IBM Series/1
Model 3 4953 Processor
and Processor Features
Description
IBM Series/1
Model 3 4953 Processor
and Processor Features
Description
Second Edition (March 1977)

This is a major revision of, and obsoletes GA34-0022-0. Significant changes in this new edition include (1) rearrangement of chapters to provide a more logical flow of information and (2) removal of 4 chapters that are now included in other publications.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, have your IBM representative confirm editions that are applicable and current.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, send your comments to IBM Corporation, Systems Publications, Department 27T, P. O. Box 1328, Boca Raton, Florida 33432. Comments become the property of IBM.

©Copyright International Business Machines Corporation 1976, 1977
Contents

Preface vii
Summary of Publication vii
Related Publications vii

Chapter 1. Introduction 1-1
IBM 4953 Processor 1-3
Processor Options 1-3
Processor Description 1-3
Input/Output Units and Features 1-6
Communications Features 1-6
Sensor Input/Output Options 1-6
Packaging and Power Options 1-6
Other Options 1-6

Chapter 2. Processing Unit Description 2-1
Main Storage 2-1
Addressing Main Storage 2-3
Arithmetic and Logic Unit (ALU) 2-3
Numbering Representation 2-3
Registers 2-6
Per-system Registers 2-6
Per-level Registers 2-6
Indicator Bits 2-7
Even, Negative and Zero Result Indicators 2-7
Even, Carry, and Overflow Indicators – Condition Code for
Operations 2-8
Carry and Overflow Indicators – Add and Subtract
Operations 2-8
Carry and Overflow Indicators – Shift Operations 2-8
Indicators – Compare Operations 2-8
Indicators – Multiple Word Operands 2-9
Testing Indicators with Conditional Branch and Jump
Instructions 2-10
Supervisor State Bit 2-11
In-process Bit 2-11
Trace Bit 2-11
Summary Mask Bit 2-11
Program Execution 2-11
Instruction Formats 2-11
Effective Address Generation 2-13
Processor State Control 2-20
Initial Program Load (IPL) 2-22
Sequential Instruction Execution 2-23
Jumping and Branching 2-23
Level Switching and Interrupts 2-23
Stack Operations 2-23

Chapter 3. Interrupts and Level Switching 3-1
Introduction 3-1
Interrupt Scheme 3-1
Automatic Interrupt Branching 3-2
I/O Interrupts 3-3
Prepare I/O Device for Interrupt 3-3
Present and Accept I/O Interrupt 3-3
Class Interrupts 3-5
Priority of Class Interrupts 3-5
Present and Accept Class Interrupt 3-5
Summary of Class Interrupts 3-8

Recovery from Error Conditions 3-8
Program Check 3-8
Storage Parity Check 3-8
CPU Control Check 3-8
I/O Check 3-9
Soft Exception Trap 3-9
Processor Status Word 3-9
Program Controlled Level Switching 3-10
Selected Level Lower Than Current Level and In-process
Flag On 3-11
Selected Level Equal to Current Level and In-process
Flag On 3-11
Selected Level Higher Than Current Level and In-process
Flag On 3-11
Selected Level Lower Than Current Level and In-process
Flag Off 3-12
Selected Level Equal to Current Level and In-process
Flag Off 3-12
Selected Level Higher Than Current Level and In-process
Flag Off 3-12
Interrupt Masking Facilities 3-13
Summary Mask 3-13
Interrupt Level Mask Register 3-13
Device Mask (l-bit) 3-13

Chapter 4. Input/Output Operations 4-1
Operate I/O Instruction 4-1
Immediate Device Control Block (IDCB) 4-3
Device Control Block (DCB) 4-5
I/O Commands 4-6
DPC Operation 4-9
Cycle Steal 4-10
Start Operation 4-10
Start Cycle Steal Status Operation 4-12
Cycle-steal Device Options 4-13
Burst Mode 4-13
Chaining 4-13
Programmed Controlled Interrupt (PCI) 4-13
Suppress Exception (SE) 4-13
Cycle-steal Termination Conditions 4-15
I/O Condition Codes and Status Information 4-15
IO Instruction Condition Codes 4-19
Interrupt Condition Codes 4-19
I/O Status Information 4-20

Chapter 5. Console 5-1
Basic Console 5-2
Keys and Switches 5-2
Indicators 5-2
Programmer Console 5-3
Console Display 5-3
Indicators 5-4
Combination Keys/Indicators 5-5
Keys and Switches 5-8
Displaying Main Storage Locations 5-12
Storing Into Main Storage 5-12
Displaying Registers 5-13
Storing Into Registers 5-13
Chapter 6. Instructions 6-1

Exception Conditions 6-1
  Program Check Condition 6-1
  Soft Exception Trap Conditions 6-1
Instruction Termination or Suppression 6-2
Compatibility 6-2
  Soft Exception Trap 6-2
  No Operation 6-2
  Program Check 6-2

Instruction Descriptions 6-3

Add Byte (AB) 6-3
Add Byte Immediate (ABI) 6-3
Add Carry Register (ACY) 6-4
Add Double Word (AD) 6-5
  Register/Storage Format 6-5
  Storage/Storage Format 6-5
Add Word (AW) 6-6
  Register/Register Format 6-6
  Register/Storage Format 6-6
  Storage to Register Long Format 6-7
  Storage/Storage Format 6-7
Add Word With Carry (AWCY) 6-8
Add Word Immediate (AWI) 6-9
  Register Immediate Long Format 6-9
  Storage Immediate Format 6-9
Branch Unconditional (B) 6-10
Branch and Link (BAL) 6-10
Branch and Link Short (BALS) 6-11
Branch On Condition (BC) 6-12
Branch On Condition Code (BCC) 6-13
Branch On Not Condition (BNC) 6-14
Branch On Not Condition Code (BNCC) 6-15
Branch On Not Overflow (BNOV) 6-16
Branch On Overflow (BOV) 6-16
Branch Indexed Short (BXS) 6-17
Compare Byte (CB) 6-18
  Register/Storage Format 6-18
  Storage/Storage Format 6-18
Compare Byte Immediate (CBI) 6-19
Compare Double Word (CD) 6-20
  Register/Storage Format 6-20
  Storage/Storage Format 6-20
Compare Byte Field Equal and Decrement (CFED) 6-21
Compare Byte Field Equal and Increment (CFEN) 6-21
Compare Byte Field Not Equal and Decrement (CFNED) 6-22
Compare Byte Field Not Equal and Increment (CFNEN) 6-22
Complement Register (CMR) 6-23
Copy Current Level (CPCL) 6-23
Copy Console Data Buffer (CPCON) 6-24
Copy Interrupt Mask Register (CPIMR) 6-24
Copy In-process Flags (CPIPF) 6-25
Copy Level Block (CPLB) 6-26
Copy Level Status Register (CPLSR) 6-27
Copy Processor Status and Reset (CPPSR) 6-27
Compare Word (CW) 6-28
  Register/Register Format 6-28
  Register/Storage Format 6-28
  Storage/Storage Format 6-28
Compare Word Immediate (CWI) 6-29
  Register Immediate Long Format 6-29
  Storage Immediate Format 6-29
Divide Byte (DB) 6-30
Divide Doubleword (DD) 6-31
Diagnose (DIAG) 6-32
Disable (DIS) 6-33
Divide Word (DW) 6-34
Enable (EN) 6-35
Fill Byte Field and Decrement (FFD) 6-36
Fill Byte Field and Increment (FFN) 6-36
Operate I/O (IO) 6-37
Interchange Registers (IR) 6-37
Jump Unconditional (J) 6-38
Jump and Link (JAL) 6-38
Jump On Condition (JC) 6-39
Jump On Count (JCT) 6-40
Jump On Not Condition (JNC) 6-41
Level Exit (LEX) 6-41
Load Multiple and Branch (LMB) 6-42
Multiply Byte (MB) 6-43
Multiply Doubleword (MD) 6-44
Move Address (MVA) 6-45
  Storage Address to Register Format 6-45
  Storage Immediate Format 6-45
Move Byte (MBV) 6-46
  Register/Storage Format 6-46
  Storage/Storage Format 6-46
Move Byte Immediate (MVI) 6-47
Move Byte and Zero (MVBZ) 6-47
Move Doubleword (MVD) 6-48
  Register/Storage Format 6-48
  Storage/Storage Format 6-48
Move Doubleword and Zero (MVDZ) 6-49
Move Byte Field and Decrement (MVFD) 6-50
Move Byte Field and Increment (MVFI) 6-50
Move Word (MWV) 6-51
  Register/Register Format 6-51
  Register/Storage Format 6-51
  Register to Storage Long Format 6-51
  Storage to Register Long Format 6-52
  Storage/Storage Format 6-52
Move Word Immediate (MWI) 6-53
  Storage to Register Format 6-53
  Storage Immediate Format 6-53
Move Word Short (MVWS) 6-54
  Register to Storage Format 6-54
  Storage to Register Format 6-54
  Move Word and Zero (MVWZ) 6-55
  Multiply Word (MW) 6-56
  No Operation (NOP) 6-57
And Word Immediate (NWI) 6-57
  And Word Immediate Format 6-57
OR Byte (OB) 6-58
  Register/Storage Format 6-58
  Storage/Storage Format 6-58
OR Doubleword (OD) 6-59
  Register/Storage Format 6-59
  Storage/Storage Format 6-59
OR Word (OW) 6-60
  Register/Register Format 6-60
  Register/Storage Format 6-60
  Storage to Register Long Format 6-61
  Storage/Storage Format 6-61
OR Word Immediate (OWI) 6-62
  Register Immediate Format 6-62
  Storage Immediate Format 6-62
Pop Byte (PB) 6-63
Pop Doubleword (PD) 6-63
Push Byte (PSB) 6-64
Push Doubleword (PSD) 6-64
Push Word (PSW) 6-65
Resert Bits Byte (RBTTB) 6-66
  Register/Storage Format 6-66
  Storage/Storage Format 6-66
Reset Bits Double Word (RBTD) 6-67
  Register/Storage Format 6-67
  Storage/Storage Format 6-67
Reset Bits Word (RBTW) 6-68
  Register/Register Format 6-68
  Register/Storage Format 6-68
  Storage to Register Long Format 6-69
  Storage/Storage Format 6-69
Reset Bits Word Immediate (RBTWI) 6-70
  Register Immediate Long Format 6-70
  Storage Immediate Format 6-70
Subtract Byte (SB) 6-71
Subtract Carry Indicator (SCY) 6-71
Subtract Doubleword (SD) 6-72
  Register/Storage Format 6-72
  Storage/Storage Format 6-72
Set Console Data Lights (SECON) 6-73
Set Interrupt Mask Register (SEIMR) 6-73
Set Indicators (SEIND) 6-74
Set Level Block (SELB) 6-75
Scan Byte Field Equal and Decrement (SFED) 6-76
Scan Byte Field Equal and Increment (SFEN) 6-76
Scan Byte Field Not Equal and Decrement (SFNED) 6-77
Scan Byte Field Not Equal and Increment (SFNEN) 6-77
Shift Left Circular (SLC) 6-78
  Immediate Count Format 6-78
  Count in Register Format 6-78
Shift Left Circular Double (SLCD) 6-79
  Immediate Count Format 6-79
  Count in Register Format 6-80
Shift Left Logical (SLL) 6-81
  Immediate Count Format 6-81
  Count in Register Format 6-81
Shift Left Logical Double (SLLD) 6-82
  Immediate Count Format 6-82
  Count in Register Format 6-82
Shift Left and Test (SLT) 6-83
Shift Left and Test Double (SLTD) 6-83
Shift Right Arithmetic (SRA) 6-84
  Immediate Count Format 6-84
  Count in Register Format 6-84
Shift Right Arithmetic Double (SRAD) 6-85
  Immediate Count Format 6-85
  Count in Register Format 6-85
Shift Right Logical (SRL) 6-86
  Immediate Count Format 6-86
  Count in Register Format 6-86
Shift Right Logical Double (SRLD) 6-87
  Immediate Count Format 6-87
  Count in Register Format 6-87
Store Multiple (STM) 6-88
Stop (STOP) 6-89
Supervisor Call (SVC) 6-89
Subtract Word (SW) 6-90
  Register/Register Format 6-90
  Register/Storage Format 6-90
  Storage to Register Long Format 6-91
  Storage/Storage Format 6-91
Subtract Word With Carry (SWCY) 6-92
Subtract Word Immediate (SWI) 6-93
  Register Immediate Long Format 6-93
  Storage Immediate Format 6-93
Test Bit (TBT) 6-94
Test Bit and Reset (TBTR) 6-94
Test Bit and Set (TBTS) 6-95
Test Bit and Invert (TBTI) 6-95
Test Word Immediate (TWI) 6-96
  Register Immediate Long Format 6-96
  Storage Immediate Format 6-96
Invert Register (VR) 6-97
Exclusive OR Byte (XB) 6-97
Exclusive OR Doubleword (XD) 6-98
Exclusive OR Word (XW) 6-99
  Register/Register Format 6-99
  Register/Storage Format 6-99
  Storage to Register Long Format 6-100
Exclusive OR Word Immediate (XIW) 6-100
Appendix A. Instruction Execution Times A-1
Appendix B. Instruction Formats B-1
Appendix C. Assembler Syntax C-1
  Coding Notes C-1
    Legend for Machine Instruction Operands C-1
Appendix D. Numbering Systems and Conversion Tables D-1
  Binary and Hexadecimal Number Notations D-1
    Binary Number Notation D-1
    Hexadecimal Number System D-1
  Hexadecimal – Decimal Conversion Tables D-2
Appendix E. Character Codes E-1
Appendix F. Carry and Overflow Indicators F-1
  Signed Numbers F-1
  Unsigned Numbers F-2
  Carry Indicator Setting F-4
    Add Operation Examples F-4
    Subtract Operation Examples F-4
  Overflow Indicator Setting F-5
    Examples F-6
Appendix G. Reference Information G-1
  Condition Codes G-1
    I/O Instruction Condition Codes G-1
    Interrupt Condition Codes G-1
  General Registers G-2
  Interrupt Status Byte (ISB) G-2
    DPC Devices G-2
    Cycle Steal Devices G-2
  Level Status Register (LSR) G-2
  Process Status Word (PSW) G-2
Index X-1
Index of Instructions by Format X-10
Index of Instructions by Name X-13
This publication describes the functional characteristics of the IBM 4953 Processor and the features associated with this processor. It assumes that the reader understands data processing terminology and is familiar with binary and hexadecimal numbering systems. The publication is intended primarily as a reference manual for experienced programmers who require machine code information to plan, correct, and modify programs written in the assembler language.

Summary of Publication
- **Chapter 1. Introduction** is an introduction to the system architecture. It contains a general description of the processor, storage, features, and a list of attachable I/O devices.
- **Chapter 2. Processing Unit Description** contains a description of the processor hardware including registers and indicators. The section on indicators includes examples of indicator results when dealing with signed and unsigned numbers.
  - Main storage data formats and addressing are presented in this chapter.
  - A section titled “Program Execution” is included and covers:
    - Basic instruction formats
    - Effective address generation
    - Processor state control
    - Initial program load (IPL)
    - Jumping and branching
    - Level switching and interrupts
    - Stack operations
- **Chapter 3. Interrupts and Level Switching** describes the priority interrupt levels and the interrupt processing for (1) I/O devices, and (2) class interrupts. Related topics are:
  - Program controlled level switching
  - Interrupt masking facilities
  - Recovery from error conditions
- **Chapter 4. Input/Output Operations** describes the I/O commands and control words that are used to operate the I/O devices. Condition codes and status information relative to the I/O operation are also explained. Specific command and status-word bit structures are contained in the I/O device description books.
- **Chapter 5. Console** describes the keys, switches, and indicators for the basic console and the optional programmer console. Typical manual operations such as storing into and displaying main storage are presented.
- **Chapter 6. Instructions** describes the basic instruction set, including indicator settings and possible exception conditions. Individual instruction word formats are included and contain bit combinations for the operation code and function fields. The instructions are arranged in alphabetical sequence based on assembler mnemonics.
- **Appendices:**
  - Instruction execution times
  - Instruction formats
  - Assembler instruction syntax
  - Numbering systems and conversion tables
  - Character codes
  - Carry and overflow indicators
  - Reference information

Related Publications
- **IBM Series/1 System Summary**, GA34-0035.
- **IBM Series/1 4962 Disk Storage and 4964 Diskette Unit Description**, GA34-0024.
- **IBM Series/1 4973 Line Printer Description**, GA34-0044.
- **IBM Series/1 4974 Printer Description**, GA34-0025.
- **IBM Series/1 4979 Display Station Description**, GA34-0026.
- **IBM Series/1 4982 Sensor Input/Output Unit Description**, GA34-0027.
- **IBM Series/1 Communications Features Description**, GA34-0028.
- **IBM Series/1 Attachment Features Description**, GA34-0031.
- **IBM Series/1 Battery Backup Unit Description**, GA34-0032.
- **IBM Series/1 User’s Attachment Manual**, GA34-0033.
Four models of the 4953 Processor are available.

- **Model A**
  - 1/2 rack width unit
  - 16K bytes main storage
  - Additional storage in 16K byte increments
  - 64K bytes maximum

- **Model B**
  - Full rack width unit
  - 16K bytes main storage
  - Additional storage in 16K byte increments
  - 64K bytes maximum

- **Model C**
  - 1/2 rack width unit
  - 32K bytes main storage
  - Additional storage in 16/32K byte increments
  - 64K byte maximum

- **Model D**
  - Full rack width unit
  - 32K bytes main storage
  - Additional storage in 16/32K byte increments
  - 64K bytes maximum

The IBM 4953 Processor is a compact, general purpose computer and has the following general characteristics:

- Four priority interrupt levels — independent registers and status indicators for each level. Automatic and program controlled level switching.

- Main storage — read or write time is 600 nanoseconds maximum (minimum 800 nanoseconds required between two storage access cycles). Odd parity by byte is maintained throughout storage.

- TTL (transistor-transistor logic) processor technology

- Microprogram control — microcycle time: 200 nanoseconds.

- Instruction set that includes: stacking and linking facilities, multiply and divide, variable field-length byte operations, and a variety of arithmetic and branching instructions.

- Supervisor and problem states.

- Packaged in a 19-inch rack mountable unit — full width or half width.

- Basic console standard in processor unit. Programmer console optional.

- Channel capability
  - Asynchronous, multidropped channel
  - 256 I/O (input/output) devices can be addressed
  - Direct program control and cycle steal operations
  - Maximum burst data rate is 666K words per second (1.332 megabytes if transmitted in pairs). When multiple cycle stealing devices are interleaved, the aggregate data rate is also 666K words.

The processor unit contains power and space for additional features and storage. The IBM 4959 Input/Output Expansion Unit is also available for additional features. The processor is described in the following sections of this chapter.
Figure 1-1. Block diagram of IBM 4953 Processor and an IBM 4959 I/O Expansion Unit
IBM 4953 Processor

Processor Options

- Storage Addition – 16,384 bytes
  - Provides additional storage in 16K byte increments for all models
  - 64K bytes maximum
- Storage Addition – 32,768 bytes
  - Provides additional storage in 32K byte increments for models C and D
  - 64K bytes maximum
- Programmer Console

Processor Description

The basic IBM 4953 Processor includes the processor, 16K bytes of storage for models A and B (32K bytes of storage for models C and D), and a basic console. These items are packaged in a unit, called the processor unit. Figure 1-1 shows a block diagram of an IBM 4953 Processor and an IBM 4959 Input/Output Expansion Unit.

The processor is microprogram controlled, utilizing a 200 nanosecond microcycle. Circuit technology is TTL.

Four priority interrupt levels are implemented in the processor. Each level has an independent set of machine registers. Level switching can occur in two ways: (1) by program control, or (2) automatically upon acceptance of an I/O interrupt request. The interrupt mechanism provides 256 unique entry points for I/O devices.

The processor instruction set contains a variety of instruction types. These include: shift, register to register, register immediate, register to (or from) storage, bit manipulation, multiple register to storage, variable byte field, and storage to storage. Supervisor and problem states are implemented, with appropriate privileged instructions for the supervisor.

The basic console is intended for dedicated systems that are used in a basically unattended environment. Only minimal controls are provided. A programmer console can be added as a feature; this console provides a variety of indicators and controls for operator-oriented systems.

Basic storage supplied is 16K bytes for models A and B; 32K bytes for models C and D. Models A and B can add additional storage in 16K byte increments up to 64K bytes maximum. Models C and D can add additional storage in 16K and/or 32K byte increments up to 64K bytes maximum. The maximum read/write access time for main storage is 600 nanoseconds. However, the minimum duration of time between successive storage cycles is 800 nanoseconds.

I/O devices are attached to the processor through the processor I/O channel. The channel directs the flow of information between the I/O devices, the processor, and main storage. The channel accommodates a maximum of 256 addressable devices.

The channel supports:

- **Direct program control operations.** Each Operate I/O instruction transfers a byte or word of data between main storage and the device. The operation may or may not terminate in an interrupt.
- **Cycle steal operations.** Each Operate I/O instruction initiates multiple data transfers between main storage and the device (65,535 bytes maximum). Cycle steal operations are overlapped with processing operations and always terminate in an interrupt.
- **Interrupt servicing.** Interrupt requests from the devices, along with cycle steal requests, are presented and polled on the interface concurrently with data transfers.

The processor is packaged in a standard 48.3 cm (19 inch) rack-mountable unit, called the processor unit. All processor units contain an integral power supply, fans, and the basic console.

Refer to the *Series/1 Installation Manual—Physical Planning*, GA34-0029, for environmental characteristics.

Four processor models are available. Figure 1-2 shows the IBM 4953 Processor Model A, Figure 1-3 shows the IBM 4953 Processor Model B, Figure 1-4 shows the IBM 4953 Processor Model C, and Figure 1-5 shows the IBM 4953 Processor Model D.

Introduction 1-3
**IBM 4953 Processor Model A**

This model occupies one-half the width of the standard rack and has 16K bytes of storage. It has the capacity for storage cards and/or I/O feature cards in any combination up to 4 additional cards. See Figure 1-2.

*Note.* Additional storage may be added in 16K byte increments up to 64K bytes maximum.

---

**IBM 4953 Processor Model B**

This model occupies the full width of the standard rack and has 16K bytes of storage. It has the capacity for storage cards and/or I/O feature cards in any combination up to 13 additional cards. See Figure 1-3.

*Note.* Additional storage may be added in 16K byte increments up to 64K bytes maximum.

---

If the A position is not used for the Channel Repower card, the following feature cards may be plugged in this position:

- Teletypewriter Adapter Feature using TTL voltage levels
- Teletypewriter Adapter Feature using isolated current loop where user supplies external ±12V power
- Timer Feature
- Customer Direct Program Control Adapter Feature
- 4982 Sensor Input/Output Unit Attachment Feature
- Integrated Digital Input/Output Non-Isolated Feature

---

**Figure 1-2.** IBM 4953 Processor Model A with a Programmer Console

**Figure 1-3.** IBM 4953 Processor Model B with a Programmer Console
IBM 4953 Processor Model C

This model occupies one-half the width of the standard rack and has 32K bytes of storage. It has the capacity for storage cards and/or I/O feature cards in any combination up to 4 additional cards. See Figure 1-4.

*Note.* Additional storage may be added in 16K/32K byte increments up to 64K bytes maximum.

---

IBM 4953 Processor Model D

This model occupies the full width of the standard rack and has 32K bytes of storage. It has the capacity for storage cards and/or I/O feature cards in any combination up to 13 additional cards. See Figure 1-5.

*Note.* Additional storage may be added in 16K/32K byte increments up to 64K bytes maximum.
**Input/Output Units and Features**

- IBM 4962 Disk Storage Unit (4 models)
  - Requires 4962 Disk Storage Unit Attachment Features
- IBM 4964 Diskette Unit
  - Requires 4964 Diskette Unit Attachment Feature
- IBM 4979 Display Station
  - Requires 4979 Display Station Attachment Feature
- IBM 4973 Line Printer (2 models)
  - Requires 4973 Printer Attachment Feature
- IBM 4974 Printer
  - Requires 4974 Printer Attachment Feature
- Timers Feature (2 timers)
- Teletypewriter Adapter Feature
- Customer Direct Program Control Adapter Feature

The feature cards for attaching the I/O units can be housed in either the processor unit or the I/O expansion unit. Information about these units and features can be found in separate publications. The order numbers for these publications are listed in the preface of this manual.

**Communications Features**

- Asynchronous Communications Single Line Control
- Binary Synchronous Communications Single Line Control
- Binary Synchronous Communications Single Line Control/High Speed
- Synchronous Data Link Control Single Line Control
- Asynchronous Communications 8 Line Control
- Asynchronous Communications 4 Line Adapter
- Binary Synchronous Communications 8 Line Control
- Binary Synchronous Communications 4 Line Adapter
- Communications Power Feature
- Communications Indicator Panel

Refer to the publication *IBM Series/1 Communications Features Description, GA34-0028*, for a description of these features.

**Sensor Input/Output Options**

- Integrated Digital Input/Output Non-Isolated Feature
- IBM 4982 Sensor Input/Output Unit
  - 4982 Sensor Input/Output Unit Attachment Feature
- Features for the 4982 Sensor I/O Unit
  - Digital Input/Process Interrupt Non-Isolated
  - Digital Input/Process Interrupt Isolated
  - Digital Output Non-Isolated
  - Analog Input Control
  - Amplifier Multirange
  - Analog Input Multiplexer — Reed Relay
  - Analog Input Multiplexer — Solid State
  - Analog Output

The integrated digital input/output non-isolated feature provides digital sensor I/O and simple attachment for non-IBM equipment. This feature card can be housed in either the processor unit or the I/O expansion unit.

The 4982 sensor input/output attachment unit feature card is housed in either the processor unit or the I/O expansion unit. Refer to the publication *IBM Series/1, 4982 Sensor Input/Output Unit Description, GA34-0027*, for a description of the 4982 and associated features.

**Packaging and Power Options**

- IBM 4959 Input/Output Expansion Unit
- IBM 4999 Battery Backup Unit
- IBM 4997 Rack Enclosure (1-metre) — 2 models
- IBM 4997 Rack Enclosure (1.8-metre) — 2 models

The IBM 4959 Input/Output Expansion Unit is available for adding I/O feature cards beyond the capacity of the processor unit. The capacity of the I/O expansion unit is either (1) fourteen I/O cards, or (2) thirteen I/O cards plus a channel repower card. A channel repower card is required to power each additional I/O expansion unit.

The IBM 4999 Battery Backup Unit permits the processor unit (excluding external devices) to operate from a user-supplied battery when a loss or dip in line power occurs.

**Other Options**

Additional options such as communications cables, customer access panel, and a channel socket adapter are also available. For a list and description of system units and features, refer to the *IBM Series/1 System Summary, GA34-0035*. 
Figure 2-1 shows the general data flow for the IBM 4953 Processor. The major functional units shown in the data flow are discussed in the following sections.

**Main Storage**

Main storage holds data and instructions for applications to be processed on the system. The data and instructions are stored in units of information called a byte. Each byte consists of eight binary data bits. Associated with each byte is a parity bit. Odd parity by byte is maintained throughout storage; even parity causes a machine check error. Formats shown in this manual exclude the parity bit(s) because they are not a part of the data flow manipulated by the instructions.

The bits within a byte are numbered consecutively, left to right, 0 through 7. When a format consists of multiple bytes, the numbering scheme is continued; for example, the bits in the second byte would be numbered 8 through 15. Leftmost bits are sometimes referred to as high-order bits and rightmost bits as low-order bits.

Bytes can be handled separately or grouped together. A *word* is a group of two consecutive bytes, beginning on an even address boundary, and is the basic building block of instructions. A *doubleword* is a group of four consecutive bytes beginning on an even address boundary.

<table>
<thead>
<tr>
<th>Byte</th>
<th>0 0 0 0 0 0 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 7 8 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Doubleword</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 7 8 15 16 23 24 31</td>
</tr>
</tbody>
</table>
Figure 2-1. Data flow for the IBM 4953 Processor
Addressing Main Storage

Each byte location in main storage is directly addressable. Byte locations in storage are numbered consecutively, starting with location zero; each number is considered the address of the corresponding byte. Storage addresses are 16-bit unsigned binary numbers. This permits a direct addressing range of 65,536 bytes:

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 0000 to 1111 1111 1111 1111</td>
<td>0000 to FFFF</td>
<td>0 to 65,535</td>
</tr>
</tbody>
</table>

Note. Addresses that overflow or underflow the addressing range address wrap modulo 65,536.

Instruction and Operand Address Boundaries

As previously stated, all storage addressing is defined by byte location. Instructions can refer to bits, bytes, byte strings, words, or doublewords as data operands. All word and doubleword operand addresses must be on even byte boundaries. All word and doubleword operand addresses point to the most significant (leftmost) byte in the operand. Bit addresses are specified by a byte address and a bit displacement from the most significant bit of the byte.

To provide maximum addressing range, some instructions refer to a word displacement that is added to the contents of a register. In these cases, the operand is a word and the register must contain an even byte address for valid results. Effective address generation is described in a subsequent section of this chapter.

All instructions must be on an even byte boundary. This implies that the effective address for all branch type instructions must be on an even byte boundary to be valid.

If any of the aforementioned rules are violated, a program check interrupt occurs with specification check set in the processor status word (PSW). The instruction is terminated.

Arithmetic and Logic Unit (ALU)

The arithmetic and logic unit (ALU) contains the hardware circuits that perform: addition; subtraction; and logical operations such as AND, OR, and exclusive OR. The ALU performs address arithmetic as well as the operations required to process the instruction operands. Operands may be regarded as signed or unsigned by the programmer. However, the ALU does not distinguish between them. Numbering representation is discussed in a subsequent section of this chapter. For many instructions, indicators are set to reflect the result of the ALU operation. The indicators are discussed in a subsequent section of this chapter.

Numbering Representation

Operands may be signed or unsigned depending on how they are used by the programmer. An unsigned number is a binary integer in which all bits contribute to the magnitude. A storage address is an example of an unsigned number. A signed number is one where the high-order bit is used to indicate the sign, and the remaining bits define the magnitude. Signed positive numbers are represented in true binary notation with the sign bit (high-order bit) set to zero. Signed negative numbers are represented in two's complement notation with the sign bit (high-order bit) set to one. The two's complement of a number is obtained by inverting each bit of the number and adding a one to the low-order bit position. Two's complement notation does not include a negative zero. The maximum positive number consists of an all-one integer field with a sign bit of zero; whereas, the maximum negative number (the negative number with the greatest absolute value) consists of an all-zero integer field with a one-bit for the sign.

The following examples show: (1) an unsigned 16-bit number, (2) a signed 16-bit positive number, and (3) a signed 16-bit negative number.
Example of an unsigned 16-bit number:

```
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Decimal value: 65535 (The largest unsigned number representable in 16 bits.)
Hexadecimal value: FFFF

Example of a signed 16-bit positive number:

```
  0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
        Sign (+)
```

Decimal value: +32767 (The largest positive signed number representable in 16 bits.)
Hexadecimal value: 7FFF

When the number is positive, all bits to the left of the most significant bit of the number, including the sign bit, are zero:

```
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
        Sign (+)
```

Decimal value: +1
Hexadecimal value: 0001

Example of a signed 16-bit negative number:

