.00 23 714.00 24 706.00</td>
</tr>
<tr>
<td>7</td>
<td>702.00 26 702.00 27 706.00 28 706.00</td>
</tr>
<tr>
<td>8</td>
<td>700.00 30 702.00 31 706.00 32 710.00</td>
</tr>
<tr>
<td>9</td>
<td>708.00 34 712.00 35 708.00 36 712.00</td>
</tr>
<tr>
<td>10</td>
<td>704.00 38 714.00 39 710.00 40 714.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE POINT VALUE POINT VALUE POINT VALUE POINT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 2 0 3 75 4 0</td>
</tr>
<tr>
<td>2</td>
<td>0 6 5 7 40 8 85</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>8.49</td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>8.51</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>8.54</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>8.55</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>8.57</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OP-GUIDE POINTS</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14158.00</td>
<td>2</td>
<td>9818.00</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5.37</td>
<td>6</td>
<td>5.10</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>5.14</td>
<td>10</td>
<td>5.27</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>38.00</td>
<td>14</td>
<td>28.00</td>
<td>15</td>
</tr>
<tr>
<td>17</td>
<td>704.00</td>
<td>18</td>
<td>718.00</td>
<td>19</td>
</tr>
<tr>
<td>21</td>
<td>702.00</td>
<td>22</td>
<td>716.00</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td>710.00</td>
<td>26</td>
<td>712.00</td>
<td>27</td>
</tr>
<tr>
<td>29</td>
<td>704.00</td>
<td>30</td>
<td>718.00</td>
<td>31</td>
</tr>
<tr>
<td>33</td>
<td>704.00</td>
<td>34</td>
<td>718.00</td>
<td>35</td>
</tr>
<tr>
<td>37</td>
<td>708.00</td>
<td>38</td>
<td>708.00</td>
<td>39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLOSED LOOP POINTS</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ONE HOUR LOG DAY 6 TIME</th>
<th>9.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.01</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.03</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.06</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.07</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.09</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.12</td>
</tr>
<tr>
<td>NORMAL SCAN DAY 6 TIME</td>
<td>9.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OP-GUIDE POINTS</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14150.00</td>
<td>2</td>
<td>9828.00</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5.37</td>
<td>6</td>
<td>5.10</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>5.14</td>
<td>10</td>
<td>5.27</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>38.00</td>
<td>14</td>
<td>28.00</td>
<td>15</td>
</tr>
<tr>
<td>17</td>
<td>704.00</td>
<td>18</td>
<td>718.00</td>
<td>19</td>
</tr>
<tr>
<td>21</td>
<td>702.00</td>
<td>22</td>
<td>716.00</td>
<td>23</td>
</tr>
<tr>
<td>25</td>
<td>710.00</td>
<td>26</td>
<td>712.00</td>
<td>27</td>
</tr>
<tr>
<td>29</td>
<td>704.00</td>
<td>30</td>
<td>718.00</td>
<td>31</td>
</tr>
<tr>
<td>33</td>
<td>704.00</td>
<td>34</td>
<td>718.00</td>
<td>35</td>
</tr>
<tr>
<td>37</td>
<td>708.00</td>
<td>38</td>
<td>704.00</td>
<td>39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLOSED LOOP POINTS</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
<th>POINT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
### LOG15 Day 6 Time 11.30

**OP-GUIDE POINTS**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14154.00</td>
</tr>
<tr>
<td>5</td>
<td>5.37</td>
</tr>
<tr>
<td>10</td>
<td>5.14</td>
</tr>
<tr>
<td>15</td>
<td>5.00</td>
</tr>
<tr>
<td>20</td>
<td>4.90</td>
</tr>
</tbody>
</table>

**CLOSED LOOP POINTS**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9260.00</td>
</tr>
<tr>
<td>5</td>
<td>5.01</td>
</tr>
<tr>
<td>10</td>
<td>5.00</td>
</tr>
<tr>
<td>15</td>
<td>5.00</td>
</tr>
<tr>
<td>20</td>
<td>5.00</td>
</tr>
</tbody>
</table>

### DAY 6 TIME 11.28

**NORMAL SCAN**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4.90</td>
</tr>
<tr>
<td>10</td>
<td>4.90</td>
</tr>
<tr>
<td>15</td>
<td>4.90</td>
</tr>
<tr>
<td>20</td>
<td>5.00</td>
</tr>
<tr>
<td>25</td>
<td>5.00</td>
</tr>
<tr>
<td>30</td>
<td>5.00</td>
</tr>
<tr>
<td>35</td>
<td>5.00</td>
</tr>
</tbody>
</table>

**NORMAL SCAN**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4.90</td>
</tr>
<tr>
<td>10</td>
<td>4.90</td>
</tr>
<tr>
<td>15</td>
<td>4.90</td>
</tr>
<tr>
<td>20</td>
<td>5.00</td>
</tr>
<tr>
<td>25</td>
<td>5.00</td>
</tr>
<tr>
<td>30</td>
<td>5.00</td>
</tr>
<tr>
<td>35</td>
<td>5.00</td>
</tr>
</tbody>
</table>

### DAY 6 TIME 11.34

**OP-GUIDE POINT**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5.10</td>
</tr>
<tr>
<td>10</td>
<td>5.10</td>
</tr>
</tbody>
</table>

**NORMAL SCAN**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5.10</td>
</tr>
<tr>
<td>10</td>
<td>5.10</td>
</tr>
<tr>
<td>15</td>
<td>5.10</td>
</tr>
<tr>
<td>20</td>
<td>5.10</td>
</tr>
<tr>
<td>25</td>
<td>5.10</td>
</tr>
<tr>
<td>30</td>
<td>5.10</td>
</tr>
</tbody>
</table>

### DAY 6 TIME 11.36

**NORMAL SCAN**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5.10</td>
</tr>
<tr>
<td>10</td>
<td>5.10</td>
</tr>
</tbody>
</table>

### DAY 6 TIME 11.38

**NORMAL SCAN**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5.10</td>
</tr>
<tr>
<td>10</td>
<td>5.10</td>
</tr>
</tbody>
</table>
TREND LOG OP-GUIDE POINT 6 PERIOD 10 COUNT 300
TREND OP-GUIDE 6 VALUE 5.01
TREND OP-GUIDE 6 VALUE 5.01

NORMAL SCAN DAY 3 TIME 9.17
TREND OP-GUIDE 6 VALUE 5.49
TREND OP-GUIDE 6 VALUE 5.65
TREND OP-GUIDE 6 VALUE 5.64
TREND OP-GUIDE 6 VALUE 6.19
TREND OP-GUIDE 6 VALUE 6.51
TREND OP-GUIDE 6 VALUE 6.73
TREND OP-GUIDE 6 VALUE 6.73

DEMAND SCAN DAY 3 TIME 9.19
HIGH LIMIT VIOLATION POINT 6 VALUE 6.7345
TREND OP-GUIDE 6 VALUE 6.73
TREND OP-GUIDE 6 VALUE 5.96

NORMAL SCAN DAY 3 TIME 9.19
HIGH LIMIT VIOLATION POINT 6 VALUE 5.5796
TREND OP-GUIDE 6 VALUE 5.57
TREND OP-GUIDE 6 VALUE 4.98
TREND OP-GUIDE 6 VALUE 5.12
TREND OP-GUIDE 6 VALUE 4.98

NORMAL SCAN DAY 3 TIME 9.21

NORMAL SCAN DAY 3 TIME 9.21
NORMAL SCAN DAY 3 TIME 9.23
Production STOP TIME 9.25 DAY 3

NORMAL SCAN DAY 3 TIME 9.25

NORMAL SCAN DAY 3 TIME 9.25

NORMAL SCAN DAY 3 TIME 9.25

START OF GRADE 4630 PRODUCTION TIME 1.25 START TIME 12.26 DAY 2
OP-GUIDE LIMITS FOR NEW GRADE
POINT HIGH LIMIT LOW LIMIT
1 12200.00 -32000.00
2 32000.00 -32000.00
3 32000.00 -32000.00
4 32000.00 -32000.00
5 5.55 4.45
6 5.46 4.45
7 5.95 4.45
8 5.46 4.45
9 5.65 4.45
10 5.65 4.45
11 5.63 4.45
12 5.62 4.45
13 32000.00 -32000.00
14 32000.00 -32000.00
15 32000.00 -32000.00
16 32000.00 -32000.00
17 32000.00 -32000.00

NORMAL SCAN DAY 2 TIME 12.20
NORMAL SCAN DAY 2 TIME 12.22
NORMAL SCAN DAY 2 TIME 12.24
<table>
<thead>
<tr>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

CLOSED LOOP POINTS FOR NEW GRADE

NORMAL SCAN DAY 3 TIME 9.06
997 09.130 RELOAD
RELOAD
PROCESS RESTART CHECK POINT

NORMAL SCAN DAY 3 TIME 9.07

LOC 15 DAY 3 TIME 9.07

OP-GUIDE POINTS

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14150.00</td>
<td>2</td>
<td>9946.00</td>
<td>3</td>
<td>24.00</td>
<td>4</td>
<td>-100.00</td>
</tr>
<tr>
<td>5</td>
<td>5.01</td>
<td>6</td>
<td>5.01</td>
<td>7</td>
<td>4.99</td>
<td>8</td>
<td>4.90</td>
</tr>
<tr>
<td>9</td>
<td>4.04</td>
<td>10</td>
<td>5.04</td>
<td>11</td>
<td>4.08</td>
<td>12</td>
<td>4.98</td>
</tr>
<tr>
<td>13</td>
<td>30.00</td>
<td>14</td>
<td>22.00</td>
<td>15</td>
<td>24.00</td>
<td>16</td>
<td>24.00</td>
</tr>
<tr>
<td>17</td>
<td>702.00</td>
<td>18</td>
<td>714.00</td>
<td>19</td>
<td>706.00</td>
<td>20</td>
<td>704.00</td>
</tr>
<tr>
<td>21</td>
<td>714.00</td>
<td>22</td>
<td>704.00</td>
<td>23</td>
<td>716.00</td>
<td>24</td>
<td>708.00</td>
</tr>
<tr>
<td>25</td>
<td>708.00</td>
<td>26</td>
<td>708.00</td>
<td>27</td>
<td>708.00</td>
<td>28</td>
<td>704.00</td>
</tr>
<tr>
<td>29</td>
<td>714.00</td>
<td>30</td>
<td>704.00</td>
<td>31</td>
<td>712.00</td>
<td>32</td>
<td>702.00</td>
</tr>
<tr>
<td>33</td>
<td>714.00</td>
<td>34</td>
<td>710.00</td>
<td>35</td>
<td>704.00</td>
<td>36</td>
<td>716.00</td>
</tr>
<tr>
<td>37</td>
<td>704.00</td>
<td>38</td>
<td>716.00</td>
<td>39</td>
<td>704.00</td>
<td>40</td>
<td>718.00</td>
</tr>
</tbody>
</table>

CLOSED LOOP POINTS

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>69</td>
</tr>
</tbody>
</table>

ONE HOUR LOG DAY 3 TIME 9.07

SHIFT END LOG DAY 3 TIME 9.08

NORMAL SCAN DAY 3 TIME 9.09

DEMAND SCAN DAY 3 TIME 9.10
HIGH LIMIT VIOLATION POINT 5 VALUE 7.3461

DEMAND SCAN DAY 3 TIME 9.10
<table>
<thead>
<tr>
<th></th>
<th>POINT SETPT</th>
<th>POINT SETPT</th>
<th>POINT SETPT</th>
<th>POINT SETPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>18</td>
<td>32000.00</td>
</tr>
<tr>
<td>18</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>19</td>
<td>32000.00</td>
</tr>
<tr>
<td>19</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>20</td>
<td>32000.00</td>
</tr>
<tr>
<td>20</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>21</td>
<td>32000.00</td>
</tr>
<tr>
<td>21</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>22</td>
<td>32000.00</td>
</tr>
<tr>
<td>22</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>23</td>
<td>32000.00</td>
</tr>
<tr>
<td>23</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>24</td>
<td>32000.00</td>
</tr>
<tr>
<td>24</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>25</td>
<td>32000.00</td>
</tr>
<tr>
<td>25</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>26</td>
<td>32000.00</td>
</tr>
<tr>
<td>26</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>27</td>
<td>32000.00</td>
</tr>
<tr>
<td>27</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>28</td>
<td>32000.00</td>
</tr>
<tr>
<td>28</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>29</td>
<td>32000.00</td>
</tr>
<tr>
<td>29</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>30</td>
<td>32000.00</td>
</tr>
</tbody>
</table>

**CLOSED LOOP SET POINTS FOR NEW GRADE**

<table>
<thead>
<tr>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>70</td>
<td>8</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>

**NORMAL SCAN DAY 4 TIME 13.48**

**NORMAL SCAN DAY 4 TIME 13.50**

**NORMAL SCAN DAY 4 TIME 13.52**

**NORMAL SCAN DAY 4 TIME 13.54**

**NORMAL SCAN DAY 4 TIME 13.56**

**RESTART**

**NORMAL SCAN DAY 4 TIME 13.58**

**LOG15 DAY 4 TIME 14.00**

**OP-GUIDE POINTS**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14156.00</td>
<td>2</td>
<td>9724.00</td>
<td>3</td>
<td>34.00</td>
<td>4</td>
<td>24.00</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6.98</td>
<td>7</td>
<td>5.00</td>
<td>8</td>
<td>4.98</td>
</tr>
<tr>
<td>3</td>
<td>6.98</td>
<td>10</td>
<td>3.04</td>
<td>11</td>
<td>4.88</td>
<td>12</td>
<td>4.98</td>
</tr>
<tr>
<td>4</td>
<td>28.00</td>
<td>14</td>
<td>26.00</td>
<td>15</td>
<td>14.00</td>
<td>16</td>
<td>18.00</td>
</tr>
<tr>
<td>5</td>
<td>706.00</td>
<td>18</td>
<td>706.00</td>
<td>19</td>
<td>708.00</td>
<td>20</td>
<td>706.00</td>
</tr>
<tr>
<td>6</td>
<td>706.00</td>
<td>22</td>
<td>704.00</td>
<td>23</td>
<td>714.00</td>
<td>24</td>
<td>708.00</td>
</tr>
<tr>
<td>7</td>
<td>704.00</td>
<td>30</td>
<td>710.00</td>
<td>31</td>
<td>704.00</td>
<td>32</td>
<td>712.00</td>
</tr>
<tr>
<td>8</td>
<td>700.00</td>
<td>34</td>
<td>714.00</td>
<td>35</td>
<td>708.00</td>
<td>36</td>
<td>702.00</td>
</tr>
<tr>
<td>9</td>
<td>712.00</td>
<td>38</td>
<td>704.00</td>
<td>39</td>
<td>716.00</td>
<td>40</td>
<td>704.00</td>
</tr>
</tbody>
</table>

**CLOSED LOOP POINTS**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>70</td>
<td>8</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>

**ONE HOUR LOG DAY 4 TIME 14.00**

**NORMAL SCAN DAY 4 TIME 14.00**

**NORMAL SCAN DAY 4 TIME 14.02**

286
ONE HOUR LOG  
DAY 3  
TIME 9.00

NORMAL SCAN  
DAY 3  
TIME 9.00

<table>
<thead>
<tr>
<th>Loop</th>
<th>New Set Point</th>
<th>Day</th>
<th>Time</th>
<th>9.02</th>
</tr>
</thead>
</table>

NORMAL SCAN  
DAY 3  
TIME 9.02
CE UNMASK CORE LOAD--PRESS START TO EXIT FROM CORE LOAD

CE 09.070
SSO ON ERR CNTRS
SS1 ON 1443
SS2 ON 2330
SS3 ON 1053
2310
UNIT USER ORIGINAL
1 00 00
2 02 02
3 00 00
2310
UNIT USER ORIGINAL
1 01 01
2 02 02
3 00 00
SWITCH PACKS

NORMAL SCAN  
DAY 3  
TIME 9.05

<table>
<thead>
<tr>
<th>Start of Grade</th>
<th>85 Production Time</th>
<th>0.20 Start Time</th>
<th>9.05 Day 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>High Limit</td>
<td>Low Limit</td>
<td>Point</td>
</tr>
<tr>
<td>1 32000.00</td>
<td>-32000.00</td>
<td>2 32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>3 32000.00</td>
<td>-32000.00</td>
<td>4 32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>5 5.50</td>
<td>4.50</td>
<td>6 5.50</td>
<td>4.50</td>
</tr>
<tr>
<td>7 5.50</td>
<td>4.50</td>
<td>8 5.50</td>
<td>4.50</td>
</tr>
<tr>
<td>9 5.50</td>
<td>4.50</td>
<td>10 5.50</td>
<td>4.50</td>
</tr>
<tr>
<td>11 5.50</td>
<td>4.50</td>
<td>12 5.50</td>
<td>4.50</td>
</tr>
<tr>
<td>13 32000.00</td>
<td>-32000.00</td>
<td>14 32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>15 32000.00</td>
<td>-32000.00</td>
<td>16 32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>17 32000.00</td>
<td>-32000.00</td>
<td>18 32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>19 32000.00</td>
<td>-32000.00</td>
<td>20 32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>21 32000.00</td>
<td>-32000.00</td>
<td>22 32000.00</td>
<td>-32000.00</td>
</tr>
</tbody>
</table>

Programming Techniques 287
<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14154.00</td>
<td>2</td>
<td>9454.00</td>
<td>3</td>
<td>26.00</td>
<td>4</td>
<td>-62.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4.97</td>
<td>6</td>
<td>4.98</td>
<td>7</td>
<td>5.00</td>
<td>8</td>
<td>4.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4.94</td>
<td>10</td>
<td>5.04</td>
<td>11</td>
<td>4.98</td>
<td>12</td>
<td>4.98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.19</td>
<td>14</td>
<td>0.02</td>
<td>15</td>
<td>0.57</td>
<td>16</td>
<td>-0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>712.00</td>
<td>18</td>
<td>710.00</td>
<td>19</td>
<td>710.00</td>
<td>20</td>
<td>710.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>714.00</td>
<td>22</td>
<td>710.00</td>
<td>23</td>
<td>710.00</td>
<td>24</td>
<td>712.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>708.00</td>
<td>26</td>
<td>712.00</td>
<td>27</td>
<td>708.00</td>
<td>28</td>
<td>706.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>710.00</td>
<td>30</td>
<td>708.00</td>
<td>31</td>
<td>700.00</td>
<td>32</td>
<td>710.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>708.00</td>
<td>34</td>
<td>706.00</td>
<td>35</td>
<td>708.00</td>
<td>36</td>
<td>708.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>708.00</td>
<td>38</td>
<td>706.00</td>
<td>39</td>
<td>710.00</td>
<td>40</td>
<td>710.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CLOSED LOOP POINTS**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>69</td>
<td>8</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MONDAY MORNING REPORT DAY 2 TIME 8.30**

**START OF GRADE 97 PRODUCTION TIME 0.20 START TIME 8.45 DAY 2**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14156.00</td>
<td>2</td>
<td>9444.00</td>
<td>3</td>
<td>36.00</td>
<td>4</td>
<td>-82.00</td>
</tr>
<tr>
<td>5</td>
<td>4.97</td>
<td>6</td>
<td>4.98</td>
<td>7</td>
<td>5.00</td>
<td>8</td>
<td>4.90</td>
</tr>
<tr>
<td>9</td>
<td>4.94</td>
<td>10</td>
<td>5.04</td>
<td>11</td>
<td>4.98</td>
<td>12</td>
<td>4.98</td>
</tr>
<tr>
<td>13</td>
<td>0.71</td>
<td>14</td>
<td>0.36</td>
<td>15</td>
<td>0.69</td>
<td>16</td>
<td>-0.00</td>
</tr>
<tr>
<td>17</td>
<td>706.00</td>
<td>18</td>
<td>708.00</td>
<td>19</td>
<td>712.00</td>
<td>20</td>
<td>712.00</td>
</tr>
<tr>
<td>21</td>
<td>710.00</td>
<td>22</td>
<td>716.00</td>
<td>23</td>
<td>710.00</td>
<td>24</td>
<td>712.00</td>
</tr>
<tr>
<td>25</td>
<td>710.00</td>
<td>26</td>
<td>716.00</td>
<td>27</td>
<td>714.00</td>
<td>28</td>
<td>712.00</td>
</tr>
<tr>
<td>29</td>
<td>710.00</td>
<td>30</td>
<td>714.00</td>
<td>31</td>
<td>710.00</td>
<td>32</td>
<td>710.00</td>
</tr>
<tr>
<td>33</td>
<td>710.00</td>
<td>34</td>
<td>710.00</td>
<td>35</td>
<td>710.00</td>
<td>36</td>
<td>712.00</td>
</tr>
<tr>
<td>37</td>
<td>710.00</td>
<td>38</td>
<td>710.00</td>
<td>39</td>
<td>710.00</td>
<td>40</td>
<td>710.00</td>
</tr>
</tbody>
</table>

**CLOSED LOOP POINTS**

<table>
<thead>
<tr>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
<th>POINT</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>36</td>
<td>8</td>
<td>63</td>
</tr>
<tr>
<td>POINT</td>
<td>VALUE</td>
<td>POINT</td>
<td>VALUE</td>
<td>POINT</td>
<td>VALUE</td>
<td>POINT</td>
<td>VALUE</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>69</td>
<td>8</td>
<td>50</td>
</tr>
</tbody>
</table>