```
  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
        Sign (-)
```

Decimal value: -32768 (The largest negative signed number representable in 16 bits.)
Hexadecimal value: 8000

Note. This form of representation yields a negative range of one more than the positive range.
When the number is negative, all bits to the left of the most significant bit of the number, including the sign bit, are set to one:

\[
\begin{array}{cccccccccccccccccc}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Binary number</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

\textbf{Sign (-)}

\textbf{Decimal value} \quad -2

\textbf{Hexadecimal value} \quad FFFE

When a signed-number operand must be extended with high-order bits, the expansion is achieved by prefixing a field in which each bit is set equal to the high-order bit of the operand.

Example of an 8-bit field extended to a 16-bit field:

\[
\begin{array}{cccccccccccccc}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Binary number</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

\textbf{Sign (-)}

\textbf{Decimal value} \quad -3

\textbf{Hexadecimal value} \quad FD

\[
\begin{array}{cccccccccccccccccc}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Binary number</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

\textbf{Sign (-)}

\textbf{Decimal value} \quad -3

\textbf{Hexadecimal value} \quad FFFD

It must be emphasized that when performing the add and subtract operations, the machine does not regard the number as either signed or unsigned, but performs the designated operation on the values presented. Whether a given add or subtract operation is to be regarded as a signed operation or an unsigned operation is determined by the programmer's view of the values being presented as operands. The carry indicator and the overflow indicator of the LSR are changed on various operations to reflect the result of that operation. This allows the programmer to make result tests for the number representation involved. The carry and overflow indicator settings are explained in a subsequent section.
Registers
Registers in the processor are provided in two categories:
1. Per-system register (the register is provided only once and is used by all priority interrupt levels)
2. Per-level register (the register is duplicated for each priority interrupt level)

Information that must be saved when a level is preempted is retained in registers supplied for a specific level. Information that pertains only to the current process is kept in registers common to all levels. The registers in each category are listed in this section. Descriptions for each of the registers appear in subsequent sections.

Registers supplied on a per-system basis:
- Console data buffer
- Current-instruction address register (CIAR)
- Mask register (interrupt level)
- Processor status word (PSW)
- Storage address register (SAR)

Registers supplied on a per-level basis:
- General registers (8 per level)
- Instruction address register (IAR)
- Level status register (LSR)

Note. For a specific level, the contents of the IAR, LSR, and the general registers are known as a level status block (LSB). The LSB is a 22-byte entity used by hardware and software for task control and task switching.

Per-system Registers
Console Data Buffer
The console data buffer is a 16-bit register associated with the programmer console feature. Details of how the buffer is used are explained in the programmer console section of Chapter 5. The contents of the console data buffer can be loaded into a specified general register by using the Copy Console Data Buffer (CPCON) instruction (see Chapter 6).

Current-Instruction Address Register (CIAR)
When the processor enters the stop state, the current-instruction address register (CIAR) contains the address of the last instruction that was executed. The CIAR is not addressable by software. It may be displayed from the optional programmer console. Refer to Stop State in this chapter for methods of entering stop state.

Mask Register
The mask register is a 4-bit register used for control of interrupts. Bit 0 controls level 0, bit 1 controls level 1, and so on.

A one bit enables interrupts on a level, while a zero bit disables interrupts. For example if bit 3 is set to a one, interrupts are enabled on level 3.

Processor Status Word (PSW)
The processor status word (PSW) is a 16-bit register used to (1) record error or exception conditions that may prevent further processing, and (2) hold certain flags that aid in error recovery. Error or exception conditions recorded in the PSW result in a class interrupt. Each bit in the PSW is described in detail in Chapter 3. The PSW can be accessed by using the Copy Processor Status and Reset (CPPSR) instruction (see Chapter 6).

Storage Address Register (SAR)
The storage address register (SAR) is a 16-bit register that contains the main-storage address for the last attempted processor storage cycle. This register is addressable by the Diagnose instruction and may be altered or displayed from the optional programmer console.

Per-level Registers
General Registers
Subsequently referred to simply as registers, the general registers are 16-bit registers available to the program for general purposes. Eight registers are provided for each level. The R and RB fields in the instructions control the selection of these registers.

Instruction Address Register (IAR)
The instruction address register (IAR) is a 16-bit register that holds the main storage address used to fetch an instruction. After an instruction has been fetched, the IAR is updated to point to the next instruction to be fetched.

Note. These registers are sometimes referred to as IAR0, IAR1, IAR2, and IAR3. The numbers represent the priority level associated with the register.

Level Status Register (LSR)
The level status register (LSR) is a 16-bit register that holds:
- Indicator bits
  - Set as a result of arithmetic, logical, or I/O operations
- A supervisor state bit
- An in-process bit
- A trace bit
- A summary mask bit.

These bits are further discussed in the following sections. Seven other bits in the LSR are not used and are always set to zero.
Indicator Bits

The indicators are located in bits 0–4 of the level status register (LSR). Figure 2-2 shows the indicators and how they are set for arithmetic operations. The indicator bits are changed or not changed depending on the instruction being executed. Some instructions do not affect the indicators, other instructions change all of the indicators, and still other instructions change only specific indicators. Refer to the individual instruction descriptions in Chapter 6 for the indicators changed by each instruction.

The indicators are changed in a specialized manner for certain operations. These operations are described briefly. Additional information is provided in subsequent sections for those operations where more detail is required.

- **Add, subtract, or logical operations.** The even, negative, and zero indicators are result indicators. For add and subtract operations, the carry and overflow indicators are changed to provide information for both signed and unsigned number representations.
- **Multiply and divide operations.** Signed number operands are always assumed for these operations. The carry indicator is used to provide a divide by zero indication for the divide instruction. The overflow indicator defines an unrepresentable product for multiply operations. Refer to the individual instruction descriptions in Chapter 6.
- **Priority interrupts and input/output operations.** The even, carry and overflow indicators are used to form a three-bit condition code that is set as a binary value.
- **Compare operations.** The indicators are set in the same manner as a subtract operation.
- **Shift operations.** The carry and overflow indicators have a special meaning for shift left logical operations.
- **Complement operations.** The overflow indicator is set if an attempt is made to complement the maximum negative number. This number is not representable.
- **Set Indicators (SEIND) and Set Level Block (SELB) instructions.** All indicators are changed by the data associated with these instructions.

Even, Negative, and Zero Result Indicators

The even, negative, and zero indicators are called the result indicators. A positive result is indicated when the zero and negative indicators are both off (set to zero). These indicators are set to reflect the result of the last arithmetic, or logical operation performed. A logical operation in this sense includes data movement instructions. See the individual instruction descriptions in Chapter 6 for the indicators changed for specific instructions.
**Even, Carry, and Overflow Indicators — Condition Code for Input/Output Operations**

The even, carry, and overflow indicators contain the I/O condition code: (1) following the execution of an Operate I/O instruction and (2) following an I/O interrupt.

These indicators are used to form a 3-bit binary number that results in a condition code value. For additional information about condition codes, refer to:

1. Branch on Condition Code (BCC) and Branch on Not Condition Code (BNCC) instructions in Chapter 6.
2. Condition codes in Chapter 4.

**Carry and Overflow Indicators — Add and Subtract Operations**

A common set of add and subtract integer operations performs both signed and unsigned arithmetic. Whether a given add or subtract operation is to be regarded as a signed operation or an unsigned operation is determined by the programmer’s view of the values being presented as operands.

The carry and overflow indicators are set to reflect the result for both cases.

**Carry Indicator Setting**

The carry indicator is used to signal overflow of the result when operands are presented as *unsigned* numbers.

**Overflow Indicator Setting**

The overflow indicator is used to signal overflow of the result when the operands are presented as *signed* numbers.

*Note.* Appendix F explains the meaning of these indicators for signed and unsigned numbers. The appendix also provides examples for setting the carry indicator and for setting the overflow indicator.

**Carry and Overflow Indicators — Shift Operations**

The carry and overflow indicators are changed for shift left logical operations and shift left and test operations. These operations affect the indicators as follows:

1. The carry indicator is set to reflect the value of the last bit shifted out of the target register (register where bits are being shifted).
2. The overflow indicator is set to one if bit-0 of the target register was changed during the shift. Otherwise it is set to zero.

**Indicators — Compare Operations**

A compare operation sets the indicators in the same manner as a subtract operation. The even, negative, and zero indicators reflect the result. The carry and overflow indicators are set as described previously.

Compare instructions provide a test between two operands (without altering either operand) so that conditional branch and jump instructions may be used to control the programming logic flow. The conditions specified in branch and jump instructions are named such that, when the condition of the “subtracted from” operand relative to the other operand is true the jump or branch occurs. Otherwise, the next sequential instruction is executed. This is illustrated in the following example.

- **Compare operation example**

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Assembler mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare word</td>
<td>CW</td>
<td>R3, R4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>0 1</td>
<td>1 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>4 5 7 8 10 11</td>
<td>R3</td>
<td>R4</td>
<td></td>
</tr>
</tbody>
</table>

  In this example, the contents of register 3 are subtracted from register 4:

  **Decimal**

<table>
<thead>
<tr>
<th>Unsigned</th>
<th>Signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>+2</td>
</tr>
<tr>
<td>65531</td>
<td>-5</td>
</tr>
<tr>
<td>-65529</td>
<td>+7</td>
</tr>
</tbody>
</table>

  **Machine operation:**

  Minuend: 0000 0000 0000 0010
  Subtrahend: 0000 0000 0000 0100
  Constant: 1

  **Result:** 0000 0000 0000 0111

  **Indicator Settings:**

  - E = 0: Result is not zero.
  - C = 1: Result is positive.
  - O = 1: Result fits operand size as a signed number.
  - N = 1: A negative result for an unsigned number.
  - Z = 1: Result is not even (low-order bit = 1).
If the programmer is comparing unsigned numbers, such as storage addresses, he should use the logical conditional tests (refer to Figure 2-3). In this example, assuming unsigned number representation, R4 is logically less than R3 and unequal to R3. Therefore, the following branch instructions would cause a transfer to symbolic location A (assuming register values shown in the example).

\[ \text{CW } R3, R4 \]
\[ \text{BLLT } A \]
\[ \text{or} \]
\[ \text{CW } R3, R4 \]
\[ \text{BNE } A \]

The complementary tests (BLGT and BE) would not cause a transfer in this case.

If the programmer is comparing signed numbers, he should use the arithmetic conditional tests (refer to Figure 2-3). In the previous compare word example, assuming signed number representation, R4 is greater than R3 and unequal to R3. The following branch instructions would cause a transfer to symbolic location A.

\[ \text{CW } R3, R4 \]
\[ \text{BGT } A \]
\[ \text{or} \]
\[ \text{CW } R3, R4 \]
\[ \text{BNE } A \]

The complementary tests (BLT and BE) would not cause a transfer.

Note. Jump instructions are also available for the logical and arithmetic conditional tests.

It must be emphasized again that the machine does not regard the numbers as either signed or unsigned. The compare word instruction results in a subtract operation being performed on the values presented. The programmer must then choose the correct conditional test (logical or arithmetic) for the number representation involved.

**Indicators – Multiple Word Operands**

A programmer may desire to work with numbers that cannot be represented in one word or in a doubleword. It may take three or more words to represent the number.

Certain register to register instructions allow the programmer to add or subtract these multi-word operands and then have the indicators reflect the multi-word result. These instructions are:

- Add Carry Register (ACY)
- Add Word With Carry (AWCY)
- Subtract Carry Register (SCY)
- Subtract Word With Carry (SWCY)

The following two examples show how the add instructions are used. A subtract operation would be similar. See Chapter 6 for details of the individual instructions.

---

**Example 1. (Equal length operands)**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4</td>
<td>R5</td>
<td>R6</td>
</tr>
</tbody>
</table>

**Operand 1/Result**

**Operand 2**

**Program steps:**

- \( \text{AW } R6, R3 \)
- \( \text{AWCY } R5, R2 \)
- \( \text{AWCY } R4, R1 \)

**Explanation:**

- **Step 1:** The contents of R6 are added to the contents of R3.
- **Step 2:** The contents of R5 are added to the contents of R2 plus any carry from the previous operation.
- **Step 3:** The contents of R4 are added to the contents of R1 plus any carry from the previous operation.

**Example 2. (Unequal length operands)**

<table>
<thead>
<tr>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R5</td>
</tr>
<tr>
<td>R6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operand 1/Result**

**Operand 2**

**Program Steps:**

- \( \text{AW } R6, R3 \)
- \( \text{AWCY } R5, R2 \)
- \( \text{ACY } R1 \)

**Explanation:**

- **Step 1:** The contents of R6 are added to the contents of R3.
- **Step 2:** The contents of R5 are added to the contents of R2 plus any carry from the previous operation.
- **Step 3:** Any carry from the previous operation is added to the contents of R1.

**Note.** In both examples the final indicator settings reflect the status of the 3-word result.

- **Even** Set on if the result low-order bit of R3 is zero.
- **Carry** Set on if the result cannot be represented as an unsigned 3-word number.
- **Overflow** Set on if the result cannot be represented as a signed 3-word number.
- **Negative** Set on if the result high-order bit of R1 is one.
- **Zero** Set on if all three result registers contain zeros.
**Testing Indicators with Conditional Branch and Jump Instructions**

The indicators are tested according to a selected condition when a conditional branch or a conditional jump instruction is executed. The conditions and the indicators tested for each condition are shown in Figure 2-3.

The conditional instructions are:
- Branch on Condition (BC)
- Branch on Not Condition (BNC)
- Jump on Condition (JC)
- Jump on Not Condition (JNC)

The assembler also provides extended mnemonics for the conditions shown in Figure 2-3. Refer to the individual instructions in Chapter 6.

<table>
<thead>
<tr>
<th>Condition tested by conditional branch or jump instruction</th>
<th>Assembler extended mnemonics</th>
<th>Indicators tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero or equal</td>
<td>BE, BZ, JE, JZ</td>
<td>0 1 2 3 4 E C O N Z</td>
</tr>
<tr>
<td>Not zero or unequal</td>
<td>BNE, BNZ, JNE, JNZ</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Positive and not zero</td>
<td>BP, JP</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Not positive</td>
<td>BNP, JNP</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Negative</td>
<td>BN, JN</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Not negative</td>
<td>BNN, JNN</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Even</td>
<td>BEV, JEV</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Not even</td>
<td>BNEV, JNEV</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Arithmetically less than</td>
<td>BLT, JLT</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Arithmetically less than or equal</td>
<td>BLE, JLE</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Arithmetically greater than or equal</td>
<td>BGE, JGE</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Arithmetically greater than</td>
<td>BGT, JGT</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Logically less than or equal</td>
<td>BLLE, JLLE</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Logically less than (carry)</td>
<td>BLLT, JLLT</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Logically greater than</td>
<td>BLGT, JLG</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Logically greater than or equal (no carry)</td>
<td>BLGE, JLG</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

Legend:  
- **LSR bit**  
  - 0 - E - Even  
  - 1 - C - Carry  
  - 2 - O - Overflow  
  - 3 - N - Negative  
  - 4 - Z - Zero  

Figure 2-3. Indicators tested by conditional branch and jump instructions
Supervisor State Bit
LSR bit 8, when set to one, indicates that the processor is in the supervisor state. This state allows privileged instructions to be executed. It is set by any of the following:

1. Class interrupt
   a. Machine check condition
   b. Program check condition
   c. Power/thermal warning
   d. Supervisor Call (SVC) instruction
   e. Soft exception trap condition
   f. Trace
   g. Console interrupt
2. I/O interrupt
3. Initial program load (IPL)

When LSR bit 8 is set to zero, the processor is in problem state. For a selected priority level, the supervisor can alter the supervisor state bit by using a Set Level Block (SELB) instruction. For additional information, refer to Processor State Control in this chapter.

In-process Bit
LSR bit 9, when set to one, indicates that a priority level is currently active or was preempted by a higher priority level before completing its task. Bit 9 is turned off by a Level Exit (LEX) instruction. Bit 9 can also be turned on or off by a Set Level Block (SELB) instruction. The in-process bit also affects level switching under program control. Refer to Chapter 3. Interrupts and Level Switching.

Trace Bit
LSR bit 10, when set to one, causes a trace class interrupt at the beginning of each instruction. The bit can be turned on or off with the Set Level Block (SELB) instruction. The trace bit aids in debugging programs. See Class Interrupts in Chapter 3.

Summary Mask Bit
LSR bit 11, when set to zero (disabled), inhibits all priority interrupts on all levels. When this bit is set to one (enabled), normal interrupt processing is allowed. Refer to Summary Mask in Chapter 3 for details relating to control of the summary mask.

Program Execution

Instruction Formats
The processor instruction formats are designed for efficient use of bit combinations to specify the operation to be performed (operation code) and the operands that participate. Some formats also include (1) an immediate data field or word, (2) an address displacement or address word, and (3) a function field that further modifies the operation code. Various combinations of these fields are used by the individual instructions. Some typical instruction formats are presented in this section. All formats are shown in the section Instruction Formats in Appendix B.

One Word Instructions
The basic instruction length is one word (16 bits). The operation code field (bits 0–4) is the only common field for all formats. This field, unless modified by a function field, specifies the operation to be performed. For a format without a function field, bits 5–15 specify the location of operands or data associated with an operand:

Example:

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Assembler mnemonic</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Byte Immediate</td>
<td>ABI</td>
<td>byte, reg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>R</td>
<td>7 8 15</td>
</tr>
</tbody>
</table>

Bits 0–4 Operation code (specifies ABI instruction).

Bits 5–7 General register (0–7).
This register contains data for the second operand.

Bits 8–15 Immediate data for the first operand.

In some cases the operation code is the same for a group of instructions. The format for this group includes a function field. The bit combinations in the function field then determine the specific operation to be performed.

In-process Bit
LSR bit 9, when set to one, indicates that a priority level is currently active or was preempted by a higher priority level before completing its task. Bit 9 is turned off by a Level Exit (LEX) instruction. Bit 9 can also be turned on or off by a Set Level Block (SELB) instruction. The in-process bit also affects level switching under program control. Refer to Chapter 3. Interrupts and Level Switching.

Trace Bit
LSR bit 10, when set to one, causes a trace class interrupt at the beginning of each instruction. The bit can be turned on or off with the Set Level Block (SELB) instruction. The trace bit aids in debugging programs. See Class Interrupts in Chapter 3.

Summary Mask Bit
LSR bit 11, when set to zero (disabled), inhibits all priority interrupts on all levels. When this bit is set to one (enabled), normal interrupt processing is allowed. Refer to Summary Mask in Chapter 3 for details relating to control of the summary mask.
Example:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Assembler mnemonic</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Word</td>
<td>AW</td>
<td>reg, reg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>011110</td>
<td></td>
<td></td>
<td>01000</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7 8 10 11</td>
</tr>
</tbody>
</table>

Bits 0–4  Operation code for a group of instructions.
Bits 5–7  General register (0–7).
This register contains data for the first operand.
Bits 8–10 General register (0–7).
This register contains data for the second operand.
Bits 11–15 Function field.
Modifies the operation code to specify the Add Word instruction.

Note. For other instruction groups, the function field may vary as to location within the format, and also the number of bits used.

Two Word Instructions
The first word of this format is identical to the one-word format. The second word (bits 16–31) contains either immediate data, an address, or a displacement. This word is used to (1) provide data for an operand, or (2) provide a main storage address or displacement for effective address generation (see Effective Address Generation in this chapter).

Example:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Assembler mnemonic</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch and Link</td>
<td>BAL</td>
<td>longaddr, reg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td></td>
<td></td>
<td>0</td>
<td>011</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7 8 10 11 12</td>
<td>15</td>
</tr>
</tbody>
</table>

Bits 0–4  Operation code
Bits 5–7  General register (0–7) for the second operand
Bits 8–10 General register (0–7) for the first operand
Bit 11  Indirect addressing bit
Bits 12–15 Function field
Bits 16–31 A main storage address used for the first operand

Note. In this example, the register designated R1 is associated with the second operand in assembler syntax.

Variable Length Instructions
Some instructions use a selectable encoded technique for generating effective addresses. This method is referred to as an address argument technique in subsequent sections. These instruction formats contain a base register (RB) field and an address mode (AM) field. If both operands are using this technique, the format contains an RB and associated AM field for each. These fields are in the first instruction word. The AM field consists of two bits and is referred to in binary notation (AM=00, 01, 10, or 11). If AM is equal to 10 or 11 an additional word is appended to the normal instruction word. For a format that contains two AM fields, two additional words may be appended. See Effective Address Generation in this chapter for a description of the appended words and how they are used.

For instructions with a single storage address argument, the RB field consists of two bits. An RB field of two bits with its associated AM field of two bits are referred to as a 4-bit address argument or addr4 in assembler syntax.

Example:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Assembler mnemonic</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare byte</td>
<td>CB</td>
<td>addr4, reg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation byte</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11000</td>
<td>4</td>
<td>5</td>
<td>7 8 9 10 11 12</td>
<td>15</td>
</tr>
</tbody>
</table>

| Appended word, AM=10 or 11 |
| 16 | 31 |

Bits 0–4  Operation Code.
Bits 5–7  General register (0–7) for the second operand.
Bits 8–9  Base register (0–3).
Bits 10–11 Address mode.
Bits 12–15 Function.
Bits 16–31 Appended word for the first operand.

Note. The register specified by the RB field is a general register that is used as a base register for effective address generation.

Some instruction formats have two storage address arguments. In this case, the first operand has a 3-bit RB field giving a 5-bit address argument (addr5 in assembler syntax) and the second operand has a 4-bit address argument.
Example:

<table>
<thead>
<tr>
<th>Instruction name</th>
<th>Assembler mnemonic</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Word</td>
<td>AW</td>
<td>addr5, addr4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

- **Shift Instructions with Immediate Count**  
  This is a shift instruction with the count field contained within the instruction word.
- **Storage Immediate Instructions**  
  One operand is in main storage. The other operand uses an immediate data field. The immediate data field is the second word of a two-word format.
- **Parametric Instructions**  
  For this instruction format, a parameter field (bits 8–15) is contained within the instruction word.

**Effective Address Generation**

For purposes of storage efficiency, certain instructions formulate storage operand addresses in a specialized manner. These instructions have self-contained fields that are used when generating effective addresses. Standard methods for deriving effective addresses are included in this section. Other methods such as bit displacements, are explained in the individual instruction descriptions in Chapter 6.

**Programming Note.** For certain instructions, the effective address points to a control block rather than an operand. These instructions are:

- Copy Level Block (CPLB)
- Load Multiple and Branch (LMB)
- Pop Byte (PB)
- Pop Doubleword (PD)
- Push Byte (PSB)
- Push Doubleword (PSD)
- Push Word (PSW)
- Pop Word (PW)
- Set Level Status Block (SELB)
- Store Multiple (STM)

**Base Register Word Displacement Short**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>WD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Base register:

- 00 Register 0
- 01 Register 1
- 10 Register 2
- 11 Register 3

Word displacement:

Range 0 to 31 (decimal)

The five-bit unsigned integer (WD) is doubled in magnitude to form a byte displacement then added to the contents of the specified base register to form the effective address. The contents of the base register must be even.
Example:

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>WD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  4  8  9  11</td>
<td>1  0  0  1  0  0</td>
<td></td>
</tr>
</tbody>
</table>

Hex Dec

Contents of register 1 (RB) 0000 0000 0110 0000 0060 0096

Word displacement (WD) doubled + 0 1000 8 8

Effective address 0000 0000 0110 0000 0068 0104

Base Register Word Displacement

Instruction format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>WD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  4  5  7  8  15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base register

000 Register 0
001 Register 1
010 Register 2
011 Register 3
100 Register 4
101 Register 5
110 Register 6
111 Register 7

Word displacement
Range +127 to -128 (decimal)

The eight-bit signed integer (WD) is doubled in magnitude to form a byte displacement then added to the contents of the specified base register to form the effective address. The contents of the base register must be even.

The word displacement can be either positive or negative; bit 8 of the instruction word is the sign bit for the displacement value. If this high-order bit of the displacement field is a 0, the displacement is positive with a maximum value of +127 (decimal). If the high-order bit of the displacement field is a 1, the displacement is negative with a maximum value of -128. The negative number is represented in twos-complement form.

Example:

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>WD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  4  5  7  8  15</td>
<td>1  1  0  1  0  1  0  1</td>
<td></td>
</tr>
</tbody>
</table>

Hex Dec

Contents of register 6 (RB) 0000 0000 0110 0000 0086 0134

Word displacement (WD) doubled (sign bit is propagated left) + 1111 1111 1111 0010

Effective address 0000 0000 0101 1000 0058 0088

Note. This example uses a negative word displacement (-17 hex) shown in two's complement.

Four-Bit Address Argument

Instruction format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  4  8  9  10  11  15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base register

00 Register 0
00 (AM=00 or 01)
00 No register
00 (AM=10 or 11)
01 Register 1
10 Register 2
11 Register 3

Address mode

The Address Mode (AM) has the following significance:

AM=00. The contents of the selected base register form the effective address.

AM=01. The contents of the selected base register form the effective address. After use, the base register contents are incremented by the number of bytes in the operand. For some instructions the effective address points to a control block rather than an operand. When the effective address points to a control block, the base register contents are incremented by two.
Example:

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1</td>
<td>0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>0080</td>
<td>0128</td>
</tr>
</tbody>
</table>

Effective address (contents of register 1) 0000 0000 1000 0000 0080 0128

Contents of register 1 after instruction execution

* Byte operand
  * 0000 0000 1000 0001 0081 0129
* Word operand
  * 0000 0000 1000 0010 0082 0130
* Double word operand
  * 0000 0000 1000 0100 0084 0132

Notes.
1. For register to storage instructions, if the register specified is the same for both operands then the register will be incremented prior to using it as an operand.
2. Certain instructions (storage-to-storage) have two address arguments. Operand 1 has a 3-bit RB field with its associated AM field. Operand 2 has a 2-bit RB field with its associated AM field. If both RB fields specify the same register and both AM fields are equal to 01, the base register contents are incremented prior to fetching operand 2 and again after fetching operand 2. Assuming the same conditions but with the operand 2 AM field not equal to 01, the base register contents are incremented prior to calculating the effective address for operand 2.
3. If the effective address points to a control block rather than an operand, the base register contents are incremented by two.

AM=10. An additional word is appended to the instruction. The word has the following format.

<table>
<thead>
<tr>
<th>Address or displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>

- If RB is zero, the appended word contains the effective address.
- If RB is non-zero, the contents of the selected base register and the contents of the appended word (displacement) are added to form the effective address.
Example:

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4</td>
<td>8 9 10 11 12 15 16 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hex Dec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contents of register 3: 0000 1000 0000 0000</td>
<td>0800 2048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contents of appended word: + 0000 0001 0000 0000</td>
<td>0100 0256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effective address: 0000 1001 0000 0000</td>
<td>0900 2304</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AM=11. An additional word is appended to the instruction.

- If RB is zero, the appended word has the format:

  **Indirect address**

  16 31

  This address points to a main storage location, on an even byte boundary, that contains the effective address.

Example:

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Indirect address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 4</td>
<td>8 9 10 11 12 15 16 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hex Dec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contents of appended word: 0000 0000 0101 0000</td>
<td>0050 0080</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effective address equals contents of storage at address 0080 (decimal): 0000 0100 0000 0000</td>
<td>0400 1024</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- If RB is non-zero, the appended word has the format:

<table>
<thead>
<tr>
<th>Displacement 1</th>
<th>Displacement 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 23 24 31</td>
<td></td>
</tr>
</tbody>
</table>

  The two displacements are unsigned eight-bit integers. Displacement 2 is added to the contents of the selected base register to generate a main storage address. The contents of this storage location are added to Displacement 1 resulting in the effective address.
**Example:**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Displacement 1</th>
<th>Displacement 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0101 0111</td>
<td>0100 0010</td>
</tr>
</tbody>
</table>

Contents of register 2: 0000 0101 0011 0101
Displacement 2: + 0000 0101 0111 0111
Storage address: 0000 0101 0111 0111

Contents of storage at address 1399 (decimal): 0000 0100 0001 0000
Displacement 1: + 0010 0101
Effective address: 0000 0100 0011 0101

Note. This example is invalid for other than a byte operand.

**Programming Note.** This addressing mode (AM=11, RB is non-zero) is useful for the directorized data concept. For the addr4 or addr5 assembler syntax, the programmer codes the form displacement 1 (register, displacement 2)*.

For addr4, the specified register is 1–3. For addr5, the specified register is 1–7. The asterisk denotes indirect addressing.
Five-Bit Address Argument

**Instruction format**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</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>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Base register**

- 000: Register 0 (AM=00 or 01)
- 000: No register (AM=10 or 11)
- 001: Register 1
- 010: Register 2
- 011: Register 3
- 100: Register 4
- 101: Register 5
- 110: Register 6
- 111: Register 7

**Address mode**

Operation of this mode is identical to the four-bit argument, but provides additional base registers.

**Base Register Storage Address**

**Instruction format**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>X</th>
<th>Address/displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Base register**

- 000: No register
- 001: Register 1
- 010: Register 2
- 011: Register 3
- 100: Register 4
- 101: Register 5
- 110: Register 6
- 111: Register 7

- If RB is zero, the address field contains the effective address.
- If RB is non-zero, the contents of the selected base register and the contents of the address field are added together to form the effective address.

**Note.** Bit 11, if a one, specifies that the effective addressing is indirect.
**Example:** Indirect address

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0 0 1</td>
<td>0 0 0 0 1 0 0 0 0 1 0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0 0 1</td>
<td>0 0 0 0 1 0 0 0 0 1 0 0 0 0</td>
</tr>
</tbody>
</table>

**Hex**  
**Dec**  
Contents of register 4: 0000 0001 0000 0000 0100 0256  
Address field: +0000 0100 0001 0000 0410 1040  
Storage address: 0000 0101 0001 0000 0510 1296  
Effective address: Contents of storage at address 1296 (decimal) 0000 0110 0100 0000 0640 1600

**Instruction Length Variations for Address Arguments**

- One-word instructions that contain a single AM field become two words in length if AM is equal to 10 or 11. The AM appended word follows the instruction word.

**Example:**

<table>
<thead>
<tr>
<th>AM=00 or 01</th>
<th>Instruction word</th>
<th>No appended word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AM=10 or 11</th>
<th>Instruction word</th>
<th>AM appended word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15 16</td>
<td>31</td>
</tr>
</tbody>
</table>

- Two-word instructions that contain a single AM field become three words in length if AM is equal to 10 or 11. The AM word is appended to the first instruction word. The data or immediate field then becomes the third word of the instruction.

**Example:**

<table>
<thead>
<tr>
<th>AM=00 or 01</th>
<th>Instruction word</th>
<th>Immediate field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15 16</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AM=10 or 11</th>
<th>Instruction word</th>
<th>AM appended word</th>
<th>Immediate field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15 16</td>
<td>31</td>
<td>31 32</td>
</tr>
</tbody>
</table>

- One-word instructions that contain two AM fields (AM1 and AM2) may be one, two, or three words in length depending on the values of AM1 and AM2. The AM1 word is appended first, then the AM2 word is appended.
Example:

<table>
<thead>
<tr>
<th>AM1=00 or 01</th>
<th>Instruction word</th>
<th>No appended word</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM2=00 or 01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AM1=10 or 11</th>
<th>Instruction word</th>
<th>AM1 appended word</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM2=00 or 01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AM1=00 or 01</th>
<th>Instruction word</th>
<th>AM2 appended word</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM2=10 or 11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AM1=10 or 11</th>
<th>Instruction word</th>
<th>AM1 appended word</th>
<th>AM2 appended word</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM2=10 or 11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Processor State Control**

The processor is always in one of the following mutually exclusive states:

- Power off
- Stop
- Load
- Wait
- Run — when in run state, programs can be executed in either:
  - Supervisor state or
  - Problem state

**Stop State**

The stop state is entered when:

1. The Stop key on the programmer console is pressed.
2. The STOP instruction is executed and the mode switch on the basic console is in the Diagnostic position. 
3. An address-compare occurs and the rate control on the programmer console is in the Stop on Address position.
4. An instruction has completed execution and the rate control on the programmer console is in the Instruction Step position.
5. An error occurs and the error control on the programmer console is in the Stop on Error position.
6. The Reset key on the programmer console is pressed.
7. Power-on reset occurs. For conditions 1–6, the display buffer contains the contents of the IAR.

Note. Any manual entry into Stop State is via the programmer console. The STOP instruction performs no operation if the programmer console is not installed.

While the processor is in the stop state: (1) the Stop light on the programmer console is on, (2) the functions provided on the console can be activated, and (3) no interrupt requests can be accepted by the processor.

If, when accessing main storage through the console while in stop state, a check condition arises:

1. Program check is suppressed.
2. The PSW bit(s) is set.
3. The check light is turned on.
4. The Display Register will be set to a default value of 0025.

The processor exits the stop state when:

1. The Load key on the basic console is pressed.
2. The Start key on the programmer console is pressed. When the Start key is pressed, the processor returns to the state that was exited before entering stop state. If the run state is entered, one instruction is executed before interrupts are accepted by the processor. If the stop state was entered because of a reset (power-on reset or reset key), pressing the Start key causes program execution to begin on level zero with the instruction in location zero of main storage. If the stop state was entered because of an error, with the Stop on Error switch turned on, (1) a system reset clears the error condition, or (2) pressing the Start key allows the error interrupt to be handled as if the Stop on Error switch were not on. For more information about system reset, see *State of Processor Following a Reset*.  

2-20 GA34-0022
**Wait State**
The processor enters wait state when: (1) Level Exit (LEX) instruction is executed and no other level is pending, or (2) a Set Level Block (SELB) instruction is executed that sets the current in-process bit off and no level is pending. While the processor is in the wait state, (1) the Wait light on the basic console is on and (2) interrupts can be accepted under control of the system mask register and the summary mask as defined by the LSR of the last active level. The processor exits the wait state when:
1. The Stop key on the programmer console is pressed.
2. The Reset key on the programmer console is pressed.
3. An I/O interrupt is accepted (the level must be enabled by the mask register).
4. A class interrupt occurs. (See Class Interrupts in Chapter 3.)

**Load State**
The processor enters the load state when initial program load (IPL) begins. This occurs:
1. When the Load key on the basic console is pressed.
2. After a power-on reset if the Mode switch is in the Auto IPL position.
3. A signal from a host system.

While the processor is in load state, the Load light on the basic console is on. The processor exits the load state and enters the run state upon successful completion of the IPL. See Initial Program Load (IPL).

**Run State**
The processor enters the run state when not in the stop, wait, or load state. Run state is entered:
1. From load state upon successful completion of IPL.
2. From wait state when an interrupt is accepted.
3. From stop state when the start key is pressed. (See Stop State.)

The processor exits run state when entering stop, wait, or load states as previously described.

**Supervisor State and Problem State**
While in run state, instructions can be executed in either supervisor state or problem state. This is determined by bit 8 of the level status register (LSR):

<table>
<thead>
<tr>
<th>State</th>
<th>LSR bit 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>1</td>
</tr>
<tr>
<td>Problem</td>
<td>0</td>
</tr>
</tbody>
</table>

Supervisor and problem states are discussed in the following sections.

**Supervisor State.** The processor enters supervisor state when:
1. A class interrupt occurs. This type of interrupt is caused by the following:
   a. Machine check condition
   b. Program check condition
   c. Power/Thermal warning
   d. Supervisor Call (SVC) instruction
   e. Soft exception trap condition
   f. Trace bit (LSR bit 10) set to one
   g. Console Interrupt key on the programmer console.
2. An I/O interrupt is accepted.
3. After initial program load (IPL) has completed.

Class interrupts and I/O interrupts are discussed in Chapter 3. Initial program load is discussed in a subsequent section of this chapter.

When the processor is in supervisor state, the full instruction set may be executed. The following privileged instructions may only be executed in supervisor state:
- Copy Console Data Buffer (CP CON) Note 1
- Copy Current Level (CP CL)
- Copy In-Process Flags (CPIPF)
- Copy Interrupt Mask Register (CPIMR)
- Copy Level Status Block (CPLB)
- Copy Processor Status and Reset (CPPSR)
- Diagnose (DIAG)
- Disable (DIS)
- Enable (EN)
- Level Exit (LEX)
- Operate I/O (IO)
- Set Console Data Lights (SECON) Note 2
- Set Interrupt Mask Register (SEIMR)
- Set Level Status Block (SELB)

**Notes.**
1. The resultant data is unpredictable if the programmer console feature is not installed.
2. Performs no operation if the programmer console feature is not installed.

**Problem State.** This is a state that does not allow the processor to execute the privileged instructions. The processor enters the problem state when the supervisor state bit (LSR bit 8) is turned off. This can be accomplished with a Set Level Status Block (SELB) instruction. This instruction can change the contents of the registers for a selected processor level.

While the processor is in problem state, privileged instructions cannot be executed. If a privileged instruction execution is attempted, the instruction is suppressed and a program check class interrupt occurs, with privilege violate (bit 2) set in the processor status word.
State of Processor Following a Reset
The term reset used in the following sections denotes the reset action that occurs during:

1. Power-on reset
2. Initial program load (IPL) reset
3. System reset initiated by pressing the Reset key on the programmer console

The following registers and conditions are not initialized by a reset and require program or operator action before they become valid:

- Console data buffer (Programmer Console Feature)
- General registers
- IAR on levels 1–3
- Main storage (above 16K)
- Address Compare Register

The following registers and conditions are initialized by a reset:

- Level Zero Indicator (turned on)
- CIAR — set to zeros
- IAR on level zero — set to zeros
- Mask register — set to ones (all levels enabled)
- LSR on level zero
  - Indicators — set to zeros
  - Supervisor state (bit 8) — set on
  - In-process (bit 9) — set on
  - Trace (bit 10) — set to zero (disabled)
  - Summary mask (bit 11) — set on (enabled)
  - All other bits — set to zeros
- PSW — set to zeros except as noted
  - Auto-IPL (bit 13) — set to zero unless the reset was caused by an Auto-IPL
  - Power/Thermal (bit 15) reflects the status of the power/thermal condition
- LSR on levels 1–3 — set to zero
- SAR — set to zeros
- Display buffer — set to all ones by Power-on Reset only (Programmer Console Feature)
- Main storage (0–16K, verified good parity Power-on Reset only)
- Check Restart (Note 1)
- Instruction Step (Note 1)
- Stop on Address (Note 1)
- Stop on Error (Note 1)

Note 1. This condition is reset by a Power-on Reset.

Initial Program Load (IPL)
An initial program load function is provided to (1) read an IPL record (set of instructions) from an external storage media, and (2) automatically execute a start-up program. An IPL record is read into storage from a local I/O device or host system. The I/O attachments for the desired IPL sources are prewired at installation time. Two local sources, primary and alternate, can be wired and either can be selected by using the IPL Source switch on the console.

IPL can be started by three methods:

1. Manually, by pressing the Load key on the console.
2. Automatically, after a power-on condition.
3. Automatically, when a signal is received from a host system. The host system can be connected through a communications adapter.

The automatic power-on IPL is selected by a mode switch on the console. When the Mode switch is in the Auto-IPL position, IPL occurs whenever power turns on (either initially or after a power failure). Power must be good to all attachments before the IPL sequence begins.

Auto IPL is useful for unattended systems. A manual IPL can be initiated at any time by pressing the load key on the console (even when in run state). The mode switch has no effect on the manual IPL. For Auto-IPL and manual IPL, the local IPL source (primary or alternate) is selected. IPL from a host system can occur at any time and is initiated by the host system. The IPL record is transferred through the host-system device; for example, the communications adapter. When an auto-IPL occurs, bit 13 of the PSW is turned on to indicate the condition to the software. When a manual or host-system IPL occurs, this bit is set to zero.

During IPL main storage is loaded starting at location zero. The length of the IPL record depends on the media used by the IPL source.

Upon successful completion of an IPL, the processor enters supervisor state and begins execution on priority level zero. The summary mask is enabled and all priority interrupt levels in the mask register are enabled. The first instruction to be executed is at main storage location zero. The IPL source has a pending interrupt request on level zero. The system program must:

1. Perform housekeeping; for example, load vector table addresses in the reserved area of storage (see Automatic Interrupt Branching in Chapter 3).
2. Issue a Level Exit (LEX) instruction. This allows the processor to accept the interrupt from the IPL source. When the interrupt is accepted, a forced branch is taken using the device-address vector table. The vector table entry is determined by the device address of the IPL source and results in a branch to the proper program routine for handling the interrupt. The device address of the IPL source is set into bits 8–15 of register 7 on level zero. Condition code 3, device end, is reported by the IPL source. For additional information, see I/O Interrupts in Chapter 3.
A system reset always occurs prior to an IPL. However, if any errors occur during the IPL, the results are unpredictable.

**Sequential Instruction Execution**

Normally, the operation of the processor is controlled by instructions taken in sequence. An instruction is fetched from the main storage location specified in the instruction address register (IAR). The instruction address in the IAR is then increased by the number of bytes in the instruction just fetched. The IAR now contains the address of the next sequential instruction. After the current instruction is executed, the same steps are repeated using the updated address in the IAR.

A change from sequential operation can be caused by branching, jumping, interrupts, level switching, or manual intervention.

**Jumping and Branching**

The normal sequential execution of instructions is changed when reference is made to a subroutine; when a two-way choice is encountered; or when a segment of coding, such as a loop, is to be repeated. All of these tasks can be accomplished with branching and jumping instructions. Provision is also made for subroutine linkage, permitting not only the introduction of a new instruction address, but also the preservation of the return address and associated information.

The conditional branch and jump instructions are used to test the indicators in the LSR. These indicators are set as the result of I/O operations and most arithmetic or logical operations. Single or multiple indicators are tested as determined by the value in a three-bit field within the instruction. Refer to: (1) **Indicators** and (2) **Testing Indicators with Conditional Branch and Jump Instructions**.

**Jumping**

Jump instructions are used to specify a new instruction address relative to the address in the IAR. The new address must be within -256 to +254 bytes of the byte following the jump instruction.

**Note**. The jump instruction contains a word displacement that is converted to a byte displacement when the instruction is executed. However, when using the assembler, the programmer specifies a byte value that is converted to a word displacement by the assembler.

**Branching**

Branch instructions are used to specify a new full-width 16-bit address. A 16-bit value, range 0 to 65535, is contained in the second word of the instruction or in a register. The value in the second word can be used as the effective branch address or added to the contents of a base register to form an effective address. (See **Base Register Storage Address** in this chapter.)

**Level Switching and Interrupts**

The processor can execute programs on four different interrupt priority levels. These levels, listed in priority sequence, are numbered 0, 1, 2, and 3 with level 0 having highest priority. The processor switches from one level to another in two ways:

1. Automatically, when an interrupt request is accepted from an I/O device operating on a higher priority level than the current level.
2. Under program control, by using the Set Level Block (SELB) instruction.

Both types of level switching are discussed in detail in Chapter 3. **Class Interrupts** and **Interrupt Masking Facilities** are also discussed in Chapter 3.

**Stack Operations**

The processing unit provides two types of stacking facilities. Each facility is briefly described in this section. Additional information appears in subsequent sections. The two types of stacking facilities are:

1. **Data Stacking**. This facility provides an efficient and simple way to handle last-in first-out (LIFO) queues of data items and/or parameters in main storage. The data items or parameters are called stack elements. For a given queue (or stack), each element is one, two, or four bytes wide. Instructions for each element size (byte, word, or doubleword) are provided to:
   a. Push an element into a stack (register to storage).
   b. Pop an element from a stack (storage to register).
2. **Linkage stacking**. This facility provides an easy method for linking subroutines to a calling program. A word stack is used for saving and restoring the status of general registers and for allocating dynamic work areas. The Store Multiple (STM) instruction stores the contents of the registers into the stack and reserves a designated number of bytes in the stack as a work area. The Load Multiple and Branch (LMB) instruction reloads the registers, releases the stack elements, and causes a branch via register 7 back to the calling program.
Data Stacking Description

Any contiguous area of main storage can be defined as a stack. Each stack is defined by a stack control block. Figure 2-4 shows a data stack and its associated stack control block. Stack control blocks must be aligned on a word boundary.

The words in the stack control block are used as follows:

High Limit Address (HLA). This word contains the address of the first byte beyond the area being used for the stack. All data in the stack has a lower address than the contents of the HLA. Note that the HLA points to the first byte beyond the bottom of an empty stack.

Low Limit Address (LLA). This word designates the lowest storage location that can be used for a stack element. Note that the LLA points to the top of a stack.

Top Element Address (TEA). This word points to the stack element that is currently on top of the stack. For empty stacks, the TEA points to the same location as the high limit address (HLA).

Note. For word stacks or double word stacks, the HLA, LLA, and TEA must all contain an even address to ensure data alignment on a word boundary.

Figure 2-4. Relationship of stack control block to data stack
**Push Operation.** When a new element is pushed into a stack, the address value in the TEA is decremented by the length of the element (one, two, or four bytes) and compared against the LLA. If the TEA is less than the LLA, a stack overflow exists. A soft exception trap interrupt occurs with `stack exception` set in the PSW. The TEA is unchanged. If the stack does not overflow, the TEA is updated and the new element is moved to the topic location defined by the TEA.

The following diagram shows how elements are pushed into a stack. Note that each push operation always places an element at a lower address in the stack than the preceding element.

Refer to Chapter 6 for descriptions of the following instructions:
- Push Byte (PSB)
- Push Word (PSW)
- Push Doubleword (PSD)

**Pop Operation.** When an element is popped from a stack, the TEA is compared against the HLA. If it is equal to or greater than the HLA, an underflow condition exists. A soft exception trap interrupt occurs with `stack exception` set in the PSW. If the stack does not underflow, the stack element defined by the TEA is moved to the specified register and the TEA is incremented by the length of the element.

The following diagram shows how elements are popped from a stack.
Refer to Chapter 6 for descriptions of the following instructions:

- Pop Byte (PB)
- Pop Word (PW)
- Pop Doubleword (PD)

Data Stacking Example – Allocating Fixed Storage Areas

Many programs require temporary main storage work areas. It is very useful to be able to dynamically assign such work-area storage to a program only when that storage is needed. Conversely, when work-area storage is no longer needed by a program, it is desirable to free that resource so it may be used by other programs. Use of the stacking mechanism can assist in the programming of the dynamic storage management function.

The following is an example of how storage areas could be allocated using the stacking mechanism.

A stack is initialized with addresses that point to a fixed area of storage. Each element in the stack represents the starting address of a block of storage consisting of 512 bytes; e.g., addresses 0200 through 03FF. As storage is needed, the starting address for a block of storage is popped from the stack. When the block of storage is no longer needed, the starting address is pushed back into the stack.

The stack control block, stack, and storage areas appear initially as follows:
Notice that each stack element is one word long; addresses of storage areas are the stack elements; the TEA points to the lowest location of the last element because the initialized stack is full. Contrast this with an empty stack, in which the TEA points to the same location as the HLA.

Now assume that program A requires a block of storage. Program A (or a storage management function at the request of program A) issues a pop word instruction against the stack control block. The TEA is updated as follows:

The word element popped is placed in the register specified by the pop word instruction executed by program A. This is the address of the 512-byte storage area beginning at address 0200.
At this time, assume that program B (operating on a different hardware level than program A) also requires a storage area. It too executes a pop word instruction against the stack. The next element is moved to the register specified and points to the next available storage area and the TEA is updated:

![Stack control block diagram]

Now, before any further requests occur, program A terminates its need for a work area. Program A then issues a push word instruction against the stack and returns the address of the area it was using for use by other programs:

![Stack control block diagram]

A similar operation will be performed by program B when it releases its storage to the stack, popping address 0400 into location 0B00. While the addresses are obviously shuffled in the stack (from the values initially established), this presents no problem since each program requires only an area of storage — it is not important where that area is located.
**Linkage Stacking Description**

As previously described, a word-stack mechanism may be used for subroutine linkage. This mechanism saves and restores registers and allocates dynamic work areas.

The letters in the following description correspond to the letters in Figure 2-5.

The Store Multiple (STM) instruction specifies:

- **A** Stack control block address
- **B** Limit register (RL) number
- **C** Number (N) of words to allocate for work areas

When the STM instruction is executed, the allocate value (N) plus the number of registers saved plus one control word is the requested block size in words. This times 2 is the size in bytes. The block size is used to decrement the TEA before an overflow check is made. If no overflow occurs, the operation proceeds. The link register (R7) and register 0 through the specified limit register (RL) are saved sequentially in the stack. If register 7 is specified as the limit register, only register 7 is stored in the stack. The dynamic work space is allocated and a pointer to the work area is returned in register RL. If no work area is specified, the returned pointer contains the location of R7 in the stack. The values of RL and N are saved as an entry in the stack. The TEA is updated to point to the new top of the stack location.

When a Load Multiple and Branch (LMB) instruction is executed, the values of RL and N are retrieved from the stack and an underflow check is made. The value of RL controls the reloading of the registers; the values of RL and N are used to restore the stack pointer (TEA) to its former status. The contents of register 7 are then loaded into the instruction address register, returning program control to the calling routine.

**Figure 2-5. Word stack for subroutine linkage**
Linkage Stacking Example – Reenterable Subroutine

A subroutine may be used by programs that operate on different interrupt levels. Rather than providing copies of the subroutine, one copy for each program that needs it, the subroutine can be made reenterable. Here, only one copy of the subroutine is provided; the single copy is used by all requesting programs. Two items must be considered in the reenterable subroutine code:

• Saving the register contents of each calling program. The subroutine is then free to use the same registers, restoring their contents to the calling-program's values just prior to returning to the calling program.

• Preserving the applicable variable data (generated by the subroutine) that is related to each call of the subroutine. That is, data associated with one call must not be disturbed when subroutine execution is restarted due to another call from a higher priority program.

The stacking mechanism, by means of the STM and LMB instructions, handles the two items just mentioned. As an example, operation could proceed as follows:

1. Program A calls the subroutine by means of a branch and link instruction (return address is in R7).

   BAL SUBRT,7

2. The subroutine, in this example, uses registers R3 and R4 during its execution. The subroutine receives (from program A) a parameter list address in R0 and the address of the stack control block in R1. Also, the subroutine requires 20 bytes of work space. Thus, the subroutine executes, upon entry, the following store multiple instruction:

   SUBRT STM 4,(1),20

   After execution of the STM, the stack contains the following:

   * The last word contains a value that specifies the last register stored (e.g., R4 in this example) and the size of the dynamic work area (in words).

   R4 (the last register stored in the stack) is automatically loaded, during the STM operation, with the address of the work area to be used by the subroutine to hold its work data.

3. When subroutine processing for this call is completed, the subroutine executes a single load multiple instruction in order to reload the registers and return (via R7) to the calling program:

   LMB (1)

   If a second call to the subroutine has occurred prior to execution of the LMB, action similar to that just stated would occur again. However, another stack area would be used. Then, when subroutine execution is completed for the second call, and all higher priority interrupt level processing is completed, a return would be made to the interrupted subroutine for completion of processing for the first call.

   Thus, multiple calls to a single subroutine are processed without interfering with the integrity of data associated with any other call to the subroutine.
Chapter 3. Interrupts and Level Switching

Introduction

Efficient operation of a central processor depends on prompt response to I/O device service requests. This is accomplished by an interrupt scheme that stops the current processor operation, branches to a device service routine, handles device service, then returns to continue the interrupted operation. One processor can control many I/O devices; therefore, an interrupt priority is established to handle the more important operations before those of lesser importance. Certain error or exception conditions (such as a machine check) also cause interrupts. These are called class interrupts and are processed in a manner similar to I/O interrupts. Both I/O and class interrupts are explained further in the following sections.

Interrupt priority is established by four priority levels of processing. These levels, listed in priority sequence, are numbered 0, 1, 2, and 3 with level 0 having highest priority. Interrupt levels are assigned to I/O devices via program control. This provides flexibility for reassigning device priority as the application changes.

Each of the four priority levels has its own set of registers. These consist of a level status register (LSR), eight general registers (R0–R7), and an instruction address register (IAR). Information pertaining to a level is automatically preserved in these hardware registers when an interrupt occurs.

Processor level switching, under program control, may be accomplished by use of the Set Level Block (SELB) instruction. Details of this method are presented in a separate section of this chapter.

I/O and class interrupts cause automatic branching to a service routine. Fixed locations in main storage are reserved for branch addresses or pointers which are referenced during interrupt processing. This storage allocation is shown in the section Automatic Interrupt Branching in this chapter.

Interrupt masking facilities provide additional program control over the four priority levels. System and level masking are controlled by the Summary Mask and the Interrupt Level Mask Register. Device masking is controlled by the Device Mask. Manipulation of the mask bits can enable or disable interrupts on all levels, a specific level, or for a specific device. See Interrupt Masking Facilities in this chapter.

Interrupt Scheme

As previously stated, four priority interrupt levels exist. Each I/O device is assigned to a level, dependent on the application. When an interrupt on a given level is accepted, that level remains active until (1) a Level Exit (LEX) instruction is executed, (2) a Set Level Block (SELB) instruction causes a level switch, or (3) a higher priority interrupt is accepted. In the first two cases, the active level at the time is cleared. In the latter case, the processor switches to the higher level, completes execution (including a LEX instruction), then automatically returns to the interrupted-from level. This automatic return can be delayed by other higher priority interrupts.

If an interrupt request is pending on the currently active level, it will not be accepted until after execution of a LEX instruction by the current program. If no other level of interrupt is pending when a Level Exit instruction is executed, the processor enters the wait state. In the wait state no processing is performed, but the processor can accept interrupts that are expected to occur. See Figure 3-1.

Class interrupts do not change priority levels. They are processed at the currently active level. If the processor is in the wait state when a class interrupt occurs, priority level 0 is used to process the interrupt.
Requests for interrupts

Level 0

Level 1

Level 2

Level 3

Priority level processing

Priority level 0

Priority level 1

Priority level 2

Priority level 3

* This interrupt request cannot be honored until after a LEX instruction has been executed on level 3 to clear the previous interrupt service.

Figure 3-1. Interrupt priority scheme

Automatic Interrupt Branching

Hardware processing of an interrupt includes automatic branching to a service routine. The processor uses a reserved storage area for branch information. The reserved area begins at main storage address 0000. The total size of the area depends on the number of interrupting devices attached. One word (two bytes) is reserved for each interrupting device and is related to a particular device by the device address. For example: device 00 causes a reference to location 0030, device 01 to location 0032, and so on. The device area begins at address 0030 (Hex); the reserved area is 0000 through 022F (Hex) if 256 devices (maximum number) are attached. These storage locations and contents are shown in Figure 3-2.

<table>
<thead>
<tr>
<th>Main storage address (Hex)</th>
<th>Contents of word</th>
</tr>
</thead>
<tbody>
<tr>
<td>022E</td>
<td>Device FF DDB pointer</td>
</tr>
<tr>
<td>0032</td>
<td>Device 01 DDB pointer</td>
</tr>
<tr>
<td>0030</td>
<td>Device 00 DDB pointer</td>
</tr>
<tr>
<td>002E</td>
<td>Reserved</td>
</tr>
<tr>
<td>002C</td>
<td>Reserved</td>
</tr>
<tr>
<td>002A</td>
<td>Reserved</td>
</tr>
<tr>
<td>0028</td>
<td>Reserved</td>
</tr>
<tr>
<td>0026</td>
<td>Reserved</td>
</tr>
<tr>
<td>0024</td>
<td>Reserved</td>
</tr>
<tr>
<td>0022</td>
<td>Soft exception trap SIA</td>
</tr>
<tr>
<td>0020</td>
<td>Soft exception trap LSB pointer</td>
</tr>
<tr>
<td>001E</td>
<td>Console interrupt SIA</td>
</tr>
<tr>
<td>001C</td>
<td>Console interrupt LSB pointer</td>
</tr>
<tr>
<td>001A</td>
<td>Trace SIA</td>
</tr>
<tr>
<td>0018</td>
<td>Trace LSB pointer</td>
</tr>
<tr>
<td>0016</td>
<td>Power failure SIA</td>
</tr>
<tr>
<td>0014</td>
<td>Power failure LSB pointer</td>
</tr>
<tr>
<td>0012</td>
<td>SVC SIA</td>
</tr>
<tr>
<td>0010</td>
<td>SVC LSB pointer</td>
</tr>
<tr>
<td>000E</td>
<td>Program check SIA</td>
</tr>
<tr>
<td>000C</td>
<td>Program check LSB pointer</td>
</tr>
<tr>
<td>000A</td>
<td>Machine check SIA</td>
</tr>
<tr>
<td>0008</td>
<td>Machine check LSB pointer</td>
</tr>
<tr>
<td>0006</td>
<td>Reserved</td>
</tr>
<tr>
<td>0004</td>
<td>Reserved</td>
</tr>
<tr>
<td>0002</td>
<td>Restart instruction word 2</td>
</tr>
<tr>
<td>0000</td>
<td>Restart instruction word 1</td>
</tr>
</tbody>
</table>

Figure 3-2. Reserved storage locations
The reserved storage locations are described as follows:

<table>
<thead>
<tr>
<th>Storage Location (Hex)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-0003</td>
<td>Restart instruction. Following IPL a forced branch is made to location 0000.</td>
</tr>
<tr>
<td>0004-0005</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0006-0007</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0008-0023</td>
<td>Addresses used for class interrupts. The Level Status Block (LSB) pointer is the first address of an area where a level status block will be stored. The Start Instruction Address (SIA) points to the first instruction of a service routine.</td>
</tr>
<tr>
<td>0024-002F</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0030-022F</td>
<td>Addresses used for I/O interrupts. The Device Data Block (DDB) pointer is the address of the first word of a device data block. This word is used to obtain the start instruction address for the service routine. See I/O Interrupts in this chapter.</td>
</tr>
</tbody>
</table>

Note. The area reserved for I/O devices varies in size depending on the number of devices. The device address determines the fixed location to be accessed. For example: Interrupts for device 01 always vector to main storage address 0032.

A device address is established by installing the appropriate connectors on the I/O feature card for the device.

I/O Interrupts

Prepare I/O Device for Interrupt

I/O device interrupt parameters are established via program control. The Operate I/O (IO) instruction initiates the device operation and in conjunction with the “Prepare” I/O command tells the device:

1. If the device can interrupt.
2. What priority level to use for interrupts. See Chapter 6 Instructions and Chapter 4 Input/Output Operations for details of the Operate I/O instruction.

Execution of the Prepare command transfers a word to the addressed device that controls its interrupt parameters. This word has the format:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–10</td>
<td>Set to zeros.</td>
</tr>
<tr>
<td>11–14</td>
<td>Level. A four-bit encoded field that assigns an interrupt priority level to the device (see note). Example: 0000 - level 0, 0001 - level 1, 0010 - level 2, 0011 - level 3.</td>
</tr>
<tr>
<td>15</td>
<td>Device mask or I-bit. This bit sets the interrupt mask in the device. When set to one, the device can interrupt. When set to zero, the device cannot request an interrupt.</td>
</tr>
</tbody>
</table>

Note. The 4953 Processor does not recognize priority levels other than zero through three; therefore, bits 11 and 12 must always be set to zero or the interrupt is lost.

An interrupting device is always able to accept and execute a Prepare command, even if it is presently busy or has an interrupt request pending from a previous command. This allows the software to change the device mask and interrupt level at any time. Any pending interrupt request is then serviced on the new interrupt level.

Present and Accept I/O Interrupt

The I/O device presents an interrupt request on its assigned priority level. This request is applied to the interrupt algorithm for acceptance determination.

For an I/O interrupt to be serviced, the following conditions must exist:

1. The summary mask must be on (enabled).
2. The mask bit (Interrupt Level Mask Register) for the interrupting level must be on (enabled).
3. For I/O interrupts the device must have its Device Mask bit on (enabled).
4. The interrupt request must be the highest priority of the outstanding requests and higher than the current level of the processor.
5. The processor must not be in the stop state.

Supervisor state is entered upon acceptance of all priority interrupts.

Following acceptance, the device sends an interrupt ID word and a condition code to the processor. The condition code is placed in the even, carry, and overflow indicators for the interrupted-to level. The ID word is placed into register 7 of the interrupted-to level. The interrupt ID word consists of an interrupt information byte (IIB) and the device address. Bits 0–7 of this word contain the interrupt information; bits 8–15 contain the device address. See Chapter 4 for condition codes and interrupt information byte (IIB) details. Hardware causes the following events to occur after the processor receives the interrupt ID word and the condition code (Figure 3-3):

- The processor hardware switches from the registers and status of the interrupted-from level to those of the interrupted-to level.
- The interrupt ID word is placed in register 7 of the interrupted-to level.
- The condition code is placed in LSR positions 0–2.
- Supervisor state is entered (LSR bit 8 is set to one).
- The processor executes an automatic branch.
  - The device address is used by hardware to fetch the DDB pointer from reserved storage.
  - The DDB pointer is placed in register 1 of the interrupted-to level.
  - The DDB pointer is used by hardware to fetch the start instruction pointer.
  - The Start Instruction Address (SIA) is loaded into the IAR of the interrupted-to level.
- Execution begins on the new level.
Figure 3-3. Example of I/O interrupt with automatic branching
Class Interrupts

System error or exception conditions can cause seven types of class interrupts:

1. Machine check, caused by a hardware error.
2. Program check, caused by a programming error.
3. Power/thermal warning, caused by a power or thermal irregularity.
4. Supervisor call, caused by execution of an SVC instruction.
5. Soft exception trap, caused by software.
6. Trace, caused by instruction execution (trace enabled in the current LSR).
7. Console, caused by a console interrupt when the optional programmer console is installed.

Machine check, program check, soft exception trap, and power/thermal warning are defined by bits in the processor status word. Software can refer to the processor status word for a specific condition and any related status information. See Processor Status Word in this chapter.

Class interrupts do not cause a change in priority level. The interrupt is serviced on the level that is active when the condition occurs. If the processor is in the wait state, the interrupt is serviced on priority level zero. Independent routines are used to handle each type of class interrupt regardless of priority level.

All class interrupts cause the processor to enter supervisor state. Refer to a subsequent section, Present and Accept Class Interrupt, for details of the hardware processing.