One Hour Log Day 2 TIME 9.00

Production STOP Time 9.10 Day 2

---

Process Job File Load Time 11.14

Grade Number 12345 Production Time 1.30

### OP-GUIDE Limits for Grade

<table>
<thead>
<tr>
<th>POINT</th>
<th>HIGH LIMIT</th>
<th>LOW LIMIT</th>
<th>POINT</th>
<th>HIGH LIMIT</th>
<th>LOW LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>2</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>3</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>4</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>5</td>
<td>5.45</td>
<td>4.96</td>
<td>6</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>7</td>
<td>5.45</td>
<td>4.96</td>
<td>8</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>9</td>
<td>5.45</td>
<td>4.96</td>
<td>10</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>11</td>
<td>5.45</td>
<td>4.96</td>
<td>12</td>
<td>5.45</td>
<td>4.96</td>
</tr>
<tr>
<td>13</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>14</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>15</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>16</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>17</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>18</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>19</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>20</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>21</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>22</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>23</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>24</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>25</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>26</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>27</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>28</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>29</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>30</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>31</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>32</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>33</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>34</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>35</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>36</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>37</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>38</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>39</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>40</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
</tbody>
</table>

Closed Loop Points for Grade

<table>
<thead>
<tr>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>69</td>
<td>8</td>
<td>50</td>
</tr>
</tbody>
</table>

---

Process Job File Load Time 11.14

Grade Number 45930 Production Time 1.25

### OP-GUIDE Limits for Grade

<table>
<thead>
<tr>
<th>POINT</th>
<th>HIGH LIMIT</th>
<th>LOW LIMIT</th>
<th>POINT</th>
<th>HIGH LIMIT</th>
<th>LOW LIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>2</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>3</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>4</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>5</td>
<td>5.45</td>
<td>4.95</td>
<td>6</td>
<td>5.45</td>
<td>4.95</td>
</tr>
<tr>
<td>7</td>
<td>5.45</td>
<td>4.95</td>
<td>8</td>
<td>5.45</td>
<td>4.95</td>
</tr>
<tr>
<td>9</td>
<td>5.45</td>
<td>4.95</td>
<td>10</td>
<td>5.45</td>
<td>4.95</td>
</tr>
<tr>
<td>11</td>
<td>5.45</td>
<td>4.95</td>
<td>12</td>
<td>5.45</td>
<td>4.95</td>
</tr>
<tr>
<td>13</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>14</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>15</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>16</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>17</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>18</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>19</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>20</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>21</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>22</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>23</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>24</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>25</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>26</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>27</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>28</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>29</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>30</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>31</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>32</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>33</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>34</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>35</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>36</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>37</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>38</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
<tr>
<td>39</td>
<td>32000.00</td>
<td>-32000.00</td>
<td>40</td>
<td>32000.00</td>
<td>-32000.00</td>
</tr>
</tbody>
</table>

Closed Loop Points for Grade

<table>
<thead>
<tr>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
<th>POINT</th>
<th>SETPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td>69</td>
<td>8</td>
<td>50</td>
</tr>
</tbody>
</table>

---

Programming Techniques 289
OP-GUIDE LIMITS FOR GRADE

POINT | HIGH LIMIT | LOW LIMIT | POINT | HIGH LIMIT | LOW LIMIT
1 | 14200.00 | 14100.00 | 2 | 9890.00 | 9880.00
3 | 30.00 | 20.00 | 4 | -10.00 | -30.00
5 | 5.10 | 4.90 | 6 | 5.10 | 4.90
7 | 5.10 | 4.90 | 8 | 5.10 | 4.90
9 | 5.10 | 4.90 | 10 | 5.10 | 4.90
11 | 5.10 | 4.90 | 12 | 5.10 | 4.90
13 | 25.00 | 15.00 | 14 | 25.00 | 15.00
15 | 32000.00 | -32000.00 | 16 | 32000.00 | -32000.00
17 | 32000.00 | -32000.00 | 18 | 32000.00 | -32000.00
19 | 32000.00 | -32000.00 | 20 | 32000.00 | -32000.00
21 | 32000.00 | -32000.00 | 22 | 32000.00 | -32000.00
23 | 32000.00 | -32000.00 | 24 | 32000.00 | -32000.00
25 | 32000.00 | -32000.00 | 26 | 32000.00 | -32000.00
27 | 32000.00 | -32000.00 | 28 | 32000.00 | -32000.00
29 | 32000.00 | -32000.00 | 30 | 720.00 | 700.00
31 | 720.00 | 700.00 | 32 | 720.00 | 700.00
33 | 32000.00 | -32000.00 | 34 | 32000.00 | -32000.00
35 | 32000.00 | -32000.00 | 36 | 32000.00 | -32000.00
37 | 32000.00 | -32000.00 | 38 | 720.00 | 700.00
39 | 720.00 | 700.00 | 40 | 720.00 | 700.00

CLOSED LOOP SET POINTS FOR GRADE

POINT | SETPT | POINT | SETPT | POINT | SETPT
1 | 0 | 2 | 0 | 3 | 0 | 4 | 0
5 | 0 | 6 | 0 | 7 | 98 | 8 | 12

PROCESS JOB FILE LOAD  TIME 11.14
GRADE NUMBER 4531  PRODUCTION TIME 0.10

PROCESS JOB FILE LOAD  TIME 11.14
GRADE NUMBER 4532  PRODUCTION TIME 5.45

CLOSED LOOP SET POINTS FOR GRADE

POINT | SETPT | POINT | SETPT | POINT | SETPT
1 | 0 | 2 | 0 | 3 | 0 | 4 | 0
5 | 0 | 6 | 0 | 7 | 98 | 8 | 12

PROCESS JOB FILE LOAD  TIME 11.15

// JOB
A

// END OF ALL JOBS FOR NOW
# Appendix A. TSX System Composition and Capabilities

<table>
<thead>
<tr>
<th></th>
<th>On-line System</th>
<th>Off-line System</th>
<th>On-line System Task Loaded from Cards</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fixed Core</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed Core Contains</td>
<td>Skeleton</td>
<td>Skeleton</td>
<td>Task</td>
</tr>
<tr>
<td>Skeleton Area of Disk Contains</td>
<td>Skeleton</td>
<td>Skeleton</td>
<td>Task</td>
</tr>
<tr>
<td>TSC in Fixed Core</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Nonprocess Monitor (NPM) Functions</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Core Load Build - Nonprocess Core Loads</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Core Load Build - Process Core Loads</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Result of Cold Start</strong></td>
<td>System Skeleton</td>
<td>System Skeleton</td>
<td>Task</td>
</tr>
<tr>
<td><strong>Execute Core Loads</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Execute Process Interrupts</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Execute Programmed Interrupts</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Execute Timer Interrupts</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Data Processing I/O Functions</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Process I/O Functions</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>NPM Error Messages</strong></td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Task Error Messages</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>EAC Error Messages</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td><strong>Control of Task Functions</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Control of NPM Functions</strong></td>
<td>TSC, Nonprocess Supervisor and Job Stack</td>
<td>Non-applicable</td>
<td>Nonprocess Supervisor and Job Stack</td>
</tr>
<tr>
<td><strong>Control of Process Functions</strong></td>
<td>System Skeleton</td>
<td>System Skeleton</td>
<td>Non-applicable</td>
</tr>
<tr>
<td><strong>Method of Calling Core Loads</strong></td>
<td>Name Card, Interrupts, Calls to: Chain, Viaq, Specl, Back, Dpart, Intex, Share, //Xeq and Link</td>
<td>Name Card, Interrupts, Calls to: Chain, Viaq, Specl, Back, Dpart, and Intex</td>
<td>Non-applicable</td>
</tr>
<tr>
<td><strong>Variable Area of Core Contains</strong></td>
<td>Process Core Loads, Nonprocess Core Loads, Nonprocess Monitor, EDP Programs, and Cold Start</td>
<td>Process Core Loads, EDP Programs, and Cold Start</td>
<td>Nonprocess Monitor, System Loader, Task Utilities, Nonprocess Core Loads and Skeleton Builder</td>
</tr>
</tbody>
</table>

Appendix A. TSX System Composition and Capabilities 291
# APPENDIX B. SUMMARY OF TSX CALL STATEMENTS

<table>
<thead>
<tr>
<th>Where Used Code</th>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>CALL INTEX</td>
<td>Causes return of control to MIC on interrupt exit.</td>
</tr>
<tr>
<td>M</td>
<td>CALL CHAIN (NAME)</td>
<td>Mainline core load designated by NAME is loaded and executed.</td>
</tr>
<tr>
<td>M</td>
<td>CALL SPECL (NAME)</td>
<td>Mainline core load containing this call is saved on disk. Mainline core load designated by NAME is loaded and executed.</td>
</tr>
<tr>
<td>M</td>
<td>CALL BACK</td>
<td>Mainline core load saved as a result of a CALL SPECL is restored to core and execution continues with the statement following the special call.</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL QIFON (NAME, P, L, I, E)</td>
<td>Specified interrupts, that have been recorded, will be queued in the order called by the CALL QIFON statement and according to its designated parameters. NAME - name of the mainline core load. P - execution priority of the named mainline core load. L - interrupt level or indicator. I - PISW bit position indicator or CALL COUNT indicators. E - error parameter to specify the action to be taken if queue is full.</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL CLEAR (M, L, I, ...., L, I,)</td>
<td>Specified interrupts will be cleared of recorded status whether they were recorded or not. M specifies the number of L and I parameters to follow. L and I are the same as designated for CALL QIFON. If M = 0, all recorded status is cleared.</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL QUEUE (NAME, P, E)</td>
<td>Mainline core load designated by NAME is entered in core load queue with priority P and error option E.</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL UNQ (NAME, P)</td>
<td>Mainline core load designated by priority P and NAME will be removed from the core load queue.</td>
</tr>
<tr>
<td>M</td>
<td>CALL VIAQ</td>
<td>Last logical statement of a mainline core load. The first core load, of the highest priority entered in the queue, is loaded and executed.</td>
</tr>
<tr>
<td>M, I, C</td>
<td>CALL MASK (I, J)</td>
<td>Interrupt levels specified by data statements for I and J are masked (no unmasking occurs).</td>
</tr>
<tr>
<td>Where Used Code</td>
<td>Statement</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>M, I, C</td>
<td>CALL UNMK (I, J)</td>
<td>Interrupt levels specified by data statements for I and J are unmasked (no masking occurs).</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL SAVMK (I, J)</td>
<td>Interrupt level mask status is saved in I and J (no masking or unmasking occurs).</td>
</tr>
<tr>
<td>M, I, C</td>
<td>CALL RESMK (I, J)</td>
<td>Interrupt levels are masked according to I and J (all others are unmasked).</td>
</tr>
<tr>
<td>M, I, C, N, C</td>
<td>CALL OPMON</td>
<td>Operation Monitor is reset.</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL TIMER (NAME, I, INT)</td>
<td>Interval timer specified by I (1 or 2) is set up to count INT intervals. After INT intervals have elapsed, ITC will branch to NAME (user's subprogram).</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL COUNT (IN, I, INB)</td>
<td>Program interval timer specified by I (1, 2, 3, ..., 9) is set to count INB intervals. Upon completion of INB intervals, the ITC will branch to the subroutine specified by IN (IN specifies a subroutine number from 0 - 31).</td>
</tr>
<tr>
<td>M, X</td>
<td>CALL SHARE (I)</td>
<td>The present core load is saved and nonprocess time-sharing is set up for I timed intervals.</td>
</tr>
<tr>
<td>M, I, C</td>
<td>CALL ENCTS</td>
<td>Time-sharing is terminated.</td>
</tr>
<tr>
<td>M, I, N*, C</td>
<td>CALL SETCL (I)</td>
<td>Programmed clock is set to equal I.</td>
</tr>
<tr>
<td>M, I, N, C</td>
<td>CALL CLOCK (I)</td>
<td>Clock is read into I.</td>
</tr>
<tr>
<td>M, I, C</td>
<td>CALL LEVEL (L)</td>
<td>Calls the programmed interrupt specified by the hardware level L (L must be between 0 - 23).</td>
</tr>
<tr>
<td>M, I, C</td>
<td>CALL DPART</td>
<td>Tests the operation level of present use and, if an interrupt level exists, executes a CALL INTEX; otherwise a CALL VIAQ is executed.</td>
</tr>
</tbody>
</table>

M - Mainline core loads only.
I - Interrupt core loads only.
N - Nonprocess core loads only.
C - Combination mainline and interrupt core load.
* - Must be an XEQ from core load area (INTERVAL TIMER CONTROL REQUIRED)
X - Must be an XEQ from core load area (TIME-SHARING REQUIRED)
This section describes the Assembler language equivalent of the FORTRAN CALL statements provided in the time-sharing executive system.

**Machine Interval Timers**

The Assembler language statements to call the TIMER subprogram are:

```
CALL TIMER
CALL NAME
DC A
DC B
```

where NAME is the name of the subprogram to be executed when the time specified by B has elapsed. A and B must be defined as:

- **A DC 1** For machine interval timer (A).
- **or**
  - **2** For machine interval timer (B).

- **B DC XX** The number of intervals to be counted before the subprogram is executed.

**Programmed Interval Timers**

The Assembler language statements to call the COUNT subprogram are:

```
CALL COUNT
DC A
DC B
DC C
```

where the parameters A, B, and C must be defined as:

- **B DC 1-9** Programmed timer number
- **C DC XX** The number of intervals to be counted before the subprogram is executed.

A **DC 0–31** Number of the subprogram to be executed when the time has elapsed.

The Assembler language statements to be used to read and to set the programmed real-time clock are:

**Read:**

```
CALL CLOCK
DC A
```

where A is the address of the location where the contents of the clock are to be stored.

**Set:**

```
CALL SETCL
DC A
```

where A must be defined as:

- **A DC XXXX** The time to be used for setting the clock. The time must be represented in hours and thousandths of hours (i.e., 00000 through 23999).

**PSC Statements**

The following Assembler language statements are equivalent to the FORTRAN language calls for core load sequencing.

```
CALL BACK
CALL ENDTS
CALL VIAQ
CALL DPART
```

No parameters are required for these calls.
Call Chain:

CALL CHAIN
CALL NAME

where NAME is the name of the core load to be executed.

Call Special:

CALL SPECL
CALL NAME

where NAME is the name of the core load to be executed.

Call Queue:

CALL QUEUE
CALL NAME
DC A
DC B

where NAME is the name of a core load to be added to the queue. A and B must be defined as follows.

A DC 1-32,767 Priority Number
B DC 1-32,766 Replace the lowest priority entry on error condition
     or
     0 Ignore the call on error condition
     32,767 Restart on error condition

Call Unqueue:

CALL UNQ
CALL NAME
DC A

where NAME is the name of a core load whose entry is to be removed from the queue. A must be defined as follows:

A DC 1-32,767 Priority Number

Call Time-Share:

CALL SHARE
DC A

where A must be defined as follows:

A DC XX Number of programmed timer base intervals to be used for nonprocess operations.

Call Programmed Settable Interrupts:

CALL LEVEL
DC A

where A must be defined as:

A DC 0-23 User specified hardware level to cause interrupt

Interrupt Calls

The following Assembler language statements are used to service and clear recorded interrupts.

Call Interrupt Exit:

CALL INTOE No parameters are required for this call

Service Recorded Interrupts

CALL QIFON
CALL NAME
DC A
DC B
DC C
DC D

where NAME is the name of the core load to be serviced if recorded. A, B, C, and D must be defined as follows:
A DC XX Priority number
B DC XX Interrupt level number or indicator
C DC XX Position within PISW or indicator
D DC 1-32,767 Replace lowest priority entry on error condition
or 0 Ignore the call on error condition
or 32,767 Restart on error condition

Clear Recorded Interrupts:

<table>
<thead>
<tr>
<th>CALL CLEAR</th>
<th>CALL CLEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC A</td>
<td>DC A (when A = 0)</td>
</tr>
<tr>
<td>DC B(1)</td>
<td></td>
</tr>
<tr>
<td>DC C(1)</td>
<td></td>
</tr>
<tr>
<td>DC B(2)</td>
<td></td>
</tr>
<tr>
<td>DC C(2)</td>
<td></td>
</tr>
</tbody>
</table>

where A, B, and C must be defined as follows:

A DC XX Number of Bs and Cs which follow. If zero, all recorded status is cleared
B DC XX Interrupt level number or indicator. Not used if A = 0
C DC XX Position within PISW or indicator. Not used if A = 0

Miscellaneous Subroutines:
The following Assembler language statements are used to link the miscellaneous subroutines.

Mask:

<table>
<thead>
<tr>
<th>CALL MASK</th>
<th>CALL MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC A</td>
<td>DC A</td>
</tr>
<tr>
<td>DC B</td>
<td>DC B</td>
</tr>
</tbody>
</table>

where A and B must be defined as:

Unmask:

<table>
<thead>
<tr>
<th>CALL UNMK</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC A</td>
</tr>
<tr>
<td>DC B</td>
</tr>
</tbody>
</table>

where A and B must be defined the same as shown for CALL MASK. The designated levels are unmasked.

Save Mask:

<table>
<thead>
<tr>
<th>CALL SAVMK</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC A</td>
</tr>
<tr>
<td>DC B</td>
</tr>
</tbody>
</table>

where A and B are the addresses of the core storage words where the contents of the interrupt mask register are to be placed:

Restore Mask:

<table>
<thead>
<tr>
<th>CALL RESMK</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC A</td>
</tr>
<tr>
<td>DC B</td>
</tr>
</tbody>
</table>

where A and B are the levels defined for the CALL MASK or CALL UNMK.

Reset Operations Monitor:

<table>
<thead>
<tr>
<th>CALL OPMON</th>
</tr>
</thead>
<tbody>
<tr>
<td>No parameters are required for this call</td>
</tr>
</tbody>
</table>
### APPENDIX D. CONTENTS OF THE FIXED AREA OF CORE