Programming Notes.

1. Two class interrupts (power/thermal warning and console) can be disabled by the summary mask.
2. If the optional programmer console is installed and Check Restart or Stop on Error are selected, machine check, power/thermal warning, and program check interrupts do not occur. See Programmer Console Feature in Chapter 5.

Priority of Class Interrupts

Although class interrupts are serviced on the current priority level, they are serviced according to an exception condition priority.

The following table lists the exception conditions in priority sequence with zero being the highest priority. Two exception conditions of the same priority, such as invalid storage address and specification check, may be reported to the PSW simultaneously. The table also shows the associated class interrupt vector for the exception conditions.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Exception Condition</th>
<th>Class Interrupt Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>CPU control check</td>
<td>Machine check</td>
</tr>
<tr>
<td>1</td>
<td>Invalid function (Note 1)</td>
<td>Program check</td>
</tr>
<tr>
<td>2</td>
<td>Privilege violate</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Invalid function (Note 2)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Not applicable on 4953 Processor</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Invalid storage address Specification check</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Storage parity</td>
<td>Machine check</td>
</tr>
<tr>
<td>7</td>
<td>Power warning</td>
<td>Power/thermal warning</td>
</tr>
<tr>
<td>8</td>
<td>Supervisor call</td>
<td>Supervisor call</td>
</tr>
<tr>
<td>9</td>
<td>Invalid function (Note 3)</td>
<td>Soft exception trap</td>
</tr>
<tr>
<td>10</td>
<td>Not applicable on 4953 Processor</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Stack exception</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Trace</td>
<td>Trace</td>
</tr>
<tr>
<td>13</td>
<td>Console</td>
<td>Console</td>
</tr>
</tbody>
</table>

Note 1. Caused by an illegal operation code or function combination.

Note 2. A Copy Segmentation Register (CPSR) or Set Segmentation Register (SESR) instruction is attempted. The translator feature is not available in the 4953 Processor.

Note 3. A floating-point instruction is attempted. The floating-point feature is not available on the 4953 Processor.

Present and Accept Class Interrupt

When a class interrupt occurs, it is serviced on the currently active level or on level zero (if in the wait state). Hardware processing of the interrupt causes the following:

- Register contents are saved
- Supervisor state is entered (LSR bit 8 is set to one)
- Trace is reset (LSR bit 10 is set to zero)
- Summary mask is disabled (LSR bit 11 is set to zero)
- An automatic branch is taken to a service routine.

Each type of class interrupt has an associated LSB pointer and SIA in the reserved area of main storage (see Figure 3-2). Reference is made to the reserved area to:

1. Store current level IAR, registers, and LSR into a level status block (LSB) in main storage.
2. Automatically branch to a service routine by using the start instruction address (SIA).

Note. Priority level zero is forced active when a class interrupt occurs in the wait state. The level zero LSB is stored into main storage. The in-process flag (LSR bit 9) is zero in the stored LSB.
Contents of the level status block are as follows:

<table>
<thead>
<tr>
<th>Main storage address (LSB) pointer)</th>
<th>Instruction address register</th>
<th>Zero</th>
<th>Level status register</th>
<th>Register 0</th>
<th>Register 1</th>
<th>Register 2</th>
<th>Register 3</th>
<th>Register 4</th>
<th>Register 5</th>
<th>Register 6</th>
<th>Register 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>+14 (Hex)</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

The instruction address (contents of IAR) stored in the LSB depends on the type of class interrupt and is shown in the following chart.

<table>
<thead>
<tr>
<th>Type of Class Interrupt</th>
<th>Contents of IAR (Stored in LSB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program check</td>
<td>Address of instruction that caused the interrupt.</td>
</tr>
<tr>
<td>Supervisor call</td>
<td>Address of the next instruction.</td>
</tr>
<tr>
<td>Trace</td>
<td></td>
</tr>
<tr>
<td>Console</td>
<td></td>
</tr>
<tr>
<td>Power/thermal warning</td>
<td></td>
</tr>
<tr>
<td>Machine check (with Sequence indicator off)</td>
<td>Address of instruction that caused the interrupt.</td>
</tr>
<tr>
<td>Machine check (with Sequence indicator on)</td>
<td>Address of instruction that was being executed at the time of the error.</td>
</tr>
</tbody>
</table>

### Machine Check

A machine check interrupt is caused by a hardware malfunction and is considered a system-wide incident. The three types are:

1. Storage parity check (PSW bit 08)
2. CPU control check (PSW bit 10)
3. I/O check (PSW bit 11)

A level status block is stored, starting at the location in main storage designated by the machine check LSB pointer (contents of storage locations hex 0008 and 0009). The contents of the storage address register (SAR) are loaded into register seven. The machine check SIA (contents of storage locations hex 000A and 000B) is then loaded into the IAR, becoming the address of the next instruction to be fetched.

**Note.** When the error condition occurs, the IAR contains the true address of the first word of the instruction; it is not incremented if the error occurs in the second or third word of a long instruction.

### Program Check

A program check interrupt is caused by a programming error. The types are:

1. Specification check (PSW bit 00).
2. Invalid storage address (PSW bit 01).
3. Privilege violate (PSW bit 02).
4. Invalid function (PSW bit 04).

A level status block is stored, starting at the location in main storage designated by the program check LSB pointer (contents of storage locations hex 000C and 000D). The contents of the storage address register (SAR) are loaded into register seven. The program check SIA (contents of storage locations hex 000E and 000F) is then loaded into the IAR, becoming the address of the next instruction to be fetched.

**Note.** A program check interrupt condition on one priority level does not affect software on other levels.
**Power/Thermal Warning (PSW Bit 15)**

A power/thermal warning class interrupt is initiated by:

1. A power warning signal that is generated when the power line decreases to about 85% of its rated value.
2. A thermal warning that occurs if the temperature limits inside the closure are exceeded.

In both cases, the instruction address that is stored in the LSB points to the next instruction to be executed.

A level status block is stored, starting at the location in main storage designated by the power failure LSB pointer (contents of storage locations hex 0014 and 0015). The power failure SIA (contents of storage locations hex 0016 and 0017) is then loaded into the IAR, becoming the address of the next instruction to be fetched.

A power/thermal warning interrupt can occur when the system is running or in the wait state, assuming (1) the summary mask is enabled and (2) the programmer console is not set to Check Restart or Stop on Error. These interrupts are not taken by the processor if either of the two conditions are not met.

If the optional battery backup unit is installed and a power warning occurs, PSW bit 15 remains on as long as power is supplied by the battery. If a thermal warning occurs, the processor will power down regardless of the battery backup unit. The minimum time before the processor powers down is 20 milliseconds. The IBM 4999 Battery Backup Unit is explained in a separate publication; *IBM Series/1 Battery Backup Unit Description*, GA34-0032.

Power/thermal warning interrupts are not taken by the processor until the first instruction is executed following a power-on reset, an IPL, or exit from stop state.

*Note.* If the processor is in the wait state when the power/thermal condition occurs:

1. The interrupt is serviced on priority level 0. The level 0 LSB is stored into main storage. Additional power/thermal interrupts are disabled at this time because the summary mask is set to zero by the class interrupt.
2. The instruction address stored in the LSB is unpredictable.

**Supervisor Call**

A supervisor call class interrupt is initiated by executing an SVC instruction. The SVC instruction is described in Chapter 6. A level status block is stored, starting at the main storage location designated by the supervisor call LSB pointer (contents of storage locations hex 0010 and 0011). The supervisor call SIA (contents of storage locations 0012 and 0013) is then loaded into the IAR, becoming the address of the next instruction to be fetched.

**Soft Exception Trap**

A soft exception trap interrupt is caused by software. The types are:

1. Invalid function (PSW bit 4)
2. Stack exception (PSW bit 6)

These exception conditions may be handled by software; therefore, they do not constitute an error condition.

A level status block is stored, starting at the location in main storage designated by the soft-exception-trap LSB pointer (contents of storage locations hex 0020 and 0021). The contents of the storage address register (SAR) are loaded into register seven. The soft-exception-trap SIA (contents of storage locations hex 0022 and 0023) is then loaded into the IAR, becoming the address of the next instruction to be fetched.

*Note.* The contents of register seven are unpredictable.

**Trace**

The trace class interrupt provides an instruction trace facility for software debugging. Instruction tracing may occur on any priority level, and is enabled by the trace bit (LSR bit 10). The tracing occurs when bit 10 of the current LSR is set to one. When trace is enabled, a trace class interrupt occurs at the beginning of each instruction. A level status block is stored, starting at the location in main storage designated by the trace LSB pointer (contents of storage locations hex 0018 and 0019). The trace SIA (contents of storage locations hex 001A and 001B) is then loaded into the IAR, becoming the address of the next instruction to be fetched.

*Note.* After the LSB is stored, and before the next instruction is fetched, supervisor state is set on (LSR bit 8), trace is turned off (LSR bit 10), and the summary mask is disabled (LSR bit 11).

**Programming Note.** When trace is enabled, a trace class interrupt occurs prior to executing each instruction. Hardware processing of the interrupt provides an automatic branch to the programmer's trace routine. To prevent retracing the same instruction, the program should exit the trace routine by using the Set Level Block (SELB) instruction with the inhibit trace (IT) bit set to one. The inhibit trace bit prevents a trace interrupt from occurring for the duration of one instruction (see SELB instruction in Chapter 6). A double trace of an instruction can also occur when the instruction is interrupted and must be reexecuted for example: a class interrupt occurs during execution of a variable field length instruction. Under this condition, exit from the class interrupt routine should be via a SELB instruction with the inhibit trace bit set to one.

The occurrence of any class interrupt or priority interrupt causes the trace bit (LSR bit 10) to be set to zero. This action permits tracing only problem state code. If the programmer desires to trace supervisor code, he must make provisions within the service routine to enable the trace bit.
The following three conditions inhibit a trace class interrupt:

1. A Set Level Block (SELB) instruction sets the trace bit on and the in-process bit on in the LSR of a selected level lower than the current level; then, when the selected level becomes active, the first instruction executed is not preceded by a trace interrupt.

2. The programmer console is in diagnose mode and a stop instruction is encountered while tracing; then, when the Start Key is depressed, a trace interrupt does not occur prior to executing the first instruction.

3. When a level is exited by either a LEX or a SELB instruction and processing is to continue on a pending level, one instruction is executed on the pending level prior to sampling for a trace interrupt.

Console

A console interrupt function is provided when the optional programmer console is installed. To recognize the interrupt, the processor must have the summary mask enabled and be in the run state or wait state. A level status block is stored, starting at the main storage location designated by the console interrupt LSB pointer (contents of storage locations hex 001C and 001D). The console interrupt SIA (contents of storage locations hex 001E and 001F) is then loaded into the IAR, becoming the address of the next instruction to be fetched.

Note. If the processor is in the wait state when a console interrupt occurs, the interrupt is serviced on priority level 0.

Summary of Class Interrupts

The following chart is a summary of class interrupt processing. Each class interrupt is fully explained in separate sections of this chapter.

---

**Recovery from Error Conditions**

Error recovery procedures, initiated by software, depend on several factors:

1. Application involved.
2. Type of error.
3. Number of recommended retries.

The error class interrupt provides an automatic branch to a service routine. This routine can interrogate the PSW for specific error and status information. The routine can then initiate corrective action or retry the failing instruction(s). If an error occurs during a priority interrupt sequence, the priority level switch is completed before the error class interrupt is processed. This facilitates automatic register retention. A reset is generated by machine check class interrupts caused by an I/O check or a CPU control check. No reset is generated by program check or power/thermal warning class interrupts. Error conditions along with error recovery information are presented in the following sections.

**Program Check**

A program check is caused by a programming error and initiates a program check class interrupt. Error retry depends on the application. All necessary parameters are made available for locating and, if required, correcting the invalid condition. There is no change to operands or priority level during a program check class interrupt. The stored LSB reflects conditions at the time the interrupt occurred and contains:

- The contents of all general registers.
- Status information (LSR contents).
- The address of the failing instruction (IAR contents).

The contents of the storage address register (SAR) are loaded into R7. The programmer must reference the PSW to determine the type of program check.

**Storage Parity Check**

A storage parity error initiates a machine check class interrupt. The error may occur when accessing a storage location that has not been validated since power on. Any retry procedure should include refreshing data in the failing location. Two unsuccessful retries are considered a permanent failure and the storage location should not be used. An IPL should be initiated.

**CPU Control Check**

A CPU control check occurs if hardware detects a malfunction of the processor controls. It is a machine-wide error and initiates a machine check class interrupt. A reset is generated to the channel, the I/O attachment features, and all attached I/O devices. The processor, sensor-based output points, and timer values are not reset. The generated reset should clear the error condition, but validity of any previous execution is not guaranteed. No retry is recommended. An IPL should be initiated.
I/O Check
An I/O check condition occurs if a hardware error is detected that may prevent further communication with I/O devices. A machine check class interrupt is initiated and a reset is generated to the I/O attachment features, the channel, and all I/O devices. Error recovery from an I/O check depends on the sequence indicator setting (PSW bit 12).

Sequence Indicator Set to Zero. The error occurred during an Operate I/O instruction. The address of the failing instruction (IAR contents) is available in the stored LSB. Retry should be attempted twice. After two unsuccessful retries, use of the device should be discontinued.

Sequence Indicator Set to One. The error occurred during an interrupt or cycle steal operation. The instruction address (IAR contents) stored in the LSB is not related to the error. The sequence of events leading to the I/O check is lost, along with all pending interrupt requests within the devices. Retry is not recommended.

Soft Exception Trap
A soft exception trap interrupt is the result of an exception condition that software may choose to handle dynamically. All necessary parameters are available to locate and correct the condition. The address of the instruction (IAR contents) causing the exception is preserved in the level status block in main storage. The processor is not reset. The programmer must reference the PSW to determine the soft-exception type.

Processor Status Word
The processor status word (PSW) is used to record error or exception conditions in the system that may prevent further processing. It also contains certain status flags related to error recovery. Error or exception conditions recorded in the PSW cause four of the possible seven class interrupts to occur. These are machine check, program check, soft exception trap, and power/thermal warning. See Class Interrupts in this chapter.

The Copy Processor Status and Reset (CPPSR) instruction can be used to examine the PSW. This instruction stores the contents of the PSW into a specified location in main storage.

The PSW is contained in a 16-bit register with the following bit representation:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Condition</th>
<th>Class</th>
<th>Interrupt</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Specification check</td>
<td>Program check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>Invalid storage address</td>
<td>Program check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>Privilege violate</td>
<td>Program check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>Not used</td>
<td></td>
<td></td>
<td>always zero</td>
</tr>
<tr>
<td>04</td>
<td>Invalid function</td>
<td>Program check or soft exception trap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>Not used</td>
<td></td>
<td></td>
<td>always zero</td>
</tr>
<tr>
<td>06</td>
<td>Stack exception</td>
<td>Soft exception trap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Not used</td>
<td></td>
<td></td>
<td>always zero</td>
</tr>
<tr>
<td>08</td>
<td>Storage parity check</td>
<td>Machine check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>Not used</td>
<td></td>
<td></td>
<td>always zero</td>
</tr>
<tr>
<td>10</td>
<td>CPU control check</td>
<td>Machine check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>I/O check</td>
<td>Machine check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Sequence indicator</td>
<td>None</td>
<td>Status flag</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Auto-IPL</td>
<td>None</td>
<td>Status flag</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Not used</td>
<td></td>
<td></td>
<td>always zero</td>
</tr>
<tr>
<td>15</td>
<td>Power/thermal warning</td>
<td>Power/thermal</td>
<td></td>
<td>Note 1</td>
</tr>
</tbody>
</table>

Note 1. The power/thermal warning class interrupt is controlled by the summary mask.

Bit 00 Specification Check. Set to one if the storage address violates the boundary requirements of the specified data type.

Bit 01 Invalid Storage Address. Set to one when an attempt is made to access a storage address outside the storage size of the system. This can occur on an instruction fetch, an operand fetch, or an operand store.

Bit 02 Privilege Violate. Set to one when a privileged instruction is attempted in the problem state (supervisor state bit in the level status register is not on).

Bit 04 Invalid Function. Set to one by one of the following conditions:

1. Attempted execution of an illegal operation code or function combination. These are:

<table>
<thead>
<tr>
<th>Op code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111</td>
<td>All</td>
</tr>
<tr>
<td>01000</td>
<td>0001, 0010, 0011, 0101, 0110, 0111</td>
</tr>
<tr>
<td>01011</td>
<td>0001, 1001 (When in supervisor state)</td>
</tr>
<tr>
<td>01011</td>
<td>0101, 0111</td>
</tr>
<tr>
<td>01100</td>
<td>111</td>
</tr>
<tr>
<td>01110</td>
<td>11000, 11010, 11011, 11100, 11110, 11111</td>
</tr>
<tr>
<td>01111</td>
<td>1X1XX, 01XXX, 1X011, 10001</td>
</tr>
<tr>
<td>10110</td>
<td>All</td>
</tr>
<tr>
<td>11011</td>
<td>All</td>
</tr>
<tr>
<td>11101</td>
<td>1100, 1101, 1110, 1111</td>
</tr>
</tbody>
</table>

Note. The preceding illegal conditions cause a program check class interrupt to occur.

2. The processor attempts to execute reserved operation codes or function combinations. These are:

<table>
<thead>
<tr>
<th>Op code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00100</td>
<td>All</td>
</tr>
<tr>
<td>01011</td>
<td>0011, 1011 (When in supervisor state)</td>
</tr>
</tbody>
</table>

Note. The preceding condition causes a soft-exception-trap class interrupt to occur.
Bit 06 Stack Exception. Set to one when an attempt has been made to pop an operand from an empty main storage stack or push an operand into a full main storage stack. A stack exception also occurs when the stack cannot contain the number of words to be stored by a Store Multiple (STM) instruction.

Bit 08 Storage Parity. Set to one when a parity error has been detected on data being read out of storage by the processor. This error may occur when accessing a storage location that has not been validated since power on.

Bit 10 CPU Control Check. A control check will occur if no levels are active but execution is continuing. This is a machine-wide error. (See I/O check note.)

Bit 11 I/O Check. Set to one when a hardware error has occurred on the I/O interface that may prevent further communication with any I/O device. PSW bit 12 (sequence indicator) is a zero if the error occurred during an Operate I/O instruction and is set to one if the error occurred during a non-DPC transfer. The sequence indicator bit is not an error in itself but reflects the last interface sequence at any time. An I/O check cannot be caused by a software error. (See note.)

Note. The machine check class interrupt initiated by a CPU control check or I/O check causes a reset. The I/O channel and all devices in the system are reset as if a Halt I/O (channel directed command) had been executed. The processor, sensor-based output points, and timer values are not reset.

Bit 12 Sequence Indicator. This bit reflects the last I/O interface sequence to occur. See “I/O Check” described above.

Bit 13 Auto IPL. Set to one by hardware when an automatic IPL occurs.

Bit 15 Power Warning and Thermal Warning. Set to one when these conditions occur (see Power/Thermal Warning class interrupt in this chapter). The power/thermal class interrupt is controlled by the summary mask.

Program Controlled Level Switching
Level switching under program control may be accomplished by using the Set Level Block (SELB) instruction. This instruction is covered in detail in Chapter 6, Instructions, and in general it will:

- Specify the location of a level status block (LSB) at an effective address in main storage.
- Specify a selected priority level associated with the main storage LSB.
- Load the main storage LSB into the hardware LSB for the selected level.

Note. The hardware LSB consists of the following hardware registers for the selected level:
1. Instruction address register
2. Level status register
3. Eight general registers (0–7)

The system programmer should become thoroughly familiar with other effects on the processor caused by execution of the SELB instruction. These effects are determined by three factors:
1. The current execution level.
2. The selected level specified in the SELB instruction.
3. The state of the in-process flag (Bit 9 of the LSR) contained in the main storage LSB.

Note. Interrupt masking, provided by the summary mask and the interrupt level mask register, does not apply to program controlled level switching.

The main storage LSB and the location of the in-process flag bit are shown in the following diagram:

<table>
<thead>
<tr>
<th>Main storage effective address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAR</td>
</tr>
<tr>
<td>Zero</td>
</tr>
<tr>
<td>LSR</td>
</tr>
<tr>
<td>Register 0</td>
</tr>
<tr>
<td>Register 1</td>
</tr>
<tr>
<td>Register 2</td>
</tr>
<tr>
<td>Register 3</td>
</tr>
<tr>
<td>Register 4</td>
</tr>
<tr>
<td>Register 5</td>
</tr>
<tr>
<td>Register 6</td>
</tr>
<tr>
<td>Register 7</td>
</tr>
</tbody>
</table>

*In-process flag (bit 9)*

- 0 = off
- 1 = on

Execution of the SELB instruction may result in level switching or a change in the pending status of a level as described in the following sections.
**Selected Level Lower Than Current Level and In-process Flag On**

These conditions cause the selected level to be pending. The main storage LSB is loaded into the hardware LSB for the selected level. Execution of a LEX instruction on the current level causes the selected level to become active providing no higher priority interrupts are being requested.

**Selected Level Equal to Current Level and In-process Flag On**

These conditions cause the selected level to become the current level. The main storage LSB is loaded into the hardware LSB for the selected level.

**Selected Level Higher Than Current Level and In-process Flag On**

These conditions cause the selected level to become the current level. The main storage LSB is loaded into the hardware LSB for the selected level. This is a level switch to the higher (selected) level and causes the lower level to be pending.
**Selected Level Lower Than Current Level and In-process Flag Off**

These conditions cause the selected level to be not pending. The main storage LSB is loaded into the hardware LSB for the selected level.

**Selected Level Equal to Current Level and In-process Flag Off**

These conditions cause an exit from the current level. This exit is identical to executing a LEX instruction with the exception that the main storage LSB is loaded into the hardware LSB for the selected level. See LEX instruction in Chapter 6.

**Selected Level Higher Than Current Level and In-process Flag Off**

The main storage LSB is loaded into the hardware LSB for the higher (selected) level.
Interrupt Masking Facilities

Three levels of priority interrupt masking are provided to the programmer for control of the interrupt processing. These consist of:

1. Summary Mask (LSR bit 11)
2. Interrupt Level Mask Register
3. Device Mask (I-bit)

Each masking facility has specific control as explained in the following sections.

Summary Mask

The summary mask provides a masking facility for priority interrupts and certain class interrupts. The state of the summary mask (enabled or disabled) is controlled by bit 11 in the level status register (LSR) of the active priority level. When bit 11 is set to zero, the summary mask is disabled and prevents (1) all priority interrupts regardless of priority level, and (2) power/thermal and console class interrupts. All other class interrupts are not masked. When bit 11 is set to one, the mask is enabled and the interrupts are allowed.

The summary mask is disabled and enabled as follows:

- Disabled (Set to Zero)
  1. When a Supervisor Call (SVC) instruction is executed, the summary mask for the active level is disabled.
  2. Execution of a Disable (DIS) instruction, with bit 15 of the instruction equal to one, causes the summary mask for the active level to be disabled.
  3. All class interrupts disable the active level summary mask.
  4. The summary mask for a selected level is disabled by executing a Set Level Block (SELB) instruction with bit 11 of the LSR to be loaded equal to zero.
  5. The summary mask bits for priority levels 1–3 are set to zero by a system reset, power-on reset, or IPL.
- Enabled (Set to One)
  1. Execution of an Enable (EN) instruction, with bit 15 of the instruction equal to one, causes the active level summary mask to be enabled.
  2. The summary mask for a selected level is enabled by executing a Set Level Block (SELB) instruction with bit 11 of the LSR to be loaded equal to one.
  3. The level zero summary mask is enabled by a system reset, power-on reset, or IPL.
  4. The summary mask for the interrupted-to level is enabled by a priority interrupt.

Note. If the processor is in the wait state, the summary mask is enabled or disabled as defined by bit 11 in the LSR of the last active priority level.

Interrupt Level Mask Register

The interrupt level mask register is a 4-bit register used for control of interrupts on specific priority levels. Each level is controlled by a separate bit of the mask register as shown below:

<table>
<thead>
<tr>
<th>Bit position</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority level</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With a bit position set to one, the corresponding priority level is enabled and permits interrupts. With a bit position set to zero, the corresponding priority level is disabled.

The Set Interrupt Mask Register (SEIMR) instruction is used to control bit settings in the interrupt level mask register. The Copy Interrupt Mask Register (CPIMR) instruction may be used to interrogate the register.

Note. All levels are enabled (set to one) by a system reset, power-on reset, or IPL.

Device Mask (I-bit)

Each interrupting device contains a one-bit mask called the device mask or interrupt bit (I-bit). Interrupts by the device are permitted when its device mask is enabled (set to one).

With the device mask bit disabled (set to zero), that device cannot cause an interrupt. The device mask is controlled by a Prepare command in conjunction with an Operate I/O instruction. See Chapter 6, Instructions, and Chapter 4, Input/Output Operations.
Input/output (I/O) operations involve the use of input/output devices. These devices are attached to the processor and main storage via the I/O channel with the channel directing the flow of information. The I/O channel can accommodate a maximum of 256 addressable devices. The general data flow is shown in Figure 4-1.

The channel supports three basic types of operations:

- **Direct Program Control (DPC) Operations** — An immediate data transfer is made between main storage and the device for each Operate I/O instruction. The data may consist of one byte or one word. The operation may or may not terminate with an interrupt.

- **Cycle Steal Operations** — An Operate I/O instruction can initiate cycle-stealing data transfers of up to 65,535 bytes between main storage and the device. Cycle steal operations are overlapped with processing operations. Word or byte transfers, DCB chaining, burst mode, and program controlled interrupt can be supported. All cycle stealing operations terminate with an interrupt.

- **Interrupt Servicing** — Four preemptive priority interrupt levels are available to facilitate device service. The device interrupt level is assignable by the program. In addition, the device interrupt capability may be masked under program control. Interrupt requests, along with cycle steal requests, are presented and polled concurrently with DPC and cycle-steal data transfers.

The channel provides comprehensive error checking including time-outs, sequence checking, and parity checking. Error, exception, and status reporting are facilitated by: (1) recording condition codes in the processor during execution of Operate I/O instructions, and (2) recording condition codes and an Interrupt Information Byte (IIB) in the processor during interrupt acceptance. Additional status words may be used by the device as necessary to describe its status (see I/O Condition Codes and Status Information in this chapter).
Operate I/O Instruction

The Operate I/O instruction initiates all I/O operations from the processor. It is a privileged instruction and is independent of specific I/O parameters. The generated effective address points to an immediate device control block (IDCB) in main storage. The IDCB consists of two words that contain an I/O command, a device address, and an immediate data field. For DPC operations, the immediate data field is used as a device data word. For cycle steal operations, the immediate data field points to a device control block (DCB) that provides additional information needed for the operation. For more details of the Operate I/O Instruction refer to Chapter 6.

Operate I/O Instruction

\[
\begin{array}{c|c|c}
0 & 1 & 1 & 0 & 1 & 0 & 0 & | & R2 & \ast & 1 & 1 & 0 & 0 & \text{Address} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
\text{IDCB} & \text{Command} & \text{Device address} & \text{Immediate data field} \\
0 & 7 & 8 & 15 & 16 & 31 \\
\end{array}
\]

*Indirect addressing bit
Immediate Device Control Block (IDCB)

The location in storage specified by the Operate I/O instruction contains the first word of the IDCB. The IDCB contains an I/O command that describes the specific nature of the I/O operation. This command is used by the channel for execution of the operation. The IDCB must always be on a word address boundary and has the following format:

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7-8-15</td>
</tr>
</tbody>
</table>

Immediate data field

| 16-31         |

Command field (bits 0–7)

Bit 0: Channel directed. If this bit is equal to one, the I/O command is directed to the channel rather than to a specific device. The Halt I/O command is the only valid channel directed command. Any other command with bit 0 set to one causes a command reject exception condition.

Bit 1: Read/Write. If this bit is equal to one, the data contained in the immediate field is transferred to the addressed I/O device. If this bit is equal to zero, the immediate field contains the data received from the I/O device at the conclusion of the I/O instruction.

Bits 2–3: Function. This field specifies the general type of I/O operation to be performed (see Figure 4-2).

Bits 4–7: Modifier. This field contains four bits for further specification of a function, if required (see Figure 4-2).

Device address field (bits 8–15)

This byte contains the I/O device address. The address range is 00 through FF (hex).

Immediate data field (bits 16–31)

This field contains a device data word for DPC operations. It contains the address of a device control block for cycle steal operations.

Figure 4-2 shows the relationship of the IDCB and the Operate I/O instruction. It also contains a chart of the various I/O commands. The Start command and the Start Cycle Steal Status command are used to initiate cycle steal operations. The remaining commands are used for DPC operations only.
### IDCB (Immediate Device Control Block)

<table>
<thead>
<tr>
<th>Command</th>
<th>Device address</th>
<th>Immediate field</th>
<th>Hex Specific command</th>
<th>Type of operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 00 Read</td>
<td>XXXX</td>
<td>0X</td>
<td>Read</td>
<td>DPC</td>
</tr>
<tr>
<td>0 0 01 Read</td>
<td>XXXX</td>
<td>1X</td>
<td>Read</td>
<td>DPC</td>
</tr>
<tr>
<td>0 0 10 Read status</td>
<td>0000</td>
<td>20</td>
<td>Read ID</td>
<td>DPC</td>
</tr>
<tr>
<td>0 0 10 Read status</td>
<td>XXXX</td>
<td>2X</td>
<td>Read status</td>
<td>DPC</td>
</tr>
<tr>
<td>0 0 11</td>
<td></td>
<td>3X</td>
<td>Unused***</td>
<td>Unused</td>
</tr>
<tr>
<td>0 1 00 Write</td>
<td>XXXX</td>
<td>4X</td>
<td>Write</td>
<td>DPC</td>
</tr>
<tr>
<td>0 1 01 Write</td>
<td>XXXX</td>
<td>5X</td>
<td>Write</td>
<td>DPC</td>
</tr>
<tr>
<td>0 1 10 Control</td>
<td>0000</td>
<td>60</td>
<td>Prepare</td>
<td>DPC</td>
</tr>
<tr>
<td>0 1 10 Control</td>
<td>XXXX</td>
<td>6X</td>
<td>Control</td>
<td>DPC</td>
</tr>
<tr>
<td>0 1 10 Control</td>
<td>1111</td>
<td>6F</td>
<td>Device reset</td>
<td>DPC</td>
</tr>
<tr>
<td>0 1 11 Start</td>
<td>XXXX</td>
<td>7X</td>
<td>Start</td>
<td>Cycle steal</td>
</tr>
<tr>
<td>0 1 11 Start</td>
<td>1111</td>
<td>7E</td>
<td>Start cycle steal status</td>
<td>Cycle steal</td>
</tr>
<tr>
<td>1 1 11 Channel</td>
<td>0000</td>
<td>F0</td>
<td>Halt I/O</td>
<td>Channel</td>
</tr>
</tbody>
</table>

*Indirect addressing bit,

**Modifier XXXX is device dependent. Other modifiers are system defined.

***To avoid future code obsolescence, this command format must not be used.

Figure 4-2. IDCB and I/O commands
Device Control Block (DCB)

This section describes the device control block that is used for a cycle steal operation. The actual cycle steal operation is explained in a later section of this chapter. The DCB is an eight-word control block residing in the supervisor area of main storage. It contains the specific parameters of a cycle steal operation. The device fetches the DCB using the cycle steal mechanism. The format of the DCB is shown in Figure 4-3.

The DCB words have the following meanings:

**Control word**

- **Bit 0** Chaining flag. If this bit is equal to one, a DCB chaining operation is indicated.
- **Bit 1** Programmed controlled interrupt (PCI). If this bit is equal to one, the device presents a programmed controlled interrupt (PCI) at the completion of the DCB fetch.
- **Bit 2** Input flag. The setting of this bit tells the device the direction of data transfer.
  - 0 = Output (main storage to device)
  - 1 = Input (device to main storage)
  For bidirectional data transfers under one DCB operation, this bit must be set to one. For control operations involving no data transfer, this bit must be set to zero.
- **Bit 3** Reserved. This bit must be set to zero to avoid future code obsolescence.
- **Bit 4** Suppress exception (SE). If this bit is equal to one, the device is allowed to suppress the reporting of certain exception conditions. The device can then take alternative action depending on the condition.
- **Bits 5–7** Cycle steal address key. Not used on the 4953 Processor.
- **Bits 8–15** Modifier. These are device dependent bits with one exception. When a device uses burst mode, it is specified in bit 15. These bits may be used for functions that are unique to a particular device.

*Chaining, PCI, and SE are device options that are available on a device feature basis. Any bit not used by the device should be set to zero although it is not checked by the device. Refer to the Cycle-Steal Device Options section of this chapter.

**Device Parameter Words 1–2**

These parameter words are device-dependent control words and are implemented as required. Refer to the individual device publications for definition.

**Device Parameter Word 3**

When PCI is specified, the high-order byte (bits 0–7) of this word is used for a DCB identifier. The device places the identifier in the interrupt information byte when the PCI is processed. The low-order byte (bits 8–15) is always device dependent. The high-order byte is device dependent when PCI is not specified.
Device Parameter Word 4
If suppress exception (SE) is used by a device, this word specifies a 16-bit main storage address called the status address. This address points to a residual status block that is stored by the device following completion of the DCB operation.

If suppress exception is not used by a device, a residual status block is not stored. In this case, parameter word 4 is device dependent. Refer to Cycle-Steal Device Option in this chapter.

Device Parameter Word 5
If the DCB chaining bit (bit 0 of the control word) is equal to one, this word specifies a 16-bit main storage address of the next DCB in the chain. If chaining is not indicated, this parameter word is device dependent.

Count
The count word contains a 16-bit unsigned integer representing the number of data bytes to be transferred for the current DCB. Count is specified in bytes with a range of 0 through 65,535. The count specification must be even for word-only devices.

Data Address
This word contains the starting main storage address for the data transfer.

Programming Considerations When Using the DCB
1. Only those words required for the cycle stealing operation are fetched by the device and they may be fetched in any order. Contents of the words must be specified correctly; if not, the device records a DCB specification check in the interrupt status byte and terminates the cycle steal operation with an exception interrupt.

2. The DCB address (in the DCB), the chain address, and the status address must be even (word boundary). If the DCB address is odd, the device records a command reject condition code and terminates the cycle steal operation. An odd chain address or status address results in a DCB specification check.

Note. Condition code and status recording are explained in detail in a separate section of this chapter.

I/O Commands
This section describes each I/O command and shows the related IDCB. The command field (bits 0–7) of the IDCB contains the binary value of the command. An X in this field means the value is device dependent.

Read

This command transfers a word or byte from the address of the device to the data word of the IDCB. If a single byte is transferred, it is placed in bits 24–31 of the data word with bits 16–23 set to zeros. Correct parity is always maintained and checked for both bytes on the I/O channel. The individual devices may use either the 0X or 1X type of read command. The two commands operate the same in the channel.

Read ID

This command transfers an identification (ID) word from the device to the data word of the IDCB. The device identification word contains physical information about the device and may be used to determine the devices that are attached to the system. This word is not related to the interrupt ID word associated with interrupt processing. The device ID word format is:

<table>
<thead>
<tr>
<th>Bits 0–12</th>
<th>Assigned code</th>
<th>C</th>
<th>CS</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unique identification code for the device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 13</td>
<td>Zero – not a controller device or the device does not report delayed command reject</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One – controller device or any device that reports delayed command reject</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 14</td>
<td>Zero – not a cycle steal device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One – cycle steal device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 15</td>
<td>Zero – IBM device</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One – OEM device</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Note.** A controller may control more than one I/O device and is not directly addressable, but is not transparent to software. That is, the controller may cause busy or exception conditions as opposed to those caused by an attached I/O device.

**Read Status**

IDCB (Immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 X X X X X X</td>
<td>X X X X X X X X X X</td>
</tr>
</tbody>
</table>

This command transfers a device status word from the device to the data word of the IDCB. Contents of the status word are device dependent.

**Write**

IDCB (Immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 X X X X</td>
<td>X X X X X X X X X</td>
</tr>
</tbody>
</table>

This command transfers a word or byte to the addressed device from the data word of the IDCB. The individual device may use either format of the command. If a single byte is to be transferred, it must be placed in bits 24–31 of the data word and bits 16–23 must be set to zero. A byte oriented device may ignore bits 16–23 (including the parity bit on the I/O channel) but these bits should be zeros to avoid future code obsolescence.

**Note.** Both bytes of the IDCB data word are fetched by the channel and placed on the I/O data bus (in good parity) even if not required by the device.

**Prepare**

IDCB (Immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0 0 0 0</td>
<td>X X X X X X X</td>
</tr>
</tbody>
</table>

This command transfers a word (to the addressed device) that controls the device interrupt parameters. The word is transferred from the immediate data field of the IDCB in the format shown. A priority interrupt level is assigned to the device by the level field. The I-bit (device mask) controls the device interrupt capability. If the I-bit equals 1, the device is allowed to interrupt. If the I-bit equals 0, the device cannot interrupt. See **Prepare I/O Device for Interrupt** in Chapter 3.

**Note.** The IBM 4953 Processor does not recognize a priority level other than 0–3. Lost interrupts result if a device is prepared for a level other than 0–3.

**Control**

IDCB (Immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 X X X</td>
<td>X X X X X X X</td>
</tr>
</tbody>
</table>

This command initiates a control action in the addressed device. A word, or byte, transfer from the data word of the IDCB to the addressed device may or may not occur, depending on device requirements. If a single byte is to be transferred it must be placed in bits 24–31 of the data word and bits 16–23 must be set to zero.

**Note.** Both bytes of the IDCB data word are fetched by the channel and placed on the I/O data bus (in good parity) even if not required by the device.
Device Reset

IDCB (immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110111111X X X X X X X X</td>
<td>6F 7 8 00-FF</td>
</tr>
</tbody>
</table>

Immediate data field

<table>
<thead>
<tr>
<th></th>
<th>Zeros</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This command resets the addressed device. A pending interrupt from this device (or a busy condition) is cleared. The device mask (I-bit) is not changed. A device must always accept and execute this command. There is no change to the assigned priority level for the device. The residual address (device status) and output sensor points are not affected. Parity checking of the IDCB data word is not performed.

Start

IDCB (immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111111111X X X X X X X X</td>
<td>7X 7 8 00-FF</td>
</tr>
</tbody>
</table>

Immediate data field

<table>
<thead>
<tr>
<th></th>
<th>DCB address</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This command initiates a cycle steal operation for the addressed device. The second word of the IDCB is transferred to the device and contains a 16-bit logical address of a device control block (DCB). See Cycle Steal in this chapter.

Halt I/O

IDCB (immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>11110000</td>
<td>F0 7 8 15</td>
</tr>
</tbody>
</table>

Immediate data field

<table>
<thead>
<tr>
<th></th>
<th>DCB address</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is a channel directed command that causes a halt of all I/O activity on the I/O channel and resets all devices. No data is associated with this command. All pending device interrupts are cleared. Device priority-interrupt-level assignments and device masks (I-bits) are unchanged. The residual address (device status) and output sensor points are not affected.

Notes.
1. The channel is always able to accept and execute this command.
2. Halt I/O is the only valid channel directed command.
**DPC Operation**

A DPC operation causes an immediate transfer of data or control information to or from an I/O device. An Operate I/O instruction must be executed for each data transfer and causes the following events to occur (refer to Figure 4-4).

1. The Operate I/O instruction points to an IDCB in main storage. A
2. The I/O channel uses the IDCB to select the addressed device and to determine the operation to perform. B
3. The I/O channel sends data to the device from main storage, or from the device to main storage. C
4. The device sends an I/O instruction condition code to the level status register (LSR) in the processor. D

**Notes.**

1. The DPC operation may end with a priority interrupt if the device has this capability. Refer to I/O Interrupts in Chapter 3.
2. There are two types of condition codes: the first is an I/O instruction condition code and is available immediately after completion of an Operate I/O instruction; the second is an interrupt condition code and is presented upon acceptance of a priority interrupt. The code significance is different for the two cases. Refer to I/O Condition Codes and Status Information in this chapter.

---

**Figure 4-4. Direct program control I/O operation**
Cycle Steal

The cycle steal mechanism allows data service to or from an I/O device while the processor is processing instructions. This overlapped operation allows multiple data transfers to be started by one Operate I/O instruction. The processor executes the Operate I/O instruction, then continues processing instructions while the I/O device steals main storage data cycles when needed. The channel resolves contention among multiple devices requesting cycle steal transfers. The operation always ends with a priority interrupt from the device.

The cycle steal operation includes certain capabilities that are provided on a device feature basis:

1. Burst mode
2. DCB chaining
3. Programmed controlled interrupt (PCI)
4. Suppress exception (SE)
5. Storage addresses and data transfers by byte or word

See the Cycle-Steal Device Options section of this chapter for details of these facilities.

All cycle steal operations terminate with a priority interrupt, providing the device has executed a successful Prepare command, with the device mask (I-bit) enabled. If the device mask is disabled, the interrupt presentation is blocked and the device remains busy until (1) the condition is cleared by a reset, or (2) the proper Prepare command is executed.

All cycle steal operations are started by an Operate I/O instruction that points to an IDCBO. The immediate data field of the IDCBO contains the address of a device control block (DCB). The DCB is fetched by the device using the cycle-steal mechanism. Within the DCB are specific parameters of the cycle steal operation. See Device Control Block in this chapter.

There are two types of cycle steal commands:
- Start
- Start Cycle Steal Status.

Start Operation

A cycle steal operation begins after successful execution of the Start command. The IDCBO, pointed to by an Operate I/O instruction, has the format:

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 X X X X X X</td>
<td>X X X X X X X X</td>
</tr>
<tr>
<td>7 8</td>
<td>15</td>
</tr>
<tr>
<td>7X</td>
<td>00-FF</td>
</tr>
</tbody>
</table>

Immediate data field

| DCB address |
| 16 | 31 |

The command modifier (X) is device dependent. The DCB address always specifies a word boundary and is the starting storage address of the DCB. This address is used by the device to fetch the DCB, using the cycle steal mechanism.

A cycle steal operation is presented in the following chart. Use Figure 4-5 in conjunction with this chart. Condition codes used in the chart are fully explained in the section I/O Condition Codes and Status Information in this chapter.

Note. An I/O device must be properly prepared (using a Prepare command), before it is allowed to interrupt.

<table>
<thead>
<tr>
<th>Cycle steal major steps</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start cycle steal</td>
<td></td>
</tr>
<tr>
<td>1. Execute IO instruction.</td>
<td></td>
</tr>
<tr>
<td>2. IDCBO contains Start command and points to a DCB. The DCB address is sent to the device.</td>
<td></td>
</tr>
<tr>
<td>3. Device presents condition code 7 (bits 0-2 in the LSR).</td>
<td></td>
</tr>
<tr>
<td>Device fetches DCB</td>
<td></td>
</tr>
<tr>
<td>1. Device uses cycle steal mechanism to fetch DCB.</td>
<td></td>
</tr>
<tr>
<td>Data transfer</td>
<td></td>
</tr>
<tr>
<td>1. Data is transferred to or from the device in word or byte format.</td>
<td></td>
</tr>
<tr>
<td>2. Transfer continues until count in DCB is exhausted.</td>
<td></td>
</tr>
<tr>
<td>Termination (no error condition)</td>
<td></td>
</tr>
<tr>
<td>1. Device presents interrupt request.</td>
<td></td>
</tr>
<tr>
<td>2. Channel polls I/O attachment feature and accepts request.</td>
<td></td>
</tr>
<tr>
<td>3. Device sends interrupt ID word and interrupt condition code 3 (device end).</td>
<td></td>
</tr>
<tr>
<td>Termination (Exception condition)</td>
<td></td>
</tr>
<tr>
<td>1. Device presents interrupt request.</td>
<td></td>
</tr>
<tr>
<td>2. Channel polls I/O attachment feature and accepts request.</td>
<td></td>
</tr>
<tr>
<td>3. Device sends interrupt ID word and interrupt condition code 2 (exception).</td>
<td></td>
</tr>
</tbody>
</table>

Note. Other events that might occur during the cycle steal operation are:

Chaining

1. Device completes the current DCB operation but does not present an interrupt request.
2. Device fetches next DCB in the chain.

Program Controlled interrupt

1. Device fetches DCB (PCI bit = 1).
2. Device initiates an interrupt and sends an interrupt ID word and interrupt condition code 1 (PCI).

Suppress exception

1. Device completes current operation.
2. Device stores status at the main storage location defined by DCB parameter word 4.
Operate I/O Instruction

\[
\begin{array}{cccc}
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 \\
0 & 0 & \times & 1 \\
1 & 0 \ 0 & \text{Address} \\
\end{array}
\]

Effective address

IDCB

Command | Device address | DCB address
---------|--------------|-------------
0200     | 0 7 8 15 16  | 0500

LSR

0 2 3 15

Data area

Chained DCB

*Indirect addressing bit

Figure 4-5. Example of cycle steal control information
Start Cycle Steal Status Operation

The purpose of this operation is to obtain data from the device if the previous cycle steal operation terminates due to an error or exception condition. The operation is initiated by a Start Cycle Steal Status command. The IDCB format is:

IDCB (immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111111</td>
<td>X X X X X X X X</td>
</tr>
<tr>
<td>7F</td>
<td>00-FF</td>
</tr>
</tbody>
</table>

Immediate data field

<table>
<thead>
<tr>
<th>DCB address</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-31</td>
</tr>
</tbody>
</table>

This command uses a special DCB format with some words and fields to set to zeros (see Figure 4-6).

Programming Note.

Concerning the DCB for the start cycle steal status operation:

1. Bits designated as zero are not checked by hardware (see Figure 4-6).
2. The count is specified in bytes.
3. The maximum count is device dependent.
4. The validity of a count value less than the maximum value is device dependent.
5. If the maximum count is exceeded, or a count value is specified that indicates the partial storing of a word length parameter, the device records a DCB specification check in the ISB and terminates the operation.
6. An odd data address also results in a DCB specification check.

Data is transferred to main storage starting at the data address specified in the DCB. This data consists of residual parameters and device dependent status information and has the following format:

<table>
<thead>
<tr>
<th>Word 0</th>
<th>Word 1</th>
<th>Word 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Residual address</td>
<td>Device cycle steal status word 1</td>
<td>Device dependent status word</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Residual Address. This word contains the main storage address of the last attempted cycle steal transfer associated with a Start command. It may be a data address, a DCB address, or a residual-status-block address. It is updated to the current cycle-steal storage address upon execution of cycle steal transfers. For word transfers, the residual address points to the high-order byte of the word. If an error occurs during a start cycle steal status operation, this address (as contained within the device) is not altered. Device reset, Halt I/O, machine check, and system reset have no effect on the residual address in the device. It is cleared by a power-on reset. Following a power-on reset the residual address is:

- 0000 (Hex) for a byte-oriented device.
- 0001 (Hex) for a word-oriented device.
Device Cycle-Steal-Status Word 1. This word contains the residual byte count of the previous cycle steal operation associated with a start command. The byte count is initialized by the count field of a DCB associated with a Start command, and is updated as each byte of data is successfully transferred via a cycle steal operation. It is not updated by cycle-steal transfers into the residual status block. The residual byte count is not altered if an error occurs during a start cycle steal status operation. It is reset by (1) power-on reset, (2) system reset, (3) device reset, (4) Halt I/O, and (5) machine check condition.

Note. The contents of the device cycle-steal-status word 1 are device dependent if the device does not: (1) implement suppress exception (SE), or (2) store a residual byte count as part of its cycle-steal status.

Device Dependent Status Words. The number and contents of these words are specified by the individual device. Three conditions can cause bits to be set in the device dependent status words (refer to individual device publications).

1. Execution of an I/O command that causes an exception interrupt.
2. Asynchronous conditions in the device that indicate an error, exception, or a state condition.
3. As defined by the individual device.

The bits are reset as follows:

1. For the first condition listed above, the bits are reset by the acceptance of the next I/O command (except Start Cycle Steal Status) following the exception interrupt. These bits are also reset by a power-on reset, system reset, or execution of a Halt I/O command.
2. For the second condition, the bits are reset on a device dependent basis.
3. For the third condition, the bits are reset as defined by the individual device.

Cycle-steal Device Options

The I/O channel supports operations such as burst mode and chaining when required by individual devices. Bits in the DCB control word are used to activate these operations. Refer to the individual device publications for the device options used. The following sections explain the operations.

Burst Mode

Burst mode, when used by a device, is specified in bit 15 of the DCB control word. If bit 15 is equal to one, the transfer of data takes place in burst mode. This mode dedicates the I/O channel to the device until the last data transfer for the DCB is completed. Cycle steal interleave, by other devices, is prevented. Burst mode also prevents any priority interrupt request from being accepted by the processor.

The maximum burst rate for the 4953 channel is 1.332 megabytes per second.

Chaining

The purpose of chaining is to allow the programmer to sequence an I/O device through a set of operations by using a chain of DCBs. Bit 0 of the DCB control word (when set to one) indicates a chaining operation. This means that the chained DCB, fetched by the device, is interpreted as a new operation (or function) to be performed. The DCB may be equal to, but not a continuation of, the operation specified by the previous DCB.

When the current DCB indicates a chaining operation, device parameter word 5 of the DCB must contain a main storage address that points to the next DCB in the chain. The device completes the current operation but does not present an interrupt request (excluding PCI) to the processor. Instead, the device fetches the next DCB in the chain and continues operation.

Note. The chaining operation has no effect on programmed controlled interrupt (PCI). These interrupts, when specified in the DCB, still occur at the completion of the DCB fetch operation.

Programmed Controlled Interrupt (PCI)

Bit 1 of the DCB control word (when set to one) tells the device to present a PCI to the processor at the completion of the DCB fetch prior to data transfer.

When the PCI is serviced, a DCB identifier byte is returned to the processor in the interrupt information byte (IIB). Refer to DCB device parameter word 3 in this chapter. Two conditions should be noted by the programmer:

1. Chaining and data transfers associated with the DCB may commence even if the PCI is pending.
2. If the PCI is pending when the device encounters the next interrupt causing condition, the PCI condition is discarded by the device and replaced with the new interrupt condition.

Suppress Exception (SE)

When a device uses this option it is allowed to suppress the reporting of certain exception conditions that would normally cause an exception interrupt. The device is then allowed to take alternative action depending on the condition. The suppressed exception conditions are reported to the programmer as status information upon completion of the operation. Refer to a subsequent section, Suppression of Exceptions, for details of the various actions a device might take.

The suppress exception option also provides for automatic logging of status information (including suppressed exceptions) into main storage. When the SE bit for a DCB is set to one, the device always stores a residual status block into main storage after successful completion of the data transfer for the DCB. Device parameter word 4 of the DCB...
must be used to specify the starting main storage address for the residual status block. Note that a residual status block is stored even if there are no exception conditions to be suppressed.

The following section shows the residual status block that is stored.

Residual Status Block
The residual status block is stored into main storage at the location pointed to by the status address (DCB word 4). The size of a residual status block is fixed for each device with a limit of 8 words total. The format is:

<table>
<thead>
<tr>
<th>Word</th>
<th>residual count</th>
<th>EOC</th>
<th>reserved</th>
<th>status flags</th>
<th>NE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>7 8</td>
<td>14 15</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>maximum</td>
<td>1 of 8 words</td>
</tr>
<tr>
<td></td>
<td>device dependent status</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Word 0 Contains the residual byte count associated with the DCB.
Word 1 EOC is the End of Chain bit and is set to one for all conditions that would terminate a chaining operation. NE is the No Exception bit and is set to one when the operation is completed and no exceptions are reported. The Status Flags are device dependent flags that indicate suppressed exception conditions.

Any additional words are device dependent as to number and content. Refer to the individual device publications for the additional status information and, also, the bit significance of the status flags.

Suppression of Exceptions
An exception condition can be suppressed by a device only when it occurs during a data transfer operation. It cannot be suppressed if it occurs during (1) a DCB fetch, (2) storing of a residual status block, or (3) a cycle steal status operation. A second requirement of a suppressible exception is that the device be capable of continuing operation in a normal and predictable manner after occurrence of the exception. If these conditions are not met, the exception condition causes an exception interrupt. When a suppressible exception is encountered, the device initiates one of a possible three types of action depending on the device and the exception condition. Note that the number of action types used by a device and the suppressible exceptions for each type are a device specification. Refer to the individual device publication. The three action types are:

1. Suppress Exception and Continue. The exception condition occurs but data transfer is allowed to proceed. At the completion of the data transfer (defined by the DCB) a residual status block is stored with word one set as follows:
   - A status flag for this exception is set to one.
   - If the DCB specifies chaining, then the EOC bit is set to zero. Otherwise, it is set to one.
   - The NE bit is set to zero.

   The device may then continue with the next DCB if chaining is specified.

2. Suppress Exception and Terminate Data Transfer. Upon detecting the exception condition, the device terminates the data transfer for this DCB. It then stores a residual status block containing:
   - A status flag for the exception condition.
   - EOC bit set to zero, if chaining. Otherwise, set to one.
   - NE bit set to zero.

   The device may then continue with the next DCB if chaining is specified.

   Programming Note. For some devices, the most common exception condition of this type is incorrect length record (ILR). For example, the data transfer is completed prior to the count reaching zero.

   In certain communications devices a short ILR is considered normal operation. When a short ILR occurs in this type device, the residual byte count is sufficient to indicate the condition; therefore, the NE bit may be set to indicate no exception.

3. Suppress Exception and Terminate Chain. Upon detecting this exception condition, the device terminates the data transfer for this DCB. It ignores any commands specifying further chaining.

   The device stores a residual status block containing:
   - A status flag for the exception condition
   - EOC bit set to one
   - NE bit set to zero.

   The device then presents a device end interrupt. Refer to Interrupt Condition Codes in a subsequent section of this chapter.

   Programming Note. In certain communication devices a change-of-direction character is considered normal operation. When a change-of-direction character occurs in this type device, the EOC bit is sufficient to indicate the condition; therefore, the NE bit may be set to indicate no exception.
Priority of Suppress Exception Actions. Multiple exceptions that are suppressible can occur during an operation. They are noted in the residual status block by setting multiple status flags. The type of action taken by a device depends on the exception/action combination with highest priority. The priority sequence is type 3, type 2, and type 1 with type 3 having the highest priority.

Cycle-steal Termination Conditions
The following chart shows the action that occurs at the end of a DCB operation depending on the function specified and the exception conditions encountered:

<table>
<thead>
<tr>
<th>CHN</th>
<th>SE</th>
<th>Suppressible exception</th>
<th>Non-Suppressible exception</th>
<th>No exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>I (XCT)</td>
<td>I (XCT)</td>
<td>I (DE)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>I (PDE)</td>
<td>I (XCT)</td>
<td>I (DE)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>I (XCT)</td>
<td>I (XCT)</td>
<td>CC</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*I (PDE)/CC</td>
<td>I (XCT)</td>
<td>I (DE)</td>
</tr>
</tbody>
</table>

CC – DCB chaining
CHN – Chaining flag (bit 0 of the DCB control word)
I (DE) – Device end interrupt
I (PDE) – Permissive device end interrupt (see device end interrupt)
I (XCT) – Exception interrupt

SE – Suppress exception (bit 4 of the DCB control word)

*Dependent on the specific exception condition in the individual device.

I/O Condition Codes and Status Information
Each time an Operate I/O instruction is issued, the device, controller, or channel immediately reports to the processor one of seven condition codes pertaining to execution of the I/O command. These codes are called I/O instruction condition codes. Three bits are used to encode a condition code value (range 0 through 7). The bits are recorded in the even, carry, and overflow positions of the LSR and may be interrogated by specific instructions such as Branch on Condition Code and Branch on Not Condition Code. (See BCC and BNCC in Chapter 6.)

For interrupting devices, condition codes are also reported during a priority interrupt. These codes are called Interrupt condition codes and pertain to operations that continue beyond execution of the Operate I/O instruction (such as cycle stealing of data). The interrupt condition codes are recorded in the LSR and interrogated in the same manner as the I/O instruction codes. Along with the interrupt condition code, the device also transfers an interrupt ID word to the processor. Bits 0 through 7 of the interrupt ID word contain status information related to the interrupt processing and are called the interrupt information byte (see Interrupt ID Word in this chapter).

Figure 4-7 presents an overall view of condition code reporting along with status information. Details of the condition codes and status information are discussed in the following sections. Note that there are two unique sets of condition codes (I/O instruction and interrupt) and that most status information is device dependent.
Figure 4-7. Condition codes, status words, and status bytes received from a device (Part 1)
Cycle steal

Residual parameters and device dependent status
Returned by the device if this is a Start Cycle Steal Status operation

Residual status block
Stored into main storage if the device uses SE and the SE bit is set to one

DCB word 7
data address

residual address
cycle steal status word 1
device dependent status
device dependent status

DCB word 4
status address

residual byte count
EOC reserved status flags NE
device dependent status

Figure 4-7. Condition codes, status words, and status bytes received from a device (Part 2)
The device reports an interrupt condition code.

Interrupt ID word
Presented by the device and placed in register 7 of the interrupted-to level.

I/O Interrupt

LSR bits 0–2

<table>
<thead>
<tr>
<th>CC</th>
<th>0</th>
<th>Controller end</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>PCI</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Exception</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Device end</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Attention</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Attention and PCI</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Attention and exception</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Attention and device end</td>
</tr>
</tbody>
</table>

CC ≠ 2 or 6 (DPC or cycle steal)

<table>
<thead>
<tr>
<th>IIB</th>
<th>device address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7 8 15</td>
</tr>
</tbody>
</table>

Bits 0–7 Device dependent status or special meaning for CC2, CC3, and CC7.

CC ≠ 2 or 6 (DPC)

<table>
<thead>
<tr>
<th>ISB</th>
<th>device address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7 8 15</td>
</tr>
</tbody>
</table>

Bit 0 Device status available*
1 Delayed command reject
2–7 Device dependent

CC ≠ 2 or 6 (cycle steal)

<table>
<thead>
<tr>
<th>ISB</th>
<th>device address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7 8 15</td>
</tr>
</tbody>
</table>

Bit 0 Device status available*
1 Delayed command reject
2 Incorrect length record
3 DCB specification check
4 Storage data check
5 Invalid storage address
6 Not used
7 Interface data check

*The available status is returned by the device when the following commands are used:
Read Status–DPC
Start Cycle Steal Status–cycle steal

Figure 4-7. Condition codes, status words, and status bytes received from a device (Part 3)
**IO Instruction Condition Codes**

These codes are reported during execution of an Operate I/O instruction.

<table>
<thead>
<tr>
<th>Condition code (CC) value</th>
<th>LSR position</th>
<th>Over-</th>
<th>Reported by</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Even Carry flow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>channel</td>
<td>Device not attached</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 0</td>
<td>device</td>
<td>Busy</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 1 0 0</td>
<td>chan/dev</td>
<td>Busy after reset</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 1 1 1</td>
<td>chan/dev</td>
<td>Command reject</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 0</td>
<td>device</td>
<td>Intervention required</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 1</td>
<td>chan/dev</td>
<td>Interface data check</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 1 0 1</td>
<td>controller</td>
<td>Controller busy</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 1 1 1</td>
<td>chan/dev</td>
<td>Satisfactory</td>
<td></td>
</tr>
</tbody>
</table>

CC=0

*Device not attached.* Reported by the channel when the addressed device is not attached to the system.

CC=1

*Busy.* Reported by the device when it is unable to execute a command because it is in the busy state.

The device enters the busy state upon acceptance of a command that requires an interrupt for termination. It exits the busy state when the processor accepts the interrupt. Certain devices also enter the busy state when an external event occurs that results in an interrupt. When this condition code is reported, a subsequent priority interrupt from the addressed device always occurs.

CC=2

*Busy after reset.* Reported by the device when it is unable to execute a command because of a reset and the device has not had sufficient time to return to the quiescent state. No interrupt occurs to indicate termination of this condition.

CC=3

*Command Reject.* Reported by the device or the channel when:

1. A command is issued (in the IDCB) that is outside the device command set.
2. The device is in an improper state to execute the command.
3. The IDCB contains an incorrect parameter. For example: an odd byte DCB address, or an incorrect function/modifier combination.

When a cycle- steal device reports command reject, it does not fetch the DCB.

CC=4

*Intervention required.* Reported by the device when it is unable to execute a command due to a condition requiring manual intervention to correct.

CC=5

*Interface data check.* Reported by the device or the channel when a parity error is detected on the I/O data bus during a data transfer.

CC=6

*Controller busy.* This condition is reported by a device controller, not the addressed device, when the controller is busy. It is reported only by controllers that have two or more devices attached (each device having a unique address). When this condition code is reported, a subsequent controller-end interrupt always occurs.

CC=7

*Satisfactory.* Reported by the device on the channel when it accepts the command.

These condition codes are mutually exclusive and have a priority sequence. That is, beginning with CC=7, each successive condition code through CC=0 takes precedence over the previous code. For example, if a device cannot accept a command because it is busy, it reports CC=1, irrespective of error conditions encountered.

*Note.* The only exception is CC=6 (controller busy). This condition code may have a variable priority depending on the particular controller.

**Interrupt Condition Codes**

These condition codes are reported by the device or controller during priority interrupt acceptance.

<table>
<thead>
<tr>
<th>Condition code (CC) value</th>
<th>LSR position</th>
<th>Over-</th>
<th>Reported by</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Even Carry flow</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>controller</td>
<td>Controller end</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 0</td>
<td>device</td>
<td>Program controlled interrupt (PCI)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 1 0 0</td>
<td>device</td>
<td>Exception</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 1 1 1</td>
<td>device</td>
<td>Device end</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 0</td>
<td>device</td>
<td>Attention</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 0</td>
<td>device</td>
<td>Attention and PCI</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 1 0 1</td>
<td>device</td>
<td>Attention and device exception</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 1 1 1</td>
<td>device</td>
<td>Attention and device end</td>
<td></td>
</tr>
</tbody>
</table>

CC=0

*Controller end.* Reported by a controller when controller busy (I/O instruction condition code) has been previously reported one or more times. It signifies that the controller is now free to accept I/O commands for devices under its control. The device address reported with controller end is always the lowest address (numerical value) of the group of devices serviced by the controller. The interrupt information byte, in the interrupt ID word, is set to zero.

CC=1

*Program controlled interrupt.* Reported when the interrupt indicates that a DCB with the PCI bit set to one has been transferred by cycle steal to the device and no error or exception condition has occurred. The device places a DCB identifier into the interrupt information byte.

CC=2

*Exception.* Reported when an error or exception condition is associated with the interrupt. The condition is described in the interrupt status byte (ISB) or in device dependent status words.

CC=3

*Device end.* Reported when no error, exception, or attention condition has occurred during the I/O operation, and the interrupt is not the result of a PCI. For example: an operation has terminated normally.

*Note.* If the device has come to a normal end while using suppress exception (SE bit set to one) and an exception was suppressed since the last Start command, then bit zero of the interrupt status byte is set to one. The condition is called permissive device end (PDE) and indicates that errors or exceptions have been suppressed. Related status information is contained in the residual status block.
**Interrupt Status Byte (ISB)**

The ISB is a special format of the interrupt information byte (IIB) and contains detailed information on the nature of the interrupt. The ISB is reported only for error or exception conditions (interrupt condition codes 2 or 6). The ISB bits are normally set as a result of:

1. Status errors that occur during a DPC operation that cannot be indicated via a condition code.
2. Status errors that occur during a cycle steal operation.