<table>
<thead>
<tr>
<th>Address</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Description of Use</th>
<th>Address</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Description of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>0000</td>
<td>0000</td>
<td>Reserved</td>
<td>00000</td>
<td>0000</td>
<td>00038</td>
<td>Physical 1443-1 device table address</td>
</tr>
<tr>
<td>00001</td>
<td>0001</td>
<td>0001</td>
<td>Branch instruction (/4400)</td>
<td>00001</td>
<td>0001</td>
<td>00039</td>
<td>Physical 1443-2 device table address</td>
</tr>
<tr>
<td>00002</td>
<td>0002</td>
<td>0002</td>
<td>CE routine entry address (EACA)</td>
<td>00002</td>
<td>0002</td>
<td>00040</td>
<td>Beginning address of MIC</td>
</tr>
<tr>
<td>00003</td>
<td>0003</td>
<td>0003</td>
<td>Selective dump word</td>
<td>00003</td>
<td>0003</td>
<td>00041</td>
<td>User time-sharing time</td>
</tr>
<tr>
<td>00004</td>
<td>0004</td>
<td>0004</td>
<td>Interval timer A</td>
<td>00004</td>
<td>0004</td>
<td>00042</td>
<td>Constant: -1</td>
</tr>
<tr>
<td>00005</td>
<td>0005</td>
<td>0005</td>
<td>Interval timer B</td>
<td>00005</td>
<td>0005</td>
<td>00043</td>
<td>Constant: -10</td>
</tr>
<tr>
<td>00006</td>
<td>0006</td>
<td>0006</td>
<td>Interval timer C</td>
<td>00006</td>
<td>0006</td>
<td>00044</td>
<td>Entry address to 1053 no response subroutine</td>
</tr>
<tr>
<td>00007</td>
<td>0007</td>
<td>0007</td>
<td>General I/O busy indicator</td>
<td>00007</td>
<td>0007</td>
<td>00045</td>
<td>Timer busy indicators</td>
</tr>
<tr>
<td>00008</td>
<td>0008</td>
<td>0008</td>
<td>Internal error interrupt branch address</td>
<td>00008</td>
<td>0008</td>
<td>00046</td>
<td>Mask register (0-13)</td>
</tr>
<tr>
<td>00009</td>
<td>0009</td>
<td>0009</td>
<td>Trace interrupt branch address</td>
<td>00009</td>
<td>0009</td>
<td>00047</td>
<td>IOCC control word for UNMK1</td>
</tr>
<tr>
<td>00010</td>
<td>000A</td>
<td>000A</td>
<td>Mainline return address from CE routine</td>
<td>00010</td>
<td>000A</td>
<td>00048</td>
<td>Mask register (14-23)</td>
</tr>
<tr>
<td>00011</td>
<td>000B</td>
<td>000B</td>
<td>Level 0 interrupt address</td>
<td>00011</td>
<td>000B</td>
<td>00049</td>
<td>IOCC control word for UNMK2</td>
</tr>
<tr>
<td>00012</td>
<td>000C</td>
<td>000C</td>
<td>Level 1 interrupt address</td>
<td>00012</td>
<td>000C</td>
<td>00050</td>
<td>Mask levels 0-13</td>
</tr>
<tr>
<td>00013</td>
<td>000D</td>
<td>000D</td>
<td>Level 2 interrupt address</td>
<td>00013</td>
<td>000D</td>
<td>00051</td>
<td>IOCC for levels 0-13</td>
</tr>
<tr>
<td>00014</td>
<td>000E</td>
<td>000E</td>
<td>Level 3 interrupt address</td>
<td>00014</td>
<td>000E</td>
<td>00052</td>
<td>Mask levels 14-23</td>
</tr>
<tr>
<td>00015</td>
<td>000F</td>
<td>000F</td>
<td>Level 4 interrupt address</td>
<td>00015</td>
<td>000F</td>
<td>00053</td>
<td>IOCC for levels 14-23</td>
</tr>
<tr>
<td>00016</td>
<td>0010</td>
<td>0010</td>
<td>Level 5 interrupt address</td>
<td>00016</td>
<td>0010</td>
<td>00054</td>
<td>Pseudo accumulator (WK4)</td>
</tr>
<tr>
<td>00017</td>
<td>0011</td>
<td>0011</td>
<td>Level 6 interrupt address</td>
<td>00017</td>
<td>0011</td>
<td>00055</td>
<td>Pseudo accumulator (WK5)</td>
</tr>
<tr>
<td>00018</td>
<td>0012</td>
<td>0012</td>
<td>Level 7 interrupt address</td>
<td>00018</td>
<td>0012</td>
<td>00056</td>
<td>1 = time-sharing is in progress; 0 = not in progress</td>
</tr>
<tr>
<td>00019</td>
<td>0013</td>
<td>0013</td>
<td>Level 8 interrupt address</td>
<td>00019</td>
<td>0013</td>
<td>00057</td>
<td>Address of magnetic tape sense control word</td>
</tr>
<tr>
<td>00020</td>
<td>0014</td>
<td>0014</td>
<td>Level 9 interrupt address</td>
<td>00020</td>
<td>0014</td>
<td>00058</td>
<td>1 = ITC is in system director; 0 = ITC is not included</td>
</tr>
<tr>
<td>00021</td>
<td>0015</td>
<td>0015</td>
<td>Level 10 interrupt address</td>
<td>00021</td>
<td>0015</td>
<td>00059</td>
<td>Non-zero indicates TASK is in core</td>
</tr>
<tr>
<td>00022</td>
<td>0016</td>
<td>0016</td>
<td>Level 11 interrupt address</td>
<td>00022</td>
<td>0016</td>
<td>00060</td>
<td>Address of timer A subroutine</td>
</tr>
<tr>
<td>00023</td>
<td>0017</td>
<td>0017</td>
<td>Level 12 interrupt address</td>
<td>00023</td>
<td>0017</td>
<td>00061</td>
<td>Address of timer B subroutine</td>
</tr>
<tr>
<td>00024</td>
<td>0018</td>
<td>0018</td>
<td>Level 13 interrupt address</td>
<td>00024</td>
<td>0018</td>
<td>00062</td>
<td>Program timer 1</td>
</tr>
<tr>
<td>00025</td>
<td>0019</td>
<td>0019</td>
<td>Level 14 interrupt address</td>
<td>00025</td>
<td>0019</td>
<td>00063</td>
<td>Timer 1 subprogram number (1-32)</td>
</tr>
<tr>
<td>00026</td>
<td>001A</td>
<td>001A</td>
<td>Level 15 interrupt address</td>
<td>00026</td>
<td>001A</td>
<td>00064</td>
<td>Timer 1 on-off branch</td>
</tr>
<tr>
<td>00027</td>
<td>001B</td>
<td>001B</td>
<td>Level 16 interrupt address</td>
<td>00027</td>
<td>001B</td>
<td>00065</td>
<td>Program timer 2</td>
</tr>
<tr>
<td>00028</td>
<td>001C</td>
<td>001C</td>
<td>Level 17 interrupt address</td>
<td>00028</td>
<td>001C</td>
<td>00066</td>
<td>Timer 2 subprogram number (1-32)</td>
</tr>
<tr>
<td>00029</td>
<td>001D</td>
<td>001D</td>
<td>Level 18 interrupt address</td>
<td>00029</td>
<td>001D</td>
<td>00067</td>
<td>Timer 2 on-off branch</td>
</tr>
<tr>
<td>00030</td>
<td>001E</td>
<td>001E</td>
<td>Level 19 interrupt address</td>
<td>00030</td>
<td>001E</td>
<td>00068</td>
<td>Program timer 3</td>
</tr>
<tr>
<td>00031</td>
<td>001F</td>
<td>001F</td>
<td>Level 20 interrupt address</td>
<td>00031</td>
<td>001F</td>
<td>00069</td>
<td>Timer 3 subprogram number (1-32)</td>
</tr>
<tr>
<td>00032</td>
<td>0020</td>
<td>0020</td>
<td>Level 21 interrupt address</td>
<td>00032</td>
<td>0020</td>
<td>00070</td>
<td>Timer 3 on-off branch</td>
</tr>
<tr>
<td>00033</td>
<td>0021</td>
<td>0021</td>
<td>Level 22 interrupt address</td>
<td>00033</td>
<td>0021</td>
<td>00071</td>
<td></td>
</tr>
<tr>
<td>00034</td>
<td>0022</td>
<td>0022</td>
<td>Level 23 interrupt address</td>
<td>00034</td>
<td>0022</td>
<td>00072</td>
<td></td>
</tr>
<tr>
<td>00035</td>
<td>0023</td>
<td>0023</td>
<td>1 = Loop on 1443 not ready during nonprocess programs; 0 = go to EAC</td>
<td>00035</td>
<td>0023</td>
<td>00073</td>
<td></td>
</tr>
<tr>
<td>00036</td>
<td>0024</td>
<td>0024</td>
<td>Logical 1443-1 device table address</td>
<td>00036</td>
<td>0024</td>
<td>00074</td>
<td></td>
</tr>
<tr>
<td>00037</td>
<td>0025</td>
<td>0025</td>
<td>Logical 1443-2 device table address</td>
<td>00037</td>
<td>0025</td>
<td>00075</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Description of Use</td>
<td>Address</td>
<td>Description of Use</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>--------------------</td>
<td>---------</td>
<td>--------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00071</td>
<td>Program timer 4</td>
<td>00105</td>
<td>Interrupt core load ending address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00072</td>
<td>Timer 4 subprogram number (1-32)</td>
<td>00106</td>
<td>CALL CHAIN entry</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00073</td>
<td>Timer 4 on-off branch</td>
<td>00107</td>
<td>System director ending address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00074</td>
<td>Program timer 5</td>
<td>00108</td>
<td>Mask word out-of-core interrupts (0-13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00075</td>
<td>Timer 5 subprogram number (1-32)</td>
<td>00109</td>
<td>Mask word for out-of-core interrupts (14-23)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00076</td>
<td>Timer 5 on-off branch</td>
<td>00110</td>
<td>System mask save area (0-13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00077</td>
<td>Program timer 6</td>
<td>00111</td>
<td>System mask save area (14-23)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00078</td>
<td>Timer 5 subprogram number (1-32)</td>
<td>00112</td>
<td>1 = A1 basic overlap feature available</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00079</td>
<td>Timer 6 on-off branch</td>
<td>00113</td>
<td>1 = A1 expander overlap feature available</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00080</td>
<td>Program timer 7</td>
<td>00114</td>
<td>Entry address for I/O error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00081</td>
<td>Timer 7 subprogram number (1-32)</td>
<td>00115</td>
<td>EAC error code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00082</td>
<td>Timer 7 on-off branch</td>
<td>00116</td>
<td>Error information</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00083</td>
<td>Program timer 8</td>
<td>00117</td>
<td>Error information</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00084</td>
<td>Timer 8 subprogram number (1-32)</td>
<td>00118</td>
<td>Error information</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00085</td>
<td>Timer 8 on-off branch</td>
<td>00119</td>
<td>Error information</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00086</td>
<td>Program timer 9</td>
<td>00120</td>
<td>Entry address of EAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00087</td>
<td>Timer 9 subprogram number (1-32)</td>
<td>00121</td>
<td>Address of queue table</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00088</td>
<td>Timer 9 on-off branch</td>
<td>00122</td>
<td>Maximum number of queue entries</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00089</td>
<td>Timer-sharing timer</td>
<td>00123</td>
<td>Number of interrupt levels used (NULEV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00090</td>
<td>Exit address of I/O routines</td>
<td>00124</td>
<td>Entry address of disk routine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00091</td>
<td>Time-sharing timer busy indicator</td>
<td>00125</td>
<td>Entry address of list printer routine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00092</td>
<td>Programmed clock</td>
<td>00126</td>
<td>Entry address of system printer routine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00093</td>
<td>Programmed clock</td>
<td>00127</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00094</td>
<td>Branch to ITC exit routine</td>
<td>00128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00095</td>
<td>Constant: -50</td>
<td>00129</td>
<td>Constant: /0600</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00096</td>
<td>Constant: 3</td>
<td>00130</td>
<td>Constant: /0500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00097</td>
<td>PAUSE routine indicator specifying that interrupt has occurred</td>
<td>00131</td>
<td>Constant: /7800</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00098</td>
<td>Entry address of I/O test routine (IOTST)</td>
<td>00132</td>
<td>Constant: /0FFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00099</td>
<td>1 = CARDN is in skeleton</td>
<td>00133</td>
<td>Constant: /0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00100</td>
<td>Mask routine indicator: 1 = out-of-core interrupts are masked</td>
<td>00134</td>
<td>Constant: /0002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101</td>
<td>Address of CALL INTEX processing routines</td>
<td>00135</td>
<td>Constant: /0004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00102</td>
<td>Beginning address of variable core</td>
<td>00136</td>
<td>Constant: /0005</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00103</td>
<td>TV location (XR3)</td>
<td>00137</td>
<td>Constant: /0007</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00104</td>
<td>Interrupt level work area address (XR3)</td>
<td>00138</td>
<td>Constant: /0FFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>00139</td>
<td>Constant: /2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>00140</td>
<td>Constant: /0180</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

298
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Description of Use</th>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Description of Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>00141</td>
<td>008D</td>
<td>Constant: 320</td>
<td>00171</td>
<td>00A8</td>
<td>TSC indicator: 0 = Call nonprocess monitor; 1 = Call program from save area</td>
</tr>
<tr>
<td>00142</td>
<td>008E</td>
<td>CALL LINK entry address</td>
<td>00172</td>
<td>00AC</td>
<td>Entry point to TYSAV</td>
</tr>
<tr>
<td>00143</td>
<td>008F</td>
<td>Address of EAC disk down message</td>
<td>00173</td>
<td>00AD</td>
<td>Entry point to TVEXT</td>
</tr>
<tr>
<td>00144</td>
<td>0090</td>
<td>Constant: 321</td>
<td>00174</td>
<td>00AE</td>
<td>Ending address of skeleton I/O area</td>
</tr>
<tr>
<td>00145</td>
<td>0091</td>
<td>Address of first word after ETV</td>
<td>00175</td>
<td>00AF</td>
<td>Address of 1053 logical table</td>
</tr>
<tr>
<td>00146</td>
<td>0092</td>
<td>Constant: /FF00</td>
<td>00176</td>
<td>00B0</td>
<td>Address of message buffer table</td>
</tr>
<tr>
<td>00147</td>
<td>0093</td>
<td>Constant: /FO00</td>
<td>00177</td>
<td>00B1</td>
<td>Not used in an on-line system. Under TASK, this is the start- ing address of the TASK core dump program.</td>
</tr>
<tr>
<td>00148</td>
<td>0094</td>
<td>Constant: /FF87</td>
<td>00178</td>
<td>00B2</td>
<td>Address of EAC constants</td>
</tr>
<tr>
<td>00149</td>
<td>0095</td>
<td>Not used</td>
<td>00179</td>
<td>00B3</td>
<td>Entry address of nonprocess monitor read-in routine</td>
</tr>
<tr>
<td>00150</td>
<td>0096</td>
<td>Program timer busy Indicator</td>
<td>00180</td>
<td>00B4</td>
<td>Disk FIO save area address</td>
</tr>
<tr>
<td>00151</td>
<td>0097</td>
<td>Not used</td>
<td>00181</td>
<td>00B5</td>
<td>Non-disk FIO save area address</td>
</tr>
<tr>
<td>00152</td>
<td>0098</td>
<td>Entry address to set timers busy</td>
<td>00182</td>
<td>00B6</td>
<td>Entry point to EXIT subroutine</td>
</tr>
<tr>
<td>00153</td>
<td>0099</td>
<td>Entry point to EAC printer routine</td>
<td>00183</td>
<td>00B7</td>
<td>Address of message buffer disk address</td>
</tr>
<tr>
<td>00154</td>
<td>009A</td>
<td>Entry point to OZSAV</td>
<td>00184</td>
<td>00B8</td>
<td>Entry for DISKN from ETV</td>
</tr>
<tr>
<td>00155</td>
<td>009B</td>
<td>Entry point to QZEXT</td>
<td>00185</td>
<td>00B9</td>
<td>Entry for TYPEN from ETV</td>
</tr>
<tr>
<td>00156</td>
<td>009C</td>
<td>High Core ending address of INSKEL COMMON, plus 1</td>
<td>00186</td>
<td>00BA</td>
<td>Entry for PRNTN from ETV</td>
</tr>
<tr>
<td>00157</td>
<td>009D</td>
<td>Length of skeleton COMMON</td>
<td>00187</td>
<td>00BB</td>
<td>Address of disk drive table</td>
</tr>
<tr>
<td>00158</td>
<td>009E</td>
<td>Address of message buffer drive code</td>
<td>00188</td>
<td>00BC</td>
<td>Address of logical drive 0 device table</td>
</tr>
<tr>
<td>00159</td>
<td>009F</td>
<td>I/O Error routine entry/return address</td>
<td>00189</td>
<td>00BD</td>
<td>Address of logical drive 1 device table</td>
</tr>
<tr>
<td>00160</td>
<td>00A0</td>
<td>Program interrupt I0CC (0-13)</td>
<td>00190</td>
<td>00BE</td>
<td>Address of logical drive 2 device table</td>
</tr>
<tr>
<td>00161</td>
<td>00A1</td>
<td>I0CC control word</td>
<td>00191</td>
<td>00BF</td>
<td>Address of physical drive 0 device table</td>
</tr>
<tr>
<td>00162</td>
<td>00A2</td>
<td>Program interrupt I0CC (14-23)</td>
<td>00192</td>
<td>00C0</td>
<td>Address of physical drive 1 device table</td>
</tr>
<tr>
<td>00163</td>
<td>00A3</td>
<td>I0CC control word</td>
<td>00193</td>
<td>00C1</td>
<td>Address of physical drive 2 device table</td>
</tr>
<tr>
<td>00164</td>
<td>00A4</td>
<td>TASK nonprocess monitor abort indicator</td>
<td>00194</td>
<td>00C2</td>
<td>Address of save area for unformatted FIO</td>
</tr>
<tr>
<td>00165</td>
<td>00A5</td>
<td>EAC printer type code: 0 = 1003, 1 = 1443</td>
<td>00195</td>
<td>00C3</td>
<td>Address of iniskel ETV</td>
</tr>
<tr>
<td>00166</td>
<td>00A6</td>
<td>List printer type code: 0 = 1003, 1 = 1443</td>
<td>00196</td>
<td>00C4</td>
<td>Address of variable ETV</td>
</tr>
<tr>
<td>00167</td>
<td>00A7</td>
<td>System printer type code: 0 = 1003, 1 = 1443</td>
<td>00197</td>
<td>00C5</td>
<td>Special save indicator</td>
</tr>
<tr>
<td>00168</td>
<td>00A8</td>
<td>Core size minus 1</td>
<td>00198</td>
<td>00C6</td>
<td>User time-sharing (TISHA) for CALL VIAQ when Queue is empty.</td>
</tr>
<tr>
<td>00169</td>
<td>00A9</td>
<td>Address of 1442 entry in interrupt branch table</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00170</td>
<td>00AA</td>
<td>Address of FORTRAN I/O table</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INDEX

Absolute loader function 13
    TASK 187
    operation 188
Absolute programs, sector break 128-130
A and B timers 43-45
ADC 191
Address protection 185
AIPTN subroutine 75
AIRN subroutine 75
AISQN subroutine 75
Analog
    input functions 206
    input log listing 264-266
    multiplexer 191
    process I/O 191
    to-digital converter (ADC) 191
Analog input basic (AIB) 137
Analog input basic with comparator (AIBC) 137
Analog input extended (AIE) 137
Analog input extended with comparator (AIEC) 137
Arithmetic and functional subroutines 10
Assemble and execute nonprocess program 80-81
Assembler
    ASM 11, 74, 122
    Call subroutines 181
    Control cards 79, 181
    implicit and explicit subroutines 169-170
    TSX calls 294-296
    Writing assembler language subroutines 181-184
Assembling programs 80-82
Assignment card restrictions 128
*ASSIGNMENT cards, function of 126-127
Assignment table (AT) and loader 122
Automatic mode, loader 188
Automatic time-sharing (VIAQ) 49
BACK 19-20
Back-up capability, EAC 60
Bootstrap for nonprocess supervisor 122
Buffering of messages 119-121
Buffer size, disk 120
Building TSX operating tables 124-125
BZ1-8 118
Calculating core size
    skeleton 172
    system director 132-133
    TASK 120-121
CALL
    BACK 19-20
    CHAIN 18
    CLEAR 41
    CLOCK 46
    COUNT 46
    DPART 31
    ENDTS 48
    EXIT 49
    INTEX 30
    LEVEL 49, 145
    LINK 49
    MASK 40
    OPMON 59
    QIFON 23-25
    QUEUE 20-23
    RESMK 40
    RETURN 33
    SAVMK 40
    SETCL 45
    SHARE 48
    SPECI 18
    TIMER 43
    UNMK 40
    UNQ 23
    VIAQ 25, 49
Call COUNT subroutines 147
CALL subroutines 186
    assembler 181
    timer 165
    type 179
    user programs 187
Calling process core load externally 106
CAR 68
Card reproducing 107
CARDN subroutine 75, 166
   SUP 72
*CCEND and dump 98
CDINS 118
CDW example 163
C.E. interrupt 199
C.E. level interrupt switch 70
CHAIN 18
Chaining technique 18-20
Channel address register (CAR) 68
Characteristics of interrupts 29
Checkpoint operation 197
CLEAR subroutine 41
Clock 46
    real time 45-46
    real time setup (SETCL) 45
Closed loop control 199
Coding, reentrant 6
Cold start
    cards 121
    CLST 9, 122, 151
    EAC 70
    logical number 109
off-line  111  
on-line  107  
program  9, 122  
COLDs routines and time-sharing  198  
Combination core load  215  
COMMON  
  COMM  179  
  area  6-7,  76  
  area,  INSKEL/interrupt/normal  6-7  
  communications area and NPM  71  
  INSKEL  75  
  LOCAL  99  
Communications area  
  Disk  72  
  nonprocess  72  
Communications linkages  101-102  
Compile and execute nonprocess programs  82  
Compile and store nonprocess programs  81-82  
Compiler, FORTRAN  11,  74  
Compiler, system  7-11  
COMSZ  
  equate card  119,  161  
Configuration, disk system  147-161  
Console  200  
  interrupt  111-112  
  interrupt logic  113  
  CONTA  118  
Contact sense  192  
Continue, EAC  71  
Control  
  Direct Digital (DDC)  195  
  Error Alert (EAC)  14  
  Interval Timer (ITC)  14,  133  
  Master Interrupt (MIC)  13,  33-39  
  Multi-level  16  
  Program Sequence (PSC)  14,  17-18  
  Programs  8-9  
  Record analyzer, monitor  72  
  Record Entry Table (CRENT)  175  
  Time-sharing (TSC)  14  
Control cards  
  Assembler  181  
  *INCLD  173  
  sequence of system loader  124  
Control cards, Monitor  77  
  assembler  79  
  DUP  78  
  FORTRAN  78  
  loader  77  
  nonprocess monitor  77-79  
  simulator  79  
Conversion subroutines  10,  76  
Copying process data, example of  99  
Core  
  contents, Fixed Area  297-299  
  Dump, EAC  59  
  exchange method, time-sharing  14  
  image formats  128  
  map, skeleton  178  
  off-line requirements  165  
  on-line (non time-sharing) requirements  165  
  on-line (time-sharing) requirements  164  
  storage factors  164  
  variable (VCORE)  8  
Core load  
  area  82,  85,  101,  150,  160  
  area, delete  84-85  
  area, file-moving  99  
  area, loading  98  
  builder (CLB)  9,  122  
  combination  84  
  concept of a  5  
  deleting and replacing  82-88  
  file area, reserving  97-98  
  interrupt  84  
  link  84  
  linkages, changing  86  
  mainline and interrupt  31,  84  
  mainline queue tables  132  
  nonprocess  84  
  queue table  14  
  rebuild  176  
Core Load Builder (CLB)  122  
Core size  
  calculating skeleton  172  
  skeleton  164-171  
  system director  132-133  
COUNT subroutine  46,  147,  166,  197  
  NITP1/NITP2  147  
Counters, update EAC error  60  
  CRDMP Relocatable Dump to Cards program  123  
  CRENT  175  
  Cycle steal  192  
DAC  192  
DACS  9,  191  
  analog process I/O  191-192  
  applications  193  
  digital process I/O  192  
DAOP subroutine  75  
Data  
  channels and masking  29  
  entry dials  202  
  format  187  
  manipulation  97-99  
  sheets, program  210-220  
Data acquisition and process control system (DACS)  9,  191  
Data files  84  
  deleting/replacing  82-88  
  dump from core load  98  
DCOM  72,  122  
  entries  125  
SUP  72  
DDC  195,  199  
Debugging  
  core loads  87  
  process programs  112  
  simulator  87-96  
  system  203