The ISB is never reported as zero unless the condition code presentation of 2 or 6 is singular in meaning for devices that do not cycle steal. After the processor has accepted the interrupt request, the device resets the ISB.

Bits 0–7 of the two special formats are explained in the following sections.

**ISB (devices that do not cycle steal):**

- **Bit 0**  
  *Device dependent status available.* This bit set to one signifies that additional status information is available from the device. The information content and method of reading is described in the individual device publications.

- **Bit 1**  
  *Delayed Command reject.* This bit is set to one if the device cannot execute the command (specified in the DCB) due to an incorrect parameter in the DCB, or it cannot execute the command due to its present state. For example: (1) the DCB specifies an incorrect function/modifier combination, or (2) the device is temporarily not ready. The operation in progress is terminated. Command reject is set in the ISB only if the device cannot report IO instruction condition codes for the condition.

**ISB (cycle stealing device):**

- **Bit 0**  
  *Device dependent status available.* This bit, when set to one, signifies that: (1) additional status information is available from the device, or (2) the device is in an improper state to execute a function specified by a DCB.

  The operation is terminated. The content and method of reading the additional status information is described in the individual device publications.

**Note.** When bit 0 of the ISB is equal to one and bits 2–7 are zeros, the contents of the residual-address word (cycle steal status) are defined by the device.

- **Bit 1**  
  *Delayed command reject.* This bit is set to one if the device cannot execute the command due to one of the following conditions:

  1. The DCB contains an incorrect parameter. Examples are (a) an odd-byte DCB address, or (b) an incorrect function/modifier combination.

  2. The present state of the device, such as a not ready condition, prevents execution of an I/O command specified in the DCB.

  Delayed command reject is set in the ISB only if the device cannot report IO instruction condition codes for the condition. The operation is terminated. The DCB is not fetched.
Bit 2  Incorrect length record. This bit is set to one when the device encounters a mismatch between byte count and actual record length after beginning execution of the DCB. For example: the byte count is reduced to zero (with chaining flag off) and no end of record encountered. Incorrect length record is not reported when the SE bit in the control word is set to one. Reporting of incorrect length record is a device dependent feature and may be implemented regardless of the suppress exception feature. The operation is terminated.

Bit 3  DCB specification check. This bit is set to one when the device cannot execute a command due to an incorrect parameter specification in the DCB. Examples are (1) an odd-byte DCB chaining or status address, (2) the byte count is odd for a word-only device, (3) an odd-byte data address for a word-only device, (4) an invalid command or invalid bit settings in the control word, or (5) an incorrect count.

The operation is terminated.

Bit 4  Storage data check. This error condition applies to cycle steal output operations only. If the bit is set to one, it indicates that the main storage location accessed during the current output cycle contained bad parity. Parity in main storage is not corrected. The device terminates the operation. The bad parity data is not transferred to the I/O data bus. No machine check condition occurs. See Figure 4-8 for other bits that may be present.

Bit 5  Invalid storage address. When set to one, this bit indicates that, during a cycle steal operation, the device has presented a main storage address that is outside the storage size of the system.

Invalid storage address can occur on a data transfer or on a DCB fetch operation. In either case, the cycle steal operation is terminated. See Figure 4-8 for other bits that may be present.

Bit 6  Not used.

Bit 7  Interface data check. This bit set to one indicates that a parity error has been detected on the I/O data bus during a cycle steal data transfer. The condition may be detected by the channel or the I/O device. In either case, the operation is terminated. See Figure 4-8 for other bits that may be present.

<table>
<thead>
<tr>
<th>Conditions</th>
<th>I/O operation</th>
<th>Invalid storage address</th>
<th>Incorrect data parity</th>
<th>Bit results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Write</td>
<td>No</td>
<td>No</td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>Yes</td>
<td>No</td>
<td>0 1 0</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>No</td>
<td>No</td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>Yes</td>
<td>No</td>
<td>0 1 0</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>No</td>
<td>Yes</td>
<td>0 0 1</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>Yes</td>
<td>Yes</td>
<td>0 1 1</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>No</td>
<td>Yes</td>
<td>1 0 1</td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>Yes</td>
<td>Yes</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

*This condition not possible.

Figure 4-8. Bit result chart
Chapter 5. Console

There are two configurations of consoles available for the IBM 4953 Processor. The Basic Console is standard, and remains with the processor. The Programmer Console is an optional feature that is added to the processor when the option is selected.

Configuration 1
Basic Console

Configuration 2
Basic Console and Programmer Console
Configuration 1 is primarily intended for those systems that are totally dedicated to a particular application, where operator intervention is not needed during the execution of the application.

Configuration 2 is aimed at operator oriented systems where various programs are entered and executed during the day. This type of environment requires a more versatile console arrangement for program and machine problem determination, and for manual alteration of data and programs in storage.

**Basic Console**

Each IBM 4953 Processor comes equipped with the standard Basic Console. The Basic Console provides the following capabilities:

- Power On/Off switch for the processor card file
- Load key for IPL (initial program load)
- Load, Wait, Run, and Power On indicators
- Mode switch to select: Diagnostic mode, Auto IPL, or Normal mode
- IPL source switch to select a primary or alternate IPL device.

**Keys and Switches**

A  Power On/Off  When this switch is placed in the On position, power is applied to the processor card file. After all power levels are up, the Power On indicator is turned on. When this switch is placed in the Off position, power is removed from the processor card file and the Power On indicator is turned off.

B  IPL Source  This switch selects the I/O device to be used for program loading. In the Primary position, the device that was pre-wired as the primary IPL device is selected. In the Alternate position, the device that was pre-wired as the alternate IPL device is selected.

C  Load  Pressing this key causes a system reset, then the initial program load (IPL) sequence is started. The Load indicator is turned on and remains on until the IPL sequence is completed. When the IPL is completed, instruction execution begins at location zero on level zero.

D  Mode  This switch has the following positions:
- Auto IPL – In this position, an IPL is initiated after a successful power-on sequence. Bit 13 of the PSW is set to indicate to the software that an automatic IPL was performed. In this mode STOP instructions are treated as no-ops.
- Normal – This position is for attended operation. In this mode STOP instructions are treated as no-ops.
- Diagnostic – This position has no function without the Programmer Console. This position places the processor in a diagnostic mode if the Programmer Console is attached. When the processor is in diagnostic mode, STOP instructions cause the processor to enter stop state.

**Indicators**

- E  Power On  On when the proper power levels are available to the system.
- F  Load  On when the machine is performing an initial program load (IPL).
- G  Wait  On when an instruction that exits the active level has been executed and no other levels or interrupts are pending.
- H  Run  On when the machine is executing instructions.
The Programmer Console

The Programmer Console is an optional feature that can be ordered with the IBM 4953 Processor or may be field installed at a later date. The Programmer Console provides the following capabilities:

- Start and stop the processor.
- Display or alter any storage location.
- System reset.
- Select any of the four interrupt levels for display or alter purposes.
- Display or alter the storage address register (SAR), instruction address register (IAR), console data buffer, or any general purpose register.
- Display but not alter the level status register (LSR), current instruction address register (CIAR), op register, or processor status word (PSW).
- Stop-on-address.
- Stop-on-error.
- Instruction step.
- Check restart.
- Request a console interrupt.
- Check indicator, on when a machine check or program check class interrupt has occurred.

The Programmer Console is touch sensitive with a tone generator providing an audio response tone whenever a key depression has been accepted and serviced by the processor.

**Console Display**

When the processor is in run state or wait state, the console data buffer is displayed in the data display indicators. The only exception to this is when in run state a Set Console Data Lights instruction writes a message to the data display. This message remains displayed until the processor enters stop state or the Data Buffer Key is pressed. When the Data Buffer Key is pressed, the console data buffer is again displayed in the data display indicators.

When the processor enters stop state, the IAR is displayed in the data display indicators. Any system resource that has a corresponding select key on the console can be displayed while in stop state. Once data has been entered into the console data buffer, it remains there until other data is entered. The console data buffer can be displayed at any time, during either run state, wait state, or stop state, by pressing the Data Buffer key.

After a power-on reset, the data display indicators are all set on, and the level indicators are set off.
**Indicators**

**A Data Display**
- When the processor is in run state, the console data buffer is displayed in the data display indicators.
- When the processor enters stop state, the IAR is displayed unless another system resource is selected.
- To display the contents of the console data buffer after a system resource has been displayed, press the Data Buffer key.

**B Check**
On when a machine check, program check, or power/thermal warning class interrupt has occurred while in process mode or in stop-on-error mode. The check indicator remains on until either the check condition is cleared, or any console key is pressed while in the stop state. The check condition is cleared by the Reset key, Load key, or the execution of a Copy Processor Status and Reset instruction (which resets the check bits). If a main storage display of a location causes a parity error, an invalid storage address, or a specification check, the check indicator is turned on, or appears to stay on.
**Combination Keys/Indicators**

There are nine combination key/indicators:

- Level 0, 1, 2, and 3
- Stop
- Stop On Address
- Instruct Step
- Check Restart
- Stop On Error

### Level 0–3

The current active level is always displayed by one of the level indicators. When in the stop state, pressing any of the level keys causes that level to be selected and the associated indicator is turned on.

### Stop

This indicator is on when the processor is in the stop state. Stop state is entered in the following ways:

- By pressing the Stop key.
  - In run state the current instruction is completed.
  - In wait state, stop state is entered directly.
- By execution of the Stop instruction (diagnostic mode only).
- When an address compare occurs in stop-on-address mode.
- When an error occurs in stop-on-error mode.
- By pressing the Reset key.
- When a power-on reset occurs.
- By selecting the Instruction step mode while in run state.
The Stop On Address key and the Instruct Step key are mutually exclusive. When one is pressed, the other is reset if it was on.

**Stop on Address**
This key places the processor in stop on address mode. Pressing the Stop On Address key a second time resets stop on address mode and turns off the indicator.

**Instruct Step**
Pressing the Instruct Step key places the processor in instruction step mode and turns the Instruct Step indicator on. The Stop On Address indicator is turned off if it was on.

If the processor is in run state, pressing this key causes the processor to enter stop state. Pressing the Instruct Step key a second time resets instruction step mode, the processor remains in stop state.

To operate in instruction step mode:
- Key the desired starting address and store into the IAR.
- Press the Instruct Step key.
- Press the Start key. The instruction located at the selected address is executed, the processor returns to stop state. The IAR is updated to the next instruction address, this address is displayed in the data display indicators.
- Each subsequent depression of the Start key causes one instruction to be executed and the IAR is updated to the next instruction address.

**Stop On Address Mode**
Processor must be in stop state to set the compare address.
1. Press Stop On Address Key.
2. Key in selected address.
3. Press Store Key. The selected address is placed in the stop on address buffer.
4. Press Start Key. Execution begins at current IAR address on the current level.

When the selected address is loaded into the IAR, the processor enters stop state. To exit stop state press the Start key; execution begins at the next sequential address.

**Note.** When running in Stop on Address Mode, instruction execution time is increased by 7.8 microseconds per instruction.
The Check Restart key and the Stop On Error key are mutually exclusive. When one is pressed, the other is reset if it was on.

**Check Restart**
Pressing this key places the processor in check restart mode. While in this mode, a program check, or machine check, or a power/thermal warning class interrupt causes the processor to be reset and execution to restart at address zero on level zero.

*Note.* The power/thermal warning class interrupt is controlled by the summary mask.

**Stop On Error**
Pressing the Stop On Error key places the processor in stop on error mode. Any program check, machine check, or power/thermal warning causes the processor to enter stop state. To determine the cause of the error, display the PSW (see table). To restart the processor, press the Reset key then the Start key. Pressing only the Start key allows the processor to proceed with the class interrupt as if stop mode has not occurred. Note that the check indicator may have been turned off while in stop state. After the class interrupt routine is completed, control may be returned to the instruction that caused the error and an attempt to re-execute the instruction may be made. Note that some instructions are not re-executable because operand registers or storage locations were changed before the instruction was terminated because of the initial error. In these cases, the operator must be familiar with the program because manual restoration of affected locations must be made before restart is attempted.

*Note.* The power/thermal warning class interrupt is controlled by the summary mask.

### Processor Status Word (PSW) Table

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Specification check</td>
<td>Program check</td>
</tr>
<tr>
<td>1</td>
<td>Invalid storage address</td>
<td>Program check</td>
</tr>
<tr>
<td>2</td>
<td>Privilege violate</td>
<td>Program check</td>
</tr>
<tr>
<td>3</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Invalid function (may be program check)</td>
<td>Soft exception</td>
</tr>
<tr>
<td>5</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Stack exception</td>
<td>Soft exception</td>
</tr>
<tr>
<td>7</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Storage parity check</td>
<td>Machine check</td>
</tr>
<tr>
<td>9</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CPU control check</td>
<td>Machine check</td>
</tr>
<tr>
<td>11</td>
<td>I/O check</td>
<td>Machine check</td>
</tr>
<tr>
<td>12</td>
<td>Sequence indicator</td>
<td>Status flag</td>
</tr>
<tr>
<td>13</td>
<td>Auto-IPL</td>
<td>Status flag</td>
</tr>
<tr>
<td>14</td>
<td>Not used</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Power/thermal warning</td>
<td>Power/Thermal</td>
</tr>
</tbody>
</table>

Bits not used are always zero.
**Keys and Switches**

**A  Reset**
This key initiates a system reset that performs the following functions:
- Interrupt mask set to all levels enabled.
- LSR on level zero – indicators set to zero, summary mask enabled, supervisor state and in-process flag turned on, trace disabled.
- LSRs for levels 1 – 3 set to zeros.
- PSW set to zero.
- SAR set to zeros.
- CIAR set to zeros.
- IAR on level zero – set to zeros.

After the system reset is completed, the processor is placed in the stop state with stop indicator on.

The following resources are not effected by system reset:
- General registers (all levels)
- IARs (levels 1 – 3)
- Main storage
- Console data buffer
- Stop on Address buffer.

**B  Store**
This key is effective only when the processor is in stop state. Pressing this key causes the last data entry to be stored in the last selected resource.

**C  Data Buffer**
Pressing this key causes the contents of the console data buffer to be displayed in the data display indicators.

**D  Console Interrupt**
The effect of this key depends on the state of the processor. If the processor is in the stop or load states, this key has no effect. If the processor is in the run or wait state and the summary mask is enabled, a console class interrupt occurs.

*Note.* If the summary mask is enabled by the program while the key is being activated, a console class interrupt occurs.

**E  Start**
Effective in stop state only. Stop state is exited and the processor resumes execution at the address in the IAR on the current level. If stop state was entered from system reset, execution begins at address zero, level zero. If stop state was entered from wait state, the processor returns to wait state.

*Note.* The Reset and Console Interrupt keys have an indication (+++) on the face of the keys. This signifies that additional pressure must be used to activate these keys. This is to minimize the possibility of the operator inadvertently activating these functions.
Pressing this key selects the processor status word. The contents of the PSW are displayed in the data display indicators. Data cannot be stored into the PSW from the console.

Pressing this key selects the Op register and displays the contents in the data display indicators. Data cannot be stored into the Op register from the console.

Pressing this key after entering stop state causes the address of the instruction just executed to be displayed. Data cannot be stored into the CIAR from the console.

Pressing this key while in stop state displays the contents of the storage address register. An address can be stored into the SAR to address main storage for display or store operations. Bit 15 of the SAR cannot be set from the console.

Pressing this key selects main storage as the facility to be accessed by the console. When this key is pressed, the contents of the main storage location addressed by the SAR is displayed in the data display indicators. Procedures for displaying and storing main storage are provided in subsequent sections of this chapter.
Level Dependent Keys

The following keys select registers that are duplicated in hardware for each of the four interrupt levels:

- LSR
- IAR
- General purpose registers 0–7

Pressing any of these keys, once a level has been selected, causes the contents of that register to be displayed in the data display indicators.

The level status register (LSR) is displayable only; data cannot be stored into this register.

The AKR key is not functional on the 4953 Processor and does not respond with an audio tone when pressed.

Bit 15 of the IARs cannot be changed from the console.

Pressing the Store key after selecting an LSR or AKR results in no action taken and no audio tone response.
Data Entry Keys
The sixteen data entry keys are used to enter data into the selected resource.

Example:
Data to be entered: F3A8

<table>
<thead>
<tr>
<th>Action</th>
<th>Data display indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Press data entry key F</td>
<td>0 1 2 3 4 5 6 7 8 9 10 12 13 14 15</td>
</tr>
<tr>
<td>Press data entry key 3</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>Press data entry key A</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>Press data entry key 8</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
</tbody>
</table>

Legend:
- Indicator on
- Indicator off
**Displaying Main Storage Locations**

- Processor must be in stop state.

1. Press the SAR key. The contents of the SAR are displayed in the data display indicators.
2. Key in the selected address (four hex characters). This address is displayed in the data display indicators.
3. Press the Store key. The address that is displayed is stored into the SAR.
4. Press the Main Storage key. The contents of the addressed storage location are displayed in the data display indicators. To display sequential main storage locations, continue pressing the Main Storage key. The storage address is incremented by +2 each time the Main Storage key is pressed, and the contents of the addressed location are displayed.

**Storing Into Main Storage**

- Processor must be in stop state.

1. Press the SAR key. The current contents of the SAR are displayed in the data display indicators.
2. Key in the selected address (four hex characters). The address is displayed in the data display indicators.
3. Press the Store key. The address displayed in the data display indicators is stored into the SAR.
4. Press the Main Storage key. The contents of the addressed storage location are displayed in the data display indicators.
5. Key in the data that is to be stored into main storage. This data is displayed in the data display indicators.
6. Press the Store key. The data that is displayed is stored at the selected storage location. Each subsequent pressing of the Store key causes the SAR to be incremented by +2, and the data stored at that location is displayed.
**Displaying Registers**

- Processor must be in stop state.

1. Select the proper level by pressing the appropriate Level key. A

   The contents of any register associated with the selected level can now be displayed by pressing the register key.

2. Press the desired register key. The contents of that register are displayed in the data display indicators. B

**Storing Into Registers**

- Processor must be in stop state.

1. Select the proper level by pressing the appropriate Level key. A

2. Press the key for the register where data is to be stored. The contents of that register are displayed in the data display indicators. B

3. Key in the data that is to be stored. This data is displayed in the data display indicators.

4. Press the Store key. C

The data that is displayed is stored into the selected register.
The instructions for the IBM 4953 Processor are described in this chapter. A complete listing of instruction formats is contained in Appendix B. Instruction timings are contained in Appendix A. Indicator settings are listed for each instruction. For additional indicator information, refer to Indicators in Chapter 2.

### Exception Conditions

Exception conditions that might occur during instruction execution are shown in abbreviated form with each instruction description. Refer to the following sections for a detailed description of these conditions.

#### Program Check Conditions

**Invalid Function**

(1) An illegal operation code or function combination is encountered during instruction execution, or (2) while in supervisor state, the processor attempts to execute one of the following instructions: operation code 01011 with a function of 0001 or 1001.

A program check class interrupt occurs with invalid function (bit 4) set in the PSW. See Processor Status Word in Chapter 3 for a list of invalid functions.

**Invalid Storage Address**

**Instruction Word or Operand.** One or more words of the instruction or the effective address is outside the installed storage size of the system. The instruction is suppressed unless otherwise noted in the individual instruction description.

A program check class interrupt occurs with invalid storage address (bit 1) set in the PSW.

**Privilege Violate**

**Privileged Instruction.** A privileged instruction is encountered while in problem state. The instruction is suppressed.

A program check class interrupt occurs with privilege violate (bit 2) set in the PSW. See Processor Status Word in Chapter 3 for a list of privileged instructions.

#### Specification Check

**Operand Address.** The generated effective address has violated an even-byte boundary requirement.

**Indirect Address.** When using addressing mode (AM=11), the indirect address is not on an even-byte boundary.

The instruction is suppressed unless otherwise noted in the individual instruction description. A program check class interrupt occurs with specification check (bit 0) set in the PSW.

**Note.** A specification check can also occur during a Supervisor Call (SVC) instruction if the SVC LSB pointer or the SVC SIA pointer violates an even-byte boundary requirement.

#### Soft Exception Trap Conditions

**Invalid Function**

(1) Operation code 00100 is attempted

(2) operation code 10110 is attempted or

(3) in supervisor state, operation code 01011 with a function of 0011 or 1011 is attempted. The instruction is suppressed. A soft-exception-trap class interrupt occurs with invalid function (bit 4) set in the PSW. See Processor Status Word in Chapter 3 for a list of invalid functions.

**Stack Exception**

(1) The stack is full and a Push instruction or a Store Multiple (STM) instruction is attempted, (2) the stack is empty and a Pop instruction or a Load Multiple and Branch (LMB) instruction is attempted, or (3) the stack cannot contain the number of words to be stored by a Store Multiple instruction.

The instruction is suppressed. A soft-exception-trap class interrupt occurs with stack exception (bit 6) set in the PSW.

**Note.** When the AM field is equal to 01, the register specified by the RB field is incremented before the class interrupt occurs.
Instruction Termination or Suppression
Exception conditions that occur during instruction processing might cause the instruction to be terminated or suppressed. When an instruction is terminated, partial execution has taken place and may have caused a change to registers, indicators, or main storage. When an instruction is suppressed, there has been no execution, therefore, no changes. Refer to Exception Conditions in the previous section.

Compatibility
The IBM 4955 Processor has more features and instructions than the IBM 4953 Processor. Consideration should be given to the differences between the processors when writing programs to permit possible future replacement with a larger system.

The following section describes the action taken by the IBM 4953 Processor when instructions that apply to the IBM 4955 Processor are encountered.

Soft Exception Trap
The following instructions cause a soft exception trap class interrupt with invalid function, bit 04 in the PSW, set to 1.

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>001100</td>
<td>all</td>
</tr>
<tr>
<td>01011</td>
<td>0011, 1011 (supervisor state only)</td>
</tr>
</tbody>
</table>

No Operation
In supervisor state the following instructions are recognized and executed as a No Operation instruction.

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01011</td>
<td>0010, 0100, 1010, 1100</td>
</tr>
<tr>
<td>01100</td>
<td>110</td>
</tr>
<tr>
<td>01111</td>
<td>10010, 11010</td>
</tr>
</tbody>
</table>

Program Check
1. In supervisor state the following instructions cause a program check class interrupt with invalid function, bit 04 in the PSW, set to 1.

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01011</td>
<td>0001, 1001</td>
</tr>
<tr>
<td>10110</td>
<td>all</td>
</tr>
</tbody>
</table>

2. In problem state the following instructions cause a program check class interrupt with privilege violate, bit 02 in the PSW, set to 1.

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01011</td>
<td>0001, 0010, 0011, 0100, 1001, 1010, 1011, 1100</td>
</tr>
<tr>
<td>01100</td>
<td>110</td>
</tr>
<tr>
<td>01111</td>
<td>10010, 11010</td>
</tr>
</tbody>
</table>
Instruction Descriptions

The following descriptions are in alphabetical sequence based on assembler mnemonics. However, extended mnemonics are listed under the appropriate machine instruction. For example: branching and jumping instructions.

Add Byte (AB)

\[
\text{AB} \quad \text{reg}, \text{addr4} \\
\text{addr4}, \text{reg}
\]

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9 10 11 12 13 15</td>
</tr>
</tbody>
</table>

An add operation is performed between the least significant byte of the register specified by the R field and the location specified by the effective address in main storage. (See Effective Address Generation in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand and high-order byte of the register are unchanged.

Indicators

**Carry.** Turned on if a carry is detected out of the high-order bit position of the byte. If no carry is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the sum cannot be represented in one byte; i.e., if the sum is less than \(-2^7\) or greater than \(+2^7\).

If an overflow occurs, the result contains the correct low-order eight bits of the sum; the carry indicator contains the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address).

Add Byte Immediate (ABI)

\[
\text{ABI} \quad \text{byte}, \text{reg}
\]

The immediate field is expanded to 16 bits by sign propagation to the eight high-order bits. The field is then added to the contents of the register specified by the R field. The result is placed in the register specified by the R field.

Indicators

**Carry.** Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}\).

If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

No program checks occur.
ACY

Add Carry Register (ACY)

ACY reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>0 0</td>
<td>0 1 1 0 0</td>
</tr>
</tbody>
</table>

The value of the carry indicator on entry is added to the contents of the register specified by the R2 field, and the result is placed in the register specified by the R2 field. Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Programming Note. This instruction can be used when adding multiple word operands. See Indicators – Multiple Word Operands in Chapter 2.

Indicators

Carry. Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than $-2^{15}$ or greater than $+2^{15}-1$.

If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

Even. Unchanged.

Negative. Changed to reflect the result.

Zero. If on at entry, changed to reflect the result. If off at entry, it remains off.

Program Check Conditions

No program checks occur.
Add Doubleword (AD)

**Register/Storage Format**

AD reg, addr4

addr4, reg

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9  10  11 12 13 15</td>
</tr>
</tbody>
</table>

1 = result to storage
0 = result to register

An add operation is performed between the register pair specified by the R field (R and R+1) and the doubleword in main storage specified by the effective address. (See *Effective Address Generation* in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

If the R field equals 7, register 7 and register 0 are used.

**Indicators**

**Carry.** Turned on if a carry is detected out of the high-order bit position of the doubleword. If no carry is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the sum cannot be represented in the doubleword; i.e., if the sum is less than \(-2^{31}\) or greater than \(+2^{31}-1\).

If an overflow occurs, the result contains the correct low-order 32 bits of the sum; the carry indicator contains the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand. The instruction is terminated. If RB1 and RB2 specify the same register and AM1=01, the register is incremented before the program check interrupt occurs.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Add Word (AW)

Register/Register Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>0 4 5 7 8 10 11 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are added to the contents of the register specified by the R2 field. The result is placed in the register specified by the R2 field. The contents of the register specified by the R1 field remain unchanged if R1 and R2 do not specify the same register.

Indicators

**Carry.** Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}-1\).

If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

No program checks occur.

Register/Storage Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 1</td>
<td>0 4 5 7 8 9 10 11 12 13 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An add operation is performed between the register, specified by the R field and the location specified by the effective address in main storage. (See Effective Address Generation in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

Indicators

**Carry.** Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}-1\).

If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Storage to Register Long Format

\[ \text{AW} \quad \text{longaddr}.\text{reg} \]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1 1 1 0</td>
</tr>
</tbody>
</table>

\[ 0 = \text{direct address} \quad 1 = \text{indirect address} \]

Address

The contents of the main storage location specified by an effective address are added to the contents of the register specified by the R1 field. The result is placed in the register specified by the R1 field.

The effective main storage address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field. If the R2 field equals zero, no register contributes to the address generation.
2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11=0 (direct address). The result from step 1 is the effective address.
   - Bit 11=1 (indirect address). The result from step 1 is the address of the main storage location that contains the effective address.

Indicators

- **Carry.** Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

- **Overflow.** Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}-1\).
  
  If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

- **Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

- **Invalid Storage Address.** Instruction word or operand.
  
  The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

- **Specification Check.** Even byte boundary violation (indirect address or operand address).

Storage/Storage Format

\[ \text{AW} \quad \text{addr5}.\text{addr4} \]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1 0 1 1</td>
<td>2 3</td>
<td>2 4</td>
<td>6 7</td>
<td>0 1</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Address/Displacement

The address arguments generate the effective addresses of two operands in main storage. (See *Effective Address Generation* in Chapter 2.) Word operand 1 is added to word operand 2. The result replaces operand 2. Operand 1 is unchanged.

Indicators

- **Carry.** Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

- **Overflow.** Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}-1\).
  
  If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

- **Even, Negative, and Zero.** Changed to reflect the result.
Add Word With Carry (AWCY)

AWCY reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>5</td>
<td>7</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

This instruction adds three terms together:

(R1) the contents of the register specified by the R1 field.
(R2) the contents of the register specified by the R2 field.
C the value of the carry indicator at entry.

The contents of the register specified by the R1 field are unchanged if R1 and R2 do not specify the same register.
The final result replaces the contents of the register specified by the R2 field.

Programming Note. This instruction can be used when adding multiple word operands. See Indicators – Multiple Word Operands in Chapter 2.

Indicators

Carry. Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}-1\).

If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

Even. Unchanged.

Zero. If on at entry, set to reflect the result. If off at entry, remains off.

Negative. Changed to reflect the result.

Program Check Conditions

No program checks occur.
Add Word Immediate (AWI)

Register Immediate Long Format
AWI word,reg[,reg]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1</td>
<td>4 5 7 8</td>
<td>10 11</td>
<td>15</td>
</tr>
</tbody>
</table>

Immediate

The immediate field is added to the contents of the register specified by the R1 field. The result is placed in the register specified by the R2 field. The contents of the register specified by the R1 field are unchanged if R1 and R2 do not specify the same register.

Indicators

Carry. Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}-1\).

If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word.

Storage Immediate Format

AWI word,addr4

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>7 8 9</td>
<td>10 11 12</td>
<td>15</td>
</tr>
</tbody>
</table>

Immediate

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>7 8 9</td>
<td>10 11 12</td>
<td>15</td>
</tr>
</tbody>
</table>

Address/Displacement

Displacement 1
Displacement 2

Immediate

32 47

The immediate field is added to the contents of the location specified by the effective address. (See Effective Address Generation in Chapter 2.) The result replaces the contents of the storage location specified by the effective address.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

The immediate operand is unchanged.

Indicators

Carry. Turned on if a carry is detected out of the high-order bit position of the word. If no carry is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the sum cannot be represented in one word; i.e., if the sum is less than \(-2^{15}\) or greater than \(+2^{15}-1\).

If an overflow occurs, the result contains the correct low-order 16 bits of the sum; the carry indicator contains the high-order (sign) bit.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Branch and Link (BAL)

BAL longaddr, reg

Extended Assembler Mnemonic

BALX vcon, reg Branch and Link External

Operation code

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101000110100001</td>
<td>0</td>
<td>1112108754</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

Address

An effective branch address is generated and loaded into the instruction address register, becoming the next instruction to be fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11 = 0. The result from step 1 is a direct address and is loaded into the instruction address register.
   - Bit 11 = 1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

   Bits 5 - 7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Indicators
No indicators are changed.

Program Check Conditions

Invalid Storage Address. Instruction word or effective branch address.

Specification Check. Even byte boundary violation (indirect address or branch address).

The updated value of the instruction address register (the address of the next sequential instruction) is stored into the register specified by the R1 field. An effective branch address is then generated and loaded into the instruction address register, becoming the next instruction to be fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11 = 0. The result from step 1 is a direct address and is loaded into the instruction address register.
   - Bit 11 = 1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

   Bits 5 - 7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Indicators
No indicators are changed.

Program Check Conditions

Invalid Storage Address. Instruction word or effective branch address. No branch is taken, but the contents of the register specified by the R1 field are still changed.

Specification Check. Even byte boundary violation (indirect address or branch address). No branch is taken but the contents of the R1 register are changed.
Branch and Link Short (BALS)

BALS (reg,jdisp)*
(reg)*
addr*

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td>0</td>
<td>4 5 7 8 15</td>
</tr>
</tbody>
</table>

The updated contents of the instruction address register (the location of the next sequential instruction) are stored in register 7.

Bit 8 of the word displacement field is propagated left by 7 bit positions and a zero is appended at the low order end, resulting in a 16-bit word. (Word displacement is converted to a byte displacement.) This value is added to the contents of the register specified by R to form an effective address. The contents of the storage location specified by the effective address are stored into the instruction address register, and become the address of the next instruction to be fetched.

**Programming Note.** If the implied register (R7) is used as a base register, the initial contents of R7 are used in effective address computation and subsequently overwritten by the return data.

**Indicators**

No indicators are changed.

**Program Check Conditions**

**Invalid Storage Address.** Effective address. Branching does not occur but storing of the updated instruction address into R7 does occur.

**Specification Check.** Even byte boundary violation (effective address). Branching does not occur but storing of the updated instruction address into R7 does occur.
Branch On Condition (BC)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand syntax</th>
<th>Instruction name</th>
<th>Condition field bits (see A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>cond,longaddr</td>
<td>Branch On Condition</td>
<td>Any value listed below</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extended Mnemonic</th>
<th>Operand syntax</th>
<th>Instruction name</th>
<th>Condition field bits (see A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BE</td>
<td>longaddr</td>
<td>Branch on Equal</td>
<td>000</td>
</tr>
<tr>
<td>BOFF</td>
<td>longaddr</td>
<td>Branch if Off</td>
<td>000</td>
</tr>
<tr>
<td>BZ</td>
<td>longaddr</td>
<td>Branch on Zero</td>
<td>000</td>
</tr>
<tr>
<td>BP</td>
<td>longaddr</td>
<td>Branch on Positive</td>
<td>001</td>
</tr>
<tr>
<td>BMIX</td>
<td>longaddr</td>
<td>Branch if Mixed</td>
<td>001</td>
</tr>
<tr>
<td>BN</td>
<td>longaddr</td>
<td>Branch if Negative</td>
<td>010</td>
</tr>
<tr>
<td>BON</td>
<td>longaddr</td>
<td>Branch if On</td>
<td>010</td>
</tr>
<tr>
<td>BEV</td>
<td>longaddr</td>
<td>Branch on Even</td>
<td>011</td>
</tr>
<tr>
<td>BLT</td>
<td>longaddr</td>
<td>Branch on Arithmetically Less Than</td>
<td>100</td>
</tr>
<tr>
<td>BLE</td>
<td>longaddr</td>
<td>Branch on Arithmetically Less Than or Equal</td>
<td>101</td>
</tr>
<tr>
<td>BLLL</td>
<td>longaddr</td>
<td>Branch on Logically Less Than or Equal</td>
<td>110</td>
</tr>
<tr>
<td>BCY</td>
<td>longaddr</td>
<td>Branch on Carry</td>
<td>111</td>
</tr>
<tr>
<td>BLLT</td>
<td>longaddr</td>
<td>Branch on Logically Less Than</td>
<td>111</td>
</tr>
</tbody>
</table>

This instruction tests the condition of the various indicators (LSR bits 0–4). If the condition tested is met, the effective branch address is loaded into the instruction address register and becomes the next address to be fetched.

If the condition tested is not met, the next sequential instruction is fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11 = 0. The result from step 1 is a direct address and is loaded into the instruction address register.
   - Bit 11 = 1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

Indicators
All indicators are unchanged.

Program Check Conditions

Invalid Storage Address. Instruction word or effective address.

Specification Check. Even byte boundary violation (indirect address or branch address).
Branch On Condition Code (BCC)

BCC cond,longaddr

Extended mnemonic
BNER longaddr Branch on Not Error
(CC field = 111)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>CC</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1</td>
<td>0 4</td>
<td>5 7 8 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

Address

The value of the CC field is compared to the even, carry, and overflow indicators. These indicators hold the I/O condition code: (1) following an I/O instruction or (2) following an I/O interrupt.

CC bit   Indicator
5        Even
6        Carry
7        Overflow

If the conditions match, an effective branch address is generated and loaded into the instruction address register, becoming the next instruction to be fetched.

If the conditions do not match the next sequential instruction is fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address.
   If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   Bit 11=0. The result from step 1 is a direct address and is loaded into the instruction address register.
   Bit 11=1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

Indicators
No indicators are changed.

Program Check Conditions

Invalid Storage Address. Instruction word or effective address.

Specification Check. Even byte boundary violation (indirect address or branch address).

I/O Condition Codes

The I/O condition codes are summarized in the following tables. Refer to Chapter 4 for a detailed description of each condition code value. Also refer to the specific I/O device descriptions because some devices do not report all condition codes.

Condition Codes Reported After I/O Instruction.

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Indicators</th>
<th>Carry</th>
<th>Overflow</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0</td>
<td>Device not attached</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td>0 1</td>
<td></td>
<td>Busy</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0</td>
<td>1 0</td>
<td></td>
<td>Busy after reset</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1</td>
<td>1 1</td>
<td></td>
<td>Command reject</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0</td>
<td>0 0</td>
<td></td>
<td>Intervention required</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1</td>
<td>0 1</td>
<td></td>
<td>Interface data check</td>
</tr>
<tr>
<td>6</td>
<td>1 1 0</td>
<td>1 0</td>
<td></td>
<td>Controller busy</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1</td>
<td>1 1</td>
<td></td>
<td>Satisfactory</td>
</tr>
</tbody>
</table>

Condition Codes Reported During an I/O Interrupt.

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Indicators</th>
<th>Carry</th>
<th>Overflow</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
<td>0 0</td>
<td>0</td>
<td>Controller end</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td>0 1</td>
<td></td>
<td>PCI (program control interrupt)</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0</td>
<td>1 0</td>
<td></td>
<td>Exception</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1</td>
<td>1 1</td>
<td></td>
<td>Device end</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0</td>
<td>0 0</td>
<td></td>
<td>Attention</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1</td>
<td>0 1</td>
<td></td>
<td>Attention and PCI</td>
</tr>
<tr>
<td>6</td>
<td>1 1 0</td>
<td>1 0</td>
<td></td>
<td>Attention and exception</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1</td>
<td>1 1</td>
<td></td>
<td>Attention and device end</td>
</tr>
</tbody>
</table>
This instruction tests the various indicators (LSR bits 0–4). If the condition tested is met, the effective branch address is loaded into the instruction address register and becomes the next address to be fetched.

If the condition tested is not met, the next sequential instruction is fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:

   Bit 11=0. The result from step 1 is a direct address and is loaded into the instruction address register.

   Bit 11=1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

**Indicators**

All indicators are unchanged.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or effective address.

**Specification Check.** Even byte boundary violation (indirect address or branch address).
Branch On Not Condition Code (BNCC)
BNCC cond,longaddr

Extended mnemonic
BER longaddr Branch on Error (CC Field=111)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>CC</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1</td>
<td>4 5 7 8 10 11 12 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

Address

The value of the CC field is compared to the even, carry, and overflow indicators. These indicators hold the I/O conditions code: (1) following an I/O instruction or (2) following an I/O interrupt.

CC bit Indicators
5 Even
6 Carry
7 Overflow

If the conditions do not match, an effective branch address is generated and loaded into the instruction address register, becoming the next instruction to be fetched.

If the conditions match, the next sequential instruction is fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11=0. The result from step 1 is a direct address and is loaded into the instruction address register.
   - Bit 11=1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

Indicators
No indicators are changed.

Program Check Conditions

Invalid Storage Address. Instruction word or effective address.

Specification Check. Even byte boundary violation (indirect address or branch address).

I/O Condition Codes
The I/O condition codes are summarized in the following tables. Refer to Chapter 4 for a detailed description of each condition code value. Also refer to the specific I/O device descriptions because some devices do not report all condition codes.

Condition Codes Reported After I/O Instruction.

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Indicators</th>
<th>Carry</th>
<th>Overflow</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td></td>
<td></td>
<td>Device not attached</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td>Busy</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0</td>
<td></td>
<td></td>
<td>Busy after reset</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1</td>
<td></td>
<td></td>
<td>Command reject</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td>Intervention required</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1</td>
<td></td>
<td></td>
<td>Interface data check</td>
</tr>
<tr>
<td>6</td>
<td>1 1 0</td>
<td></td>
<td></td>
<td>Controller busy</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1</td>
<td></td>
<td></td>
<td>Satisfactory</td>
</tr>
</tbody>
</table>

Condition Codes Reported During an I/O Interrupt.

<table>
<thead>
<tr>
<th>Condition Code</th>
<th>Indicators</th>
<th>Carry</th>
<th>Overflow</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td></td>
<td></td>
<td>Controller end</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td>PCI (program controlled interrupt)</td>
</tr>
<tr>
<td>2</td>
<td>0 1 0</td>
<td></td>
<td></td>
<td>Exception</td>
</tr>
<tr>
<td>3</td>
<td>0 1 1</td>
<td></td>
<td></td>
<td>Device end</td>
</tr>
<tr>
<td>4</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td>Attention</td>
</tr>
<tr>
<td>5</td>
<td>1 0 1</td>
<td></td>
<td></td>
<td>Attention and PCI</td>
</tr>
<tr>
<td>6</td>
<td>1 1 0</td>
<td></td>
<td></td>
<td>Attention and Exception</td>
</tr>
<tr>
<td>7</td>
<td>1 1 1</td>
<td></td>
<td></td>
<td>Attention and device end</td>
</tr>
</tbody>
</table>
BNOV

**Branch On Not Overflow (BNOV)**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1 0 0 0</td>
<td>4 5 7 8 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 = direct address

1 = indirect address

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>

The overflow indicator is tested. If the indicator is off, the effective branch address is loaded into the instruction address register and becomes the next address to be fetched.

If the overflow indicator is on, the next sequential instruction is fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11=0. The result from step 1 is a direct address and is loaded into the instruction address register.
   - Bit 11=1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

Bits 5−7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

**Indicators**

All indicators are unchanged.

**Program Check Conditions**

Invalid Storage Address. Instruction word or effective address.

Specification Check. Even byte boundary violation (indirect address or branch address).

---

BOV

**Branch On Overflow (BOV)**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1 0 0 0</td>
<td>0 4 5 7 8 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 = direct address

1 = indirect address

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>

The overflow indicator is tested. If the indicator is on, the effective branch address is loaded into the instruction address register and becomes the next address to be fetched.

If the overflow indicator is off, the next sequential instruction is fetched.

The effective branch address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11=0. The result from step 1 is a direct address and is loaded into the instruction address register.
   - Bit 11=1. The result from step 1 is an indirect address. The contents of the main storage location specified by the result are loaded into the instruction address register.

Bits 5−7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

**Indicators**

All indicators are unchanged.

**Program Check Conditions**

Invalid Storage Address. Instruction word or effective address.

Specification Check. Even byte boundary violation (indirect address or branch address).
Branch Indexed Short (BXS)

BXS  (reg1-7,disp)
    (reg1-7)
    addr

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 0</td>
<td>45</td>
<td>8 15</td>
</tr>
<tr>
<td></td>
<td>1-7</td>
<td></td>
</tr>
</tbody>
</table>

Bit 8 of the word displacement field is propagated left seven bit positions and a zero is appended at the low order end, resulting in a 16-bit word. (Word displacement is converted to a byte displacement.) This value is added to the contents of the register specified by the R field, and the result is stored into the instruction address register, becoming the address of the next instruction to be fetched.

Note. The hardware format of this instruction is identical to the format used for the Jump Unconditional (J) and No Operation (NOP) instructions.

Indicators

No indicators are changed.

Program Check Conditions

Invalid Storage Address. Effective address.

Specification Check. Even byte boundary violation (branch address).
Compare Byte (CB)

Register/Storage Format

CB addr4, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0</td>
<td>0 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Addr4, Displac.

The contents of the location specified by the effective address in main storage are subtracted from the least significant byte of the register specified by the R field. (Effective Address Generation is explained in Chapter 2.) Neither operand is changed.

Indicators

**Carry**. Turned on by the detection of a borrow beyond the high-order bit position of the byte. If no borrow is detected, the carry indicator is reset.

**Overflow**. Cleared, then turned on if the difference cannot be represented in one byte; i.e., if the difference is less than $-2^7$ or greater than $2^7-1$.

**Even, Negative, and Zero**. Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address**. Instruction word or operand.

**Specification Check**. Even byte boundary violation (indirect address).

Storage/Storage Format

CB addr5, addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0</td>
<td>4 5</td>
<td>7 8 9</td>
<td>10 11 12 13 14 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Displacemen.

The address arguments generate the effective addresses of the two operands in main storage. (Effective Address Generation is explained in Chapter 2.) Byte operand 1 is subtracted from byte operand 2. Neither operand is changed.

Indicators

**Carry**. Turned on by the detection of a borrow beyond the high-order bit position of the byte. If no borrow is detected, the carry indicator is reset.

**Overflow**. Cleared, then turned on if the difference cannot be represented in one byte; i.e., if the difference is less than $-2^7$ or greater than $2^7-1$.

**Even, Negative, and Zero**. Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address**. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

**Specification Check**. Even byte boundary violation (indirect address).
Compare Byte Immediate (CBI)

\[ \text{CBI byte,reg} \]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
<td>1 5</td>
</tr>
</tbody>
</table>

The immediate field is extended to 16 bits by sign propagation to the eight high-order bit positions. The result is subtracted from the contents of the register specified by the R field. Neither operand is changed.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15}\).-1.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

No program checks occur.
Compare Doubleword (CD)

Register/Storage Format

CD addr4,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1 0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7 8 9</td>
</tr>
</tbody>
</table>

The contents of the doubleword in main storage specified by the effective address are subtracted from the contents of the register pair specified by the R field and R+1. *(Effective Address Generation is explained in Chapter 2.)*

Neither operand is changed.

Bit 12 of the instruction is not used and must be set to zero to avoid future code obsolescence. If the R field equals 7, register 7 and register 0 are used.

Indicators

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the doubleword. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in the doubleword; i.e., if the difference is less than $-2^{31}$ or greater than $+2^{31}-1$.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).

Storage/Storage Format

CD addr5,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

The address arguments generate the effective addresses of two operands in main storage. *(Effective Address Generation is explained in Chapter 2.)* Doubleword operand 1 is subtracted from doubleword operand 2. Neither operand is changed.

Indicators

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the operand. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one doubleword; i.e., if the difference is less than $-2^{31}$ or greater than $+2^{31}-1$.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Compare Byte Field Equal and Decrement (CFED)

Compare Byte Field Equal and Increment (CFEN)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>I</th>
<th>D</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 0 1</td>
<td>4 5 7 8 10 11 12 13 14 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 for CFED or CFEN
0 for CFED: decrement contents of R1 & R2.
1 for CFEN: increment contents of R1 & R2.

This instruction compares two fields in main storage on a byte for byte basis. Register 7 contains the number of bytes to be compared. This number is decremented after each byte is compared. The register specified by R1 contains the address of operand 1. The register specified by R2 contains the address of operand 2. Operand 1 is subtracted from operand 2, but neither operand is changed. After each byte is compared, the addresses in R1 and R2 are incremented or decremented (determined by bit 13 of the instruction). The operation terminates when either:

1. An equal condition is detected, or
2. The number of bytes specified in register 7 has been compared.

When an equality occurs, the addresses in the registers point to the next operands to be compared, but the count in R7 is not updated.

Bit 11 of the instruction is not used and must be set to zero to avoid future code obsolescence.

See Scan Byte Field Equal and Decrement (SFED) and Scan Byte Field Equal and Increment (SFEN) for other versions of this machine instruction.

Notes.
1. If the specified count in R7 is zero, the instruction performs no operation (No-op).
2. Variable field length instructions can be interrupted. When this occurs and the interrupted level resumes operation, the processor treats the uncompleted instruction as a new instruction with the remaining byte count specified in register 7.

Indicators

Carry. Turned on by the detection of a borrow beyond the high-order bit position of the byte. If no borrow is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the difference cannot be represented in one byte; i.e., if the difference is less than $-2^7$ or greater than $+2^7-1$.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Operand. The instruction is terminated.
CFNED

Compare Byte Field Not Equal and Decrement (CFNED)

Compare Byte Field Not Equal and Increment (CFNEN)

CFNED (reg),(reg)
CFNEN (reg),(reg)

This instruction compares two fields in main storage on a byte for byte basis. Register 7 contains the number of bytes to be compared. This number is decremented after each byte is compared. The register specified by R1 contains the address of operand 1. The register specified by R2 contains the address of operand 2. Operand 1 is subtracted from operand 2, but neither operand is changed. After each byte is compared, the addresses in R1 and R2 are incremented or decremented (determined by bit 13 of the instruction). The operation terminates when either:
1. An unequal condition is detected, or
2. The number of bytes specified in register 7 has been compared.

When an inequality occurs, the addresses in the registers point to the next operands to be compared, but the count in R7 is not updated.

Bit 11 is not used and must be set to zero to avoid future code obsolescence.

See Scan Byte Field Not Equal and Decrement (SFNED) and Scan Byte Field Not Equal and Increment (SFNEN) for other versions of this machine instruction.
Complement Register (CMR)

CMR reg[,reg]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>0</td>
<td>0</td>
<td>0 0 1 1 0</td>
</tr>
<tr>
<td>0 4 5 7 8 10 11 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are converted to the two's complement. The result is placed in the register specified by the R2 field. The contents of the register specified by the R1 field are unchanged if R1 and R2 do not specify the same register.

Indicators

- **Carry.** Reset. Then turned on if the number to be complemented is zero.
- **Overflow.** Reset. Then turned on if the number to be complemented is the maximum negative number representable.
- **Even, Negative, and Zero.** Unchanged.

Program Check Conditions

No program checks occur.

Copy Current Level (CPCL)

CPCL reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 1 1 0 1 0 1 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 4 5 7 8 10 11 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The register specified by the R2 field is loaded as follows:
- Bits 0–13 are set to zero.
- Bits 14–15 are set to the binary-encoded current level. For example if the current level is three, bits 14–15 are set to 11.
- Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Indicators

All indicators are unchanged.

Program Check Conditions

Privilege Violate. Privileged instruction.
Copy Console Data Buffer (CPCON)

CPCON \[\text{reg}\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Operation code} & 0 & 1 & 1 & 1 & 1 & 0 & 0 & R2 & 1 & 1 & 0 & 0 & 0 & \text{Function} \\
\hline
0 & & 4 & 5 & 7 & 8 & 10 & 11 & 15 & \hline
\end{array}
\]

The contents of the console data buffer are loaded into the register specified by the R2 field. The contents of the buffer are unchanged.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence. If the programmer console is not installed, the data loaded into the specified register is undefined.

**Indicators**

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Privilege Violate.** Privileged instruction.

Copy Interrupt Mask Register (CPIMR)

CPIMR \[\text{addr4}\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Operation code} & 0 & 1 & 0 & 1 & 1 & 0 & 0 & RB & AM & 1 & 0 & 0 & 0 & \text{Function} \\
\hline
0 & 4 & 5 & 7 & 8 & 9 & 10 & 11 & 12 & 15 & \hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{Address Displacement} & 16 & 23 & 24 & \text{Displacement 2} & 31 & \hline
\end{array}
\]

The contents of the interrupt mask register are stored at the word location in main storage specified by the effective address. (See *Effective Address Generation* in Chapter 2.) The interrupt mask register is unchanged.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

The mask is represented in a bit significant manner as follows:

<table>
<thead>
<tr>
<th>Mask bit</th>
<th>Interrupt level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Bits 4–15 are set to zero.

A mask bit set to "1" indicates that the level is enabled.

A mask bit set to "0" indicates that the level is disabled.

**Indicators**

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand.

**Privilege Violate.** Privileged instruction.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Copy In-process Flags (CPIPF)

CPIPF addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1 0 0 0</td>
<td>1 1 0 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address/Disp

The in-process flags for each level are stored at the word location in main storage specified by the effective address. (Effective Address Generation is explained in Chapter 2.)

The in-process flags are not changed. The flags are stored in a bit significant manner with bit zero representing level zero, and so on. Bits 4-15 are set to zero.

Bits 5-7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

This instruction permits the supervisor on the current level to inspect the in-process flags of the other levels. The in-process flag, bit 9 of the level status register, is on when a level is active or pending (previously interrupted by a higher level).

Indicators

All indicators are unchanged.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Privilege Violate. Privileged instruction.

Specification Check. Even byte boundary violation (indirect address or operand address).
Copy Level Block (CPLB)

CPLB reg,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8  9 10 11 12 13</td>
</tr>
</tbody>
</table>

This instruction stores a level status block (LSB) into 11 words of main storage beginning with the location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) The contents of the LSB are not changed.

The register specified by the R field contains the binary encoded level of the LSB to be stored. The binary encoded level is placed in bits 14–15 of the register. Bits 0–13 are not used and must be zero.

Using this one instruction, the supervisor can copy the information contained in the hardware registers assigned to a program operating on any level. Most instructions are restricted to the registers associated with the current level. After executing a CPLB instruction, the supervisor can:

1. Use the information just stored; for example, the contents of the general registers or the LSR.
2. Assign the level to another task by executing a Set Level Block (SELB) instruction that points to a different level status block.

In the second case, the supervisor can restart the preempted program at a later time by executing another SELB instruction that points to the previously stored level status block.

Programming Note. If the AM field equals 01, the contents of the register specified by the RB field are incremented by 2.

Indicators
All indicators are unchanged.

Program Check Conditions
Invalid Storage Address. Instruction word or the 11 word main storage area. The instruction is terminated. If the main storage area being accessed is partially outside the installed storage size, a partial data transfer occurs.

Privilege Violate. Privileged instruction.

Specification Check. Even byte boundary violation (indirect address or operand address).

Level Status Block Format

<table>
<thead>
<tr>
<th>EA</th>
<th>IAR</th>
<th>Zeros</th>
<th>LSR</th>
<th>Register 0</th>
<th>Register 1</th>
<th>Register 2</th>
<th>Register 3</th>
<th>Register 4</th>
<th>Register 5</th>
<th>Register 6</th>
<th>Register 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA+20 (+14 hex)</td>
<td>EA=effective address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format of Register Specified by R in CPLB Instruction

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>13 14 15</td>
<td>Level 0</td>
</tr>
<tr>
<td></td>
<td>Level 1</td>
</tr>
<tr>
<td></td>
<td>Level 2</td>
</tr>
<tr>
<td></td>
<td>Level 3</td>
</tr>
<tr>
<td></td>
<td>Level 4</td>
</tr>
</tbody>
</table>

Level 0

Level 1

Level 2

Level 3
Copy Level Status Register (CPLSR)

CPLSR  reg

Operation code  R2  Function
0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 1 0

The level status register is loaded into the register specified by the R2 field. The level status register is unchanged.

Indicators
All indicators are unchanged.

Program Check Conditions
No program checks occur.

Copy Processor Status and Reset (CPPSR)

CPPSR  addr4

Operation code  RB  AM  Function
0 1 0 1 1 0 0 0 0 1 1 1 1

The contents of the processor status word (PSW) are stored at the word location in main storage specified by the effective address. (Effective Address Generation is explained in Chapter 2.)

This instruction resets bits 0–12 of the PSW. Bits 13–15 are unchanged.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.
**Compare Word (CW)**

**Register/Register Format**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>00101</td>
<td>10</td>
<td>11</td>
<td>15</td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are subtracted from the contents of the register specified by the R2 field. The contents of both registers are unchanged.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15}\). -1.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

No program checks occur.

**Register/Storage Format**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10111215</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>15</td>
</tr>
</tbody>
</table>

The address arguments generate the effective addresses of two operands in main storage. (See Effective Address Generation in Chapter 2.) Word operand 1 is subtracted from word operand 2. Neither operand is changed.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15}\). -1.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Compare Word Immediate (CWI)

Register Immediate Long Format

CWI word, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 1</td>
<td>7 8 10 11 15</td>
<td></td>
</tr>
</tbody>
</table>

The immediate field is subtracted from the contents of the register specified by the R1 field. The contents of the register specified by the R1 field are unchanged.

Bits 8–10 are not used and must be set to zero to avoid future code obsolescence.

Indicators

Carry. Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than $-2^{15}$ or greater than $+2^{15} - 1$.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word.

Storage Immediate Format

CWI word, addr4

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>7 8 9 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>7 8 9 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The immediate word is subtracted from the contents of the location specified by the effective address. (Effective Address Generation is explained in Chapter 2.)

Bits 5–7 are not used and must be set to zero to avoid future code obsolescence. Both operands are unchanged.

Indicators

Carry. Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than $-2^{15}$ or greater than $+2^{15} - 1$.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Divide Byte (DB)

DB addr4, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>111010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A divide operation is performed between the word dividend contained in the register specified by the R field and the byte divisor at the location specified by the effective address. *(Effective Address Generation is explained in Chapter 2.)* The 1-word quotient replaces the contents of the specified register while the 1-word remainder is placed in the register specified by R+1. If the R field specifies register 7, the remainder is placed in register 0.

**Indicators**

**Overflow.** Cleared, then turned on if division by zero is attempted, or if the quotient cannot be represented in one word. If overflow occurs, the remaining indicators and the contents of the specified register are undefined.

**Carry.** Cleared, then turned on (together with the overflow indicator) if the overflow was caused by an attempt to divide by zero.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand. If the AM field equals 01, the contents of the register specified by the RB field are incremented.

**Specification Check.** Even byte boundary violation (indirect address).
A divide operation is performed between the doubleword dividend contained in the registers, specified by the R field and R+1, and the word divisor at the location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) The doubleword quotient replaces the contents of the specified registers (least significant word is in R+1). The one-word remainder is placed in the register specified by R+2.

The R field wraps from 7 to 0; e.g., if R specifies register 6, registers 6, 7, and 0 are used.

*Programming Note.* If the AM field equals 01, the contents of the register specified by the RB field are incremented by 2.

**Indicators**

**Overflow.** Cleared, then turned on if division by zero is attempted, or if the quotient cannot be represented in a doubleword. If overflow occurs, the remaining indicators and the contents of the specified registers are undefined.

**Carry.** Cleared, then turned on (together with the overflow indicator) if the overflow was caused by an attempt to divide by zero.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
The Diagnose instruction is used for controlling or testing various hardware functions in a machine dependent manner.

The parameter field has the following bit significance.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>8&amp;9 = 00</td>
<td>Storage select - word</td>
</tr>
<tr>
<td>10 = unused</td>
<td>Must be set to zero</td>
</tr>
<tr>
<td>11 = unused</td>
<td>Must be set to zero</td>
</tr>
<tr>
<td>12 = 0</td>
<td>Storage to register</td>
</tr>
<tr>
<td>13 = 1</td>
<td>Set system ID</td>
</tr>
<tr>
<td>14 = 0</td>
<td>Disable</td>
</tr>
<tr>
<td>15 = unused</td>
<td>Must be set to zero</td>
</tr>
</tbody>
</table>

**Local Storage Register Select.** This function transfers data between main storage and any local storage location by directly addressing local storage. Two additional words are appended to the Diagnose instruction when this function is specified.

The bits in these words are defined as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>16–25</td>
<td>Must be set to zero</td>
</tr>
<tr>
<td>26–31</td>
<td>Local storage address</td>
</tr>
<tr>
<td>32-47</td>
<td>Immediate data field</td>
</tr>
</tbody>
</table>

Bit 12 of the Parameter field specifies the direction of the data transfer.

The following chart shows the addresses for the registers in local storage.

6-32 GA34-0022
I/O Interface Select. This function, with bit 14 equal to 0, disables and logically isolates the interrupt and cycle steal request lines on the I/O interface from the channel. When bit 14 equals 1, these lines are enabled.

Storage to Register.

Storage to Storage. This function specifies the direction of data transfer for storage and local storage functions.

Set System ID. This function sets the system model number into bits 14–15 of register 0 of the current active level. Bits 0–13 are set to zeros.

If this function is specified, all other functions in the parameter field are ignored. The system ID for the 4953 processor is 03 (hex) and register 0 is set as follows:

<table>
<thead>
<tr>
<th>Register 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 1 1</td>
</tr>
</tbody>
</table>

Disable.

Enable. This function disables or enables parity generation checking. See functions Storage Select and I/O Interface Select.

Program Check Conditions

Privilege Violate. Privileged instruction.

---

**Disable (DIS)**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Function</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0</td>
<td>0 1 1</td>
<td>0 7 8 15</td>
</tr>
</tbody>
</table>

The facilities designated by one bits in the parameter field are disabled. The bits in the parameter field have the following significance:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Facility</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Not used</td>
</tr>
<tr>
<td>9</td>
<td>Not used</td>
</tr>
<tr>
<td>10</td>
<td>Not used</td>
</tr>
<tr>
<td>11</td>
<td>Not used</td>
</tr>
<tr>
<td>12</td>
<td>Not used</td>
</tr>
<tr>
<td>13</td>
<td>Not used</td>
</tr>
<tr>
<td>14</td>
<td>Not used</td>
</tr>
<tr>
<td>15</td>
<td>Summary mask</td>
</tr>
</tbody>
</table>

Note. Bits not used must be set to zero to avoid future code obsolescence.

Indicators

No indicators are changed.

Program Check Conditions

Privilege Violate. Privileged instruction.
Divide Word (DW)

\[
\begin{array}{cccccc}
\text{Operation code} & R & RB & AM & \text{Function} \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 4 & 5 & 7 & 8 & 9 & 10 & 11 & 12 & 15 \\