Index  301
*DEDIT control card 127
parameters 127
*DEFINE CONFIG 151-156, 164
control card 151
*DEFINE PAKDK (DUP) 150
operation 107
*DEFINE REMOV 150
*DELETE 82
Delete mainline 84-85
combination core loads 84-85
interrupt core loads 84-85
Deleting
core loads 82-88
data files 82-88
programs 82-88
Deleting from queue (UNQ) 23
Design considerations, system 115-143
Devices with no interrupt 127
DICMP subroutine 75
DIEXP subroutine 76
Digital
control, direct 195
data display 202
input switches 202
process I/O 192
to-Analog Converter (DAC) 192
DINP subroutine 76
Direct digital control (DDC) 195, 199
Director, system 8, 130-143
Disk
areas, assignment 159
buffer size 120
cartridge initialization 157-159
cartridges, changing of 148
configuration examples 154-156
drives and logical number 109
edit phase 128
file organization, example 205
hardware 148
layout 149-151
layout editing 125
map for system programs 122
organization 148-151
sector distribution 159
storage 159-161
system configuration 147-161
units conversion 148
utility program (DUP) 10-11, 74, 122
Disk Communications Area (DCOM) 72, 122
DISKN subroutine 3, 75, 166
EDP relation 70
DLABL
disk initialization 158
DWRAD, TDWA 159
functions of 159
LET/FLET 158
DMPAN Dump analysis program 123
DMPAI Dump analysis program 123
DMPA2 Dump analysis program 123
DPART routine 31
DP and process I/O subroutines 10, 166
Dummy interrupt core load 111
Dump
core storage 59
EAC 134
from relocatable area 98-99
from user area 98-99
LET/FLET table 107
*DUMP and *DUMPDATA 98
function of DUP 59
Dumping from core load 98
*DUMPLET
LET/FLET 107
listing 235, 267, 278
use of 84
DUP 10, 11, 74, 122
control cards 78
facilities 11
ICLT 175
sectors on system cartridge 74
// DUP control record 74
DWRAD
disk initialization 157
functions of 157-158
TDWA, DLABL comparison 159
EAC 14, 59-71
dump 134
error action 68-71
error codes 61
features of 59-60
on-line recovery 62-67
printer 199
program breakdown 60-68
system director 131
EBPA 102
EBT and ETV 163
LIBF 164
ECO 192
ECPT2 118
Edit phase, disk 128
Editing and LET/FLET, DCOM 125
Editing disk layout 125
EDP 151
DISKN relation 70
Electronic contact operate (ECO) 192
//END card and SUP 72
//END OF ALL JOBS card 25, 26
VIAQ 49
ENDTS 48-49
Entry to MIC 33-37
EPDM 150
Entry to SUP 73
EPSV 150
Equate cards and system director 131
TASK 117-119
Error
action, EAC 68-71
alert Control (EAC) 14, 59-71
codes, EAC 61
condition, servicing 4
counters, updating 60
decision subroutines 68, 70
disk program (EDP) 68
Dump Area (EDPM) 150
procedures, I/O 198
sample 197
programs 123
programs (EDP) 151
save area (EPSV) 150
subroutine, unused 60
ETV 162, 175
EBT 163
element 163
LIB 162
PNT 163
Event sequence 15
Examples
Nonprocess Monitor 79-114
Execute highest priority core load (VIAQ) 25
Execute nonprocess
from core load area 82
from temporary area 80-81
Executive programs, functions of 12-114
EXIT 49
Exit
EAC 68
from MIC 37-39
procedures from interrupt 33
through interrupt level 71
Explicit subroutines 166
External calling of process core load 106
External interrupt level 134
FADD subroutine 143
Features, EAC 59-60
File area, reserving core load 97-98
File protection 149
*FILES control record 97
Fixed area 161
core contents 297-299
IOCC 145
FLIT 82
definition of 122
entries 109
table dump 107
tables 122
FLIP 10, 101
FOR 11, 74, 123
FORTRAN
compiler (FOR) 11, 74, 123
corel cards 78
DP I/O 166
files, sample use 206
implicit and explicit subroutines 167-168
I/O save area 150, 160
I/O subroutines 10
I/O table, example 163
sectors on system cartridge 74

Function buttons 202
Functional simulate-class subroutines 76
Functions of executive programs 12-114
Functions of system director 133-135

General TSX subroutines 76
Guard interrupt core load 111

Handling of interrupts 27-41
Hardware timers 43-45
HOLL 102

IAC
code example 126
code and I/O interrupt 182
codes and *ASSIGNMENT cards 126
codes and I/O 127
IBT 39
element 163
ICI 176, 179
ICLT 14, 35, 84, 144
core load rebuild 176
IST 31
map 179
map interpretation 177
NIL card and DUP 175
PISW 143
ICNT Indicator in INSHEL COMMON 50
ILSW 30, 126
assignment 136
bits and *ASSIGNMENT cards 126
interrupt level 35
PISW 143
Implementation of LOCALs 99-102
Implicit subroutines 166
*INCLD control card
core load rebuild 176
use of 173
In-core, EAC 68
In-core interrupt (ICI) 176, 179
Index registers and MIC 34
Initiating nonprocess monitor operation 109-110
Initiating time-sharing 48-50
Insert into queue (QUEUE) 20
INSK 179
INSHEL
call COUNT 210, 211, 213, 224-225, 228-229, 234
call TIMER 210, 227
COMMON 4, 50, 179, 203
interrupt subroutines 210-214, 225, 228, 229, 233
INSV 150
Internal interrupt level 134
Internal machine error, causes 68
Interprogram communication 7
Interrupt 7, 27-29
action of MIC 36
assignment restrictions 137
Branch Table (IBT) 35, 39
characteristics of 29
Control, Master (MIC) 13, 33-39

Index 303
core load 31-32, 215-218, 257-264
core load, delete 84-85
Core Load Table (ICLT) 14
exit (INTEX) 33
handling of 4, 27-41, 71
initiation of 15-16
I/O 27, 39
levels 28, 135, 185
levels exit 71
levels masking 29
levels, out-of-core 146
level and PISW group number 144
level, sensed by ILSW 35
masking 39-40, 186
per level 137
periodic 14
philosophy 7, 27-29
priority 135
procedures 33
programs, Mainline/Skeleton 30
*RCORD card 42
recording 41-42
restrictions 40, 71
routine exit (DPART) 33
routines, Mainline/Skeleton 30
Save Area (INSV) 37, 150
servicing subroutines (ISS) 30, 41, 42, 144
Status Table (IST) 31, 35
structure 7
use of console 111-112
Interval timer control (ITC) 14, 133
Interval timers, use of 42-47
INTEX 30
INTKI, INTK2 117, 118
Input/Output (I/O)
  analog process 191
  associated conversion subroutines 166
  device subroutines 30
  digital process 192
  error procedures 198
  interrupts 27, 39
  simulator subroutines 75
  skeleton 12-13, 161
  subroutines 182
  unit table (IOUT) and loader 122
  unit table (IOUT) and LUN 125
IOC C locations in Fixed Area 145
ISS subroutine 144, 176
  example of 183-184
  I/O subroutine 182
IST 31, 35
ICLT 31
ITC 14
  functions of 42
  system director 131
JOB 79-80
  //JOB card and DCOM 72
  SUP 72
Job deck 79-80
JTEST indicator in INSkel COMMON 46
Keyboard Request, see INTKI, 117, 118
Language translators 11
LD(I) subroutine and timers 44, 45
LET 74, 80, 82
  entries 109
  /FLET and DLABEL 158, 159
  /FLET entries 109, 125
  table dump 107
  tables 122
  use of 81-82
Level
  interrupt 134
  interrupts per 137
  interrupt switch 70
LEVEL subroutine 10, 49, 145
Level work areas 14, 137, 185
  fixed section 139
  layout 138
  overlay section 138, 140-143
LIBF 101, 179
FLIP, LPT, VTV 101
  subroutines 182, 186
  user programs 187
Library Function Subroutines (LIBF) 101, 179, 182, 186, 187
Library, subroutine 123
LINK 49
Linkages, changing core load 86
Linkages, communication 101-102
LINS parameter 165
Loader
  control cards 77
  operation 123-125
  system 9, 122
Loading
  into core load area 98
  into NPWS 99
  nonprocess system 123-124
Load Monitor function 13
LOCAL
  calls 101
  COMMON 99
  definition of 5
  implementation of 99-102
  restrictions on use of 102
  subprograms 5
  uses of 102-103
Location Equivalence Table (LET) 74, 80-82, 107, 109, 110, 122,
  125, 158-159
Log description, sample program 206
Logical number and disk drives 109
LPT (Local Parameter Table) 101
LSKEL parameter 164
LSKEL and VCORE 164
LUN
  IAC codes 127
  IOUT 125
maximum LUN on TSX 126
number assignment change 128
numbers and *ASSIGNMENT card 126

Machine features 1-2
Machine Timers: A, B, C 14
MAGT subroutine 75
Mainline Core Load 31, 213, 217
debbuging with 112
listing of 238-247, 250-257
queue table 14, 132
Mainline process, delete 84-85
Mainline segmentation 22
Manipulation, data 97-99
Manual mode, loader 188
MASK subroutine 10, 39
Masking and data channels 29
Masking Interrupts 39-40, 186
Master Branch Table (MBT) 125
Master Communications Area and Skeleton Common 175
Master Communications Words (MCW) 175
Master Interrupt Control (MIC) 13, 33-39, 131, 144, 185
MBT (Master Branch Table) 175
MBT and SKIIBT 164
MCW 175
MESS 150
MESS and DEFINE CONFG 153
Message
buffer (MESS) 150
buffer area 160
buffer size, change 128
buffering 119-121
unit size 119
Method of operation, SUP 72-73
MIC 13, 33-39, 185
entry to 33-37
exit from 37-39
index registers 34
interrupt action 36
maximum number of serviced interrupts 39
NB card 33
sensing of ILSW/PISW 144
system director 131
Minimum system requirements 1-2
Miscellaneous subroutines 10
Modes of operation (of TSX) 3
Monitor
ccontrol cards 77
ccontrol record analyzer 72
Nonprocess (NPM) 71
Operations 42, 59, 134
Process 4
Monitoring, Process 4
Moving files in core load area 99
Multi-drive TSX on-line system 156
Multi-interrupt priority 28-29
Multi-level control 16
Multi-level programming 7
Multi-process control 17
NIL cards and ICLT 175
NIL labels 143
NOBUF 118
NOCYL 118
Non-interrupt devices 127
Nonprocess
Communications Area 72
compile, store 80-82
core load 84, 85, 219-220, 268-277
core load listing 268-277
execute 90-92
Monitor (NPM) 25, 71-114
Monitor control cards 77-79
Monitor debugging 112
Monitor examples 79-114
Monitor initiating operation 109-110
Monitor utility functions 107-114
program debugging 87
programs 80
save area (NPSV) 150
supervisor (SUP) 9, 72-73, 122
supervisor bootstrap 122
system 121-130
system loading 123-124
working storage (NPWS) 98-99
*NONPROCESS PROGRAM control card 81
Non-synchronous periodic scheduling 26
NORSP, System Director Equate card 132, 134
NPM 71
operation initiation 110
SHARE, VIAQ 110
NPSV 150
NPWS 98, 150, 159
NULEV 135
ICLL1/ICL12 cards 146
NUMBE 118
Number of interrupt levels 135
Object core size, change 128
Off-line
cold start 111
execution of user program 187
mode 3
skeleton rebuild 175
SYSGEN 116
OG/SC 194
ONLIN 119
On-line
core load rebuilding 85-86
EAC error codes 61
mode 3
output, sample system 208
process output listing, sample system 280-290
SYSGEN 115
system, multi-drive 156
Operations Monitor 42, 59, 134
time interval 59
Operator Guide Control 200