\end{array}
\]

A divide operation is performed between the word dividend contained in the register specified by the R field and the word divisor at the location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) The one word quotient replaces the contents of the specified register. The one word remainder is placed in the register specified by R+1.

The R field wraps from 7 to 0; that is, if R specifies register 7, registers 7 and 0 are used.

**Indicators**

**Overflow.** Cleared, then turned on if division by zero is attempted, or if the quotient cannot be represented in one word. If overflow occurs, the remaining indicators and the contents of the specified registers are undefined.

**Carry.** Cleared, then turned on (together with the overflow indicator) if the overflow was caused by an attempt to divide by zero.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Enable (EN)

EN ubyte

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Function</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0</td>
<td>0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

The facilities designated by one bits in the parameter field are enabled.

The bits in the parameter field have the following significance:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Facility</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Not used</td>
</tr>
<tr>
<td>9</td>
<td>Not used</td>
</tr>
<tr>
<td>10</td>
<td>Not used</td>
</tr>
<tr>
<td>11</td>
<td>Not used</td>
</tr>
<tr>
<td>12</td>
<td>Not used</td>
</tr>
<tr>
<td>13</td>
<td>Not used</td>
</tr>
<tr>
<td>14</td>
<td>Not used</td>
</tr>
<tr>
<td>15</td>
<td>Summary mask</td>
</tr>
</tbody>
</table>

*Note.* Bits not used must be set to zero to avoid future code obsolescence.

**Indicators**

No indicators are changed.

**Program Check Conditions**

**Privilege Violate.** Privileged instruction.
Fill Byte Field and Decrement (FFD)

Fill Byte Field and Increment (FFN)

This instruction fills all bytes of a field in main storage with the same bit configuration in each byte. Register 7 contains the number of bytes to be filled (field length). If a field length of zero is specified, the instruction is a no-op. The register specified by R1 contains, in bits 8-15, the byte used to fill the field. The register specified by R2 contains the starting address of the field in main storage.

After each byte in the field is filled:

1. The address in R2 is either incremented or decremented, determined by bit 13 of the instruction. This permits filling the field in either direction.
2. The length count in R7 is decremented.

The operation ends when the specified field length has been filled (contents of R7 equal zero). At this time, the address in R2 has been updated and points to the byte adjacent to the end of the field.

Note. Variable field length instructions can be interrupted. When this occurs and the interrupted level resumes operation, the processor treats the uncompleted instruction as a new instruction with the remaining byte count specified in register 7.

Indicators

Carry. Unchanged.

Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect that result of the last byte moved.

Program Check Conditions

Invalid Storage Address. Operand. The instruction is terminated.
Operate I/O (IO)
Refer to Chapter 4 for a detailed description concerning the operation of this instruction.

IO  longaddr

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1 0 0</td>
<td>0 0</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

An effective main storage address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field. If the R2 field equals zero, no register contributes to the address generation.
2. Instruction bit 11 is tested for direct or indirect addressing:

**Bit 11=0 (Direct Address).** The result from step 1 is the effective address.

**Bit 11=1 (Indirect Address).** The result from step 1 is the address of the main storage location that contains the effective address.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

The effective address specifies the location of a two-word control block, called the immediate device control block (IDCB). The IDC contains the command, device address, and a one-word immediate data field:

IDCB (immediate device control block)

<table>
<thead>
<tr>
<th>Command field</th>
<th>Device address field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 7 8 15</td>
<td></td>
</tr>
</tbody>
</table>

The immediate data field serves two purposes:

1. For direct program control (DPC) operations, it holds the data transferred to or from the I/O device.
2. For cycle steal operations, it holds the address of the device control block (DCB).

Refer to Chapter 4 for additional information.

Interchange Registers (IR)
IR  reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0 0</td>
<td>0 1 1 1 1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 and R2 fields are interchanged.

Indicators

**Even, Carry, and Overflow.** Changed to reflect the condition code. See Branch on Condition Code (BCC) or Branch on Not Condition Code (BNCC) instructions.

**Negative and Zero.** These indicators are not changed.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand.

**Privilege Violate.** Privileged instruction.

**Specification Check.** Even byte boundary violation (indirect address or operand address).

Refer to Chapter 4 for additional information.
Jump Unconditional (J)

\[
J \\
\text{jdisp} \\
\text{jaddr}
\]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 0</td>
<td></td>
<td>0 4 5 7 8</td>
</tr>
</tbody>
</table>

Bit 8 of the word displacement field is propagated left seven bit positions and a zero is appended at the low order end, resulting in a 16-bit word. (Word displacement is converted to a byte displacement.) This value is added to the updated value of the instruction address register becoming the address of the next instruction to be fetched.

Note. The hardware format of this instruction is identical to the format used for the Branch Indexed Short (BXS) instruction.

Indicators

No indicators are changed.

Program Check Conditions

Invalid Storage Address. Effective address.

Specification Check. Even byte boundary violation (branch address).

Jump and Link (JAL)

\[
\text{JAL} \\
\text{jdisp,reg} \\
\text{jaddr,reg}
\]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 1</td>
<td></td>
<td>0 4 5 7 8</td>
</tr>
</tbody>
</table>

The updated value of the instruction address register (the location of the next sequential instruction) is stored into the register specified by the R field. Bit 8 of the word displacement field is propagated left by seven bit positions and a zero is appended at the low order end, resulting in a 16-bit word. (The word displacement is converted to a byte displacement.) This value is added to the updated contents of the instruction address register, and the result is stored in the instruction address register, becoming the address of the next instruction to be fetched.

Indicators

No indicators are changed.

Program Check Conditions

Invalid Storage Address. Effective address. The instruction is terminated. Branching does not occur, but the storing of the updated instruction address into the register specified by the R field still occurs.
### Jump On Condition (JC)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand syntax</th>
<th>Instruction Name</th>
<th>Condition field bits (see A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JC</td>
<td>cond,jdisp</td>
<td>Jump on Condition</td>
<td>Any value listed below</td>
</tr>
<tr>
<td></td>
<td>cond,jaddr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extended Mnemonic</th>
<th>Operand syntax</th>
<th>Instruction Name</th>
<th>Condition field bits (see A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JE</td>
<td>jdisp</td>
<td>Jump on Equal</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOFF</td>
<td>jdisp</td>
<td>Jump if Off</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JZ</td>
<td>jdisp</td>
<td>Jump on Zero</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMIX</td>
<td>jdisp</td>
<td>Jump if Mixed</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JP</td>
<td>jdisp</td>
<td>Jump on Positive</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JON</td>
<td>jdisp</td>
<td>Jump if On</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JN</td>
<td>jdisp</td>
<td>Jump on Negative</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JEV</td>
<td>jdisp</td>
<td>Jump on Even</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JLT</td>
<td>jdisp</td>
<td>Jump on Arithmetically Less Than</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JLE</td>
<td>jdisp</td>
<td>Jump on Arithmetically Less Than or Equal</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JLLT</td>
<td>jdisp</td>
<td>Jump on Logically Less Than or Equal</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JCY</td>
<td>jdisp</td>
<td>Jump on Carry</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JLLT</td>
<td>jdisp</td>
<td>Jump on Logically Less Than</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This instruction tests the condition of the various indicators set by a previously executed instruction (for example: an arithmetic, compare, test bit, or test word type of instruction).

If the condition tested is met, bit 8 of the word displacement field is propagated left by seven bit positions and a zero is appended at the low-order end resulting in a 16-bit word. (Word displacement is converted to a byte displacement.) This value is added to the updated value of the instruction address register, becoming the address of the next instruction to be fetched. If the condition tested is not met, the next sequential instruction is fetched.

For additional information about the indicator settings for the various conditions, see Chapter 2.

**Indicators**

No indicators are changed.

**Program Check Conditions**

**Invalid Storage Address.** Effective address.
Jump On Count (JCT)

JCT jdisp,reg
jaddr,reg

This instruction tests the contents of the register specified by the R field.
If the register contents are not zero, the contents are decremented by one. If the register contents are still not zero, the word displacement is converted to a byte displacement and added to the updated contents of the updated instruction address register (IAR). This value indicates the location of the next instruction to be fetched.
If the register contents are zero when initially tested, no decrementing occurs. In this case, or when the register contents are zero after decrementing, the next sequential instruction is fetched.

Note. When the register contents are not zero, the word displacement is converted to a byte displacement as follows. Bit 8 of the word displacement field is propagated left by seven bit positions, and a zero is appended at the low-order end. This results in a 16-bit word that has been doubled in magnitude.

Indicators
No indicators are changed.

Program Check Conditions
Invalid Storage Address. Effective address. The jump does not occur, but the contents of the register specified by the R field are still decremented by one.
Jump On Not Condition (JNC)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand syntax</th>
<th>Instruction name</th>
<th>Condition field bits (see A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNC</td>
<td>cond,disp</td>
<td>Jump on Not</td>
<td>Any value listed below</td>
</tr>
<tr>
<td></td>
<td>cond,jaddr</td>
<td>Condition</td>
<td></td>
</tr>
</tbody>
</table>

Extended Mnemonic

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operand syntax</th>
<th>Instruction name</th>
<th>Condition field bits (see A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNE</td>
<td>jdisp</td>
<td>Jump on Not</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Equal</td>
<td></td>
</tr>
<tr>
<td>JNOFF</td>
<td>jdisp</td>
<td>Jump if Not</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Off</td>
<td></td>
</tr>
<tr>
<td>JNZ</td>
<td>jdisp</td>
<td>Jump on Not</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Zero</td>
<td></td>
</tr>
<tr>
<td>JNMIX</td>
<td>jdisp</td>
<td>Jump on Not</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Mixed</td>
<td></td>
</tr>
<tr>
<td>JNP</td>
<td>jdisp</td>
<td>Jump on Not</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Positive</td>
<td></td>
</tr>
<tr>
<td>JNON</td>
<td>jdisp</td>
<td>Jump if Not</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>On</td>
<td></td>
</tr>
<tr>
<td>JNN</td>
<td>jdisp</td>
<td>Jump on Not</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Negative</td>
<td></td>
</tr>
<tr>
<td>JNEV</td>
<td>jdisp</td>
<td>Jump on Not</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Even</td>
<td></td>
</tr>
<tr>
<td>JGE</td>
<td>jdisp</td>
<td>Jump on Arithm</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>cally Greater</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Than</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>or Equal</td>
<td></td>
</tr>
<tr>
<td>JGT</td>
<td>jdisp</td>
<td>Jump on Arithm</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>cally Greater</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Than</td>
<td></td>
</tr>
<tr>
<td>JLG</td>
<td>jdisp</td>
<td>Jump on</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Logically Greater</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Than</td>
<td></td>
</tr>
<tr>
<td>JLGE</td>
<td>jdisp</td>
<td>Jump on</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Logically Greater</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Than</td>
<td>or Equal</td>
</tr>
<tr>
<td>JNCY</td>
<td>jdisp</td>
<td>Jump on No</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>jaddr</td>
<td>Carry</td>
<td></td>
</tr>
</tbody>
</table>

This instruction tests the condition of the various indicators set by a previously executed instruction (for example: an arithmetic, compare, test bit, or test word type of instruction).

If the condition tested is met, bit 8 of the word displacement field is propagated left by seven bit positions and a zero is appended at the low-order end resulting in a 16-bit word. (Word displacement is converted to a byte displacement.) This value is added to the updated value of the instruction address register, becoming the address of the next instruction to be fetched.

If the condition tested is not met, the next sequential instruction is fetched.

For additional information about the indicator settings for the various conditions, see Chapter 2.

Indicators
No indicators are changed.

Program Check Conditions
Invalid Storage Address. Effective address.

Level Exit (LEX)
LEX [ubyte]

When this instruction is executed, the processor exits the current level. The in-process flag (LSR bit 9) for the current level is turned off. Next the instruction tests for (1) pending levels or outstanding priority interrupt requests, (2) the condition of the summary mask (LSR bit 11), and (3) the condition of the level mask bits.

- If pending levels or outstanding requests exist and the summary mask and level mask is enabled:
  - A branch is executed to the address contained in the IAR of the highest pending or requesting level.
  - This level then becomes the current level and processing resumes.
- If processing levels or outstanding requests exist and the summary mask is disabled:
  - The priority interrupts are not allowed.
  - The highest pending level becomes the current level and processing resumes.
  - If no levels are pending, the processor goes to the wait state.
- If no levels are pending and no interrupt requests are outstanding, the processor goes to the wait state.

For additional information on level switching, refer to Chapter 3.

Programming Note. When a level is exited by a LEX instruction and processing is to continue on a pending level, one instruction is executed on the pending level prior to sampling for a trace class interrupt.

Indicators
No indicators are changed.

Program Check Conditions
Privilege Violate. Privileged instruction.
LMB

Load Multiple and Branch (LMB)

Refer to Stack Operations in Chapter 2 for a detailed description concerning the operation of this instruction. The LMB instruction is used in conjunction with the Store Multiple (STM) instruction described later in this chapter.

LMB  addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>7 8 9 10 11 12 13 14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the registers for the current level are loaded from the stack defined by the stack control block pointed to by the effective address. (Effective Address Generation is explained in Chapter 2.) The registers to be loaded are defined by the stack entry previously stored by a Store Multiple (STM) instruction. The next instruction is fetched from the storage address contained in register 7.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Programming Note. If the AM field equals 01 the contents of the register specified by the RB field are incremented by 2.

Indicators

No indicators are changed.

Program Check Conditions

Invalid Storage Address. Instruction word or stack control block. The instruction is terminated.

Specification Check.
1. Even byte boundary violation (indirect address, stack control block, or stack element).
2. Address in R7 is odd.

Soft Exception Trap Condition

Stack Exception. Stack is empty. If the AM field equals 01, the contents of the register specified by the RB field are incremented. The instruction is terminated.
Multiply Byte (MB)

MB addr4,reg

A multiply operation is performed between the word multiplicand contained in the register specified by the R field and the byte multiplier at the location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) The word product replaces the contents of the register.

Indicators

**Carry.** Reset.

**Overflow.** Cleared, then turned on if the result cannot be represented in 16 bits. If overflow occurs, the contents of the specified register are undefined.

**Even, Negative, and Zero.** Set to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand. If the AM field equals 01, the contents of the register specified by the RB field are incremented.

**Specification Check.** Even Byte boundary violation (indirect address).
Multiply Doubleword (MD)

MD addr4, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11101</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

A multiply operation is performed between the doubleword multiplicand contained in the registers specified by the R field and R+1 and the word multiplier at the location specified by the effective address. *(Effective Address Generation is explained in Chapter 2.)* The doubleword product replaces the contents of the registers with the least significant word in R+1.

The R field wraps from 7 to 0; that is, if R specifies register 7, registers 7 and 0 are used.

**Programming Note.** If AM=01, the register specified by the RB field is incremented by 2.

**Indicators**

- **Carry.** Reset.
- **Overflow.** Cleared, then turned on if the result cannot be represented in 32 bits. If overflow occurs, the contents of the specified registers are undefined.
- **Even, Negative, and Zero.** Set to reflect the result.

**Program Check Conditions**

- **Invalid Storage Address.** Instruction word or operand.
- **Specification Check.** Even byte boundary violation (indirect address or operand address).
Move Address (MVA)

Storage Address to Register Format

MVA       addr4, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

The effective address is loaded into the register specified by the R field. *(Effective Address Generation is explained in Chapter 2.)*

Indicators

- **Carry and Overflow.** Unchanged.
- **Even, Negative, and Zero.** Changed to reflect the operand loaded into the register specified by the R field.

Program Check Conditions

- **Invalid Storage Address.** Second Instruction word.
- **Specification Check.** Even byte boundary violation (indirect address).

Storage Immediate Format

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

The operand in the immediate field replaces the contents of the location specified by the effective address. *(Effective Address Generation is explained in Chapter 2.)*

- Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.
- The immediate operand is not changed.

Indicators

- **Carry and Overflow.** Unchanged.
- **Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

- **Invalid Storage Address.** Instruction word or operand.
- **Specification Check.** Even byte boundary violation (indirect address or operand address).
MVB

Move Byte (MVB)

Register/Storage Format

MVB reg, addr4
addr4, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Rb</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9 10 11 12 13 15</td>
</tr>
</tbody>
</table>

\(1 = \text{result to storage}\)
\(0 = \text{result to register}\)

A byte is moved between the least significant byte of the register specified by the R field and the location specified by the effective address in main storage. (Effective Address Generation is explained in Chapter 2.) Bit 12 of the instruction specifies the direction of the move:

- **Bit 12 = 0.** The byte is moved from storage to register. The high-order bit of the byte (sign) is propagated to the eight high order bits of the register. This permits the Compare Byte Immediate (CBI) instruction to be used for byte compare operations. The operand in storage is unchanged.
- **Bit 12 = 1.** The byte is moved from register to storage. The contents of the register specified by the R field are not changed.

Indicators

- **Carry and Overflow.** Unchanged.
- **Even, Negative, and Zero.** Changed to reflect the byte moved.

Program Check Conditions

- **Invalid Storage Address.** Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RBI is incremented.

Specification Check. Even byte boundary violation (indirect address).

Storage/Storage Format

MVB addr5, addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Address/Displacement

\(16\ 23\ 24\ \text{Displacement 1}\)
\(25\ 26\ \text{Displacement 2}\)
\(31\ \text{Displacement 2}\)

The address arguments generate the effective addresses of two operands in main storage. (Effective Address Generation is explained in Chapter 2.) A byte is moved from operand 1 to operand 2. Operand 1 is unchanged.

Indicators

- **Carry and Overflow.** Unchanged.
- **Even, Negative, and Zero.** Changed to reflect the byte moved.

Program Check Conditions

- **Invalid Storage Address.** Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RBI is incremented.

Specification Check. Even byte boundary violation (indirect address).
Move Byte Immediate (MVBI)

MVBI  byte,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 4 5 7 8 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The register specified by the R field is loaded with the immediate operand.

The immediate field of the instruction forms the operand to be loaded. The immediate field is expanded to a sixteen bit operand by propagating the sign bit value through the high order bit positions; this operand is loaded into the register specified by the R field.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the operand loaded into the register.

Program Check Conditions

No program checks occur.

Move Byte and Zero (MVBZ)

MVBZ  addr4,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>0 0 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The byte specified by the effective address is loaded into the least significant byte of the register specified by the R field. (Effective Address Generation is explained in Chapter 2.) The high order bit of the byte (sign) is propagated to the eight high order bits within the register.

The byte specified by the effective address is then set to zeros.

Bit 12 of the instruction is not used and must be set to zero to avoid future code obsolescence.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the operand loaded into the register.

Program Check Conditions.

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address).
MVD

Move Doubleword (MVD)

Register/Storage Format

MVD   addr4,reg
     reg,addr4

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 0 0</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>14 15</td>
</tr>
</tbody>
</table>

1 = result to storage
0 = result to register

A doubleword is moved between the contents of the register pair specified by the R field (R and R+1) and the doubleword location specified by the effective address in main storage. (Effective Address Generation is explained in Chapter 2.) The source operand is unchanged.

Bit 12 of the instruction specifies the direction of the move:

- Bit 12 = 0. The doubleword is moved from storage to the register pair.
- Bit 12 = 1. The doubleword is moved from the register pair to storage.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the doubleword moved.

Program Check Conditions

Invalid Storage Address. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RBI is incremented.

Specification Check. Even byte boundary violation (indirect address or operand address).

Storage/Storage Format

MVD   addr5,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1 RB2 AM1 AM2 Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0</td>
<td>4 5 7 8 9 10 11 12 13 14 15</td>
</tr>
</tbody>
</table>

Address/Displacement

<table>
<thead>
<tr>
<th>Displacement 1</th>
<th>Displacement 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>23 24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address/Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displacement 1</td>
</tr>
<tr>
<td>32 39 40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address/Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displacement 2</td>
</tr>
<tr>
<td>31</td>
</tr>
</tbody>
</table>

The address arguments generate the effective addresses of two operands in main storage. (Effective Address Generation is explained in Chapter 2.) A doubleword is moved from operand 1 to operand 2.Operand 1 is unchanged.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the doubleword moved.

Program Check Conditions

Invalid Storage Address. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RBI is incremented.

Specification Check. Even byte boundary violation (indirect address or operand address).
Move Doubleword and Zero (MVDZ)

MVDZ    addr4, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11010</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9 10 11 12 15</td>
</tr>
</tbody>
</table>

The doubleword specified by the effective address is loaded into the register pair specified by the R field (R and R+1). *(Effective Address Generation is explained in Chapter 2.)* The R field wraps from 7 to 0; that is, if R specifies register 7, registers 7 and 0 are used.

The doubleword specified by the effective address is then set to zeros.

Bit 12 of the instruction is not used and must be set to zero to avoid future code obsolescence.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the operand loaded into the register pair.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Move Byte Field and Decrement (MVFD)

Move Byte Field and Increment (MVFN)

MVFD (reg),(reg)
MVFN (reg),(reg)

This instruction moves a specified number of bytes (one byte at a time) from one storage location to another. Register 7 contains the number of bytes to be moved (field length). If a field length of zero is specified, the instruction is a no-op. The register specified by R1 contains the address of operand 1; the register specified by R2 contains the address of operand 2. Operand 1 is moved to operand 2.

After each byte is moved:

1. The addresses in R1 and R2 are either incremented or decremented, determined by bit 13 of the instruction. This allows the field to be moved in either direction.
2. The length count in R7 is decremented.

The operation ends when the specified field length has been filled (contents of R7 equal zero). At this time, the addresses in R1 and R2 have been updated and point to the next operands.

Bits 11 and 15 of the instructions are not used and must be set to zero to avoid future code obsolescence.

See Fill Byte Field and Decrement (FFD) and Fill Byte Field and Increment (FFN) for other versions of this machine instruction.
**Move Word (MVW)**

### Register/Register Format

**MVW**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4 5</td>
<td>7 8</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field replace the contents of the register specified by the R2 field. The contents of the register specified by the R1 field are unchanged.

### Indicators

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the result.

### Program Check Conditions

No program checks occur.

### Register/Storage Format

**MVW**

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>10 11 12 13 15</td>
</tr>
</tbody>
</table>

1 = result to storage

0 = result to register

A word is moved between the contents of the register specified by the R field and the location specified by the effective address in main storage. *(Effective Address Generation is explained in Chapter 2.)* The source operand is unchanged.

### Indicators

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the operand moved.

### Program Check Conditions

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Storage to Register Long Format

MVW longaddr,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1</td>
<td>4 5 7 8 10 11 12 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

Address

The register specified by the R1 field is loaded with the contents of the main storage location specified by an effective address. This effective address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field. If the R2 field equals zero, no register contributes to the address generation.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit = 0 (direct address). The result from step 1 is the effective address.
   - Bit = 1 (indirect address). The result from step 1 is the address of the main storage location that contains the effective address.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result loaded into the register specified by the R1 field.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).

Storage/Storage Format

MVW addr5,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1</td>
<td>4 5 7 8 9 10 11 12 13 14 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The address arguments generate the effective addresses of two operands in main storage. (Effective Address Generation is explained in Chapter 2.) A word is moved from operand 1 to operand 2. Operand 1 is unchanged.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the word moved.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Move Word Immediate (MVWI)

**Storage to Register Format**

MVWI \( \text{word,reg} \)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td>0 1 0 0 0</td>
</tr>
</tbody>
</table>

The effective address value is loaded into the register specified by the R field. (*Effective Address Generation* is explained in Chapter 2.) This value is equal to the value of *word* as specified by the programmer.

**Indicators**

- **Carry and Overflow**. Unchanged.
- **Even, Negative, and Zero**. Changed to reflect the operand loaded into the register specified by the R field.

**Program Check Conditions**

- **Invalid Storage Address**. Second instruction word.
- **Specification Check**. Even byte boundary violation (indirect address).

**Immediate Format**

MVWI \( \text{word,addr4} \)

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td>0 0 0 0 0</td>
</tr>
</tbody>
</table>

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td>0 0 0 0 0</td>
</tr>
</tbody>
</table>

The operand in the immediate field replaces the contents of the location specified by the effective address. (*Effective Address Generation* is explained in Chapter 2.)

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence. The immediate operand is not changed.

**Indicators**

- **Carry and Overflow**. Unchanged.
- **Even, Negative, and Zero**. Changed to reflect the result.

**Program Check Conditions**

- **Invalid Storage Address**. Instruction word or operand.
- **Specification Check**. Even byte boundary violation (indirect address or operand address).
Move Word Short (MVWS)

**Register to Storage Format**

MVWS \( \text{reg,shortaddr} \)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>( R1 )</th>
<th>( RB )</th>
<th>( X )</th>
<th>Wd displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 0</td>
<td>4 5 7 8 9 10 11 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( 0 = \text{direct address} \)
\( 1 = \text{indirect address} \)

The contents of the register specified by \( R1 \) are stored into the main storage location specified by the effective address. The contents of the register are unchanged.

The effective address is generated as follows:
1. The five bit unsigned integer (word displacement) is doubled in magnitude (converted to a byte displacement).
2. The result from step 1 is added to the contents of the base register (\( RB \)) to form a main storage address.
3. Instruction bit 10 is tested for direct or indirect addressing:
   - Bit 10 = 0 (direct address). The result from step 2 is the effective address.
   - Bit 10 = 1 (indirect address). The result from step 2 is the address of the main storage location that contains the effective address.

**Indicators**

*Carry and Overflow.* Unchanged.

*Even, Negative, and Zero.* Changed to reflect the operand stored into main storage.

**Program Check Conditions**

*Invalid Storage Address.* Operand.

*Specification Check.* Even byte boundary violation (indirect address or operand address).

**Storage to Register Format**

MVWS \( \text{shortaddr,reg} \)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>( R1 )</th>
<th>( RB )</th>
<th>( X )</th>
<th>Wd displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 0</td>
<td>4 5 7 8 9 10 11 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( 0 = \text{direct address} \)
\( 1 = \text{indirect address} \)

The contents of the main storage location specified by the effective address are loaded into the register specified by the \( R1 \) field. The contents of the main storage location remain unchanged.

The effective address is generated as follows:
1. The five bit unsigned integer (word displacement) is doubled in magnitude (converted to a byte displacement).
2. The result from step 1 is added to the contents of the base register (\( RB \)) to form a main storage address.
3. Instruction bit 10 is tested for direct or indirect addressing:
   - Bit 10 = 0 (direct address). The result from step 2 is the effective address.
   - Bit 10 = 1 (indirect address). The result from step 2 is the address of the main storage location that contains the effective address.

**Indicators**

*Carry and Overflow.* Unchanged.

*Even, Negative, and Zero.* Changed to reflect the operand loaded into the register specified by the \( R1 \) field.

**Program Check Conditions**

*Invalid Storage Address.* Operand.

*Specification Check.* Even byte boundary violation (indirect address or operand address).
Move Word and Zero (MVWZ)

MVWZ     addr4,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 10 01</td>
<td></td>
<td>5</td>
<td>8</td>
<td>0 1 01</td>
</tr>
</tbody>
</table>

The word specified by the effective address is loaded into the register specified by the R field. *(Effective Address Generation is explained in Chapter 2.)*

The word specified by the effective address is then set to zeros.

Bit 12 of the instruction is not used and must be set to zero to avoid future code obsolescence.

**Indicators**

*Carry and Overflow.* Unchanged.

*Even, Negative, and Zero.* Changed to reflect the results of the operand loaded.

**Program Check Conditions**

*Invalid Storage Address.* Instruction word or operand.

*Specification Check.* Even byte boundary violation (indirect address or operand address).
Multiply Word (MW)

MW addr4,reg

A multiply operation is performed between the word multiplier contained in the register specified by the R field and the word multiplicand at the location specified by the effective address. *(Effective Address Generation is explained in Chapter 2.)* The word product replaces the contents of the register.

**Indicators**

**Carry.** Unchanged

**Overflow.** Cleared, then turned on if the result cannot be represented in 16 bits. If overflow occurs, the contents of the specified register are undefined.

**Even, Negative, and Zero.** Set to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
No Operation (NOP)
NOP

The hardware format of this instruction is identical to the format used for the Branch Indexed Short (BXS) and Jump Unconditional (J) instructions. When bits 5–15 are all zeros, the instruction performs no operation.

Indicators
No indicators are changed.

Program Check Conditions
No program checks occur.

And Word Immediate (NWI)
NWI word,reg [,reg]

The immediate field is ANDed bit by bit with the contents of the register specified by the R1 field. The result is placed in the register specified by the R2 field. The contents of the register specified by R1 are unchanged unless R1 and R2 specify the same register.

Indicators
Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions
Invalid Storage Address. Instruction word.
OR Byte (OB)

Register/Storage Format

OB  reg,addr4
    addr4,reg

Operation Code

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11000</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9  10 11 12 13 14 15</td>
</tr>
</tbody>
</table>

A logical OR operation is performed between the least significant byte of the register specified by the R field and the location specified by the effective address in main storage. (Effective Address Generation is explained in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged. Also, when going from storage to register, bits 0 through 7 of the register are unchanged.

Indicators

Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.
Specification Check. Even byte boundary violation (indirect address).
OR Doubleword (OD)

Register/Storage Format

OD \[ \text{addr4,reg} \]
reg,\text{addr4}  

A logical OR operation is performed between the contents of the register pair specified by the R field (R and R+1) and the doubleword in main storage specified by the effective address. (Effective Address Generation is explained in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

If the R field equals 7, register 7 and register 0 are used.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result of the OR operation.

Program Check Conditions

- Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or direct address).

Storage/Storage Format

OD \[ \text{addr5,addr4} \]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

The address arguments generate the effective addresses of two operands in main storage. (Effective Address Generation is explained in Chapter 2.) A doubleword logical OR operation is performed between operand 1 and operand 2. The result replaces operand 2. Operand 1 is unchanged.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result of the OR operation.

Program Check Conditions

Invalid Storage Address. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RBI is incremented.

Specification Check. Even byte boundary violation (indirect address or operand address).
OR Word (OW)

Register/Register Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4 5</td>
<td>7 8</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are ORed bit by bit with the contents of the register specified by the R2 field. The result is placed in the register specified by the R2 field. The contents of the register specified by the R1 field remain unchanged unless R1 and R2 specify the same register.

Indicators

- Carry and Overflow. Unchanged.
- Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

No program checks occur.

Register/Storage Format

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 1</td>
<td>4 5</td>
<td>7 8</td>
<td>9 10 11 12 13 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ l = \text{result to storage} \]
\[ 0 = \text{result to register} \]

A logical OR operation is performed between the contents of the register specified by the R field and the location specified by the effective address in main storage. (See Effective Address Generation in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

Indicators

- Carry and Overflow. Unchanged.
- Even, Negative, and Zero. Changed to reflect the result of the OR operation.

Program Check Conditions

- Invalid Storage Address. Instruction word or operand.
- Specification Check. Even byte boundary violation (indirect address or operand address).
Storage to Register Long Format

OW  longaddr,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7 8 10 11 12 13 14 15</td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

A logical OR operation is performed between the contents of the main storage location specified by an effective address and the contents of the register specified by the R1 field. The result is placed in the register specified by the R1 field.

The effective main storage address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field. If the R2 field equals zero, no register contributes to the address generation.
2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11=0 (direct address). The result from step 1 is the effective address.
   - Bit 11=1 (indirect address). The result from step 1 is the address of the main storage location that contains the effective address.

Indicators

Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

Specification Check. Even byte boundary violation (indirect address or operand address).

Storage/Storage Format

OW  addr5,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1</td>
<td>0 4</td>
<td>5 7</td>
<td>8 9</td>
<td>10 11 12 13 14 15</td>
<td>0 1</td>
</tr>
</tbody>
</table>

The address arguments generate the effective addresses of two operands in main storage. (See Effective Address Generation in Chapter 2.) A one word logical OR operation is performed between operand 1 and operand 2. The result replaces operand 2. Operand 1 is unchanged.

Indicators

Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

Specification Check. Even byte boundary violation (indirect address or operand address).
Register Immediate Format

OWI word, reg[.reg]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>011111</td>
<td>0</td>
<td>0</td>
<td>00011</td>
</tr>
</tbody>
</table>

Immediate

16 31

The immediate field is ORed bit by bit with the contents of the register specified by the R1 field. The result is placed in the register specified by the R2 field. The contents of the register specified by R1 are unchanged unless R1 and R2 specify the same register.

Indicators

- Carry and Overflow. Unchanged.
- Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

- Invalid Storage Address. Instruction word or operand.

Storage Immediate Format

OWI word, addr4

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>0</td>
<td>0</td>
<td>1100</td>
</tr>
</tbody>
</table>

Immediate

16 31

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>010000</td>
<td>0</td>
<td>0</td>
<td>1100</td>
</tr>
</tbody>
</table>

Immediate

32 47

A logical OR operation is performed between the immediate field and the contents of the main storage location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) The result replaces the contents of the storage location.

Bits 5-7 of the instruction are not used and must be set to zero to avoid future code obsolescence. The immediate operand is unchanged.

Indicators

- Carry and Overflow. Unchanged.
- Even, Negative, and Zero. Changed to reflect the result of the OR operation.

Program Check Conditions

- Invalid Storage Address. Instruction word or operand.
- Specification Check. Even byte boundary violation (indirect address or operand address).
Pop Byte (PB)
Refer to *Stack Operations* in Chapter 2 for additional information about the operation of this instruction and the associated stack control block.

PB \( \text{addr4,reg} \)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

The top element of a byte stack is popped from the stack and loaded into the least significant byte of the register specified by the R field. The stack is defined by the stack control block pointed to by the effective address. (*Effective Address Generation* is explained in Chapter 2.)

**Programming Note.** If AM equals 01, the register specified by the RB field is incremented by two.

**Indicators**
No indicators are changed.

**Program Check Conditions**

Invalid Storage Address. Instruction word, stack control block, or operand.

Specification Check. Even byte boundary violation (indirect address or stack control block).

**Soft Exception Trap Condition**

Stack Exception. Stack is empty. If AM equals 01, the contents of the register specified by the RB field are incremented.

Pop Doubleword (PD)
Refer to *Stack Operations* in Chapter 2 for additional information about the operation of this instruction and the associated stack control block.

PD \( \text{addr4,reg} \)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

The top element of a doubleword stack is popped from the stack and loaded into the register pair specified by the R field (R and R+1). The stack is defined by the stack control block pointed to by the effective address. (*Effective Address Generation* is explained in Chapter 2.)

If the R field equals 7, registers 7 and 0 are used.

**Programming Note.** If AM equals 01, the register specified by the RB field is incremented by two.

**Indicators**
No indicators are changed.

**Program Check Conditions**

Invalid Storage Address. Instruction word, stack control block, or operand.

Specification Check. Even byte boundary violation (indirect address, stack control block, or stack element).

**Soft Exception Trap Condition**

Stack Exception. Stack is empty. If the AM field equals 01, the contents of the register specified by the RB field are incremented.
**PSB**

**PSD**

**Push Byte (PSB)**

Refer to *Stack Operations* in Chapter 2 for additional information about the operation of this instruction and the associated stack control block.

PSB \( \text{reg,addr}4 \)

---

**Push Doubleword (PSD)**

Refer to *Stack Operations* in Chapter 2 for additional information about the operation of this instruction and the associated stack control block.

PSD \( \text{reg,addr}4 \)

---

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11101</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

\[ \text{Displacement 1} = 16 \]
\[ \text{Displacement 2} = 23, 24 \]

The least significant byte of the register specified by the R field is pushed into the stack. The stack is defined by the stack control block pointed to by the effective address. *(Effective Address Generation is explained in Chapter 2.)*

**Programming Note.** If AM equals 01, the register specified by the RB field is incremented by two.

**Indicators**

No indicators are changed.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word, stack control block, or operand.

**Specification Check.** Even byte boundary violation (indirect address or stack control block).

**Soft Exception Trap Condition**

**Stack Exception.** Stack is full. If AM equals 01, the contents of the register specified by the RB field are incremented.

---

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11101</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

\[ \text{Displacement 1} = 16 \]
\[ \text{Displacement 2} = 23, 24 \]

The doubleword operand contained in the register pair specified by the R field (R and R+1) is pushed into the stack. The stack is defined by the stack control block pointed to by the effective address. *(Effective Address Generation is explained in Chapter 2.)*

If the R field equals 7, registers 7 and 0 are used.

**Programming Note.** If AM equals 01, the register specified by the RB field is incremented by two.

**Indicators**

No indicators are changed.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word, stack control block, or operand.

**Specification Check.** Even byte boundary violation (indirect address, stack control block, or stack element).

**Soft Exception Trap Condition**

**Stack Exception.** Stack is full. If the AM field equals 01, the contents of the register specified by the RB field are incremented.
Push Word (PSW)
Refer to *Stack Operations* in Chapter 2 for additional information about the operation of this instruction and the associated stack control block.
PSW \( \text{reg,addr4}\)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>45</td>
<td>789</td>
<td>10121314</td>
<td>0100</td>
</tr>
</tbody>
</table>

The word operand contained in the register specified by the R field is pushed into the stack. The stack is defined by the stack control block pointed to by the effective address. *(Effective Address Generation is explained in Chapter 2.)*

*Programming Note.* If AM equals 01, the register specified by the RB field is incremented by two.

Indicators
No indicators are changed.

Program Check Conditions
Invalid Storage Address. Instruction word, stack control block, or operand.

Specification Check. Even byte boundary violation (indirect address, stack control block, or stack element).

Soft Exception Trap Condition
Stack Exception. Stack is full. If the AM field equals 01, the contents of the register specified by the RB field are incremented.

Pop Word (PW)
Refer to *Stack Operations* in Chapter 2 for additional information about the operation of this instruction and the associated stack control block.
PW \( \text{addr4,reg}\)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>045</td>
<td>789</td>
<td>10121314</td>
<td>0111</td>
</tr>
</tbody>
</table>

The top element of a word stack is popped from the stack and loaded into the register specified by the R field. The stack is defined by the stack control block pointed to by the effective address. *(Effective Address Generation is explained in Chapter 2.)*

*Programming Note.* If AM equals 01, the register specified by the RB field is incremented by two.

Indicators
No indicators are changed.

Program Check Conditions
Invalid Storage Address. Instruction word, stack control block, or operand.

Specification Check. Even byte boundary violation (indirect address, stack control block, or stack element).

Soft Exception Trap Condition
Stack Exception. Stack is empty. If the AM field equals 01, the contents of the register specified by the RB field are incremented.
RBTB

Reset Bits Byte (RBTB)

Register/Storage Format
RBTB addr4,reg
reg,addr4

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>= storage to register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>= register to storage</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This instruction operates either:
1. Storage to register (instruction bit 12 equals 0) or
2. Register to storage (instruction bit 12 equals 1).

Storage to Register. The specified bits are reset in the least significant byte of the register specified by the R1 field. The bit positions turned off correspond to the bit positions containing one-bits in the main storage byte location specified by the effective address. The remaining bits in the low-order byte of the register are unchanged. Also, bits 0–7 of the register and the storage operand are unchanged.

Register to Storage. The specified bits are reset in the main storage byte location specified by the effective address. The bits turned off correspond to the bit positions containing one-bits in the least significant byte of the register specified by the R field. The remaining bits in the storage location are unchanged. The register operand is unchanged.

Note. Effective Address Generation is explained in Chapter 2.

Indicators
Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions
Invalid Storage Address. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, R1 is incremented.
Specification Check. Even byte boundary violation (indirect address).

Storage/Storage Format
RBTB addr5,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>0 4 5</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

The address arguments generate the effective addresses of two operands in main storage. (Effective Address Generation is explained in Chapter 2.) The bit positions containing one-bits in byte operand 1 determine the bit positions turned off in byte operand 2. The remaining bits in operand 2 are unchanged. The result replaces operand 2. Operand 1 is unchanged.

Indicators
Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions
Invalid Storage Address. Instruction word or operand.
The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, R1 is incremented.
Specification Check. Even byte boundary violation (indirect address).
Reset Bits Doubleword (RBTD)

Register/Storage Format

RBTD addr4,reg
reg,addr4

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 1 0 1 0</td>
<td>0</td>
<td>4 5</td>
<td>7 8 9</td>
<td>10 11 12 13 15</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

0 = storage to register
1 = register to storage

This instruction operates either:
1. Storage to register (instruction bit 12 equals 0) or
2. Register to storage (instruction bit 12 equals 1)

Storage to Register. The specified bits are reset in the register pair specified by the R field (R and R+1). The bit positions turned off correspond to the bit positions containing one-bits in the doubleword main storage location specified by the effective address. The remaining bits in the register pair are unchanged. The storage operand is unchanged.

Register to Storage. The specified bits are reset in the doubleword main storage location specified by the effective address. The bit positions turned off correspond to the bit positions containing one-bits in the register pair specified by the R field (R and R+1). The remaining bits in the storage operand are unchanged. The register operand is unchanged. If the R field equals 7, registers 7 and 0 are used.

Note. Effective Address Generation is explained in Chapter 2.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

Specification Check. Even byte boundary violation (indirect address or operand address).
RBTW

Reset Bits Word (RBTW)

Register/Register Format

RBTW reg, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>001110</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

The bit positions containing one-bits in the register specified by the R1 field determine the bit positions turned off in the register specified by the R2 field. The remaining bits in the register specified by the R2 field are unchanged. The contents of the register specified by the R1 field are unchanged unless R1 and R2 specify the same register.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

No program checks occur.

Register/Storage Format

RBTW addr4, reg

reg, addr4

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>011001</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

0 = storage to register
1 = register to storage

This instruction operates either:

1. Storage to register (instruction bit 12 equals 0) or
2. Register to storage (instruction bit 12 equals 1)

Storage to Register. The specified bits are reset in the register specified by the R field. The bit positions turned off correspond to the bit positions containing one-bits in the main storage word location specified by the effective address. The remaining bits in the register are unchanged. The storage operand is unchanged.

Register to Storage. The specified bits are reset in the main storage word location specified by the effective address. The bit positions turned off correspond to the bit positions containing one-bits in the register specified by the R field. The remaining bits in the storage operand are unchanged. The register operand is unchanged.

Note. Effective Address Generation is explained in Chapter 2.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Storage to Register Long Format
RBTW longadr,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>8 10 11 12 15</td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

Address
16 31

The bit positions containing one-bits in the main storage word location specified by the effective address determine the bit positions turned off in the register specified by the R1 field. The remaining bits in the register specified by the R1 field are unchanged. The storage operand is unchanged.

The effective address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field to form a main storage address. If the R2 field equals zero, no register contributes to the address generation. The contents of R2 are not changed.

2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11 = 0 (direct address). The result from step 1 is the effective address.
   - Bit 11 = 1 (indirect address). The result from step 1 is the address of the main storage location that contains the effective address.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).

---

Storage/Storage Format
RBTW addr5,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>23</td>
<td>24</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Address/Displacement
16 23 24 31

Address/Displacement
32 39 40 47

The address arguments generate the effective addresses of two operands in main storage. (Effective Address Generation is explained in Chapter 2.) The bit positions containing one-bits in word operand 1 determine the bit positions turned off in word operand 2. The remaining bits in operand 2 are unchanged. The result replaces operand 2. Operand 1 is unchanged.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.
The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

Specification Check. Even byte boundary violation (indirect address or operand address).
RBTWI

Reset Bits Word Immediate (RBTWI)
Register Immediate Long Format

RBTWI word,reg[,reg]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 1</td>
<td>4 5 7 8 10 11 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Immediate

16 31

The bit positions containing one-bits in the immediate field determine the bit positions to be reset. These bit positions are reset in the operand from the contents of the register specified by the R1 field. The result is placed in the register specified by the R2 field.

Example:

Contents of immediate field 0000 0000 0000 1111
Contents of R1 register 0101 0101 0101 0101
Result in R2 register 0101 0101 0101 0000

The contents of the register specified by the R1 field are unchanged unless R1 and R2 specify the same register.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word.

Storage Immediate Format

RBTWI word,addr4

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>0 4 5 7 8 9 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Immediate

16 31

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>0 4 5 7 8 9 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address/Displacement

16 23 24 31

Immediate

32 47

The bit positions containing one-bits in the immediate field determine the bit positions turned off in the main storage location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) The immediate operand is unchanged. Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Subtract Byte (SB)

\[ SB \text{ reg,addr4} \]
\[ addr4,\text{reg} \]

**Operation code**

\[
\begin{array}{cccccc}
\text{I} & \text{I} & \text{I} & \text{I} & \text{I} & \text{I} \\
\text{R} & \text{RB} & \text{AM} & \text{X} & \text{Function} \\
0 & 4 & 5 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 15 \\
\end{array}
\]

1 = result to storage
0 = result to register

Displacement 1
Displacement 2

A subtract operation is performed between the least significant byte of the register specified by the R field and the location specified by the effective address in main storage. (See Effective Address Generation in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand and high-order byte of the register are unchanged.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the byte. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one byte; i.e., if the difference is less than \(-2^7\) or greater than \(+2^7 - 1\).

If an overflow occurs, the result contains the correct low-order eight bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address).

Subtract Carry Indicator (SCY)

\[ SCY \text{ reg} \]

**Operation code**

\[
\begin{array}{cccccc}
\text{I} & \text{I} & \text{I} & \text{I} & \text{I} & \text{I} \\
\text{R} & \text{RB} & \text{AM} & \text{X} & \text{Function} \\
0 & 4 & 5 & 7 & 8 & 9 & 10 & 11 & 15 \\
\end{array}
\]

The value of the carry indicator on entry is subtracted from the contents of the register specified by the R2 field. The result is placed in the register specified by the R2 field. Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

**Programming Note.** This instruction can be used when subtracting multiple word operands. See Indicators – Multiple Word Operands in Chapter 2.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word: i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15} - 1\).

If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even.** Unchanged.

**Negative.** Changed to reflect the result.

**Zero.** If on at entry, changed to reflect the result. If off at entry, it remains off.

**Program Check Conditions**

No program checks occur.
Subtract Doubleword (SD)

**Register/Storage Format**

<table>
<thead>
<tr>
<th>SD</th>
<th>reg, addr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr4, reg</td>
<td></td>
</tr>
</tbody>
</table>

**Storage/Storage Format**

| SD | addr5, addr4 |

**Operation code**

<table>
<thead>
<tr>
<th>1 0 1 0</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

1 = result to storage
0 = result to register

A subtract operation is performed between the register pair specified by the R field (R and R+1) and the doubleword in main storage specified by the effective address. (See *Effective Address Generation* in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

If the R field equals 7, register 7 and register 0 are used.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the doubleword. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in the doubleword; i.e., if the difference is less than $-2^{31}$ or greater than $+2^{31} - 1$.

If an overflow occurs, the result contains the correct low-order 32 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).

The address arguments generate the effective addresses of two operands in main storage. (See *Effective Address Generation* in Chapter 2.) Doubleword operand 1 is subtracted from doubleword operand 2. The result replaces operand 2. Operand 1 is unchanged.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the doubleword. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in the doubleword; i.e., if the difference is less than $-2^{31}$ or greater than $+2^{31} - 1$.

If an overflow occurs, the result contains the correct low-order 32 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or operand.

The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Set Console Data Lights (SECON)
SECON reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 1 0 0 0</td>
<td></td>
<td>1 0 0 0 0</td>
</tr>
</tbody>
</table>

The contents of the register specified by R2 are stored in the console data lights. The contents of the register are unchanged.

Bits 5-7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

If the Programmer Console is not installed, the instruction performs no operation.

Indicators
All indicators are unchanged.

Program Check Conditions
Privilege Violate. Privileged instruction.

Set Interrupt Mask Register (SEIMR)
SEIMR addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1 0 0 0</td>
<td></td>
<td>0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

Bits 0-3 of the word location in main storage specified by the effective address are loaded into the interrupt mask register. (Effective Address Generation is explained in Chapter 2.) Bits 4-15 of the word in main storage are not used. The contents of main storage are unchanged.

Bits 5-7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

The mask is represented in a bit significant manner as follows:

<table>
<thead>
<tr>
<th>Mask bit</th>
<th>Interrupt level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

A mask bit set to "1" indicates that the level is enabled. A mask bit set to "0" indicates that the level is disabled.

Indicators
All indicators are unchanged.

Program Check Conditions
Invalid Storage Address. Instruction word or operand.

Privilege Violate. Privileged instruction.

Specification Check. Even byte boundary violation (indirect address or operand address).
SEIND

Set Indicators (SEIND)
SEIND reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0 0 0 0</td>
<td>0 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

Bits 0–4 of the register specified by the R2 field are loaded into bits 0–4 of the current level status register (indicators). Bits 5–15 of the register specified by R2 are ignored. Bits 5–15 of the level status register are unchanged.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

The following table shows the indicator bits of the level status register (LSR):

<table>
<thead>
<tr>
<th>LSR bit</th>
<th>Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Even</td>
</tr>
<tr>
<td>1</td>
<td>Carry</td>
</tr>
<tr>
<td>2</td>
<td>Overflow</td>
</tr>
<tr>
<td>3</td>
<td>Negative</td>
</tr>
<tr>
<td>4</td>
<td>Zero</td>
</tr>
</tbody>
</table>

Indicators
Changed as specified by the R2 register.

Program Check Conditions
No program checks occur.
Set Level Block (SELB)

Execution of the SELB instruction can cause the processor to change levels. Also, the processor may exit supervisor state. For additional information concerning the processor action when executing this instruction, refer to *Program Controlled Level Switching* in Chapter 3.

SELB reg,addr4

This instruction loads a level status block (LSB) from 11 words of main storage beginning with the location specified by the effective address. (*Effective Address Generation* is explained in Chapter 2.) The contents of the storage locations are not changed.

The register specified by the R field contains the binary encoded level of the LSB to be loaded. The binary encoded level is placed in bits 14–15 of the register. Bits 1–13 are not used and must be zero to avoid future code obsolescence.

Bit 0 of the register specified by the R field is the inhibit trace (IT) interrupt bit. If bit 0 is a one and the trace bit (bit 10) in the LSR of the target LSB is a one, then both the Set Level Block instruction and the instruction pointed to by the IAR in the target LSB are executed before trace interrupts are allowed.

If bit 0 is zero and the trace bit in the LSR of the target LSB is a one, the Set Level Block instruction is executed and then trace interrupts are allowed.

The target LSB is defined by either (1) the effective address, if the in-process bit is set to one in the LSR of the target LSB and the specified R field level is higher than or equal to the current level, or (2) the currently active LSB when condition (1) is not met.

**Level Status Block Format**

EA IAR
Zeros
LSR
Register 0
Register 1
Register 2
Register 3
Register 4
Register 5
Register 6
Register 7
EA+20 (+14 hex)

EA=effective address

**Format of Register Specified by R in Instruction**

<table>
<thead>
<tr>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

**Programming Notes.**

1. The Set Level Block instruction with the IT bit equal to one should be used to return from the trace interrupt routine and from a class interrupt routine when the instruction causing the interrupt is to be reexecuted. This is necessary to prevent a double trace of the instruction.

2. If the Set Level Block instruction sets the current level in-process bit to zero and the current level trace bit to one, no trace interrupt occurs as the level is exited.

3. The registers and LSR for the current level are not changed if the specified R field level is other than the current level.

4. If the AM field equals 01, the contents of the register specified by the RB field are incremented by 2.

**Indicators**

All indicators are unchanged if the specified level is other than the current level.

**Program Check Conditions**

**Invalid Storage Address.** Instruction word or level status block. The instruction is terminated.

**Privilege Violate.** Privileged instruction.

**Specification Check.** Even byte boundary violation (indirect address or level status block address). The instruction is terminated.
**SFED**
**SFEN**

**Scan Byte Field Equal and Decrement (SFED)**

**Scan Byte Field Equal and Increment (SFEN)**

SFED: `reg,(reg)`  
SFEN: `reg,(reg)`

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>I</th>
<th>D</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>

1 for SFED or SFEN  
0 for SFED; decrement contents of R2.  
1 for SFEN; increment contents of R2.

This instruction compares a field in main storage against a single byte contained in a register. This comparison is made one byte at a time. Register 7 contains the number of bytes to be compared. This number is decremented after each byte is compared.

The register specified by R1 contains, in bits 8–15, the single byte of operand 1. The register specified by R2 contains the starting address of operand 2. Operand 1 is subtracted from operand 2, but neither operand is changed.

After each byte is compared, the address in R2 is incremented or decremented (determined by bit 13 of the instruction). The operation terminates when either:

1. An equal condition is detected, or
2. The number of bytes specified in register 7 has been compared.

When an equality occurs, the address in the register specified by R2 points to the next operand to be compared, but the count in R7 is not updated.

Bit 11 of the instruction is not used and must be set to zero to avoid future code obsolescence.

See Compare Byte Field Equal and Decrement (CFED) and Compare Byte Field Equal and Increment (CFEN) for other versions of this machine instruction.

**Notes.**
1. Variable field length instructions can be interrupted. When this occurs and the interrupted level resumes operation, the processor treats the uncompleted instruction as a new instruction with the remaining byte count specified in register 7.
2. If the specified count in R7 is zero, the instruction performs no operation (no-op).

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the byte. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one byte; i.e., if the difference is less than $-2^7$ or greater than $+2^7-1$.

**Even, Negative, and Zero.** Changed to reflect the result.

**Program Check Conditions**

**Invalid Storage Address.** Operand. The instruction is terminated.
Scan Byte Field Not Equal and Decrement (SFNED)

Scan Byte Field Not Equal and Increment (SFNEN)

SFNED   reg.(reg)
SFNEN   reg.(reg)

This instruction compares a field in main storage against a single byte contained in a register. This comparison is made one byte at a time. Register 7 contains the number of bytes to be compared. This number is decremented after each byte is compared.

The register specified by R1 contains, in bits 8–15, the single byte of operand 1. The register specified by R2 contains the starting address of operand 2. Operand 1 is subtracted from operand 2, but neither operand is changed. After each byte is compared, the address in R2 is incremented or decremented (determined by bit 13 of the instruction). The operation terminates when either:

1. An unequal condition is detected, or
2. The number of bytes specified in register 7 has been compared.

When an inequality occurs, the address in the register specified by R2 points to the next operand to be compared, but the count in R7 is not updated.

Bit 11 of the instruction is not used and must be set to zero to avoid future code obsolescence.

See Compare Byte Field Not Equal and Decrement (CFNED) and Compare Byte Field Not Equal and Increment (CFNEN) for other versions of this machine instruction.

Notes.
1. Variable field length instructions can be interrupted. When this occurs and the interrupted level resumes operation, the processor treats the uncompleted instruction as a new instruction with the remaining byte count specified in register 7.
2. If the specified count in R7 is zero, the instruction performs no operation (no-op).

Indicators

Carry. Turned on by the detection of a borrow beyond the high-order bit position of the byte. If no borrow is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the difference cannot be represented in one byte; i.e., if the difference is less than \(-2^{7}\) or greater than \(+2^{7}-1\).

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Operand. The instruction is terminated.
Shift Left Circular (SLC)

Immediate Count Format

SLC cnt16,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0</td>
<td>4</td>
<td>5 7 8</td>
<td>12 13 15</td>
</tr>
</tbody>
</table>

The bits in the register specified by the R field are shifted left by the number of bit positions specified in the count field. The bits shifted out of the high-order bit (bit 0) re-enter at the low-order bit (bit 15). A count of zero causes no shifting to take place.

Although the register to be shifted contains only 16 bits, shift count values of 0–31 may be specified. Shift counts greater than 16 provide an effective shift of modulo 16.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the final contents of the register.

Program Check Conditions

No program checks occur.

Example:

Instruction

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0</td>
<td>4</td>
<td>5 7 8</td>
<td>12 13 15</td>
</tr>
</tbody>
</table>

R3 before shift

0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1

R3 after shift

0 0 0 1 0 0 1 0 0 0 1 1 0 0 0

Count in Register Format

SLC reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4</td>
<td>5 7 8</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

The bits in the register specified by the R1 field are shifted left by the number of bits specified by the shift count. This count is obtained from bits 8 through 15 of the register specified by the R2 field.

The contents of the register specified by the R2 field are unchanged unless the R1 and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the register to be shifted contains only 16 bits, shift count values of 0–255 may be specified. Shift counts greater than 16 provide an effective shift of modulo 16.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the final contents of the register specified by the R1 field.

Program Check Conditions

No program checks occur.

Example:

Instruction

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4</td>
<td>5 7 8</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

R3 before shift

0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1

R4 contains shift count

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Count = 8

R3 before shift

0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1

R3 after shift

0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1

6-78  GA34-0022
### Shift Left Circular Double (SLCD)

#### Immediate Count Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0</td>
<td>5</td>
<td>7 8 12 13 15</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R field and R+1 are shifted left by the number of bit positions specified in the count field.

Within the register pair, the register specified by the R field contains the high-order word (bits 0–15); the register specified by R+1 contains the low-order word (bits 16–31). The bits shifted out of the high-order bit (bit 0) re-enter at the low-order bit (bit 31).

If the count is zero, no shifting occurs. If the R field equals 7, registers 7 and 0 are used for the register pair.

#### Indicators

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the final contents of the two registers.

#### Program Check Conditions

No program checks occur.

#### Example:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation code</th>
<th>R1</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 1 1 0</td>
<td>0 1 1</td>
<td>1 0 1 0 0</td>
<td>1 0 0</td>
</tr>
</tbody>
</table>

R3  Count = 20

Register pair before shift

<table>
<thead>
<tr>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 1 0 0 1 0 0 0 1 1</td>
<td>0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1</td>
</tr>
</tbody>
</table>

Register pair after shift

<table>
<thead>
<tr>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 0 1 1 0 0 1 1 1 0 0 0 0</td>
<td>0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 0 0</td>
</tr>
</tbody>
</table>

Instructions 6-79
**Count in Register Format**

**SLCD** \[\text{reg},\text{reg}\]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R1 field and R1+1 are shifted left by the number of bits specified by the shift count. This count is obtained from bits 8 through 15 of the register specified by the R2 field.

Within the register pair, the register specified by the R1 field contains the high-order word (bits 0–15); the register specified by R1+1 contains the low-order word (bits 16–31). The bits shifted out of the high-order bit (bit 0) re-enter at the low-order bit (bit 31).

If the count is zero, no shifting occurs. If the R1 field equals 7, registers 7 and 0 are used for the register pair.

The contents of the register specified by the R2 field are unchanged unless the R1 (or R1 + 1) and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the registers to be shifted represent 32 bits, shift count values of 0–255 may be specified. Shift count values greater than 32 provide an effective shift of modulo 32.

**Indicators**

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the final contents of the two registers.

**Program Check Conditions**

No program checks occur.

**Example:**

```
Instruction

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110</td>
<td>11</td>
<td>10</td>
<td>01000000</td>
</tr>
</tbody>
</table>

R4 contains shift count

<table>
<thead>
<tr>
<th>0000000000000000101000</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000000101000</td>
</tr>
</tbody>
</table>

Count = 20
```

Register pair before shift

R7

```
00000000000000001001000011101000101011001111
```

Register pair after shift

R7

```
010101100111000001100100100011010100
```
Shift Left Logical (SLL)

Immediate Count Format

SLL  cnt16,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0</td>
<td>5</td>
<td>7  8</td>
<td>12 13 15</td>
</tr>
</tbody>
</table>

The bits in the register specified by the R field are shifted left by the number of bit positions specified in the count field. The vacated low-order bit positions of the register are set to zero. A count of zero causes no shifting to take place.

Although the register to be shifted contains only 16 bits, shift count values of 0–31 may be specified. Shift counts greater than 17 provide an effective shift of 17.

Indicators

Carry. Set to reflect the last bit shifted out of bit 0. If the count is zero, the carry indicator is reset.

Overflow. First reset, then set to a one if the most significant bit in the register (bit 0) has changed during the operation.

Even, Carry, and Overflow. Changed to reflect the final contents of the register.

Program Check Conditions

No program checks occur.

Count in Register Format

SLL  reg.reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>5</td>
<td>8</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

The bits in the register specified by the R1 field are shifted left by the number of bits specified by the shift count. This count is obtained from bits 8–15 of the register specified by the R2 field. The vacated low-order bits of the register specified by the R1 field are set to zero.

The contents of the register specified by the R2 field are unchanged unless the R1 and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the register shifted contains only 16 bits, shift count values of 0–255 may be specified. Shift counts greater than 17 provide an effective shift of 17.

Indicators

Carry. Set to reflect the last bit shifted out of bit 0. If the count is zero, the carry indicator is reset.

Overflow. First reset, then set to a one if the most significant bit in the register (bit 0) has changed during the operation.

Even, Negative, and Zero. Changed to reflect the final contents of the register specified by the R1 field.

Program Check Conditions

No program checks occur.
SLLD

Shift Left Logical Double (SLLD)

Immediate Count Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0</td>
<td>7</td>
<td>8</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R field and R+1 are shifted left by the number of bit positions specified in the count field. The vacated low-order bits of the register pair are set to zero.

Within the register pair, the register specified by the R field contains the high-order word (bits 0–15); the register specified by R+1 contains the low-order word (bits 16–31).

If the shift count is zero, no shifting occurs. If the R field equals 7, registers 7 and 0 are used for the register pair.

Indicators

**Carry.** First reset, then set to reflect the last bit shifted out of the most significant bit position (bit 0) in the register pair.

**Overflow.** First reset, then set to one if the most significant bit position in the register pair (bit 0) has changed during the operation.

**Even, Negative, and Zero.** Changed to reflect the final contents of the two registers.

Program Check Conditions

No program checks occur.

Count in Register Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4 5 7 8</td>
<td>10 11 15</td>
<td></td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R1 field and R1+1 are shifted left by the number of bit positions specified by the shift count. This count is obtained from bits 8 through 15 of the register specified by the R2 field. The vacated low-order bit positions of the register pair are set to zero.

Within the register pair, the register specified by the R1 field contains the high-order word (bits 0–15); the register specified by R1+1 contains the low-order word (bits 16–31).

If the shift count is zero, no shifting occurs. If the R1 field equals 7, registers 7 and 0 are used for the register pair.

The contents of the register specified by the R2 field are unchanged unless the R1 (or R1+1) and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the registers to be shifted represent 32 bits, shift count values of 0–255 may be specified. Shift counts greater than 33 provide an effective shift of 33.

Indicators

**Carry.** First reset, then set to reflect the last bit shifted out of the most significant bit position (bit 0) in the register pair.

**Overflow.** First reset, then set to a one if the most significant bit in the register pair (bit 0) has changed during the operation.

**Even, Negative, and Zero.** Changed to reflect the final contents of the two registers.

Program Check Conditions

No program checks occur.
Shift Left and Test (SLT)

SLT  reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>11</td>
<td>15</td>
</tr>
</tbody>
</table>

The bits in the register specified by the R1 field are shifted left. The vacated low-order bit positions of the register are set to zero.

Shifting continues until either one of the following occurs:

1. The number of bits specified by the shift count have been shifted. This count is obtained from bits 8 through 15 of the register specified by the R2 field. No shifting occurs if the shift count is zero.
2. A one-bit is shifted from the high-order bit (bit 0) to the carry indicator. In this case, the remaining shift count is loaded into bits 8 through 15 of the register specified by the R2 field.

Bits 0 through 7 of the register specified by the R2 field are unchanged; these bits must be set to zero to avoid future code obsolescence.

If the R1 and R2 fields specify the same register, the bits in the register are shifted as specified and, when shifting is complete, the remaining shift count replaces the shifted result.

Although the register to be shifted contains only 16 bits, shift count values of 0–255 may be specified.

Indicators

Carry. Set to reflect the last bit shifted out of bit 0. If the count is zero, the carry indicator is reset.

Overflow. First reset, then set to a one if the most significant bit in the register (bit 0) has changed during the operation.

Even, Negative, and Zero. Changed to reflect the final contents of the register specified by the R2 field.

Program Check Conditions
No program checks occur.

Shift Left and Test Double (SLTD)

SLTD  reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4</td>
<td>5</td>
<td>7 8</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R1 field and R1+1 are shifted left. The vacated low-order bit positions of the register pair are set to zero.

Shifting continues until either one of the following occurs:

1. The number of bits specified by the shift count have been shifted. This count is obtained from bits 8 through 15 of the register specified by the R2 field. No shifting occurs if the shift count is zero.
2. A one-bit is shifted from the high-order bit to the carry indicator. In this case, the remaining shift count is loaded into bits 8 through 15 of the register specified by the R2 field.

Bits 0 through 7 of the register specified by the R2 field are unchanged; these bits must be set to zero to avoid future code obsolescence.

Within the register pair, the register specified by the R1 field contains the high-order word (bits 0–15); the register specified by R1+1 contains the low-order word (bits 16–31). If the R1 field equals 7, registers 7 and 0 are used for the register pair.

If the R1 (or R1+1) and R2 fields specify the same register, the bits in the register are shifted as specified and, when shifting is complete, the remaining shift count replaces the shifted result.

Although the registers to be shifted contain only 32 bits, shift count values of 0–255 may be specified.

Indicators

Carry. Set to reflect the last bit shifted out of bit 0. If the count is zero, the carry indicator is reset.

Overflow. First reset, then set to a one if the most significant bit in the register (bit 0) has changed during the operation.

Even, Negative, and Zero. Changed to reflect the final contents of the register specified by the R2 field.

Program Check Conditions
No program checks occur.
Shift Right Arithmetic (SRA)

Immediate Count Format

\[ \text{SRA cnt16,reg} \]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  0  1  1  0</td>
<td>4</td>
<td>5  7  8</td>
</tr>
<tr>
<td>0  1  1  1  1</td>
<td>12 13 15</td>
<td></td>
</tr>
</tbody>
</table>

The bits in the register specified by the R field are shifted right by the number of bit positions specified in the count field. The value of the sign (the high-order bit) is entered into the vacated high-order bit positions of the register specified by the R field. If the shift count is zero, no shifting takes place.

Although the register to be shifted contains only 16 bits, shift count values of 0–31 may be specified. Shift counts greater than 16 provide an effective shift of 16.

Indicators

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the final contents of the register.

Program Check Conditions

No program checks occur.

---

Count in Register Format

\[ \text{SRA reg,reg} \]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  1  1  1  0</td>
<td>4</td>
<td>5</td>
<td>7  8  10 11 15</td>
</tr>
<tr>
<td>0  1  0  0  1  1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The bits in the register specified by the R1 field are shifted right by the number of bit positions specified by the shift count. This count is obtained from bits 8 through 15 of the register specified by the R2 field. The value of the sign (the high-order bit) is entered into the vacated high-order bit positions of the register specified by the R1 field. If the shift count is zero, no shifting takes place.

The contents of the register specified by the R2 field are unchanged unless the R1 and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the register to be shifted is 16 bits, shift count values of 0–255 may be specified. Shift counts greater than 16 provide an effective shift of 16.

Indicators

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the final contents of the register specified by the R1 field.

Program Check Conditions

No program checks occur.
Shift Right Arithmetic Double (SRAD)

Immediate Count Format
SRAD cnt31,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 i i i</td>
<td>4</td>
<td>5 7 8</td>
</tr>
<tr>
<td>0 1</td>
<td>12</td>
<td>13 15</td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R field and R+1 are shifted right by the number of bit positions specified in the count field. The value of the sign (the high-order bit) is entered into the vacated high-order bit positions of the register pair.

Within the register pair, the register specified by the R field contains the high-order word (bits 0–15); the register specified by R+1 contains the low-order word (bits 16–31).

If the shift count is zero, no shifting occurs. If the R field equals 7, registers 7 and 0 are used for the register pair.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the final contents of the register pair.

Program Check Conditions
No program checks occur.

Count in Register Format
SRAD reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>5</td>
<td>7</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R1 field and R1+1 are shifted right by the number of bit positions specified by the shift count. This count is obtained from bits 8 through 15 of the register specified by the R2 field. The value of the sign (the high-order bit) is entered into the vacated high-order bit positions of the register pair.

Within the register pair, the register specified by the R1 field contains the high-order word (bits 0–15); the register specified by R1+1 contains the low-order word (bits 16–31).

If the shift count is zero, no shifting occurs. If the R field equals 7, registers 7 and 0 are used for the register pair.

The contents of the register specified by the R2 field are unchanged unless the R1 (or R1+1) and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the registers to be shifted represent 32 bits, shift count values of 0–255 may be specified. Shift counts greater than 32 provide an effective shift of 32.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the final contents of the register pair.

Program Check Conditions
No program checks occur.
**SRL**

**Immediate Count Format**

SRL cnt16,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1 0</td>
<td>5</td>
<td>8</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

The bits in the register specified by the R field are shifted right by the number of bit positions specified in the count field. The vacated high-order bit positions of the register are set to zero. A count of zero causes no shifting to take place.

Although the register to be shifted contains only 16 bits, shift count values of 0–31 may be specified. Shift counts greater than 16 provide an effective shift of 16.

**Indicators**

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the final contents of the register.

**Program Check Conditions**

No program checks occur.

---

**Count in Register Format**

SRL reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4 5</td>
<td>7 8</td>
<td>1 0 0 1 0</td>
</tr>
</tbody>
</table>

The bits in the register specified by the R1 field are shifted right by the number of bit positions specified by the shift count. This count is obtained from bits 8 through 15 of the register specified by the R2 field. The vacated high-order bit positions of the register specified by the R1 field are set to zero. A count of zero causes no shifting to take place.

The contents of the register specified by the R2 field are unchanged unless the R1 and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the register to be shifted contains only 16 bits, shift count values of 0–255 may be specified. Shift counts greater than 16 provide an effective shift of 16.

**Indicators**

**Carry and Overflow.** Unchanged.

**Even, Negative, and Zero.** Changed to reflect the final contents of the register specified by the R1 field.

**Program Check Conditions**

No program checks occur.
Shift Right Logical Double (SRLD)

**Immediate Count Format**

SRLD cnt31,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0</td>
<td>4</td>
<td>5</td>
<td>7 8 12 13 15</td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R field and R+1 are shifted right by the number of bit positions specified in the count field. The vacated high-order bits of the register pair are set to zero.

Within the register pair, the register specified by the R field contains the high-order word (bits 0–15); the register specified by R+1 contains the low-order word (bits 16–31).

If the shift count is zero, no shifting occurs. If the R field equals 7, registers 7 and 0 are used for the register pair.

**Indicators**

*Carry and Overflow*. Unchanged.

*Even, Negative, and Zero*. Changed to reflect the final contents of the register.

**Program Check Conditions**

No program checks occur.

---

**Count in Register Format**

SRLD reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0</td>
<td>4  5 7 8</td>
<td>10 11</td>
<td>15</td>
</tr>
</tbody>
</table>

The bits in the register pair specified by the R1 field and R1+1 are shifted right by the number of bit positions specified by the shift count. This count is obtained from bits 8 through 15 of the register specified by the R2 field. The vacated high-order bits of the register pair are set to zero.

Within the register pair, the register specified by the R1 field contains the high-order word (bits 0–15); the register specified by R1+1 contains the low-order word (bits 16–31).

If the shift count is zero, no shifting occurs. If the R1 field equals 7, registers 7 and 0 are used for the register pair.

The contents of the register specified by the R2 field are unchanged unless the R1 (or R1+1) and R2 fields specify the same register. In this case, the register contents are shifted as specified.

Although the registers to be shifted represent 32 bits, shift count values of 0–255 may be specified. Shift counts greater than 32 provide an effective shift of 32.

**Indicators**

*Carry and Overflow*. Unchanged.

*Even, Negative, and Zero*. Changed to reflect the final contents of the register pair.

**Program Check Conditions**

No program checks occur.
Store Multiple (STM)
Refer to Stack Operations in Chapter 2 for additional information about the operation of this instruction. The STM instruction is used in conjunction with the Load Multiple and Branch (LMB) instruction described previously in this chapter.

STM reg,addr4[,]abcnt

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0</td>
<td>0 5 7 8 9 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RL</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>18 19 31</td>
</tr>
</tbody>
</table>

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0</td>
<td>0 5 7 8 9 10 11 12 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address/Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displacement 1</td>
</tr>
<tr>
<td>Displacement 2</td>
</tr>
<tr>
<td>16 22 24 31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RL</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>34 35 47</td>
</tr>
</tbody>
</table>

The STM instruction stores the contents of a specified number of registers for the current level into a stack. This stack is defined by the stack control block pointed to by the effective address. (Effective Address Generation is explained in Chapter 2.)

The RL field specifies the last register to be stored. Register 7 is stored first, then registers 0 through the register specified by RL. If RL specifies register 7, only register 7 is stored.

The N field specifies the number of words to be allocated in the stack as a dynamic work area. A value of zero is valid.

The new top element address of the stack (incremented by two) is loaded into the last register stored; that is, the register specified by RL. This address points to the low storage end of the dynamic work area (or the last register stored if N=0).

Bits 5-7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Programming Note. If the AM field equals 01, the contents of the register specified by the RB field are incremented by 2.

Indicators
No indicators are changed.

Program Check Conditions

Invalid Storage Address. Instruction word or stack control block. The instruction is terminated.

Specification Check. Even byte boundary violation (indirect address, stack control block, or stack element).

Soft Exception Trap Condition

Stack Exception.
1. Stack is full.
2. Stack cannot contain the number of words to be stored; that is:
   a. Number of words specified by the N field, plus
   b. The number of registers to be moved, plus
   c. One control word.

If the AM field equals 01, the contents of the register specified by the RB field are incremented.
Stop (STOP)

STOP [ubyte]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Function</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0 1 0 0</td>
<td></td>
<td>0 1 1 0 0</td>
</tr>
</tbody>
</table>

The parameter field is ignored by the hardware, and may be used for software flags or indicators.

This instruction is executed only when the Programmer Console is installed and the Mode switch is in the Diagnostic position. Otherwise this instruction performs no operation (no-op). The processor enters the stop state following execution of this instruction. The indicators are unchanged.

**Indicators**
No indicators are changed.

**Program Check Conditions**
No program checks occur.

Supervisor Call (SVC)

Execution of this instruction causes a *class interrupt*. Additional information appears in Chapter 3.

SVC ubyte

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Function</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0 0 0 0</td>
<td></td>
<td>0 1 1 0 0</td>
</tr>
</tbody>
</table>

The instruction address register is incremented by two. The current level status block (LSB) is stored starting at the main storage location specified by the contents of the SVC LSB pointer that resides in main storage location 0010 hexadecimal. The instruction also causes the following events:

- The summary mask (LSR bit 11) is disabled.
- Supervisor state (LSR bit 8) is turned on.
- Trace (LSR bit 10) is turned off.

The parameter field (bits 8–15) is under control of the Programming System. This field is loaded into the low-order byte of register 1. The high-order byte of register 1 is set to zero.

Subsequently, the contents of main storage location 0012 hexadecimal (SVC start instruction address) are loaded into the instruction address register, becoming the address of the next instruction to be fetched.

**Indicators**
No indicators are changed.

**Program Check Conditions**

**Specification Check.** Even byte boundary violation (LSB or SIA pointers).
Subtract Word (SW)

Register/Register Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110</td>
<td></td>
<td></td>
<td>01010</td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are subtracted from the contents of the register specified by the R2 field. The result is placed in the register specified by the R2 field. The contents of the register specified by the R1 field remain unchanged unless R1 and R2 specify the same register.

Indicators

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the register. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15}\) - 1.

If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

No program checks occur.

Register/Storage Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111</td>
</tr>
</tbody>
</table>

A subtract operation is performed between the register specified by the R field and the location specified by the effective address in main storage. (See Effective Address Generation in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

Indicators

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15}\) - 1.

If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Storage to Register Long Format

**SW** longaddr.reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>01101</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

0 = direct address
1 = indirect address

Address

The contents of the main storage word location specified by an effective address are subtracted from the contents of the register specified by the R1 field. The result is placed in the register specified by the R1 field.

The effective main storage address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field. If the R2 field equals zero, no register contributes to the address generation.
2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11=0 (direct address). The result from step 1 is the effective address.
   - Bit 11=1 (indirect address). The result from step 1 is the address of the main storage location that contains the effective address.

Indicators

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15}\). If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

**Specification Check.** Even byte boundary violation (indirect address or operand address).

Storage/Storage Format

**SW** addr5,addr4

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Address

The address arguments generate the effective addresses of two operands in main storage. (See *Effective Address Generation* in Chapter 2.) Word operand 1 is subtracted from word operand 2. The result replaces operand 2.

Indicators

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than \(-2^{15}\) or greater than \(+2^{15}\). If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even, Negative, and Zero.** Changed to reflect the result.

Program Check Conditions

**Invalid Storage Address.** Instruction word or operand. The instruction is terminated. If AM1 equals 01 and the operand 2 effective address is invalid, RB1 is incremented.

**Specification Check.** Even byte boundary violation (indirect address or operand address).
Subtract Word with Carry (SWCY)

**SWCY** reg,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>7</td>
<td>8</td>
<td>10 11</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

If the carry indicator is on at entry (denoting a borrow), a positive one is subtracted from the contents of the register specified by the R2 field. Then the contents of R1 are subtracted from the intermediate result. If the carry indicator is off at entry, the contents of R1 are subtracted from the contents of the register specified by R2. The contents of the register specified by the R1 field are unchanged unless R1 and R2 specify the same register. The final result replaces the contents of the register specified by the R2 field.

*Programming Note.* This instruction can be used when subtracting multiple word operands. See *Indicators – Multiple Word Operands* in Chapter 2.

**Indicators**

**Carry.** Turned on by the detection of a borrow beyond the high-order bit position of the register. If no borrow is detected, the carry indicator is reset.

**Overflow.** Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than $-2^{15}$ or greater than $+2^{15}-1$.

If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

**Even.** Unchanged.

**Zero.** If on at entry, set to reflect the result. If off at entry, remains off.

**Negative.** Changed to reflect the result.

**Program Check Conditions**

No program checks occur.
Subtract Word Immediate (SWI)

Register Immediate Long Format
SWI word,reg[,reg]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Immediate

The immediate field is subtracted from the contents of the register specified by the R1 field. The result is placed in the register specified by the R2 field. The contents of the register specified by the R1 field are unchanged unless R1 and R2 specify the same register.

Indicators

Carry. Turned on by the detection of a borrow beyond the high-order bit position of the register. If no borrow is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than $-2^{15}$ or greater than $+2^{15}-1$.

If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word.

Storage Immediate Format

SWI word,addr4

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>0</td>
<td>1 1 1 0</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

Immediate

Format with appended word for effective addressing (AM = 10 or 11)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>0</td>
<td>1 1 1 0</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

Address/Displacement

Displacement 1

Displacement 2

Immediate

The immediate field is subtracted from the contents of the main storage location specified by the effective address. (See Effective Address Generation in Chapter 2.) The result replaces the contents of the storage location specified by the effective address.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

The immediate operand is unchanged.

Indicators

Carry. Turned on by the detection of a borrow beyond the high-order bit position of the word. If no borrow is detected, the carry indicator is reset.

Overflow. Cleared, then turned on if the difference cannot be represented in one word; i.e., if the difference is less than $-2^{15}$ or greater than $+2^{15}-1$.

If an overflow occurs, the result contains the correct low-order 16 bits of the difference; the carry indicator contains the complement of the high-order (sign) bit.

Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
The bit displacement is added to the byte address contained in the register specified by the R field to form an effective bit address. The bit displacement field is an unsigned six-bit binary integer. The bit at the effective bit address is tested, and the zero and negative indicators are set to reflect the result.

**Indicators**

**Zero and Negative.** First reset, then set as follows:

<table>
<thead>
<tr>
<th>Value of tested bit</th>
<th>Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
</tr>
</tbody>
</table>

**Even, Carry, and Overflow.** Unchanged.

**Program Check Conditions**

**Invalid Storage Address.** Operand.

---

The bit displacement is added to the byte address contained in the register specified by the R field to form an effective bit address. The bit displacement field is an unsigned six-bit integer.

The bit at the effective bit address is tested, and the zero and negative indicators are set to reflect the result.

Following the preceding test, the addressed bit is unconditionally set to zero.

**Indicators**

**Zero and Negative.** First reset, then set as follows:

<table>
<thead>
<tr>
<th>Value of tested bit</th>
<th>Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
</tr>
</tbody>
</table>

**Even, Carry, and Overflow.** Unchanged.

**Program Check Conditions**

**Invalid Storage Address.** Operand.
Test Bit and Set (TBTS)

```
TBTS (reg, bitdisp)
```

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Fun</th>
<th>Bit displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The bit displacement is added to the byte address contained in the register specified by the R field to form an effective bit address. The bit displacement field is an unsigned six-bit binary integer.

The bit at the effective address is tested, and the zero and negative indicators are set to reflect the result.

Following the preceding test, the addressed bit is unconditionally set to one.

**Indicators**

Zero and Negative. First reset, then set as follows:

<table>
<thead>
<tr>
<th>Value of tested bit</th>
<th>Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
</tr>
</tbody>
</table>

Even, Carry, and Overflow. Unchanged.

**Program Check Conditions**

Invalid Storage Address. Operand.

Test Bit and Invert (TBTV)