Index 305
Operator Guide/Supervisory Control OG/SC  194
OPMON  59
Out-of-core interrupt levels  146
Overview of TSX system  1-11

Packing user area  107
PAPIN subroutine  75
Parameters, *DEDIT control card  127
Patch Area (PTCH)  162, 179
Periodic program scheduler  197
Periodic scheduling, non-synchronous  26
Phase, Disk Edit  128
Philosophy, interrupt  27-29
Physical disk and logical number  109
PISW  30
  assignment, combination of  145
  ILSW  143
  indicators  143
  restrictions of use  144-146
  sensing of interrupt  35
PLOTX subroutine  75
PNT  84, 162
ETV  163
Preparing dummy interrupt core loads  111
PRCS  118, 166
Primary (Interrupt) Time Base  44, 133
Printers, 1053/1816  14
Priority
  assignments 29
  interrupt  135
  interrupt levels  28
  multi-interrupt  28-29
  techniques  20-25
PRNTN subroutine  3, 75, 166
Process
  control  191
  core load  86, 106
  interrupt  32, 92, 144, 145
  Interrupt Status Word (PISW)  143
Monitoring  4
  program, definition of  80
  program debugging  87
Save Area (PRSV)  151
Simulator  200-202
  work storage (PRWS)  150
Processing programs  9-11
Program
  breakdown, EAC  60-68
  cold start  9
  compiling  80-82
  control  8-9
  data sheets  210-220
  disk utility (DUP)  10, 11, 74
  dump from core load area  98
  interrupt, included with mainline  30
  name table (PNT)  84
  process and nonprocess  80
  processing  9-11
  scheduler, periodic  197
  scheduling  14-26
  sequence  15
  Sequence Control (PSC)  14, 17-18
  service  9-10
  simulator  75-77
  skeleton interrupt  30
  user written  4
  Programmed interrupts  145
  Programmed timers  46
  Programming, multi-level  7
  reentrant coding  185-187
  techniques  181-290
Protecting addresses  185
PRSV  151
PRWS  150
PSC  14, 17-18, 162
  System Director  131
PTCH  179
 QIFON  23-25
  use of  24
Queue
  core load on indicator (QIFON)  23
  definition of  14
  subroutine  20-23
  table, core load  14
  table, mainline core load  132
  Queueing statements, use of  21
  Queueing techniques  20-25
Read-and-transfer function  206
Real-time
  clock  14, 45-46, 71
  clock setup (SETCL)  46
  subroutines  10
  systems  191
Rebuilding FLET  128
Rebuilding process core loads  87
*RCORD and LOCALs  103
*RCORD card and interrupts  42
Recording Interrupts  41-42
Recovery procedures, EAC  62-67
Recursive entry to programs  14
Reentrant
coding  6, 185-187
  control  185
  program, MIC  13
  subroutine work level requirements  140-142
Reload, EAC  71
Relocatable
  Area  81-82, 98-99
  program area  159
  program, replace  85
  programs  82-88
  subroutine area  107
Relocation, TSX components  175
Repacking relocatable subroutine area  107
Replacing relocatable programs, coreloads, and data files 82-88
Reproducing cards 107
Reserving core load file area 97-98
RESPK 39
Response time 165
mainline interrupt 7
Restrictions
assignment card 128
interrupt assignment 137
LOCAL 101
Simulator 77
Restart, EAC 71
RETURN 33
Return saved mainline (CALL BACK) 19
RPQ devices 30
Rules governing interrupts 42
Sample system, TSX 195
SAVMK 39
Scheduling
periodic, non-synchronous 26
program 14-26
system 17
Secondary Time Base (CBASE) 133, 134
Sector break records for Absolute Programs 128-130
Sector, definition of 148
numbering 148
Segmenting mainlines 22
Selectable time-sharing (SHARE) 48
Selective Dump Word 70, 297
Selective replacement of PNT 86
*SEQCH function 86
Sequence checking 18
Sequence Control, Program (PSC) 14, 17-18
Sequence technique 18-20
Service programs 9-10
Servicing error conditions 4
interrupts 41
subroutines 30-32
SETCL subroutine 45
Setpoint station movement indicators 202
SFIO 175
SHARE and NPM initiation 110
SHARE subroutine 48
//SIM control record 75
SIM subroutine 123
Simulator (SIM) 11, 123
control cards 79
core load debugging 87
debugging 75, 87-96
functional simulate subroutines 76
hardware requirements 75
program 75
restrictions 77
TASK 87
termination subroutines 76
*SIMULCI and SUP 72
*SIMULCI control record 75
SKA 175
SKEL 151
SKEL core map 236-238
Skeleton
Area (SKEL) 151
build function 13
build prerequisites 173
build process 173-180
build supervisor phase (SKA) 175
builder 9, 13, 123, 161, 173-180
COMMON 161
core map interpretation 178
core size 164-171, 172
executive, role of 3-4
FORTRAN I/O table (SFIO) 175
interrupt branch table (SKIBT) 164
interrupt programs 30
I/O 3, 12-13, 161
rebuild, on-line 175
rebuilt core map 178
subroutine map (SK-SUB) 125
subroutine name table 175
subroutines 125, 164
Supervisor 72
table building 175
SKIBT and MBT 164
SKSUB 125, 175
Special Save Area (SPSV) 151
SPECL subroutine 18
SPSV 151
*START SIMULATION control card 87
Statements not used in error subroutines 60
*STORECI and SUP 72
*STOREDATA function 84
*STOREMD operation 85
Subprograms, LOCAL 5
Subroutines
arithmetic 10, 76
Assembler implicit and explicit 169-170
assignment, interrupt 7
conversion 10, 76
DP and process I/O 10
FORTRAN implicit and explicit 167-168
FORTRAN I/O 10
function 10
general TSX 76
in skeleton 164
I/O device 30
library 9, 10, 123
miscellaneous 10
servicing 30-32
simulator 75-76
user-error 59-60
user-written 76, 162
SUP 72, 122
functions 72
sectors on system cartridge 72
Supervisor, Nonprocess 9, 72-73
skeleton 72
SYSGEN 115-117
listing, sample system 221-279
overview 116
steps, sample system 206-208

Index 307
System
 area allocation 153
 components 7-11
 concepts of TSX 3-7
 configuration versatility 5
 design 196
 design considerations 115-143
 documentation 203
 flowchart, sample system 203-204
 requirements, minimum 1
 System director 4, 8, 13-71, 123, 130-143, 162
 equate cards 131
 functions of 133-135
 size of 131-133
 work areas 131
 System loader 9, 122, 123-125, 221-223
 listing, sample system 221-223
 operation 123-125
 System skeleton 12, 160, 161-180
 area 160
 constitution of 161-164
 layout at build time 174
 modify on-line 175
 rebuilding of 175
 Tables, LET/FLET 122
 Table of variables, sample system 205
 TASK (Temporary Assembled Skeleton) 8, 12-13, 115, 123
 absolute loader 187
 absolute program, example 189-190
 conversion routines 13
 core size calculation 120-121
 director 13
 disk write address (TDWA) 157
 EBPRT 13
 equate cards 117-119
 Error Alert Control (TEAC) 13
 functions 8
 HOLEB 13
 initialization 175
 master interrupt control (TMIC) 13
 program set 13
 simulator 87
 Skeleton Builder 161
 System Skeleton 12
 utilities 13
 TBASE 49
 TDWA
 DWRAD, DLABL comparison 159
 use of 149, 157
 TEAC 13
 Techniques
 sequence or chaining 18-20
 queuing and priority 20-25
 TEMP 82
 Temporary area 80-81
 Termination-class (simulator) subroutines 76
 Time Bases
 interrupt 45
 primary 45, 133
 secondary 134
 timers, hardware 43-45
 Time dependence 15
 Timer 45, 71
 A, B, and C 14, 43-45, 71
 comparison of 48
 core location of 43
 interval control program 14
 programmed 46, 71
 time bases 43
 use of 42-47, 71
 Time-shared NPM operation 114
 Time-sharing 4, 134
 Automatic Method 49
 COLDS, sample system 198
 control (TSC) 14
 initiating 48-50
 optimum 200
 selectable 48
 use of 47-50
 TMIC 13
 Transducers 191
 Translators, language 11
 TRPR subroutine 10
 TSC 14, 25
 system director 131
 TSX
 CALL statements 292-293
 component relocation 175
 core size tables 175
 sample system 195
 subroutine library 9-10
 system composition and capabilities 291
 TYPEN subroutine 3, 75
 /WRTYN 166
 UNMK 39
 UNQ 23
 Update error counters 60
 User
 area 85, 98-99, 107
 disk area 149
 error subroutine 59-60
 programs under TASK absolute loader 187-190
 subroutine listing 230-233, 247-250
 subroutines 162, 211-213, 230-233, 247-250
 writing assembler language subroutines 181
 written programs 4
 Utilities, stand-alone 123
 Utility functions, nonprocess monitor (general) 107-114
 Utility Program, Disk (DUP) 74
 UT7A Relocatable Dump program 123
 Variable core 8
 Variable Transfer Vector (VTV) 101
 Versatility, System 5
VIAQ  25, 49
//END OF ALL JOBS card  49
NPM initiation  110
use of  25
Voltage sense  192
VTV  101, 163

Work areas, level  14, 137

Work level
reentrant subroutines  140-143
requirements  140-143
sections  138
Writing User Assembler subroutines  181-184
WRYTN subroutine  75

//XEQ card and SUP  72
IBM 1800 Time-Sharing Executive System
Concepts and Techniques
©IBM Corp. 1967,1970

This Technical Newsletter, a part of Version 3, Modification 9 of the IBM 1800 Time-Sharing Executive System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are:

25, 26
63, 64
97, 98
117, 118
143–146
199, 200
293, 294

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments
This TNL incorporates technical changes required by the release of Version 3, Modification 9 of the TSX system.

Note. Please file this cover letter at the back of the manual to provide a record of changes.
READER'S COMMENT FORM

IBM 1800 Time-Sharing Executive System
Concepts and Techniques

How did you use this publication?

As a reference source □
As a classroom text □
As a self-study text □

Based on your own experience, rate this publication . . .

As a reference source:

- Very
- Good
- Fair
- Poor
- Very
- Poor

As a text:

- Very
- Good
- Fair
- Poor
- Very
- Poor

What is your occupation? .................................................................

We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
YOUR COMMENTS, PLEASE . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications Dept. D78