```
TBTV (reg, bitdisp)
```

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Fun</th>
<th>Bit displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The bit displacement is added to the byte address contained in the register specified by the R field to form an effective bit address. The bit displacement field is an unsigned six-bit binary integer.

The bit at the effective address is tested, and the zero and negative indicators are set to reflect the result.

Following the preceding test, the addressed bit is unconditionally inverted.

**Indicators**

Zero and Negative. First reset, then set as follows:

<table>
<thead>
<tr>
<th>Value of tested bit</th>
<th>Indicators</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1</td>
</tr>
</tbody>
</table>

Even, Carry, and Overflow. Unchanged.

**Program Check Conditions**

Invalid Storage Address. Operand.
Test Word Immediate (TWI)

Register Immediate Long Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 1</td>
<td>0 0 0</td>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>0 4 5 7 8 10 11 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are tested against the mask contained in the immediate word of the instruction. The contents of the register specified by the R1 field are not changed. Mask bits set to one select the bits to be tested in the register.

Example:

Mask: 0000 0000 0101 1100
Register: 0000 0000 0011 0101
Selected bits: 0 1 0 1

The selected bits are tested for the following: (1) all bits zero, (2) all bits ones, or (3) a combination of one and zero bits (mixed). The zero and negative indicators are set to reflect the result as shown under Indicators.

Instruction bits 8 through 10 are not used and must be set to zero to avoid future code obsolescence.

Indicators

Zero and Negative. Reset, then set as follows:

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Selected bits</th>
<th>Zero</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>All zeros*</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All ones</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixed</td>
<td>0 0 (positive)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Also applies when the mask bits are all zeros.

Even, Carry, and Overflow. Unchanged.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Storage Immediate Format

Format without appended word for effective addressing (AM = 00 or 01)

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 1 0 1 1 1</td>
<td>1 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 4 5 7 8 9 10 11 12 15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the storage location specified by the effective address are tested against the mask in the immediate word of the instruction. (Effective Address Generation is explained in Chapter 2.) Both operands remain unchanged. Mask bits set to one select the bits to be tested in the storage operand.

Bits 5–7 of the instruction are not used and must be set to zero to avoid future code obsolescence.

Indicators

Zero and Negative. Reset, then set as follows:

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Selected bits</th>
<th>Zero</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>All zeros*</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All ones</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixed</td>
<td>0 0 (positive)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Also applies when the mask bits are all zeros.

Even, Carry, and Overflow. Unchanged.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Invert Register (VR)
VR reg[,reg]

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4 5</td>
<td>7 8</td>
<td>10 11 15</td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are ones complemented. The result is placed in the register specified by the R2 field. The contents of the register specified by the R1 field are unchanged.

Indicators
Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions
No program checks occur.

Exclusive OR Byte (XB)
XB reg,addr4
addr4,reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0</td>
<td>4 5</td>
<td>7 8</td>
<td>9 10 11 12 13 14 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[1 = \text{result to storage}\]
\[0 = \text{result to register}\]

A logical exclusive OR operation is performed between the least significant byte of the register specified by the R field and the main storage location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged. Also, when going from storage to register, bits 0–7 of the register are unchanged.

Example of Exclusive OR Byte:
Register contents 0000 1010 1100 0011
Storage operand 0110 0101
Result 1010 0110

Rule: Either but not both bits.

Indicators
Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result of the exclusive OR operation.

Program Check Conditions
Invalid Storage Address. Instruction word or operand.
Specification Check. Even byte boundary violation (indirect address).
Exclusive OR Doubleword (XD)

XD reg, addr4
addr4, reg

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 1 1 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>0 1 1 1 1</td>
</tr>
</tbody>
</table>

1 = result to storage
0 = result to register

Example of Exclusive OR Doubleword:

Register pair contents 0000 0000 1010 1100 0000 0000 1110 1111
Storage operand 0000 0000 1101 0011 0000 0000 1101 0000
Result 0000 0000 0111 1111 0000 0000 0011 1111

Rule: Either but not both bits.

Indicators

Carry and Overflow. Unchanged.

Even, Negative, and Zero. Changed to reflect the result of the exclusive OR operation.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).

A logical exclusive OR operation is performed between the contents of the register pair specified by the R field (R and R+1) and the doubleword in main storage specified by the effective address. (Effective Address Generation is explained in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

If the R field equals 7, registers 7 and 0 are used as the register pair.
Exclusive OR Word (XW)

Register/Register Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0</td>
<td>4 5 7 8 10 11 15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The contents of the register specified by the R1 field are exclusive ORed bit by bit with the contents of the register specified by the R2 field. The result is placed in the register specified by the R2 field. The contents of the register specified by R1 are unchanged unless R1 and R2 specify the same register.

Example of Exclusive OR Word:

Register contents (R1) 1111 0000 1010 0000
Register contents (R2) 0011 1111 0111 1111
Result 1100 1111 1101 1111

Rule: Either but not both bits.

Indicators

Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result of the exclusive OR operation.

Program Check Conditions

No program checks occur.

Register/Storage Format

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 1</td>
<td>4 5 7 8 9 10 11 12 13 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 = result to storage
0 = result to register

A logical exclusive OR operation is performed between the contents of the register specified by the R field and the main storage location specified by the effective address. (Effective Address Generation is explained in Chapter 2.) Bit 12 of the instruction specifies the destination of the result. The source operand is unchanged.

Example of Exclusive OR Word:

Register contents (R) 1111 0000 1010 0000
Storage operand 0011 1111 0111 1111
Result 1100 1111 1101 1111

Rule: Either but not both bits.

Indicators

Carry and Overflow. Unchanged.
Even, Negative, and Zero. Changed to reflect the result of the exclusive OR operation.

Program Check Conditions

Invalid Storage Address. Instruction word or operand.
Specification Check. Even byte boundary violation (indirect address or operand address).
A logical exclusive OR operation is performed between the contents of the register specified by the R1 field and the contents of the main storage word location specified by the effective address. The result is placed in the register specified by the R1 field.

The effective main storage address is generated as follows:

1. The address field is added to the contents of the register specified by the R2 field. If the R2 field equals zero, no register contributes to the address generation.
2. Instruction bit 11 is tested for direct or indirect addressing:
   - Bit 11=0 (direct address). The result from step 1 is the effective address.
   - Bit 11=1 (indirect address). The result from step 1 is the address of the main storage location that contains the effective address.

Example of Exclusive OR Word:

Register contents (R1) 1111 0000 1010 0000
Storage operand 0011 1111 0111 1111
Result 1100 1111 1101 1111

Rule: Either but not both bits.

Indicators

- Carry and Overflow. Unchanged.
- Even, Negative, and Zero. Changed to reflect the result.

Program Check Conditions

- Invalid Storage Address. Instruction word or operand.

Specification Check. Even byte boundary violation (indirect address or operand address).
Appendix A. Instruction Execution Times

This appendix contains instruction execution times.

The instructions are in alphabetic sequence based on assembler mnemonics. Figure A-I is the additional time required when executing register/storage instructions or storage/storage instructions and assembler syntax for addressing modes.

When running in Stop on Address Mode, instruction execution time is increased by 7.8 microseconds per instruction.

Key to symbols for tables in this appendix:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM1</td>
<td>Additional time for addressing mode 1 (see Figure A-I).</td>
</tr>
<tr>
<td>AM2</td>
<td>Additional time for addressing mode 2 (see Figure A-I).</td>
</tr>
<tr>
<td>CL</td>
<td>Current level.</td>
</tr>
<tr>
<td>CT</td>
<td>The count value at the beginning of instruction execution.</td>
</tr>
<tr>
<td>RL</td>
<td>Limit register (LMB and STM instructions).</td>
</tr>
<tr>
<td>RS</td>
<td>Additional addressing-mode time for register/storage instructions (see Figure A-I).</td>
</tr>
<tr>
<td>SL</td>
<td>Selected level.</td>
</tr>
<tr>
<td>*</td>
<td>Indirect address.</td>
</tr>
</tbody>
</table>

Note 1. The instruction is interruptable.
Note 2. If CT equals zero, use 5.4 for total time.
Note 3. Add 0.6 to the time if AM=01.
Instructions that use addressing mode (AM) for effective address generation require additional time that must be added to the base time for execution.

• RS—the additional time for register/storage instructions

<table>
<thead>
<tr>
<th>AM</th>
<th>Time (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>2.0</td>
</tr>
<tr>
<td>01</td>
<td>2.6</td>
</tr>
<tr>
<td>10 RB=0</td>
<td>2.6</td>
</tr>
<tr>
<td>10 RB=0</td>
<td>3.2</td>
</tr>
<tr>
<td>11 RB=0</td>
<td>3.2</td>
</tr>
<tr>
<td>11 RB=0</td>
<td>6.2</td>
</tr>
</tbody>
</table>

• AM1, AM2—the additional time for storage/storage instructions

<table>
<thead>
<tr>
<th>AM1</th>
<th>AM2</th>
<th>Time (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>3.6</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>4.2</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>5.4</td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>6.2</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>4.2</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>4.8</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>6.0</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>6.8</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>5.4</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>6.0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>7.2</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>8.0</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>6.2 (If RB1=00, add 0.2)</td>
</tr>
<tr>
<td>11</td>
<td>01</td>
<td>6.8 (If RB1=00, add 0.2)</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>8.0</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>8.8</td>
</tr>
</tbody>
</table>

For each AM field=11, add 1.6 if its corresponding RB≠0.

• Assembler syntax for address modes

<table>
<thead>
<tr>
<th>Assembler Syntax</th>
<th>Address Modes (see Note 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr4</td>
<td>addr5</td>
</tr>
<tr>
<td>(reg&lt;sup&gt;-3&lt;/sup&gt;)</td>
<td>(reg)</td>
</tr>
<tr>
<td>(reg&lt;sup&gt;-3&lt;/sup&gt;)*</td>
<td>(reg)*</td>
</tr>
<tr>
<td>addr</td>
<td>addr</td>
</tr>
<tr>
<td>(reg&lt;sup&gt;-3&lt;/sup&gt;,waddr)</td>
<td>(reg&lt;sup&gt;-3&lt;/sup&gt;,waddr)</td>
</tr>
<tr>
<td>addr*</td>
<td>addr*</td>
</tr>
<tr>
<td>disp1(reg&lt;sup&gt;-3&lt;/sup&gt;,disp2)*</td>
<td>disp1(reg&lt;sup&gt;-7&lt;/sup&gt;,disp2)*</td>
</tr>
<tr>
<td>disp(reg&lt;sup&gt;-3&lt;/sup&gt;)*</td>
<td>disp(reg&lt;sup&gt;-7&lt;/sup&gt;)*</td>
</tr>
<tr>
<td>(reg&lt;sup&gt;-3&lt;/sup&gt;)*</td>
<td>(reg&lt;sup&gt;-7&lt;/sup&gt;)*</td>
</tr>
<tr>
<td>(reg&lt;sup&gt;-3&lt;/sup&gt;,disp)*</td>
<td>(reg&lt;sup&gt;-7&lt;/sup&gt;,disp)*</td>
</tr>
</tbody>
</table>

Note 1. Register/storage instructions use assembler syntax addr4 for address mode (AM).

Storage/storage instructions use assembler syntax:

1. addr5 for address mode for operand 1 (AM1), and
2. addr4 for address mode for operand 2 (AM2).

Figure A-1. Additional instruction times for addressing mode and assembler syntax for addressing modes
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction name</th>
<th>Syntax</th>
<th>Execution time (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>Add Byte</td>
<td>reg, addr4, reg</td>
<td>11.0+RS Note 3</td>
</tr>
<tr>
<td>ABI</td>
<td>Add Byte Immediate</td>
<td>byte, reg</td>
<td>4.2</td>
</tr>
<tr>
<td>ACY</td>
<td>Add Carry Register</td>
<td>reg</td>
<td>4.8</td>
</tr>
<tr>
<td>AD</td>
<td>Add Double Word</td>
<td>addr4, reg</td>
<td>10.6+RS Note 3</td>
</tr>
<tr>
<td>AD</td>
<td>Add Double Word</td>
<td>addr4, reg</td>
<td>9.4+RS Note 3</td>
</tr>
<tr>
<td>AD</td>
<td>Add Double Word</td>
<td>addr5, addr4</td>
<td>8.2+KS Note 3</td>
</tr>
<tr>
<td>AW</td>
<td>Add Word</td>
<td>reg</td>
<td>11.2+AM1+AM2 Note 3</td>
</tr>
<tr>
<td>AW</td>
<td>Add Word</td>
<td>reg, addr4, reg</td>
<td>4.2</td>
</tr>
<tr>
<td>AW</td>
<td>Add Word</td>
<td>reg, addr4, reg</td>
<td>5.8+RS</td>
</tr>
<tr>
<td>AW</td>
<td>Add Word</td>
<td>addr5, addr4</td>
<td>5.2+RS</td>
</tr>
<tr>
<td>AW</td>
<td>Add Word</td>
<td>longaddr, reg</td>
<td>6.8+AM1+AM2 Note 3</td>
</tr>
<tr>
<td>AW</td>
<td>Add Word</td>
<td>longaddr*, reg</td>
<td>7.2</td>
</tr>
<tr>
<td>AW CY</td>
<td>Add Word With Carry</td>
<td>reg</td>
<td>7.8</td>
</tr>
<tr>
<td>AWI</td>
<td>Add Word Immediate</td>
<td>word, reg[reg]</td>
<td>4.8</td>
</tr>
<tr>
<td>AWI</td>
<td>Add Word Immediate</td>
<td>word, addr4</td>
<td>5.4</td>
</tr>
<tr>
<td>AWI</td>
<td>Add Word Immediate</td>
<td>word, addr4</td>
<td>6.6+RS</td>
</tr>
<tr>
<td>B</td>
<td>Branch Unconditional</td>
<td>addr4, reg</td>
<td>4.8</td>
</tr>
<tr>
<td>B</td>
<td>Branch Unconditional</td>
<td>addr4, reg</td>
<td>5.4</td>
</tr>
<tr>
<td>BAL</td>
<td>Branch and Link</td>
<td>longaddr, reg</td>
<td>6.6</td>
</tr>
<tr>
<td>BAL</td>
<td>Branch and Link</td>
<td>longaddr*, reg</td>
<td>7.2</td>
</tr>
<tr>
<td>BALS</td>
<td>Branch and Link Short</td>
<td>(reg, disp)*</td>
<td>5.4</td>
</tr>
<tr>
<td>BALS</td>
<td>Branch and Link Short</td>
<td>(reg)*</td>
<td>5.4</td>
</tr>
<tr>
<td>BALS</td>
<td>Branch and Link Short</td>
<td>addr*</td>
<td>5.4</td>
</tr>
<tr>
<td>BALX</td>
<td>Branch and Link External</td>
<td>longaddr*, reg</td>
<td>See BAL</td>
</tr>
<tr>
<td>BC</td>
<td>Branch on Condition</td>
<td>cond, longaddr</td>
<td>4.2</td>
</tr>
<tr>
<td>BC</td>
<td>Branch on Condition</td>
<td>cond, longaddr*</td>
<td>4.2</td>
</tr>
<tr>
<td>BCC</td>
<td>Branch on Condition Code</td>
<td>cond, longaddr</td>
<td>4.2</td>
</tr>
<tr>
<td>BCC</td>
<td>Branch on Condition Code</td>
<td>cond, longaddr*</td>
<td>4.2</td>
</tr>
<tr>
<td>BCY</td>
<td>Branch on Carry</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>Branch on Equal</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BER</td>
<td>Branch on Error</td>
<td>See BNCC</td>
<td></td>
</tr>
<tr>
<td>BEV</td>
<td>Branch on Even</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BGE</td>
<td>Branch on Arithmetically Greater Than or Equal</td>
<td>See BNC</td>
<td></td>
</tr>
<tr>
<td>BGT</td>
<td>Branch on Arithmetically Greater Than</td>
<td>See BNC</td>
<td></td>
</tr>
<tr>
<td>BLE</td>
<td>Branch on Arithmetically Less Than or Equal</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BLGE</td>
<td>Branch on Logically Greater Than</td>
<td>See BNC</td>
<td></td>
</tr>
<tr>
<td>BLGE</td>
<td>Branch on Logically Greater Than</td>
<td>See BNC</td>
<td></td>
</tr>
<tr>
<td>BLGT</td>
<td>Branch on Logically Greater Than</td>
<td>See BNC</td>
<td></td>
</tr>
<tr>
<td>BLLE</td>
<td>Branch on Logically Less Than</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BLLT</td>
<td>Branch on Logically Less Than</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BLT</td>
<td>Branch on Arithmetically Less Than</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BMIX</td>
<td>Branch if Mixed</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BN</td>
<td>Branch on Negative</td>
<td>See BC</td>
<td></td>
</tr>
<tr>
<td>BNC</td>
<td>Branch on Not Condition</td>
<td>cond, longaddr</td>
<td>4.2</td>
</tr>
<tr>
<td>BNC</td>
<td>Branch on Not Condition</td>
<td>cond, longaddr*</td>
<td>4.2</td>
</tr>
</tbody>
</table>

Instruction Execution Times  A-3
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction name</th>
<th>Syntax</th>
<th>Execution time (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Test Condition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No Branch</td>
</tr>
<tr>
<td>BNCC</td>
<td>Branch on Not Condition Code</td>
<td>cond,longaddr</td>
<td>4.2</td>
</tr>
<tr>
<td>BNCY</td>
<td>Branch on No Carry</td>
<td>cond,longaddr*</td>
<td>4.2</td>
</tr>
<tr>
<td>BNE</td>
<td>Branch on Not Equal</td>
<td></td>
<td>See BNC</td>
</tr>
<tr>
<td>BNER</td>
<td>Branch if Not Error</td>
<td></td>
<td>See BNC</td>
</tr>
<tr>
<td>BNEV</td>
<td>Branch on Not Even</td>
<td></td>
<td>See BNC</td>
</tr>
<tr>
<td>BNMIX</td>
<td>Branch if Not Mixed</td>
<td></td>
<td>See BNC</td>
</tr>
<tr>
<td>BNN</td>
<td>Branch on Not Negative</td>
<td></td>
<td>See BNC</td>
</tr>
<tr>
<td>BNOV</td>
<td>Branch on Not Overflow</td>
<td>longaddr</td>
<td>3.6</td>
</tr>
<tr>
<td>BNP</td>
<td>Branch on Not Positive</td>
<td>longaddr*</td>
<td>3.6</td>
</tr>
<tr>
<td>BNZ</td>
<td>Branch on Not Zero</td>
<td></td>
<td>See BNC</td>
</tr>
<tr>
<td>BOFF</td>
<td>Branch if Off</td>
<td></td>
<td>See BC</td>
</tr>
<tr>
<td>BON</td>
<td>Branch if ON</td>
<td></td>
<td>See BC</td>
</tr>
<tr>
<td>BOV</td>
<td>Branch on Overflow</td>
<td>longaddr</td>
<td>3.6</td>
</tr>
<tr>
<td>BP</td>
<td>Branch on Positive</td>
<td>longaddr*</td>
<td>3.6</td>
</tr>
<tr>
<td>BX</td>
<td>Branch external</td>
<td>vcon</td>
<td>see BC</td>
</tr>
<tr>
<td>BXS</td>
<td>Branch Indexed Short</td>
<td>(reg1-7),disp</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(reg1-7)</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr</td>
<td>4.2</td>
</tr>
<tr>
<td>BZ</td>
<td>Branch on Zero</td>
<td></td>
<td>See BC</td>
</tr>
<tr>
<td>CB</td>
<td>Compare Byte</td>
<td>addr4,reg</td>
<td>8.8+RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>7.2+AM1+AM2</td>
</tr>
<tr>
<td>CBI</td>
<td>Compare Byte Immediate</td>
<td>byte,reg</td>
<td>3.6</td>
</tr>
<tr>
<td>CD</td>
<td>Compare Double Word</td>
<td>addr4,reg</td>
<td>8.2+RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>9.8+AM1+AM2</td>
</tr>
<tr>
<td>CFED</td>
<td>Compare Byte Field Equal and Decrement</td>
<td>(reg),(reg)</td>
<td>8.4+(7.8 x CT)</td>
</tr>
<tr>
<td>CFEN</td>
<td>Compare Byte Field Equal and Increment</td>
<td>(reg),(reg)</td>
<td>8.4+(7.8 x CT)</td>
</tr>
<tr>
<td>CFNED</td>
<td>Compare Byte Field Not Equal and Decrement</td>
<td>(reg),(reg)</td>
<td>8.4+(7.8 x CT)</td>
</tr>
<tr>
<td>CFNEN</td>
<td>Compare Byte Field Not Equal and Increment</td>
<td>(reg),(reg)</td>
<td>8.4+(7.8 x CT)</td>
</tr>
<tr>
<td>CMR</td>
<td>Complement Register</td>
<td>reg[,reg]</td>
<td>4.8</td>
</tr>
<tr>
<td>CPC</td>
<td>Copy Current Level</td>
<td>reg</td>
<td>5.4</td>
</tr>
<tr>
<td>CPCON</td>
<td>Copy Console Data Buffer</td>
<td>reg</td>
<td>4.8</td>
</tr>
<tr>
<td>CPIIMR</td>
<td>Copy Interrupt Mask Register</td>
<td>addr4</td>
<td>6.0+RS</td>
</tr>
<tr>
<td>CPIPF</td>
<td>Copy In-Process Flags</td>
<td>addr4</td>
<td>9.4+RS</td>
</tr>
<tr>
<td>CPLB</td>
<td>Copy Level Block</td>
<td>reg,addr4</td>
<td>23.4+RS SL=CL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>24.6+RS SL#CL</td>
</tr>
<tr>
<td>CPLSR</td>
<td>Copy Level Status Register</td>
<td>reg</td>
<td>6.0</td>
</tr>
<tr>
<td>CPPSR</td>
<td>Copy Processor Status and Reset</td>
<td>addr4</td>
<td>5.8+RS</td>
</tr>
<tr>
<td>CW</td>
<td>Compare Word</td>
<td>reg,reg</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>5.8+RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>5.8+AM1+AM2</td>
</tr>
</tbody>
</table>

Note. 7.8 if operand 2 is at an odd address.
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction name</th>
<th>Syntax</th>
<th>Execution time (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWI</td>
<td>Compare Word Immediate</td>
<td>word,reg</td>
<td>5.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word,addr4</td>
<td>5.8+RS</td>
</tr>
<tr>
<td>DB</td>
<td>Divide Byte</td>
<td>addr4,reg</td>
<td>7.0+RS Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>88.6+RS Maximum</td>
</tr>
<tr>
<td>DD</td>
<td>Divide Double Word</td>
<td>addr4,reg</td>
<td>7.0+RS Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>159.4+RS Maximum</td>
</tr>
<tr>
<td>DIAG</td>
<td>Diagnose</td>
<td>ubyte</td>
<td>4.8 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10.8 Maximum</td>
</tr>
<tr>
<td>DIS</td>
<td>Disable</td>
<td>ubyte</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Note)</td>
</tr>
<tr>
<td>DW</td>
<td>Divide Word</td>
<td>addr4,reg</td>
<td>7.0+RS Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>86.8+RS Maximum</td>
</tr>
<tr>
<td>EN</td>
<td>Enable</td>
<td>ubyte</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Note)</td>
</tr>
<tr>
<td>FFD</td>
<td>Fill Byte Field and Decrement</td>
<td>reg,(reg)</td>
<td>8.4+(3.6 x CT) Note 1</td>
</tr>
<tr>
<td>FFN</td>
<td>Fill Byte Field and Increment</td>
<td>reg,(reg)</td>
<td>8.4+(3.6 x CT) Note 1</td>
</tr>
<tr>
<td>IO</td>
<td>Operate I/O</td>
<td>longaddr</td>
<td>9.5 Typical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16.8 Halt I/O Note 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>longaddr*</td>
<td>10.1 Typical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>17.4 Halt I/O Note 2</td>
</tr>
</tbody>
</table>

*Note.* The range within which the OIO and OIO Indirect Instructions may be successfully completed is 9.1 min/9.7 min to 28.9 max/29.5 max respectively. The minimum times do not reflect any delays which may be induced by the attachment and/or device to which the command is directed.

| IR       | Interchange Registers             | reg,reg         | 4.8                          |
| J        | Jump Unconditional                | jdisp           | 4.2                          |
|          |                                   | jaddr           | 4.2                          |
| JAL      | Jump and Link                     | jdisp,reg       | 5.4                          |
|          |                                   | jaddr,reg       | 5.4                          |
|          |                                   |                 | **Test Condition**           |
|          |                                   |                 | **No Branch**                |
|          |                                   |                 | **Branch**                   |
| JC       | Jump on Condition                 | cond,jdisp      | 4.2                          |
|          |                                   |                 | 6.0                          |
|          |                                   | cond,jaddr      | 4.2                          |
|          |                                   |                 | 6.0                          |
| JCT      | Jump on Count                     | jdisp,reg       | 5.4                          |
|          |                                   |                 | 4.8                          |
|          |                                   |                 | 6.6                          |
| JCY      | Jump on Carry                     | jadd,reg        | 5.4                          |
|          |                                   |                 | 4.8                          |
|          |                                   |                 | 6.6                          |
| JE       | Jump on Equal                     | See JC          |
| JEV      | Jump on Even                      | See JC          |
| JGE      | Jump on Arithmetically Greater    | See JNC         |
|          | Than Equal                        |                 |
| JGT      | Jump on Arithmetically Greater    | See JNC         |
|          | Than                               |                 |
| JLE      | Jump on Arithmetically Less       | See JC          |
|          | Than or Equal                     |                 |
| JLGE     | Jump on Logically Greater Than     | See JNC         |
|          | Equal                              |                 |
| JLGT     | Jump on Logically Greater Than     | See JNC         |
|          | Than                               |                 |
| JLE      | Jump on Logically Less Than        | See JC          |
|          | Equal                              |                 |

*Note.* 5.4 if bit 15 of parameter field equals zero.
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction name</th>
<th>Syntax</th>
<th>Execution time (microseconds)</th>
<th>Test Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>JLLT</td>
<td>Jump on Logically Less Than</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>JLT</td>
<td>Jump on Arithmetically Less Than</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>JMIX</td>
<td>Jump if Mixed</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>JN</td>
<td>Jump on Negative</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump on Not Condition</td>
<td>cond_jdisp</td>
<td>4.2</td>
<td>Branch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cond_jaddr</td>
<td>4.2</td>
<td>No Branch</td>
</tr>
<tr>
<td>JNCY</td>
<td>Jump on No Carry</td>
<td></td>
<td></td>
<td>See JNC</td>
</tr>
<tr>
<td>JNE</td>
<td>Jump on Not Equal</td>
<td></td>
<td></td>
<td>See JNC</td>
</tr>
<tr>
<td>JNEV</td>
<td>Jump on Not Even</td>
<td></td>
<td></td>
<td>See JNC</td>
</tr>
<tr>
<td>JNMIX</td>
<td>Jump if Not Mixed</td>
<td></td>
<td></td>
<td>See JNC</td>
</tr>
<tr>
<td>JNN</td>
<td>Jump on Not Negative</td>
<td></td>
<td></td>
<td>See JNC</td>
</tr>
<tr>
<td>JNP</td>
<td>Jump on Not Positive</td>
<td></td>
<td></td>
<td>See JNC</td>
</tr>
<tr>
<td>JNZ</td>
<td>Jump on Not Zero</td>
<td></td>
<td></td>
<td>See JNC</td>
</tr>
<tr>
<td>JOFF</td>
<td>Jump if Off</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>JON</td>
<td>Jump if On</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>JP</td>
<td>Jump on Positive</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>JZ</td>
<td>Jump on Zero</td>
<td></td>
<td></td>
<td>See JC</td>
</tr>
<tr>
<td>LEX</td>
<td>Level Exit</td>
<td>[ubyte]</td>
<td>8.8 Minimum</td>
<td>Branch</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19.6 Maximum</td>
<td>No Branch</td>
</tr>
<tr>
<td>LMB</td>
<td>Load Multiple and Branch</td>
<td>addr4</td>
<td>17.2+RS</td>
<td>(Note)</td>
</tr>
</tbody>
</table>

**Note.** For each specified register from 0 through 6, add an additional 3.2 per register.

<p>| MB       | Multiply Byte           | addr4,reg    | 11.2+RS Minimum              | Branch         |
|          |                         |              | 26.2+RS Maximum              | No Branch      |
| MD       | Multiply Doubleword     | addr4,reg    | 10.0+RS Minimum              | See JNC        |
|          |                         | addr5,addr4  | 98.2+RS Maximum              | See JNC        |
| MVA      | Move Address            | addr4,reg    | 5.2+RS                       | Note 3         |
|          |                         | addr5,addr4  | 5.4+RS                       | Note 3         |
| MVB      | Move Byte               | reg,addr4    | 6.6+RS                       | Note 3         |
|          |                         | addr4,reg    | 8.2+RS                       | Note 3         |
|          |                         | addr5,addr4  | 5.4+AM1+AM2                  | Note 3         |
| MVBI     | Move Byte Immediate     | byte,reg     | 3.6                          |                |
| MVBI     | Move Byte and Zero      | addr4,reg    | 8.8+RS                       | Note 3         |
| MVD      | Move Doubleword         | reg,addr4    | 8.4+RS                       | Note 3         |
|          |                         | addr4,reg    | 7.6+RS                       | Note 3         |
|          |                         | addr5,addr4  | 8.4+AM1+AM2                  | Note 3         |
| MVDZ     | Move Doubleword and Zero| addr4,reg    | 9.6+RS                       | Note 3         |
| MVFD     | Move Byte Field and Decrement | (reg),(reg) | 8.4+(5.4 x CT)               | Note 1         |
| MVFN     | Move Byte Field and Increment | (reg),(reg) | 8.4+(5.4 x CT)               | Note 1         |
| MVW      | Move Word               | reg,reg      | 4.2                          | Note 2         |
|          |                         | reg,addr4    | 5.4+RS                       | Note 2         |
|          |                         | addr4,reg    | 5.2+RS                       | Note 2         |
|          |                         | addr5,addr4  | 4.8+AM1+AM2                  | Note 2         |
|          |                         | reg,longaddr | 7.6                          |                |
|          |                         | reg,longaddr*| 8.2                          |                |
|          |                         | longaddr,reg | 7.2                          |                |
|          |                         | longaddr*,reg| 7.8                          |                |
| MVWI     | Move Word Immediate     | word,reg     | 5.2                          |                |
|          |                         | word,addr4   | 5.4+RS                       |                |</p>
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction name</th>
<th>Syntax</th>
<th>Execution time (microseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVWS</td>
<td>Move Word Short</td>
<td>reg,shortaddr</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,shortaddr*</td>
<td>6.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>shortaddr,reg</td>
<td>5.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>shortaddr*,reg</td>
<td>6.2</td>
</tr>
<tr>
<td>MVWL</td>
<td>Move Word and Zero</td>
<td>addr4,reg</td>
<td>5.2+RS</td>
</tr>
<tr>
<td>MW</td>
<td>Multiply Word</td>
<td>addr4,reg</td>
<td>9.4+RS Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>40.6+RS Maximum</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td></td>
<td>4.2</td>
</tr>
<tr>
<td>NWI</td>
<td>And Word Immediate</td>
<td>word,reg[,reg]</td>
<td>5.4</td>
</tr>
<tr>
<td>OB</td>
<td>OR Byte</td>
<td>reg,addr4</td>
<td>8.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>8.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>7.8+AM1+AM2</td>
</tr>
<tr>
<td>OD</td>
<td>OR Doubleword</td>
<td>reg,addr4</td>
<td>9.0+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>7.6+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>9.8+AM1+AM2</td>
</tr>
<tr>
<td>OW</td>
<td>OR Word</td>
<td>reg,reg</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,addr4</td>
<td>5.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>5.2+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>6.0+AM1+AM2</td>
</tr>
<tr>
<td>OWI</td>
<td>OR Word Immediate</td>
<td>word,reg[,reg]</td>
<td>5.4</td>
</tr>
<tr>
<td>PB</td>
<td>Pop Byte</td>
<td>addr4,reg</td>
<td>6.6+RS Note 3</td>
</tr>
<tr>
<td>PD</td>
<td>Pop Doubleword</td>
<td>addr4,reg</td>
<td>10.4+RS Note 3</td>
</tr>
<tr>
<td>PSB</td>
<td>Push Byte</td>
<td>reg,addr4</td>
<td>10.6+RS Note 3</td>
</tr>
<tr>
<td>PSD</td>
<td>Push Doubleword</td>
<td>reg,addr4</td>
<td>11.0+RS Note 3</td>
</tr>
<tr>
<td>PSW</td>
<td>Push Word</td>
<td>reg,addr4</td>
<td>9.8+RS Note 3</td>
</tr>
<tr>
<td>PW</td>
<td>Pop Word</td>
<td>addr4,reg</td>
<td>9.2+RS Note 3</td>
</tr>
<tr>
<td>RBTB</td>
<td>Reset Bits Byte</td>
<td>reg,addr4</td>
<td>8.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>8.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>8.0+AM1+AM2</td>
</tr>
<tr>
<td>RBD</td>
<td>Reset Bits Doubleword</td>
<td>reg,addr4</td>
<td>9.0+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>8.0+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>9.8+AM1+AM2</td>
</tr>
<tr>
<td>RBTW</td>
<td>Reset Bits Word</td>
<td>reg,reg</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,addr4</td>
<td>5.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>5.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>6.6+AM1+AM2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>longaddr,reg</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>longaddr*,reg</td>
<td>7.8</td>
</tr>
<tr>
<td>RBTWI</td>
<td>Reset Bits Word</td>
<td>word,reg[,reg]</td>
<td>5.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word,addr4</td>
<td>6.6+RS Note 3</td>
</tr>
<tr>
<td>SB</td>
<td>Subtract Byte</td>
<td>reg,addr4</td>
<td>11.0+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>10.6+RS Note 3</td>
</tr>
<tr>
<td>SCY</td>
<td>Subtract Carry Indicator</td>
<td>reg</td>
<td>4.8</td>
</tr>
<tr>
<td>SD</td>
<td>Subtract Doubleword</td>
<td>reg,addr4</td>
<td>9.4+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>8.8+RS Note 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>11.2+AM1+AM2</td>
</tr>
<tr>
<td>SECON</td>
<td>Set Console Data Lights</td>
<td>reg</td>
<td>6.0</td>
</tr>
<tr>
<td>SEIMR</td>
<td>Set Interrupt Mask Register</td>
<td>addr4</td>
<td>8.0+RS Note 3</td>
</tr>
<tr>
<td>SEIND</td>
<td>Set Indicators</td>
<td>reg</td>
<td>6.0</td>
</tr>
<tr>
<td>SELB</td>
<td>Set Level Block</td>
<td>reg,addr4</td>
<td>24.4 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>43.6 Maximum</td>
</tr>
<tr>
<td>SFED</td>
<td>Scan Byte Field Equal and Decrement</td>
<td>reg,(reg)</td>
<td>8.4+(6.0 x CT) Note 1</td>
</tr>
<tr>
<td>SFEN</td>
<td>Scan Byte Field Equal and Increment</td>
<td>reg,(reg)</td>
<td>8.4+(6.0 x CT) Note 2</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Instruction name</td>
<td>Syntax</td>
<td>Execution time (microseconds)</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>--------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>SFNED</td>
<td>Scan Byte Field Not Equal and Decrement</td>
<td>reg,(reg)</td>
<td>8.4+(6.0 x CT) Note 1</td>
</tr>
<tr>
<td>SFNEN</td>
<td>Scan Byte Field Not Equal and Increment</td>
<td>reg,(reg)</td>
<td>8.4+(6.0 x CT) Note 2</td>
</tr>
<tr>
<td>SLC</td>
<td>Shift Left Circular</td>
<td>cnt16,reg</td>
<td>6.6 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>7.8 Maximum</td>
</tr>
<tr>
<td>SLCD</td>
<td>Shift Left Circular Double</td>
<td>cnt31,reg</td>
<td>15.0 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>18.0 Maximum</td>
</tr>
<tr>
<td>SLL</td>
<td>Shift Left Logical</td>
<td>cnt16,reg</td>
<td>5.4 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>12.0 Maximum</td>
</tr>
<tr>
<td>SLLD</td>
<td>Shift Left Logical Double</td>
<td>cnt31,reg</td>
<td>6.6 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>18.6 Maximum</td>
</tr>
<tr>
<td>SLT</td>
<td>Shift Left and Test</td>
<td>reg,reg</td>
<td>7.2 Minimum</td>
</tr>
<tr>
<td>SLTD</td>
<td>Shift Left and Test Double</td>
<td>reg,reg</td>
<td>7.8 Minimum</td>
</tr>
<tr>
<td>SRA</td>
<td>Shift Right Arithmetic</td>
<td>cnt16,reg</td>
<td>6.6 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>10.2 Maximum</td>
</tr>
<tr>
<td>SRAD</td>
<td>Shift Right Arithmetic Double</td>
<td>cnt31,reg</td>
<td>10.8 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>16.8 Maximum</td>
</tr>
<tr>
<td>SRL</td>
<td>Shift Right Logical</td>
<td>cnt16,reg</td>
<td>5.4 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>9.0 Maximum</td>
</tr>
<tr>
<td>SRLD</td>
<td>Shift Right Logical Double</td>
<td>cnt31,reg</td>
<td>10.2 Minimum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,reg</td>
<td>16.2 Maximum</td>
</tr>
<tr>
<td>STM</td>
<td>Store Multiple</td>
<td>reg,addr4[,abcnt]</td>
<td>21.8+RS (Note)</td>
</tr>
</tbody>
</table>

*Note.* For each specified register from 0 through 6, add an additional 3.0 per register.

<table>
<thead>
<tr>
<th>STOP</th>
<th>Stop</th>
<th>[ubyte]</th>
<th>4.2 (Note)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVC</td>
<td>Supervisor Call</td>
<td>ubyte</td>
<td>26.0</td>
</tr>
<tr>
<td>SW</td>
<td>Subtract Word</td>
<td>reg,reg</td>
<td>4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reg,addr4</td>
<td>6.2+RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr4,reg</td>
<td>5.8+RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>addr5,addr4</td>
<td>6.8+AM1+AM2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>longaddr,reg</td>
<td>7.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>longaddr*,reg</td>
<td>8.4</td>
</tr>
<tr>
<td>SWCY</td>
<td>Subtract Word With Carry</td>
<td>reg,reg</td>
<td>4.8</td>
</tr>
<tr>
<td>SWI</td>
<td>Subtract Word Immediate</td>
<td>word,addr4</td>
<td>6.6+RS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>word,reg[,reg]</td>
<td>5.4</td>
</tr>
<tr>
<td>TBT</td>
<td>Test Bit</td>
<td>(reg,bitdisp)</td>
<td>8.4</td>
</tr>
<tr>
<td>TBTR</td>
<td>Test Bit and Reset</td>
<td>(reg,bitdisp)</td>
<td>9.6</td>
</tr>
<tr>
<td>TBTS</td>
<td>Test Bit and Set On</td>
<td>(reg,bitdisp)</td>
<td>9.6</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Instruction name</td>
<td>Syntax</td>
<td>Execution time (microseconds)</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>TBTV</td>
<td>Test Bit and Invert</td>
<td>(reg,bitdisp)</td>
<td>9.6</td>
</tr>
<tr>
<td>TWI</td>
<td>Test Word Under Mask</td>
<td>word,reg</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td>Immediate</td>
<td>word,addr4</td>
<td>7.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7.0+RS All bits = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7.6+RS All bits = 1</td>
</tr>
<tr>
<td>VR</td>
<td>Invert Register</td>
<td>reg[,reg]</td>
<td>4.8</td>
</tr>
<tr>
<td>XB</td>
<td>Exclusive OR Byte</td>
<td>reg,addr4</td>
<td>8.8+RS Note 3</td>
</tr>
<tr>
<td>XD</td>
<td>Exclusive OR Doubleword</td>
<td>reg,addr4</td>
<td>8.8+RS Note 3</td>
</tr>
<tr>
<td>XD</td>
<td></td>
<td>addr4,reg</td>
<td>9.0+RS Note 3</td>
</tr>
<tr>
<td>XD</td>
<td></td>
<td></td>
<td>8.2+RS Note 3</td>
</tr>
<tr>
<td>XW</td>
<td>Exclusive OR Word</td>
<td>reg,reg</td>
<td>4.2</td>
</tr>
<tr>
<td>XW</td>
<td></td>
<td>reg,addr4</td>
<td>5.8+RS Note 3</td>
</tr>
<tr>
<td>XW</td>
<td></td>
<td>addr4,reg</td>
<td>5.2+RS Note 3</td>
</tr>
<tr>
<td>XW</td>
<td></td>
<td>longaddr,reg</td>
<td>7.2</td>
</tr>
<tr>
<td>XW</td>
<td></td>
<td>longaddr*,reg</td>
<td>7.8</td>
</tr>
<tr>
<td>XWI</td>
<td>Exclusive OR Word</td>
<td>word,reg[,reg]</td>
<td>5.4</td>
</tr>
</tbody>
</table>

Note 3
The following instruction formats are shown in ascending sequence based on operation code. Bits zero through four of the first instruction word comprise the operation code field. Bit combinations are shown for each operation code along with the hexadecimal representation.

Some instructions contain a function field that modifies the operation code to form individual instructions within a group. Each chart shows the function field bit combinations in hexadecimal and in ascending sequence. The assembler mnemonic, assembler syntax, and instruction name are listed for the individual instructions. The asterisk shown with the assembler syntax indicates indirect addressing.

Refer to Chapter 2, Effective Address Generation, for a description of the Address Mode (AM) appended words.
### 1xxx

#### 2xxx

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Cond</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 1 0</td>
<td></td>
<td>0 4 5 7 8 15</td>
</tr>
<tr>
<td>0 1 0 7 X</td>
<td></td>
<td>0 7 X X</td>
</tr>
</tbody>
</table>

- **Operation**: Jump on Condition
- **Extended mnemonics**: JCY, JE, JEV, JLE, JLTE, JLT, JMIX, JN, JOFF, JON, JP, JZ

<table>
<thead>
<tr>
<th>Operation code</th>
<th>Cond</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 1</td>
<td></td>
<td>0 4 5 7 8 15</td>
</tr>
<tr>
<td>0 1 8 F X</td>
<td></td>
<td>0 8-F X X</td>
</tr>
</tbody>
</table>

- **Operation**: Jump on Not Condition
- **Extended mnemonics**: JCE, JCT, JLCE, JLCT, JNCY, JNE, JNEV, JNMIX, JNN, JNP, JNZ

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Fun</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 0 0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2 0 3 X</td>
<td></td>
<td>0-3</td>
<td>X</td>
<td>0-F</td>
<td>AM appended word</td>
</tr>
</tbody>
</table>

- **Illegal operation code (invalid function)**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Func</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 0 0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 4-7 X</td>
<td></td>
<td></td>
<td>0-F</td>
<td></td>
</tr>
</tbody>
</table>

- **Illegal operation code (invalid function)**
<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>I</th>
<th>D</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010101</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>8-F</td>
<td>X</td>
<td>0-F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Function**: Move Byte Field and Decrement

**Operation code**: 0010101

**R1**: (reg)

**R2**: (reg)

**Count**: 0-7

**Function**: Move Byte Field and Decrement

**Instruction Formats**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00110</td>
<td>0</td>
<td>4 5</td>
<td>Shift Left Circular</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>8</td>
<td>Shift Left Logical</td>
</tr>
<tr>
<td>3</td>
<td>0-7</td>
<td>X</td>
<td>Shift Right Logical</td>
</tr>
<tr>
<td>1</td>
<td>cnt16,reg</td>
<td>Shift Right Arithmetic</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>cnt16,reg</td>
<td>Shift Right Logical</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>cnt16,reg</td>
<td>Shift Right Circular Double</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cnt31,reg</td>
<td>Shift Left Logical Double</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>cnt31,reg</td>
<td>Shift Right Logical Double</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>cnt31,reg</td>
<td>Shift Right Arithmetic Double</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>cnt31,reg</td>
<td>Shift Right Circular Double</td>
<td></td>
</tr>
</tbody>
</table>

**Instruction Formats**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00110</td>
<td>0</td>
<td>4 5</td>
<td>Shift Left Circular</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>8</td>
<td>Shift Left Logical</td>
</tr>
<tr>
<td>3</td>
<td>0-7</td>
<td>X</td>
<td>Shift Right Logical</td>
</tr>
<tr>
<td>1</td>
<td>cnt16,reg</td>
<td>Shift Right Arithmetic</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>cnt16,reg</td>
<td>Shift Right Logical</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>cnt16,reg</td>
<td>Shift Right Circular Double</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cnt31,reg</td>
<td>Shift Left Logical Double</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>cnt31,reg</td>
<td>Shift Right Logical Double</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>cnt31,reg</td>
<td>Shift Right Arithmetic Double</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>cnt31,reg</td>
<td>Shift Right Circular Double</td>
<td></td>
</tr>
</tbody>
</table>
### Operation Code

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Move Address</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Move Word Immediate</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Move Address (Note 1)</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Move Word Immediate (Note 1)</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Store Multiple</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Add Word Immediate</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Load Multiple and Branch (Note 1)</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Test Word Under Mask Immediate</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>OR Word Immediate</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Reset Bits Word Immediate</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Subtract Word Immediate</td>
</tr>
<tr>
<td>000000000000101</td>
<td></td>
<td></td>
<td></td>
<td>Compare Word Immediate</td>
</tr>
</tbody>
</table>

Note 1. Use format without immediate field.
### 4xxx

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Fun</th>
<th>Bit displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 1</td>
<td>4</td>
<td>8–F</td>
<td>0–F X</td>
</tr>
</tbody>
</table>

- **4 8–F 0–3 X**
  - TBT (reg, bitdisp)
  - Test Bit
  - Test Bit and Set On
  - Test Bit and Reset
  - Test Bit and Invert

### 5xxx

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 0</td>
<td>5</td>
<td>0–7 X X</td>
</tr>
</tbody>
</table>

- **5 0 0 0**
  - NOP No Operation

- **0 X X**
  - J disp
  - Jump Unconditional

- **1–7 X X**
  - BXSR (reg 1–7 disp)
  - Branch Indexed Short

- **1–7 X X**
  - BXSR (reg 1–7)
  - Branch Indexed Short

- **1–7 X X**
  - BXSR addr
  - Branch Indexed Short
<table>
<thead>
<tr>
<th>Operation code</th>
<th>K</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## AM appended word

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8-F</td>
<td>X</td>
<td>0-F</td>
</tr>
</tbody>
</table>

### Byte 5 8-F X

<table>
<thead>
<tr>
<th></th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SEIMR, addr4</td>
</tr>
<tr>
<td>1</td>
<td>(Note 2)</td>
</tr>
<tr>
<td>2</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>3</td>
<td>(Note 4)</td>
</tr>
<tr>
<td>4</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>5</td>
<td>(invalid)</td>
</tr>
<tr>
<td>6</td>
<td>SELB, reg,addr4</td>
</tr>
<tr>
<td>7</td>
<td>(invalid)</td>
</tr>
<tr>
<td>8</td>
<td>CPMR, addr4</td>
</tr>
<tr>
<td>9</td>
<td>(Note 2)</td>
</tr>
<tr>
<td>A</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>B</td>
<td>(Note 4)</td>
</tr>
<tr>
<td>C</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>D</td>
<td>CIPF, addr4</td>
</tr>
<tr>
<td>E</td>
<td>CPLB, reg,addr4</td>
</tr>
<tr>
<td>F</td>
<td>CPPSR, addr4</td>
</tr>
</tbody>
</table>

### Notes

1. **Supervisor state:** program check, invalid function
   **Problem state:** program check, privilege violate

2. **Supervisor state:** No-op
   **Problem state:** program check, privilege violate

3. **Supervisor state:** soft exception trap, invalid function
   **Problem state:** program check, privilege violate
<table>
<thead>
<tr>
<th>Operation code</th>
<th>Function</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0</td>
<td>0 4 5 7 8 15</td>
<td>6 0 7 X X</td>
</tr>
</tbody>
</table>

6 0 X X

SVC ubyte Supervisor Call
LEX [ubyte] Level Exit
EN ubyte Enable
DIS ubyte Disable
STOP [ubyte] Stop
DIAG ubyte Diagnose
(Note 5)
(invalid)

Note 5. Supervisor state: No-op
Problem state: program check, privilege violate
### 6xxx

**R1, condition, or condition code**

0 = Direct address, 1 = Indirect address

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R2</th>
<th>Function</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>011010</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>8</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8-F</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>8-F</th>
<th>0</th>
<th>BC</th>
<th>cond, longaddr</th>
<th>Branch on Condition (Note 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BNC</td>
<td>1</td>
<td>BNC</td>
<td>cond, longaddr</td>
<td>Branch on Not Condition (Note 7)</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>2</td>
<td>B</td>
<td>longaddr</td>
<td>Branch Unconditional (Note 8)</td>
</tr>
<tr>
<td>3</td>
<td>BAL</td>
<td>3</td>
<td>BAL</td>
<td>longaddr, reg</td>
<td>Branch and Link (Note 9)</td>
</tr>
<tr>
<td>4</td>
<td>BCC</td>
<td>4</td>
<td>BCC</td>
<td>cond, longaddr</td>
<td>Branch on Condition Code (Note 10)</td>
</tr>
<tr>
<td>5</td>
<td>BNCC</td>
<td>5</td>
<td>BNCC</td>
<td>cond, longaddr</td>
<td>Branch on Not Condition Code (Note 11)</td>
</tr>
<tr>
<td>6</td>
<td>BOV</td>
<td>6</td>
<td>BOV</td>
<td>longaddr</td>
<td>Branch on Overflow</td>
</tr>
<tr>
<td>7</td>
<td>BNOV</td>
<td>7</td>
<td>BNOV</td>
<td>longaddr</td>
<td>Branch on Not Overflow</td>
</tr>
<tr>
<td>8</td>
<td>MVW</td>
<td>8</td>
<td>MVW</td>
<td>longaddr, reg</td>
<td>Move Word</td>
</tr>
<tr>
<td>9</td>
<td>OW</td>
<td>9</td>
<td>OW</td>
<td>longaddr, reg</td>
<td>OR Word</td>
</tr>
<tr>
<td>A</td>
<td>RBTW</td>
<td>A</td>
<td>RBTW</td>
<td>longaddr, reg</td>
<td>Reset Bits Word</td>
</tr>
<tr>
<td>B</td>
<td>XW</td>
<td>B</td>
<td>XW</td>
<td>longaddr, reg</td>
<td>Exclusive OR Word</td>
</tr>
<tr>
<td>C</td>
<td>IO</td>
<td>C</td>
<td>IO</td>
<td>longaddr</td>
<td>Operate I/O</td>
</tr>
<tr>
<td>D</td>
<td>MVW</td>
<td>D</td>
<td>MVW</td>
<td>reg, longaddr</td>
<td>Move Word</td>
</tr>
<tr>
<td>E</td>
<td>AW</td>
<td>E</td>
<td>AW</td>
<td>longaddr, reg</td>
<td>Add Word</td>
</tr>
<tr>
<td>F</td>
<td>SW</td>
<td>F</td>
<td>SW</td>
<td>longaddr, reg</td>
<td>Subtract Word</td>
</tr>
</tbody>
</table>

**Note 6.** Extended mnemonics: BCY, BE, BEV, BLE, BLLE, BLLT, BLT, BMIX, BN, BOFF, BON, BP, BZ

**Note 7.** Extended mnemonics: BGE, BGT, BLGE, BLGT, BNCY, BNE, BNEV, BMIX, BNN, BNOFF, BNON, BNP, BNZ

**Note 8.** Extended mnemonic: BX

**Note 9.** Extended mnemonic: BALX

**Note 10.** Extended mnemonic: BNER

**Note 11.** Extended mnemonic: BER
Instruction Formats
<table>
<thead>
<tr>
<th>Operation code</th>
<th>R1</th>
<th>R2</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>01</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7 10</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0 7 15</td>
</tr>
<tr>
<td>7 0-7</td>
<td>0</td>
<td>RBTW reg,reg</td>
<td>Reset Bits Word</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>OW  reg,reg</td>
<td>OR Word</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>SCY  reg</td>
<td>Subtract Carry Indicator</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>XW  reg,reg</td>
<td>Exclusive OR Word</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>MVW  reg,reg</td>
<td>Move Word</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>CW  reg,reg</td>
<td>Compare Word</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>CMR  reg[reg]</td>
<td>Complement Register</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>IR  reg,reg</td>
<td>Interchange Registers</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>AW  reg,reg</td>
<td>Add Word</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>AWCY reg,reg</td>
<td>Add Word With Carry</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>SW  reg,reg</td>
<td>Subtract Word</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>SWCY reg,reg</td>
<td>Subtract Word With Carry</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>ACY  reg</td>
<td>Add Carry Register</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>VR  reg[reg]</td>
<td>Invert Register</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>CPLSR reg</td>
<td>Copy Level Status Register</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>SEIND reg</td>
<td>Set Indicators</td>
</tr>
<tr>
<td>Operation code</td>
<td>R1</td>
<td>R2</td>
<td>Function</td>
</tr>
<tr>
<td>---------------</td>
<td>----</td>
<td>----</td>
<td>--------------------------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0-7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>SLC</td>
<td>reg,reg</td>
<td>Shift Left Circular</td>
</tr>
<tr>
<td>1</td>
<td>SLL</td>
<td>reg,reg</td>
<td>Shift Left Logical</td>
</tr>
<tr>
<td>2</td>
<td>SRL</td>
<td>reg,reg</td>
<td>Shift Right Logical</td>
</tr>
<tr>
<td>3</td>
<td>SRA</td>
<td>reg,reg</td>
<td>Shift Right Arithmetic</td>
</tr>
<tr>
<td>4</td>
<td>SLCD</td>
<td>reg,reg</td>
<td>Shift Left Circular Double</td>
</tr>
<tr>
<td>5</td>
<td>SLLD</td>
<td>reg,reg</td>
<td>Shift Left Logical Double</td>
</tr>
<tr>
<td>6</td>
<td>SRLD</td>
<td>reg,reg</td>
<td>Shift Right Logical Double</td>
</tr>
<tr>
<td>7</td>
<td>SRAD</td>
<td>reg,reg</td>
<td>Shift Right Arithmetic Double</td>
</tr>
<tr>
<td>8</td>
<td>(invalid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SLT</td>
<td>reg,reg</td>
<td>Shift Left and Test</td>
</tr>
<tr>
<td>A</td>
<td>(invalid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>(invalid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>(invalid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>(invalid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>SLTD</td>
<td>reg,reg</td>
<td>Shift Left and Test Double</td>
</tr>
<tr>
<td>F</td>
<td>(invalid)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation code</td>
<td>R1</td>
<td>R2</td>
<td>Function</td>
</tr>
<tr>
<td>---------------</td>
<td>----</td>
<td>----</td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8–F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0–F</td>
</tr>
<tr>
<td>7</td>
<td>8–F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>NWI</td>
<td>word,reg [,reg]</td>
<td>And Word Immediate</td>
</tr>
<tr>
<td>1</td>
<td>AWI</td>
<td>word,reg [,reg]</td>
<td>Add Word Immediate</td>
</tr>
<tr>
<td>2</td>
<td>SWI</td>
<td>word,reg [,reg]</td>
<td>Subtract Word Immediate</td>
</tr>
<tr>
<td>3</td>
<td>OWI</td>
<td>word,reg [,reg]</td>
<td>OR Word Immediate</td>
</tr>
<tr>
<td>4</td>
<td>RBTWI</td>
<td>word,reg [,reg]</td>
<td>Reset Bits Word Immediate</td>
</tr>
<tr>
<td>5</td>
<td>XWI</td>
<td>word,reg [,reg]</td>
<td>Exclusive OR Word Immediate</td>
</tr>
<tr>
<td>6</td>
<td>CWI</td>
<td>word,reg</td>
<td>Compare Word Immediate</td>
</tr>
<tr>
<td>7</td>
<td>TWI</td>
<td>word,reg</td>
<td>Test Word Under Mask Immediate</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>(invalid)</td>
<td></td>
</tr>
</tbody>
</table>
The document contains tables and text sections describing operation codes and functions. Here is a transcription of the content:

### Operation Code Table: 01111000 R2 Function

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

### Operation Code Table: C IC IC IC IC IC IC IC IC 0 1 2 3 4 5 7 8 10 11 15

<table>
<thead>
<tr>
<th>Operation code</th>
<th>K</th>
<th>R</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>8-F</td>
<td>1, 3, 5, 7, 9, B, D, F</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8-F</td>
<td>0-F</td>
<td></td>
</tr>
</tbody>
</table>

#### Function Descriptions

- **Set Console Data Lights**
  - 7 8-F 0 SECON reg
  - (invalid)
  - (invalid)
  - (invalid)
  - (invalid)
  - (invalid)
  - (invalid)
  - (invalid)
  - (invalid)
  - Note 12.
  - Supervisor state: No-op
  - Problem state: program check, privilege violate

- **Copy Console Data Buffer**
  - 7 8-F 8 CPOCON reg

- **Copy Current Level**
  - 7 8-F 9 CPCL reg

- **Move Byte**
  - 8 0-7 X 04 MVB addr5,addr4

- **OR Byte**
  - 8 0-7 X 13 OB addr5,addr4

- **Reset Bits Byte**
  - 8 0-7 X 23 RBTB addr5,addr4

- **Compare Byte**
  - 8 0-7 X 33 CB addr5,addr4

Note: The tables and descriptions are extracted from the document and are laid out in a natural text format.
### Operation code

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**AM appended words**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td>10010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**AM appended words**

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>10011</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 8xxx

- **8-F X 0.4**
  - **MVW**: `addr5,addr4`
  - **Move Word**
- **1.5**
  - **OW**: `addr5,addr4`
  - **OR Word**
- **2.5**
  - **RBTW**: `addr5,addr4`
  - **Reset Bits Word**
- **3.7**
  - **CW**: `addr5,addr4`
  - **Compare Word**

### 9xxx

- **9 0-7 X 0.4**
  - **MVD**: `addr5,addr4`
  - **Move Double Word**
- **1.5**
  - **OD**: `addr5,addr4`
  - **OR Double Word**
- **2.6**
  - **RBTW**: `addr5,addr4`
  - **Reset Bits Double Word**
- **3.7**
  - **CD**: `addr5,addr4`
  - **Compare Double Word**

### JAL

- **9 8-F X X**
  - **JAL**: `jdisp,reg`
  - **Jump and Link**
  - **JAL**: `jaddr,reg`
  - **Jump and Link**
### Axxx

**Bxxx**

#### Instruction Formats A-15

**Operation code**

<table>
<thead>
<tr>
<th>1 0 1 0 0</th>
<th>R1</th>
<th>RB</th>
<th>0</th>
<th>Word disp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0-11, 15</td>
</tr>
</tbody>
</table>

0=Direct address; 1=Indirect address

**Operation code**

<table>
<thead>
<tr>
<th>1 0 1 0 0</th>
<th>R1</th>
<th>RB</th>
<th>1</th>
<th>Word disp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0-11, 15</td>
</tr>
</tbody>
</table>

0=Direct address; 1=Indirect address

**Operation code**

<table>
<thead>
<tr>
<th>1 0 1 0 1</th>
<th>RB1</th>
<th>RB2</th>
<th>AM1</th>
<th>AM2</th>
<th>Fun</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A 8-F X 0-F AM appended words

**Operation code**

<table>
<thead>
<tr>
<th>1 0 1 1 0</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-7</td>
<td>X</td>
</tr>
</tbody>
</table>

B 0-7 X X Unsupported operation code (Soft exception trap condition)
<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 1 1</td>
<td>4 5 7 8 15</td>
<td></td>
</tr>
</tbody>
</table>

| B 8-F X X | JCT | jdisp,reg | Jump on Count |
| B 8-F X X | JCT | jaddr,reg | Jump on Count |

0=Storage to register; 1=Register to storage

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0 0</td>
<td>4 5 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C 0-7 X</td>
<td>0-B, E, F</td>
<td>AM appended word</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| C 0-7 X | 0 | MVB | addr4,reg | Move Byte |
| 1 | OB | addr4,reg | OR Byte |
| 2 | RBTB | addr4,reg | Reset Bits Byte |
| 3 | XB | addr4,reg | Exclusive OR Byte |
| 4 | CB | addr4,reg | Compare Byte |
| 5 | MVBZ | addr4,reg | Move Byte and Zero |
| 6 | AB | addr4,reg | Add Byte |
| 7 | SB | addr4,reg | Subtract Byte |
| 8 | MVB | reg,addr4 | Move Byte |
| 9 | OB | reg,addr4 | OR Byte |
| A | RBTB | reg,addr4 | Reset Bits Byte |
| B | XB | reg,addr4 | Exclusive OR Byte |
| E | AB | reg,addr4 | Add Byte |
| F | SB | reg,addr4 | Subtract Byte |
### Instruction Formats

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>X</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

- **0** = Storage to register; **1** = Register to storage

#### C8-FX

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MVW</td>
<td>Move Word</td>
</tr>
<tr>
<td>1</td>
<td>OW</td>
<td>OR Word</td>
</tr>
<tr>
<td>2</td>
<td>RBTW</td>
<td>Reset Bits Word</td>
</tr>
<tr>
<td>3</td>
<td>XW</td>
<td>Exclusive OR Word</td>
</tr>
<tr>
<td>4</td>
<td>CW</td>
<td>Compare Word</td>
</tr>
<tr>
<td>5</td>
<td>MVWZ</td>
<td>Move Word and Zero</td>
</tr>
<tr>
<td>6</td>
<td>AW</td>
<td>Add Word</td>
</tr>
<tr>
<td>7</td>
<td>SW</td>
<td>Subtract Word</td>
</tr>
<tr>
<td>8</td>
<td>MVW</td>
<td>Move Word</td>
</tr>
<tr>
<td>9</td>
<td>OW</td>
<td>OR Word</td>
</tr>
<tr>
<td>A</td>
<td>RBTW</td>
<td>Reset Bits Word</td>
</tr>
<tr>
<td>B</td>
<td>XW</td>
<td>Exclusive OR Word</td>
</tr>
<tr>
<td>E</td>
<td>AW</td>
<td>Add Word</td>
</tr>
<tr>
<td>F</td>
<td>SW</td>
<td>Subtract Word</td>
</tr>
</tbody>
</table>

---

Instruction Formats  B-17
### Operation code table

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0 0 1 0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

- **D** = Storage to register; **I** = Register to storage
- **0–7** = Function
- **0–B, E, F** = AM appended word

#### D 0–7 X

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>MVD addr4,reg</td>
</tr>
<tr>
<td>1</td>
<td>OD addr4,reg</td>
</tr>
<tr>
<td>2</td>
<td>RBTD addr4,reg</td>
</tr>
<tr>
<td>3</td>
<td>XD addr4,reg</td>
</tr>
<tr>
<td>4</td>
<td>CD addr4,reg</td>
</tr>
<tr>
<td>5</td>
<td>MVDZ addr4,reg</td>
</tr>
<tr>
<td>6</td>
<td>AD addr4,reg</td>
</tr>
<tr>
<td>7</td>
<td>SD addr4,reg</td>
</tr>
<tr>
<td>8</td>
<td>MVD reg,addr4</td>
</tr>
<tr>
<td>9</td>
<td>OD reg,addr4</td>
</tr>
<tr>
<td>A</td>
<td>RBTD reg,addr4</td>
</tr>
<tr>
<td>B</td>
<td>XD reg,addr4</td>
</tr>
<tr>
<td>E</td>
<td>AD reg,addr4</td>
</tr>
<tr>
<td>F</td>
<td>SD reg,addr4</td>
</tr>
</tbody>
</table>

#### D 8–F X X

**Illegal operation code (Program check condition)**
**Exxx**

**Operation code**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Operation Code**

0 = Direct address; 1 = Indirect address

**Format**

- **MVWS shortaddr, reg**
  - **MVWS**: Move Word Short
  - **shortaddr**: Short address
  - **reg**: Register

**Examples**

1. **E 0-7 X**
   - Move Word Short
2. **E 0-7 X**
   - Move Word Short
## Exxx

### Operation code

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>RB</th>
<th>AM</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 0 1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 10 1 0 0</td>
<td>E</td>
<td>1 0</td>
<td>00</td>
<td>AM appended word</td>
</tr>
</tbody>
</table>

### PSB

- **Register Address 4**

### MB

- **Register Address 4**, **Register**

### DB

- **Register Address 4**, **Register**

### PB

- **Register Address 4**, **Register**

### PSW

- **Register**, **Register Address 4**

### MW

- **Register Address 4**, **Register**

### DW

- **Register Address 4**, **Register**

### PW

- **Register Address 4**, **Register**

### PSD

- **Register**, **Register Address 4**

### MD

- **Register Address 4**, **Register**

### DD

- **Register Address 4**, **Register**

### PD

- **Register Address 4**, **Register**

### C

- **Invalid**

### D

- **Invalid**

### E

- **Invalid**

### F

- **Invalid**

## Fxxx

### Operation code

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 0</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>0 4 5 7 8</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

### CBI

- **Byte, Register**

### Compare Byte Immediate

### Operation code

<table>
<thead>
<tr>
<th>Operation code</th>
<th>R</th>
<th>Word displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>0 4 5 7 8</td>
<td>8-F</td>
<td></td>
</tr>
</tbody>
</table>

### BALS

- **Register, Register Address**

### Branch and Link Short
Coding Notes

1. Data flow, when it modifies a field, is always from left to right.
2. Registers used in effective address calculations are always in parentheses.
3. An address specification followed by an asterisk indicates indirect addressing. Here, the effective address is the contents of the addressed storage location.
4. The (reg)+ format indicates that, after use, the contents of reg are increased by the number of bytes addressed.
5. AM indicates address mode.

Legend for Machine Instruction Operands

<table>
<thead>
<tr>
<th>abcnt</th>
<th>addr</th>
<th>addr4</th>
<th>addr5</th>
</tr>
</thead>
<tbody>
<tr>
<td>An absolute value or expression representing the size of a work storage area to be allocated by the Store Multiple (STM) instruction. The value you code must be an even number in the range 0–16382.</td>
<td>An address value. Code an absolute or relocatable expression in the range 0–65535.</td>
<td>An address value that you code in one of the following forms: (reg 0–3) The effective address is the contents of the reg 0–3. (AM=00) (reg 0–3) + The effective address is the contents of the register reg 0–3. After an instruction uses it, the contents of the register are increased by the number of bytes addressed by the instruction. (AM=01) addr The effective address is the value of addr, unless the instruction and addr are within the range of the same USING statement. If they are, the assembler computes the effective address as a displacement (-32768 to +32767 or 0 to 65535) from the base register, which must be reg 1–3. (AM=10) addr* The effective address is the contents of storage at the address defined by addr, unless the instruction and addr are within the domain and range of the same USING statement. If they are, the assembler computes the effective address as the contents of storage at the address defined by a displacement (0 255) from the base register, which must be reg 1–3. (AM=11)</td>
<td></td>
</tr>
</tbody>
</table>
addr* The effective address is the contents of storage at the address defined by addr, unless the instruction and addr are within the domain and range of the same USING statement. If they are, the assembler computes the effective address as the contents of storage at the address defined by a displacement (0-255) from the base register, which must be reg 1-7. (AM=10)

(reg 1-7 ,waddr) The effective address is the contents of reg 1-7 , added to the value of waddr. (AM=10)

disp(reg 1-7 ,disp2)* The effective address is calculated as follows: The contents of the register reg 1-7 are added to the value of the displacement disp2 to form an address. The contents of that storage location are added to the value of disp2 to form the effective address. (AM=11)

disp(reg 1-7 )* The effective address is the contents of storage at the address defined by the contents of reg 1-7 , added to the value of disp. (AM=11)

(reg 1-7 )* The effective address is the contents of storage at the address defined by the contents of reg 1-7. (AM=11)

(reg 1-7 ,disp)* The contents of reg 1-7 are added to disp, forming an address. The contents of storage at that address form the effective address. (AM=11)

For byte addressing, the effective address can be even or odd. For word or doubleword addressing, the effective address must be even.

bitdisp A displacement into a bit field. Code an absolute value or expression in the range 0–63.

byte A byte value. Code an absolute value or expression in the range -128 to +127 or 0 to 255.

cnt16 A single word (one register) shift count. Code an absolute value or expression in the range 0–16.

cnt31 A doubleword (register pair) shift count. Code an absolute value or expression in the range 0–31.

cond A condition code value. Code an absolute value or expression in the range 0–7.

disp A byte address displacement. Code an absolute value or expression in the range 0–255.

freg A floating-point register. Code either a predefined floating register symbol (FR0–FR3) or a symbol that is equated to the desired register number (0, 1, 2, or 3). Symbols are equated with EQUR statements, which must precede the instruction using the register symbol.

jaddr The address of an instruction that is within -256 to +254 bytes of the byte following a jump instruction. Code a relocatable expression.

jdisp A displacement from the byte following a jump instruction. Code an absolute value or expression in the range -256 to +254.

longaddr An address value that you code in one of the following forms:

addr* The effective address is the contents of storage at the address defined by addr, unless the instruction and addr are within the domain and range of the same USING statement. If they are, the assembler computes the effective address as the contents of storage at the address defined by a displacement (-32768 to +32767 or 0 to 65535) from the base register, which must be reg 1-7.

(reg 1-7 ,waddr) The effective address is the contents of reg 1-7 , added to the value of waddr.

(reg 1-7 ,waddr)* The contents of the reg 1-7 , plus waddr, form an address. The contents of storage at that location form the effective address.

(reg 1-7 ) The effective address is the contents of the register reg 1-7.

(reg 1-7 )* The effective address is the contents of storage at the address defined by the contents of reg 1-7.

raddr An address value. Code a relocatable expression in the range 0–65535.

reg A general-purpose register. Code either a predefined register symbol (R0–R7) or a symbol that is equated to the desired register number (0, 1, 2, 3, 4, 5, 6, or 7). Symbols are equated with EQUR statements, which must precede the instruction using the register symbol.

reg0-3 A general-purpose register. Code either a predefined register symbol (R0–R3) or a symbol that is equated to the desired register number (0, 1, 2, or 3). Symbols are equated with EQUR statements, which must precede the instruction using the register symbol.
A general-purpose register. Code either a predefined register symbol (R1–R3) or a symbol that is equated to the desired register number (1, 2, or 3). Symbols are equated with EQUR statements, which must precede the instruction using the register symbol.

A general-purpose register. Code either a predefined register symbol (R1–R7) or a symbol that is equated to the desired register number (1, 2, 3, 4, 5, 6, or 7). Symbols are equated with EQUR statements, which must precede the instructions using the register symbol.

An address value that you code in one of the following forms:

- \((\text{reg}^{0-3}, \text{wdisp})\) The effective address is the value of \(\text{wdisp}\) added to the contents of \(\text{reg}^{0-3}\).
- \((\text{reg}^{0-3}, \text{wdisp})^*\) The effective address is the contents of storage at the address defined by the value of \(\text{wdisp}\) added to the contents of \(\text{reg}^{0-3}\).
- \((\text{reg}^{0-3})\) The effective address is the contents of \(\text{reg}^{0-3}\).
- \((\text{reg}^{0-3})^*\) The effective address is the contents of storage at the address defined by the contents of \(\text{reg}^{0-3}\).

To use this form, the instruction and addr must be in the domain and range of the same USING statement. The assembler computes a displacement (0–62) and register combination that refers to the requested location.

Same as addr, except the assembler computes the effective address as the contents of storage at the address defined by a displacement (0–62) and register combination.

Note: For addr and addr*, the base register must be \(\text{reg}^{0-3}\).

An unsigned byte value or mask. Code an absolute value or expression in the range 0–255.

An ordinary symbol that is defined externally from the current source program.

A one-word address value. Code an absolute or relocatable expression in the range –32768 to +32767 or 0 to 65535.

An even byte address displacement. Code an absolute value or expression in the range 0–62.

A word value. Code an absolute value or expression in the range –32768 to +32767 or 0 to 65535.
Binary and Hexadecimal Number Notations

**Binary Number Notation**

A binary number system, such as is used in Series/I, uses a base of two. The concept of using a base of two can be compared with the base of ten (decimal) number system.

<table>
<thead>
<tr>
<th>Decimal number</th>
<th>Binary number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>= 0</td>
</tr>
<tr>
<td>1</td>
<td>= 1</td>
</tr>
<tr>
<td>2</td>
<td>= 10</td>
</tr>
<tr>
<td>3</td>
<td>= 11</td>
</tr>
<tr>
<td>4</td>
<td>= 100</td>
</tr>
<tr>
<td>5</td>
<td>= 101</td>
</tr>
<tr>
<td>6</td>
<td>= 110</td>
</tr>
<tr>
<td>7</td>
<td>= 111</td>
</tr>
<tr>
<td>8</td>
<td>= 1000</td>
</tr>
<tr>
<td>9</td>
<td>= 1001</td>
</tr>
</tbody>
</table>

**Example of a decimal number:**

\[
\begin{array}{c}
1 \times 10^3 \\
2 \times 10^2 \\
3 \times 10^1 \\
9 \times 10^0 \\
\end{array}
\]

\[+ 9 \text{ units position} + 30 \text{ tens position} + 200 \text{ hundreds position} + 1000 \text{ thousands position} = 1239 = \text{decimal number}\]

As shown above, the decimal number system allows counting to ten in each position from units to tens to hundreds to thousands, etc. The binary system allows counting to two in each position. Register displays in the Series/I are in binary form: a bit light on is a 1; a bit light off is a 0.

**Example of a binary number:**

\[
\begin{array}{c}
1 \times 2^3 \\
0 \times 2^2 \\
0 \times 2^1 \\
1 \times 2^0 \\
\end{array}
\]

\[+0001 = \text{decimal 1} +0000 = \text{decimal 0} +0000 = \text{decimal 0}
\]

\[1000 = \text{decimal 8} \quad 1001 = \text{decimal 9}\]

**Hexadecimal Number System**

It has been noted that binary numbers require about three times as many positions as decimal numbers to express the equivalent number. This is not much of a problem to the computer; however, in talking and writing or in communicating with the computer, these binary numbers are bulky. A long string of 1's and 0's cannot be effectively transmitted from one individual to another. Some shorthand method is necessary.

The hexadecimal number system fills this need. Because of the simple relationship of hexadecimal to binary, numbers can be converted from one system to another by inspection. The base or radix of the hexadecimal system is 16. This means there are 16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The letters A, B, C, D, E, and F represent the 10-base system values of 10, 11, 12, 13, 14, and 15, respectively.

Four binary positions are equivalent to one hexadecimal position. The following table shows the comparable values of the three number systems.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

At this point, all 16 symbols have been used, and a carry to the next higher position of the number is necessary. For example:

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0001 0000</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>0001 0001</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>0001 0010</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>0001 0011</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>0001 0100</td>
<td>14</td>
</tr>
<tr>
<td>21</td>
<td>0001 0101</td>
<td>15</td>
</tr>
</tbody>
</table>

—and so on—
Remember that as far as the internal circuitry of the computer is concerned, it understands only binary. But an operator can look at a series of lights on the computer console showing binary 1's and 0's, for example: 0001 1110 0001 0011, and say that the lights represent the hexadecimal value $\text{1E13}$, which is easier to state than the string of 1's and 0's.

### Hexadecimal—Decimal Conversion

#### Tables

The table in this appendix provides for direct conversion of decimal and hexadecimal number in these ranges:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0002</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>FFFF</td>
<td>4095</td>
</tr>
</tbody>
</table>

For numbers outside the range of the table, add the following values to the tables figures:

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>16384</td>
</tr>
<tr>
<td>5000</td>
<td>20480</td>
</tr>
<tr>
<td>6000</td>
<td>24576</td>
</tr>
<tr>
<td>7000</td>
<td>28672</td>
</tr>
<tr>
<td>8000</td>
<td>32768</td>
</tr>
</tbody>
</table>

<p>| 1000        | 4096    |
| 2000        | 8192    |
| 3000        | 12288   |
| 4000        | 16384   |
| 5000        | 20480   |
| 6000        | 24576   |
| 7000        | 28672   |
| 8000        | 32768   |</p>
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>2560</td>
<td>2561</td>
<td>2562</td>
<td>2563</td>
<td>2564</td>
<td>2565</td>
</tr>
<tr>
<td>A1</td>
<td>2570</td>
<td>2571</td>
<td>2572</td>
<td>2573</td>
<td>2574</td>
<td>2575</td>
</tr>
<tr>
<td>A2</td>
<td>2580</td>
<td>2581</td>
<td>2582</td>
<td>2583</td>
<td>2584</td>
<td>2585</td>
</tr>
<tr>
<td>A3</td>
<td>2590</td>
<td>2591</td>
<td>2592</td>
<td>2593</td>
<td>2594</td>
<td>2595</td>
</tr>
<tr>
<td>A4</td>
<td>2600</td>
<td>2601</td>
<td>2602</td>
<td>2603</td>
<td>2604</td>
<td>2605</td>
</tr>
<tr>
<td>A5</td>
<td>2610</td>
<td>2611</td>
<td>2612</td>
<td>2613</td>
<td>2614</td>
<td>2615</td>
</tr>
<tr>
<td>A6</td>
<td>2620</td>
<td>2621</td>
<td>2622</td>
<td>2623</td>
<td>2624</td>
<td>2625</td>
</tr>
<tr>
<td>A7</td>
<td>2630</td>
<td>2631</td>
<td>2632</td>
<td>2633</td>
<td>2634</td>
<td>2635</td>
</tr>
<tr>
<td>A8</td>
<td>2640</td>
<td>2641</td>
<td>2642</td>
<td>2643</td>
<td>2644</td>
<td>2645</td>
</tr>
<tr>
<td>A9</td>
<td>2650</td>
<td>2651</td>
<td>2652</td>
<td>2653</td>
<td>2654</td>
<td>2655</td>
</tr>
<tr>
<td>A10</td>
<td>2660</td>
<td>2661</td>
<td>2662</td>
<td>2663</td>
<td>2664</td>
<td>2665</td>
</tr>
<tr>
<td>A11</td>
<td>2670</td>
<td>2671</td>
<td>2672</td>
<td>2673</td>
<td>2674</td>
<td>2675</td>
</tr>
<tr>
<td>A12</td>
<td>2680</td>
<td>2681</td>
<td>2682</td>
<td>2683</td>
<td>2684</td>
<td>2685</td>
</tr>
<tr>
<td>A13</td>
<td>2690</td>
<td>2700</td>
<td>2710</td>
<td>2720</td>
<td>2730</td>
<td>2740</td>
</tr>
<tr>
<td>A14</td>
<td>2750</td>
<td>2760</td>
<td>2770</td>
<td>2780</td>
<td>2790</td>
<td>2800</td>
</tr>
<tr>
<td>A15</td>
<td>2810</td>
<td>2820</td>
<td>2830</td>
<td>2840</td>
<td>2850</td>
<td>2860</td>
</tr>
<tr>
<td>A16</td>
<td>2870</td>
<td>2880</td>
<td>2890</td>
<td>2900</td>
<td>2910</td>
<td>2920</td>
</tr>
<tr>
<td>A17</td>
<td>2930</td>
<td>2940</td>
<td>2950</td>
<td>2960</td>
<td>2970</td>
<td>2980</td>
</tr>
<tr>
<td>A18</td>
<td>2990</td>
<td>3000</td>
<td>3010</td>
<td>3020</td>
<td>3030</td>
<td>3040</td>
</tr>
<tr>
<td>A19</td>
<td>3050</td>
<td>3060</td>
<td>3070</td>
<td>3080</td>
<td>3090</td>
<td>3100</td>
</tr>
<tr>
<td>A20</td>
<td>3110</td>
<td>3120</td>
<td>3130</td>
<td>3140</td>
<td>3150</td>
<td>3160</td>
</tr>
<tr>
<td>B0</td>
<td>4000</td>
<td>4010</td>
<td>4020</td>
<td>4030</td>
<td>4040</td>
<td>4050</td>
</tr>
<tr>
<td>B1</td>
<td>4060</td>
<td>4070</td>
<td>4080</td>
<td>4090</td>
<td>4100</td>
<td>4110</td>
</tr>
<tr>
<td>B2</td>
<td>4120</td>
<td>4130</td>
<td>4140</td>
<td>4150</td>
<td>4160</td>
<td>4170</td>
</tr>
<tr>
<td>B3</td>
<td>4180</td>
<td>4190</td>
<td>4200</td>
<td>4210</td>
<td>4220</td>
<td>4230</td>
</tr>
<tr>
<td>B4</td>
<td>4240</td>
<td>4250</td>
<td>4260</td>
<td>4270</td>
<td>4280</td>
<td>4290</td>
</tr>
<tr>
<td>B5</td>
<td>4300</td>
<td>4310</td>
<td>4320</td>
<td>4330</td>
<td>4340</td>
<td>4350</td>
</tr>
<tr>
<td>B6</td>
<td>4360</td>
<td>4370</td>
<td>4380</td>
<td>4390</td>
<td>4400</td>
<td>4410</td>
</tr>
<tr>
<td>B7</td>
<td>4420</td>
<td>4430</td>
<td>4440</td>
<td>4450</td>
<td>4460</td>
<td>4470</td>
</tr>
<tr>
<td>B8</td>
<td>4480</td>
<td>4490</td>
<td>4500</td>
<td>4510</td>
<td>4520</td>
<td>4530</td>
</tr>
<tr>
<td>B9</td>
<td>4540</td>
<td>4550</td>
<td>4560</td>
<td>4570</td>
<td>4580</td>
<td>4590</td>
</tr>
</tbody>
</table>

### Numbering Systems and Conversion Tables D-5
<table>
<thead>
<tr>
<th>$2^n$</th>
<th>$2^m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
</tr>
<tr>
<td>32</td>
<td>0.3125</td>
</tr>
<tr>
<td>64</td>
<td>0.1562</td>
</tr>
<tr>
<td>128</td>
<td>0.0078</td>
</tr>
<tr>
<td>256</td>
<td>0.0039</td>
</tr>
<tr>
<td>512</td>
<td>0.0019</td>
</tr>
<tr>
<td>1,024</td>
<td>0.0009</td>
</tr>
<tr>
<td>2,048</td>
<td>0.0004</td>
</tr>
<tr>
<td>4,096</td>
<td>0.0002</td>
</tr>
<tr>
<td>8,192</td>
<td>0.0001</td>
</tr>
<tr>
<td>16,384</td>
<td>0.00006</td>
</tr>
<tr>
<td>32,768</td>
<td>0.00003</td>
</tr>
<tr>
<td>65,536</td>
<td>0.000015</td>
</tr>
<tr>
<td>131,072</td>
<td>0.000007</td>
</tr>
<tr>
<td>262,144</td>
<td>0.0000037</td>
</tr>
<tr>
<td>524,288</td>
<td>0.0000019</td>
</tr>
<tr>
<td>1,048,576</td>
<td>0.00000095</td>
</tr>
<tr>
<td>2,097,152</td>
<td>0.00000048</td>
</tr>
<tr>
<td>4,194,304</td>
<td>0.00000024</td>
</tr>
<tr>
<td>8,388,608</td>
<td>0.00000012</td>
</tr>
<tr>
<td>16,777,216</td>
<td>0.00000006</td>
</tr>
<tr>
<td>33,554,432</td>
<td>0.00000003</td>
</tr>
<tr>
<td>67,108,864</td>
<td>0.000000015</td>
</tr>
<tr>
<td>134,217,728</td>
<td>0.000000007</td>
</tr>
<tr>
<td>268,435,456</td>
<td>0.0000000037</td>
</tr>
<tr>
<td>536,870,912</td>
<td>0.0000000018</td>
</tr>
<tr>
<td>1,073,741,824</td>
<td>0.0000000009</td>
</tr>
<tr>
<td>2,147,483,648</td>
<td>0.00000000045</td>
</tr>
<tr>
<td>4,294,967,296</td>
<td>0.00000000023</td>
</tr>
<tr>
<td>8,589,934,592</td>
<td>0.00000000012</td>
</tr>
<tr>
<td>17,179,869,184</td>
<td>0.00000000006</td>
</tr>
<tr>
<td>34,359,738,368</td>
<td>0.00000000003</td>
</tr>
<tr>
<td>68,719,476,736</td>
<td>0.000000000015</td>
</tr>
<tr>
<td>137,438,953,472</td>
<td>0.000000000007</td>
</tr>
<tr>
<td>274,877,906,944</td>
<td>0.0000000000037</td>
</tr>
<tr>
<td>549,755,813,888</td>
<td>0.0000000000019</td>
</tr>
<tr>
<td>1,099,511,627,776</td>
<td>0.0000000000009</td>
</tr>
<tr>
<td>2,199,023,255,552</td>
<td>0.00000000000046</td>
</tr>
<tr>
<td>4,398,046,511,104</td>
<td>0.00000000000023</td>
</tr>
<tr>
<td>8,796,093,022,208</td>
<td>0.00000000000012</td>
</tr>
<tr>
<td>17,592,186,044,416</td>
<td>0.00000000000006</td>
</tr>
<tr>
<td>35,184,372,088,832</td>
<td>0.00000000000003</td>
</tr>
<tr>
<td>70,368,744,177,664</td>
<td>0.000000000000015</td>
</tr>
<tr>
<td>140,737,488,355,328</td>
<td>0.000000000000007</td>
</tr>
<tr>
<td>281,474,976,710,656</td>
<td>0.0000000000000035</td>
</tr>
<tr>
<td>562,949,953,421,312</td>
<td>0.00000000000000176</td>
</tr>
<tr>
<td>1,125,899,906,842,624</td>
<td>0.000000000000000881</td>
</tr>
<tr>
<td>2,251,799,813,685,248</td>
<td>0.0000000000000004441</td>
</tr>
<tr>
<td>4,503,599,627,370,496</td>
<td>0.0000000000000002234</td>
</tr>
<tr>
<td>9,007,199,254,740,992</td>
<td>0.0000000000000001117</td>
</tr>
<tr>
<td>18,014,398,508,914,988</td>
<td>0.0000000000000000555</td>
</tr>
<tr>
<td>36,028,797,918,993,976</td>
<td>0.0000000000000000277</td>
</tr>
<tr>
<td>72,057,594,037,927,936</td>
<td>0.0000000000000000139</td>
</tr>
<tr>
<td>144,115,188,075,855,872</td>
<td>0.0000000000000000069</td>
</tr>
<tr>
<td>288,230,376,151,711,844</td>
<td>0.0000000000000000346</td>
</tr>
<tr>
<td>576,460,752,303,123,688</td>
<td>0.0000000000000000173</td>
</tr>
<tr>
<td>1,152,921,504,606,814,784</td>
<td>0.0000000000000000086</td>
</tr>
<tr>
<td>2,305,843,009,213,699,568</td>
<td>0.0000000000000000043</td>
</tr>
<tr>
<td>4,611,686,018,427,388,110</td>
<td>0.0000000000000000021</td>
</tr>
<tr>
<td>9,223,372,036,854,775,820</td>
<td>0.0000000000000000010</td>
</tr>
<tr>
<td>18,446,744,073,709,551,641</td>
<td>0.0000000000000000005</td>
</tr>
</tbody>
</table>

Powers of Two Table
### Powers of Two Table

<table>
<thead>
<tr>
<th>$n$</th>
<th>(2^n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
</tr>
<tr>
<td>18</td>
<td>262144</td>
</tr>
<tr>
<td>19</td>
<td>524288</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
</tr>
</tbody>
</table>

... (Continued with more rows)
## Appendix E. Character Codes

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
<th>EBCDIC</th>
<th>ASCII</th>
<th>Eight bit data interchange</th>
<th>PTTC/EBCD</th>
<th>PTTC/Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0000</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td></td>
<td>space</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>0001</td>
<td>SOH</td>
<td>SOH</td>
<td>space</td>
<td></td>
<td>space</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0010</td>
<td>STX</td>
<td>STX</td>
<td>@</td>
<td>1</td>
<td>1,1</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>0011</td>
<td>ETX</td>
<td>ETX</td>
<td>space</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>0100</td>
<td>PF</td>
<td>EOT</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>0101</td>
<td>HT</td>
<td>ENQ</td>
<td>space</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>0110</td>
<td>LC</td>
<td>ACK</td>
<td>space</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>0111</td>
<td>DEL</td>
<td>BEL</td>
<td>space</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>1000</td>
<td>BS</td>
<td></td>
<td>space</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>1001</td>
<td>RLF</td>
<td>HT</td>
<td>space</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>1010</td>
<td>SMM</td>
<td>LF</td>
<td>(even parity)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>1011</td>
<td>VT</td>
<td>VT</td>
<td>(odd parity)</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>1100</td>
<td>FF</td>
<td>FF</td>
<td>0 (even parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>1101</td>
<td>CR</td>
<td>CR</td>
<td>0 (odd parity)</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>1110</td>
<td>SO</td>
<td>SO</td>
<td>space</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>1111</td>
<td>SI</td>
<td>SI</td>
<td>space</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>0001 0000</td>
<td>DLE</td>
<td>DLE</td>
<td>space</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>0001</td>
<td>DC1</td>
<td>DC1</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>0010</td>
<td>DC2</td>
<td>DC2</td>
<td>H (even parity)</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>0011</td>
<td>TM</td>
<td>DC3</td>
<td>H (odd parity)</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>0100</td>
<td>RES</td>
<td>DC4</td>
<td>(even parity)</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>0101</td>
<td>NL</td>
<td>NAK</td>
<td>(odd parity)</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>0110</td>
<td>BS</td>
<td>SYN</td>
<td>(EOA)</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>0111</td>
<td>IL</td>
<td>ETB</td>
<td>(EOA),9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>1000</td>
<td>CAN</td>
<td>CAN</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>1001</td>
<td>EM</td>
<td>EM</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>1010</td>
<td>CC</td>
<td>SUB</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>1011</td>
<td>CU1</td>
<td>ESC</td>
<td>X (upper case)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>1100</td>
<td>IFS</td>
<td>FS</td>
<td>upper case</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>1101</td>
<td>IGS</td>
<td>GS</td>
<td>(EOA)</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>1110</td>
<td>RS</td>
<td>RS</td>
<td>(EOA)</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>1111</td>
<td>IUS</td>
<td>US</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>20</td>
<td>0010 0000</td>
<td>DS</td>
<td>space</td>
<td>space</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
<td>0001</td>
<td>SOS</td>
<td>!</td>
<td>EOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>22</td>
<td>0010</td>
<td>FS</td>
<td>&quot;</td>
<td>D (even parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>23</td>
<td>0011</td>
<td>#</td>
<td>D (odd parity)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>24</td>
<td>0100</td>
<td>BYP</td>
<td>$</td>
<td>S (even parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>25</td>
<td>0101</td>
<td>LF</td>
<td>%</td>
<td>S (odd parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>26</td>
<td>0110</td>
<td>ETB</td>
<td>&amp;</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>27</td>
<td>0111</td>
<td>ESC</td>
<td>'</td>
<td>@</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>28</td>
<td>1000</td>
<td></td>
<td>(</td>
<td>u</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>29</td>
<td>1001</td>
<td></td>
<td>)</td>
<td>e</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
<td>1010</td>
<td>SM</td>
<td>*</td>
<td>v</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
<td>1101</td>
<td>CU2</td>
<td>+</td>
<td>w</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
<td>1100</td>
<td></td>
<td></td>
<td>k</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
<td>1101</td>
<td>ENQ</td>
<td>-</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
<td>1110</td>
<td>ACK</td>
<td>.</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
<td>1111</td>
<td>BFL</td>
<td>/</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>30</td>
<td>0011 0000</td>
<td>0</td>
<td>form feed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>31</td>
<td>0001</td>
<td>1</td>
<td>form feed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>0010</td>
<td>SYN</td>
<td>2</td>
<td>y</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>51</td>
<td>33</td>
<td>0011</td>
<td>3</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>34</td>
<td>0100</td>
<td>PN</td>
<td>4</td>
<td>z</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>53</td>
<td>35</td>
<td>0101</td>
<td>RS</td>
<td>5</td>
<td>h</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>54</td>
<td>36</td>
<td>0110</td>
<td>UC</td>
<td>6</td>
<td>space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>Hex</td>
<td>Binary</td>
<td>EBCDIC</td>
<td>ASCII</td>
<td>Eight bit data interchange</td>
<td>PTTC/EBCD</td>
<td>PTTC/Correspondence</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------------</td>
<td>--------</td>
<td>-------</td>
<td>----------------------------</td>
<td>-----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>55</td>
<td>37</td>
<td>0011 0111</td>
<td>EOT</td>
<td>7</td>
<td>(\text{SOA}, 	ext{comma})</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>38</td>
<td>1000</td>
<td></td>
<td>8</td>
<td>(\text{even parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>39</td>
<td>1001</td>
<td></td>
<td>9</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>3A</td>
<td>1010</td>
<td></td>
<td></td>
<td>(\text{even parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>3B</td>
<td>1011 CU</td>
<td>(\text{odd parity})</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>3C</td>
<td>1100 DC4</td>
<td>&lt;</td>
<td>&lt;</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>3D</td>
<td>1101 NAK</td>
<td>=</td>
<td>&lt;</td>
<td>(\text{even parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>3E</td>
<td>1110</td>
<td>&gt;</td>
<td></td>
<td>(\text{even parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>3F</td>
<td>1111 SUB</td>
<td>?</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>40</td>
<td>0100 0000</td>
<td>space</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>41</td>
<td>0001</td>
<td></td>
<td>A</td>
<td>(\text{even parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>42</td>
<td>0010</td>
<td>B</td>
<td>(\text{odd parity})</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>43</td>
<td>0011</td>
<td>C</td>
<td>i</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>44</td>
<td>0100</td>
<td>D</td>
<td>(\text{even parity})</td>
<td>(\text{even parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>45</td>
<td>0101</td>
<td>E</td>
<td>(\text{odd parity})</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>46</td>
<td>0110</td>
<td>F</td>
<td>l</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>47</td>
<td>0111</td>
<td>G</td>
<td>(\text{odd parity})</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>48</td>
<td>1000</td>
<td>H</td>
<td>m</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>49</td>
<td>1001</td>
<td>I</td>
<td>n</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>4A</td>
<td>1010</td>
<td>J</td>
<td>(\text{odd parity})</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>4B</td>
<td>1011</td>
<td>K</td>
<td>o</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>4C</td>
<td>1100</td>
<td>L</td>
<td>i</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>4D</td>
<td>1101</td>
<td>M</td>
<td>2</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>4E</td>
<td>1110 +</td>
<td>N</td>
<td>6</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>4F</td>
<td>1111</td>
<td>O</td>
<td>6</td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>50</td>
<td>0101 0000</td>
<td>&amp;</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>51</td>
<td>0001</td>
<td>P</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>52</td>
<td>0010</td>
<td>R</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>53</td>
<td>0011</td>
<td>S</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>54</td>
<td>0100</td>
<td>T</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>55</td>
<td>0101</td>
<td>U</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>56</td>
<td>0110</td>
<td>V</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>57</td>
<td>0111</td>
<td>W</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>58</td>
<td>1000</td>
<td>X</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>59</td>
<td>1001</td>
<td>Y</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>5A</td>
<td>1010 !</td>
<td>Z</td>
<td></td>
<td>(\text{odd parity})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>5B</td>
<td>1011 $</td>
<td>\Z(\text{even parity})\</td>
<td></td>
<td>\Z(\text{odd parity})\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>5C</td>
<td>1100 *</td>
<td>{\text{odd parity}} \</td>
<td></td>
<td>{\text{odd parity}}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>5D</td>
<td>1101 +</td>
<td>\1\text{odd parity}\ \</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>5E</td>
<td>1110 :</td>
<td>A</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>5F</td>
<td>1111 -</td>
<td>-</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>60</td>
<td>0110 0000</td>
<td>-</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>61</td>
<td>0001</td>
<td>a</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>62</td>
<td>0010</td>
<td>b</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>63</td>
<td>0011</td>
<td>c</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>64</td>
<td>0100</td>
<td>d</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>65</td>
<td>0101</td>
<td>e</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>66</td>
<td>0110</td>
<td>f</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>67</td>
<td>0111</td>
<td>g</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>68</td>
<td>1000</td>
<td>h</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>69</td>
<td>1001</td>
<td>i</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>6A</td>
<td>1010 !</td>
<td>j</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>6B</td>
<td>1011 .</td>
<td>k</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>6C</td>
<td>1100 %</td>
<td>l</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>6D</td>
<td>1101 -</td>
<td>m</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>6E</td>
<td>1110 &gt;</td>
<td>n</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>6F</td>
<td>1111 ?</td>
<td>o</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>70</td>
<td>0111 0000</td>
<td>p</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>71</td>
<td>0001</td>
<td>q</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>72</td>
<td>0010</td>
<td>r</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>73</td>
<td>0011</td>
<td>s</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>74</td>
<td>0100</td>
<td>t</td>
<td></td>
<td>\1\text{odd parity}\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>Hex</td>
<td>Binary</td>
<td>EBCDIC</td>
<td>ASCII</td>
<td>Eight bit data interchange</td>
<td>PTTC/EBCD</td>
<td>PTTC/Correspondence</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
<td>---------</td>
<td>--------</td>
<td>-------</td>
<td>-----------------------------</td>
<td>-----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>117</td>
<td>75</td>
<td>0111 0101</td>
<td>u</td>
<td>. (odd parity)</td>
<td>[ ]</td>
<td>.period</td>
<td>-</td>
</tr>
<tr>
<td>118</td>
<td>76</td>
<td>0110</td>
<td>v</td>
<td>[ ]</td>
<td>.period</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>77</td>
<td>0111</td>
<td>w</td>
<td>[ ]</td>
<td>.period</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>78</td>
<td>1000</td>
<td>x</td>
<td>[ ]</td>
<td>.period</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>79</td>
<td>1001</td>
<td>y</td>
<td>[ ]</td>
<td>.period</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>7A</td>
<td>1010</td>
<td>z</td>
<td>[ ]</td>
<td>.period</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>7B</td>
<td>1011</td>
<td>#</td>
<td>{ }</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>7C</td>
<td>1100</td>
<td>@</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>7D</td>
<td>1101</td>
<td>:</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>7E</td>
<td>1110</td>
<td>=</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>7F</td>
<td>1111</td>
<td>DEL</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>80</td>
<td>1000 0000</td>
<td>``</td>
<td>space</td>
<td>space</td>
<td>z, [</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>81</td>
<td>0001</td>
<td>a</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>82</td>
<td>0010</td>
<td>b</td>
<td>A (even parity)</td>
<td>=</td>
<td>[ ]</td>
<td>-</td>
</tr>
<tr>
<td>131</td>
<td>83</td>
<td>0011</td>
<td>c</td>
<td>A (odd parity)</td>
<td>@</td>
<td>[ ]</td>
<td>-</td>
</tr>
<tr>
<td>132</td>
<td>84</td>
<td>0100</td>
<td>d</td>
<td>! (even parity)</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>85</td>
<td>0101</td>
<td>e</td>
<td>! (odd parity)</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>86</td>
<td>0110</td>
<td>f</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>87</td>
<td>0111</td>
<td>g</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>88</td>
<td>1000</td>
<td>h</td>
<td>X-ON</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>89</td>
<td>1001</td>
<td>i</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>8A</td>
<td>1010</td>
<td>Q</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>8B</td>
<td>1011</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>8C</td>
<td>1100</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>8D</td>
<td>1101</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>8E</td>
<td>1110</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>8F</td>
<td>1111</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>90</td>
<td>1001 0000</td>
<td>horiz tab</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>91</td>
<td>0001</td>
<td>j</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>92</td>
<td>0010</td>
<td>k</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>93</td>
<td>0011</td>
<td>l</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>94</td>
<td>0100</td>
<td>m</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>95</td>
<td>0101</td>
<td>n</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>96</td>
<td>0110</td>
<td>o</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>97</td>
<td>0111</td>
<td>p</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>98</td>
<td>1000</td>
<td>q</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>99</td>
<td>1001</td>
<td>r</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>9A</td>
<td>1010</td>
<td>Y (even parity)</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>9B</td>
<td>1011</td>
<td>Y (odd parity)</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>9C</td>
<td>1100</td>
<td>9 (even parity)</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>9D</td>
<td>1101</td>
<td>9 (odd parity)</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>9E</td>
<td>1110</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>9F</td>
<td>1111</td>
<td>[ ]</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>A0</td>
<td>1010 0000</td>
<td>WRU (even)</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>A1</td>
<td>0001</td>
<td>s</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>A2</td>
<td>0010</td>
<td>t</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>A3</td>
<td>0011</td>
<td>u</td>
<td>E</td>
<td>?</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>A4</td>
<td>0100</td>
<td>v</td>
<td>%</td>
<td>S</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>A5</td>
<td>0101</td>
<td>w</td>
<td>%</td>
<td>S</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>A6</td>
<td>0110</td>
<td>x</td>
<td>%</td>
<td>S</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>167</td>
<td>A7</td>
<td>0111</td>
<td>y</td>
<td>%</td>
<td>S</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>A8</td>
<td>1000</td>
<td>z</td>
<td>%</td>
<td>S</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>169</td>
<td>A9</td>
<td>1001</td>
<td>[ ]</td>
<td>U</td>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>AA</td>
<td>1010</td>
<td>U (even parity)</td>
<td>V</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>171</td>
<td>AB</td>
<td>1011</td>
<td>U (odd parity)</td>
<td>W</td>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>AC</td>
<td>1100</td>
<td>S (even parity)</td>
<td>W</td>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>AD</td>
<td>1101</td>
<td>S (odd parity)</td>
<td>W</td>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>AE</td>
<td>1110</td>
<td>[ ]</td>
<td>return</td>
<td>Y</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>AF</td>
<td>1111</td>
<td>[ ]</td>
<td>M (even parity)</td>
<td>Z</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>B0</td>
<td>1011 0000</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>B1</td>
<td>0001</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>178</td>
<td>B2</td>
<td>0010</td>
<td>[ ]</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Character Codes  E-3
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
<th>EBCDIC</th>
<th>ASCII</th>
<th>Eight bit data interchange</th>
<th>PTTC/EBCD</th>
<th>PTTC/Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td>179</td>
<td>B3</td>
<td>1011 0011</td>
<td>M (odd parity)</td>
<td>- (even parity)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>B4</td>
<td>0100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>B5</td>
<td>0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>B6</td>
<td>0110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>B7</td>
<td>0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>B8</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>B9</td>
<td>1001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>186</td>
<td>BA</td>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>187</td>
<td>BB</td>
<td>1011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>BC</td>
<td>1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>189</td>
<td>BD</td>
<td>1101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td>BE</td>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>191</td>
<td>BF</td>
<td>1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>C0</td>
<td>1100 0000</td>
<td>{</td>
<td></td>
<td>EOM (even) EOM (odd)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>193</td>
<td>C1</td>
<td>0001</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>194</td>
<td>C2</td>
<td>0010</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>195</td>
<td>C3</td>
<td>0011</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>196</td>
<td>C4</td>
<td>0100</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>197</td>
<td>C5</td>
<td>0101</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>198</td>
<td>C6</td>
<td>0110</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>199</td>
<td>C7</td>
<td>0111</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>C8</td>
<td>1000</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>201</td>
<td>C9</td>
<td>1001</td>
<td>I</td>
<td></td>
<td>X-OFF</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>CA</td>
<td>1010</td>
<td>S (even parity)</td>
<td>N (odd parity)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>203</td>
<td>CB</td>
<td>1011</td>
<td>S (odd parity)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>CC</td>
<td>1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>205</td>
<td>CD</td>
<td>1101</td>
<td>3 (even parity)</td>
<td>3 (odd parity)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>206</td>
<td>CE</td>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>207</td>
<td>CF</td>
<td>1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>208</td>
<td>D0</td>
<td>1101 0000</td>
<td>}</td>
<td></td>
<td>vertical tab</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>209</td>
<td>D1</td>
<td>0001</td>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td>D2</td>
<td>0010</td>
<td>K</td>
<td></td>
<td>K (even parity)</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>211</td>
<td>D3</td>
<td>0011</td>
<td>L</td>
<td></td>
<td>K (odd parity)</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>212</td>
<td>D4</td>
<td>0100</td>
<td>M</td>
<td></td>
<td>+ (even parity)</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>213</td>
<td>D5</td>
<td>0101</td>
<td>N</td>
<td></td>
<td>+ (odd parity)</td>
<td>U</td>
<td></td>
</tr>
<tr>
<td>214</td>
<td>D6</td>
<td>0110</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>215</td>
<td>D7</td>
<td>0111</td>
<td>P</td>
<td></td>
<td>!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>D8</td>
<td>1000</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>D9</td>
<td>1001</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>DA</td>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>219</td>
<td>DB</td>
<td>1011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>DC</td>
<td>1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>221</td>
<td>DD</td>
<td>1101</td>
<td></td>
<td></td>
<td>backspace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>222</td>
<td>DE</td>
<td>1110</td>
<td></td>
<td></td>
<td>idle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>DF</td>
<td>1111</td>
<td></td>
<td></td>
<td>PAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>224</td>
<td>E0</td>
<td>1110 0000</td>
<td>\</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>225</td>
<td>E1</td>
<td>0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>226</td>
<td>E2</td>
<td>0010</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>227</td>
<td>E3</td>
<td>0011</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>228</td>
<td>E4</td>
<td>0100</td>
<td>U</td>
<td></td>
<td>, (even parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>229</td>
<td>E5</td>
<td>0101</td>
<td>V</td>
<td></td>
<td>, (odd parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td>E6</td>
<td>0110</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>231</td>
<td>E7</td>
<td>0111</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>232</td>
<td>E8</td>
<td>1000</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>233</td>
<td>E9</td>
<td>1001</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>EA</td>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>235</td>
<td>EB</td>
<td>1011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>236</td>
<td>EC</td>
<td>1100</td>
<td>R</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>ED</td>
<td>1101</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>EE</td>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>EF</td>
<td>1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>Hex</td>
<td>Binary</td>
<td>EBCDIC</td>
<td>ASCII</td>
<td>Eight bit data interchange</td>
<td>PTTC/EBCD</td>
<td>PTTC/Correspondence</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
<td>------------</td>
<td>--------</td>
<td>-------</td>
<td>-----------------------------</td>
<td>-----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>240</td>
<td>F0</td>
<td>1111 0000</td>
<td>0</td>
<td></td>
<td>shift in (even)</td>
<td>H</td>
<td>?</td>
</tr>
<tr>
<td>241</td>
<td>F1</td>
<td>0001</td>
<td>1</td>
<td></td>
<td>shift in (odd)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>F2</td>
<td>0010</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>F3</td>
<td>0011</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>244</td>
<td>F4</td>
<td>0100</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>F5</td>
<td>0101</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>F6</td>
<td>0110</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>F7</td>
<td>0111</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>F8</td>
<td>1000</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>F9</td>
<td>1001</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>FA</td>
<td>1010</td>
<td>LVM</td>
<td></td>
<td>&lt;= (even parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>FB</td>
<td>1011</td>
<td></td>
<td></td>
<td>&lt;= (odd parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>FC</td>
<td>1100</td>
<td></td>
<td></td>
<td>? (even parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>FD</td>
<td>1101</td>
<td></td>
<td></td>
<td>? (odd parity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>254</td>
<td>FE</td>
<td>1110</td>
<td></td>
<td></td>
<td>delete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>FF</td>
<td>1111</td>
<td></td>
<td></td>
<td>rub out</td>
<td></td>
<td>delete</td>
</tr>
</tbody>
</table>

Character Codes E-5
Appendix F. Carry and Overflow Indicators

This appendix explains the meaning of the carry and overflow indicators for signed and unsigned numbers. Examples for setting these indicators are also provided.

Signed Numbers

For signed addition and subtraction, the overflow indicator signals a result that exceeds the representation capability of the system for the result operand size. When overflow is indicated, the carry indicator and the resulting operand together form a valid result with the carry indicator being the most significant bit. For addition, the carry indicator is the sign (high-order bit) of this result. For subtraction, the carry indicator is the complement of the sign (high-order bit) of the result. A negative result appears in two's complement form. When no overflow is indicated, the carry indicator provides no information about the result.

Figure F-1 shows how the carry and overflow indicators are set for an add operation when using 16-bit operands. Figure F-2 provides the same information for a subtract operation.

### Signed Numbers

#### ADD OPERATION—All possible results (16-bit example)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000</td>
<td>-65536</td>
</tr>
<tr>
<td></td>
<td>7FFF</td>
<td>-32769</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>-32768</td>
</tr>
<tr>
<td></td>
<td>FFFF</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7FFF</td>
<td>+32767</td>
</tr>
<tr>
<td></td>
<td>8000</td>
<td>+32768</td>
</tr>
<tr>
<td></td>
<td>FFFF</td>
<td>+65534</td>
</tr>
</tbody>
</table>

(See Note 1)  
16-bit representable range

**Notes.**

1. When overflow occurs, the carry indicator and the result together form a valid 17-bit signed number, of which the carry is the sign, and the result is the magnitude. A negative result is in two’s complement form. When no overflow occurs, no useful information is provided by the carry indicator.

2. The carry indicator may be on or off depending on the operands.

Figure F-1. All possible results of an add operation regarding the operands as signed 16-bit numbers
**SIGNED NUMBERS**

**SUBTRACT OPERATION—All possible results (16-bit example)**

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Hexadecimal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflow</td>
<td>Carry</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>-65535</td>
</tr>
</tbody>
</table>

(See Note 1)

1. When overflow occurs, the carry indicator and the result form a valid 17-bit signed number, of which the carry is the complement of the correct sign, and the result is the magnitude. A negative result is in two's complement form. When no overflow occurs, no useful information is provided by the carry indicator.

2. The carry indicator may be on or off depending on the operands.

---

**Unsigned Numbers**

For unsigned addition and subtraction, the carry indicator signals that:

1. On an add instruction, a carry out of the high-order bit position has occurred (result exceeds result operand size). The carry indicator and the resulting operand together form a valid result of which the carry indicator is the most significant bit.

2. On a subtract operation, a borrow beyond the high-order bit position has occurred. A borrow during a subtract operation is defined as either of the following:
   - No carry is generated out of the high-order bit position when a two's complement of the subtrahend and add is performed to accomplish the subtract operation.
   - The most significant digit of the minuend must be made larger to generate a difference of zero or one when subtracting the most significant digit of the subtrahend; for example, 1 subtracted from 0.

When a borrow is signalled on a subtract operation, the result is in two's complement form.

The overflow indicator provides no useful information about unsigned operations.

Figure F-3 shows how the carry and overflow indicators are set for an add operation when using 16-bit operands. Figure F-4 provides the same information for a subtract operation.

---

Figure F-2. All possible results of a subtract operation regarding the operands as signed 16-bit numbers.
### UNSIGNED NUMBERS

#### ADD OPERATION—All possible results (16-bit example)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Result value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflow</td>
<td>Carry</td>
</tr>
<tr>
<td>(Note 2)</td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Notes

1. With the carry indicator on, the result and carry form a valid 17-bit unsigned number of which the carry is the most significant bit.
2. The overflow indicator may be set; however, it provides no useful information.

Figure F-3. All possible results of an add operation regarding the operands as unsigned 16-bit numbers

#### SUBTRACT OPERATION—All possible results (16-bit example)

<table>
<thead>
<tr>
<th>Indicators</th>
<th>Result value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflow</td>
<td>Carry</td>
</tr>
<tr>
<td>(Note 2)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Notes

1. With carry (borrow) on, the result and carry indicator form a valid 17-bit negative number of which the carry is the sign and result is the magnitude in normal two’s complement form.
2. The overflow indicator may be set; however, it provides no useful information.

Figure F-4. All possible results of a subtract operation regarding the operands as unsigned 16-bit numbers
**Carry Indicator Setting**

The carry indicator is used to signal overflow of the result when operands are presented as (unsigned) numbers. (The machine does not regard the numbers as either signed or unsigned, but performs the designated operation (add or subtract) on the values presented. The programmer must interpret the condition of the result for the number representation involved.) The machine detects the carry condition during the operation in two ways:

1. **Add operation** — when a carry out of the high-order bit position of the result operand occurs.
2. **Subtract operation** — when a borrow beyond the high-order bit position of the result operand occurs.

### Add Operation Examples

A four-bit operand size is used in the following examples. Note that the unsigned number range for this operand is 0 to 15. No other unsigned number values may be represented for this size operand.

- **Addition (carry indicator is not set)**
  
  **Desired operation:** $6 + 9 = 15$
  
  **Machine operation:**
  
  Augend $0110$
  
  Addend $1001$
  
  Result $1111$
  
  High-order bit carry $= 0$
  
  The result fits as an unsigned number. The carry indicator is not set ($C=0$).

- **Addition (carry indicator is set)**
  
  **Desired operation:** $15 + 1 = 16$
  
  **Machine operation:**
  
  Augend $1111$
  
  Addend $0001$
  
  Result $1110$
  
  High-order bit carry $= 1$
  
  The result does not fit as an unsigned number. The carry indicator is set ($C=1$).

- **Addition (carry indicator is set)**
  
  **Desired operation:** $15 + 15 = 30$
  
  **Machine operation:**
  
  Augend $1111$
  
  Addend $1111$
  
  Result $1110$
  
  High-order bit carry $= 1$
  
  Result does not fit as an unsigned number. The carry indicator is set ($C=1$).

*Note.* The result of adding the two largest numbers can be contained in the operand size and the carry indicator. The carry indicator represents the most significant bit.

**Subtract Operation Examples**

The processor performs subtraction by using the complement addition method. The second operand is complemented (two's complement) then an add operation is performed. This is actually a three-way add operation between the minuend, the subtrahend (one's complement), and a constant of one. To provide the correct carry (borrow) indication for the subtraction, the carry result of the complement add operation must be inverted to determine the carry indicator setting. The following examples use a four-bit operand with an unsigned number range of 0 to 15.

- **Subtract (carry indicator is not set)**
  
  **Desired operation:** $15 - 1 = 14$
  
  **Machine operation:**
  
  Minuend $1111$
  
  Subtrahend $1110$ one's complement
  
  Constant $1$ for two's complement
  
  Result $1110$
  
  High-order bit carry $= 1$ invert for carry indicator
  
  The result fits as an unsigned number. The carry indicator is not set ($C=0$).

*Note.* The carry indicator setting ($C=0$) for this subtract operation was determined by inverting the complement-add carry.

- **Subtract (carry indicator is not set)**
  
  **Desired operation:** $15 - 15 = 0$
  
  **Machine operation:**
  
  Minuend $1111$
  
  Subtrahend $0000$ one's complement
  
  Constant $1$ for two's complement
  
  Result $0000$
  
  High-order bit carry $= 1$ invert for carry indicator
  
  The result fits as an unsigned number. The carry indicator is not set ($C=0$).

- **Subtract (carry indicator is set)**
  
  The following two examples show the case of a negative result (subtrahend greater than minuend). This negative result cannot be represented in the operand width because all operand bits are used to represent the unsigned number. To flag this condition the carry indicator is set.
Example 1:
Desired operation: \( 0 - 1 = -1 \)
Machine operation:
- Minuend: 0000
- Subtrahend: 1110 one's complement
- Constant: 1 for two's complement
- Result: 1111

High-order bit carry = 0
invert for carry indicator

The result does not fit as an unsigned number. The carry indicator is set \((C=1)\).

Example 2:
Desired operation: \( 0 - 15 = -15 \)
Machine operation:
- Minuend: 0000
- Subtrahend: 0000 one's complement
- Constant: 1 for two's complement
- Result: 0001

High-order bit carry = 0
invert for carry indicator

The result does not fit as an unsigned number. The carry indicator is set \((C=1)\).

Note. When a negative result occurs on a subtract operation, the values may be useful to the programmer. The carry indicator and the result form a signed number. The carry indicator is the sign and the result is the number in two's complement form (see Figure F-4).

Overflow Indicator Setting
The overflow indicator is used to signal overflow of the result when the operands are presented as signed numbers. The machine does not regard the numbers as either signed or unsigned, but performs the designated operation (add or subtract) on the values presented. The programmer must interpret the condition of the result for the number representation involved. The machine detects this condition by inspection of any carry into and out of the high-order bit (sign position) of the result operand during the operation. The overflow indicator is set \((O = 1)\) for the two cases where the carries disagree:
1. A carry into, but no carry out of the sign position.
2. No carry into, but a carry out of the sign position.

The overflow indicator is not set \((O = 0)\) for the remaining two cases where the carries agree:
1. A carry into and out of the sign position.
2. No carry into and no carry out of the sign position.


Examples

A four-bit operand size is used in the following examples. Note that the signed number range for a four-bit operand is -8 to +7. No other signed number values may be represented.

- **Addition (overflow indicator is not set)**
  - Desired operation: +5 + (+2) = +7
  - Machine operation:
    - Augend: 0101
    - Addend: 0010
    - Result: 0111
  - Carry into sign position = 0
  - Carry out of sign position = 0 carries agree
  - The result fits as a signed number. The overflow indicator is not set (O = 0).

- **Subtraction (overflow indicator is not set)**
  - Desired operation: +7 - (+2) = +5
  - Machine operation:
    - Minuend: 0111
    - Subtrahend: 1101 one's complement
    - Constant: 1 for two's complement
    - Result: 0110
  - Carry into sign position = 1
  - Carry out of sign position = 0 carries agree
  - The result fits as a signed number. The overflow indicator is not set (O = 0).

- **Addition (overflow indicator is set)**
  - Desired operation: +4 + (+4) = +8
  - Machine operation:
    - Augend: 0100
    - Addend: 0100
    - Result: 1000
  - Carry into sign position = 1
  - Carry out of sign position = 0 carries disagree
  - The result does not fit as a signed number. The overflow indicator is set (O = 1).

- **Subtraction (overflow indicator is set)**
  - Desired operation: +7 - (-2) = +9
  - Note: -2 is equal to 1110
  - Machine operation:
    - Minuend: 0111
    - Subtrahend: 0001 one's complement
    - Constant: 1 for two's complement
    - Result: 1011
  - Carry into sign position = 1
  - Carry out of sign position = 0 carries disagree
  - The result does not fit as a signed number. The overflow indicator is set (O = 1).
Appendix G. Reference Information

This appendix contains the following reference information:

- Condition codes
- General registers
- Interrupt status byte
- Level status register (LSR)
- Processor status word (PSW)

Condition Codes

I/O Instruction Condition Codes

These codes are reported during execution of an Operate I/O instruction.

<table>
<thead>
<tr>
<th>Condition code (CC) value</th>
<th>LSR position</th>
<th>Overflow</th>
<th>Reported by</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>channel</td>
<td>Device not attached</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 1</td>
<td>device</td>
<td>Busy</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 1 0 1</td>
<td>device</td>
<td>Busy after reset</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 1 1 1</td>
<td>chan/dev</td>
<td>Command reject</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 1</td>
<td>device</td>
<td>Intervention required</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 1</td>
<td>chan/dev</td>
<td>Interface data check</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 1 0 0</td>
<td>controller</td>
<td>Controller busy</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 1 1 1</td>
<td>chan/dev</td>
<td>Satisfactory</td>
<td></td>
</tr>
</tbody>
</table>

Interrupt Condition Codes

These condition codes are reported by the device or controller during priority interrupt acceptance.

<table>
<thead>
<tr>
<th>Condition code (CC) value</th>
<th>LSR position</th>
<th>Overflow</th>
<th>Reported by</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>controller</td>
<td>Controller end</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 1</td>
<td>device</td>
<td>Program controlled interrupt (PCI)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 1 0 1</td>
<td>device</td>
<td>Exception</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 1 1 1</td>
<td>device</td>
<td>Device end</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 0 0 1</td>
<td>device</td>
<td>Attention</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1 0 1 1</td>
<td>device</td>
<td>Attention and PCI exception</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 1 0 0</td>
<td>device</td>
<td>Attention and device end</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 1 1 1</td>
<td>device</td>
<td>Attention and device end</td>
<td></td>
</tr>
</tbody>
</table>
**General Registers**

<table>
<thead>
<tr>
<th>R or RB* field value</th>
<th>Register selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Register 0</td>
</tr>
<tr>
<td>001</td>
<td>Register 1</td>
</tr>
<tr>
<td>010</td>
<td>Register 2</td>
</tr>
<tr>
<td>011</td>
<td>Register 3</td>
</tr>
<tr>
<td>100</td>
<td>Register 4</td>
</tr>
<tr>
<td>101</td>
<td>Register 5</td>
</tr>
<tr>
<td>110</td>
<td>Register 6</td>
</tr>
<tr>
<td>111</td>
<td>Register 7</td>
</tr>
</tbody>
</table>

*The RB field sometimes contains only the two low-order bits. In this case, registers 4 through 7 cannot be specified.

**Level Status Register (LSR)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Even indicator</td>
</tr>
<tr>
<td>1</td>
<td>Carry indicator</td>
</tr>
<tr>
<td>2</td>
<td>Overflow indicator</td>
</tr>
<tr>
<td>3</td>
<td>Negative result indicator</td>
</tr>
<tr>
<td>4</td>
<td>Zero result indicator</td>
</tr>
<tr>
<td>5</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>6</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>7</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>8</td>
<td>Supervisor state</td>
</tr>
<tr>
<td>9</td>
<td>In process</td>
</tr>
<tr>
<td>10</td>
<td>Trace</td>
</tr>
<tr>
<td>11</td>
<td>Summary mask</td>
</tr>
<tr>
<td>12</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>13</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>14</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>15</td>
<td>(not used, always zero)</td>
</tr>
</tbody>
</table>

**Interrupt Status Byte (ISB)**

**DPC Devices**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Device status available</td>
</tr>
<tr>
<td>1</td>
<td>Delayed command reject</td>
</tr>
<tr>
<td>2</td>
<td>Device dependent</td>
</tr>
<tr>
<td>3</td>
<td>Device dependent</td>
</tr>
<tr>
<td>4</td>
<td>Device dependent</td>
</tr>
<tr>
<td>5</td>
<td>Device dependent</td>
</tr>
<tr>
<td>6</td>
<td>Device dependent</td>
</tr>
<tr>
<td>7</td>
<td>Device dependent</td>
</tr>
</tbody>
</table>

**Cycle Steal Devices**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Device status available</td>
</tr>
<tr>
<td>1</td>
<td>Delayed command reject</td>
</tr>
<tr>
<td>2</td>
<td>Incorrect length record</td>
</tr>
<tr>
<td>3</td>
<td>DCB specification check</td>
</tr>
<tr>
<td>4</td>
<td>Storage data check</td>
</tr>
<tr>
<td>5</td>
<td>Invalid storage address</td>
</tr>
<tr>
<td>6</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>7</td>
<td>Interface data check</td>
</tr>
</tbody>
</table>

**Processor Status Word (PSW)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Specification check</td>
</tr>
<tr>
<td>1</td>
<td>Invalid storage address</td>
</tr>
<tr>
<td>2</td>
<td>Privilege violate</td>
</tr>
<tr>
<td>3</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>4</td>
<td>Invalid function</td>
</tr>
<tr>
<td>5</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>6</td>
<td>Stack exception</td>
</tr>
<tr>
<td>7</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>8</td>
<td>Storage parity check</td>
</tr>
<tr>
<td>9</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>10</td>
<td>CPU control check</td>
</tr>
<tr>
<td>11</td>
<td>I/O check</td>
</tr>
<tr>
<td>12</td>
<td>Sequence indicator</td>
</tr>
<tr>
<td>13</td>
<td>Auto-IPL</td>
</tr>
<tr>
<td>14</td>
<td>(not used, always zero)</td>
</tr>
<tr>
<td>15</td>
<td>Power/thermal warning</td>
</tr>
</tbody>
</table>
Index

add byte (AB) instruction 6-2
add byte immediate (ABI) instruction 6-3
add carry register (ACY) instruction 6-4
add double word (AD) instruction
    register/storage format 6-5
    storage/storage format 6-5
add word (AW) instruction
    register/register 6-6
    register/storage format 6-6
    storage/storage format 6-7
storage to register long format 6-7
add word immediate (AWl) instruction
    register immediate long format 6-9
    storage immediate format 6-9
add word with carry (AWCY) instruction 6-8
address generation, effective 2-13
address mode (AM) 2-14
    assembler syntax, conversion to A-2
AKR key, console 5-10
alternate IPL source, console switch 5-2
ALU (see arithmetic and logic unit)
AM (see address mode)
and word immediate (NWI) instruction 6-57
arithmetic and logic unit (ALU) 2-3
assembler syntax, summary of C-1
attention and device end condition code
attention and exception condition code 4-20
attention and PCI condition code 4-20
attention condition code 4-20
auto IPL, bit in PSW 3-10
auto IPL mode, console switch 5-2
base register (RB)
    used for effective address generation 2-13, 2-14
basic console 5-2
branch and link (BAL) instruction 6-10
branch and link external (BALX) instruction 6-10
branch and link short (BALS) instruction 6-11
branch external (BX) instruction 6-10
branch if mixed (BMIX) instruction 6-12
branch if negative (BN) instruction 6-12
branch if not off (BOFF) instruction 6-14
branch if not on (BON) instruction 6-14
branch if off (BOFF) instruction 6-12
branch if on (BON) instruction 6-12
branch indexed short (BXS) instruction 6-17
branch/jump instructions
    branch and link (BAL) 6-10
    branch and link external (BALX)
    branch and link short (BALS) 6-11
    branch indexed short (BXS) 6-17
    branch on condition (BC) 6-12
    branch if mixed (BMIX)
    branch if negative (BN)
    branch if off (BOFF)
branch on condition (BC) (continued)
    branch if on (BON)
    branch on arithmetically less than (BLT)
    branch on arithmetically less than or equal (BLE)
    branch on carry (BCY)
    branch on equal (BE)
    branch on even (BEV)
    branch on logically less than (BLLT)
    branch on logically less than or equal (BLLE)
    branch on positive (BP)
    branch on zero (BZ)
branch on condition code (BCC) 6-13
    branch on not error (BNER)
branch on not condition (BNC) 6-14
    branch if not off (BNOFF)
    branch if not on (BNON)
    branch on arithmetically greater than (BGT)
    branch on arithmetically greater than or equal (BGE)
    branch on logically greater than (BLGT)
    branch on logically greater than or equal (BLGE)
    branch on no carry (BNCY)
    branch on not equal (BNE)
    branch on not even (BNEV)
    branch on not mixed (BNMIX)
    branch on not negative (BNN)
    branch on not positive (BNP)
    branch on not zero (BNZ)
branch on not condition code (BNC) 6-15
    branch on error (BER)
branch on not overflow (BNOV) 6-16
branch on overflow (BOV) 6-16
branch unconditional (B) 6-16
    branch external (BX)
    jump and link (JAL) 6-38
jump on condition (JC) 6-39
    jump if mixed (JMIX)
    jump if off (JOFF)
    jump if on (JON)
    jump on arithmetically less than (JTL)
    jump on arithmetically less than or equal (JLE)
    jump on carry (JCY)
    jump on equal (JE)
    jump on even (JEV)
    jump on logically less than (JLLT)
    jump on logically less than or equal (JLLE)
    jump on negative (JN)
    jump on positive (JP)
    jump on zero (JZ)
jump on count (JCT) 6-40
jump on not condition (JNC) 6-41
    jump if not off (JNOC)
    jump if not on (JNON)
    jump on arithmetically greater than (JGT)
    jump on arithmetically greater than or equal (JGE)
    jump on logically greater than (JLGT)
branch/jump instructions (continued)

jump on not condition (JNC) (continued)
  jump on logically greater than or equal (JLGE)
  jump on no carry (JNVC)
  jump on not equal (JNE)
  jump on not even (JNEV)
  jump on not mixed (JNMIX)
  jump on not negative (JNN)
  jump on not positive (JNP)
  jump on not zero (JNZ)

jump unconditional (J) 6-38
no operation (NOP) 6-57

branch on arithmetically greater than (BGT)
  instruction 6-14
branch on arithmetically greater than or equal
  (BGE) instruction 6-14
branch on arithmetically less than (BLT)
  instruction 6-12
branch on carry (BCY) instruction 6-12
branch on condition (BC) instruction 6-12
branch on condition code (BCC) instruction 6-13
branch on equal (BE) instruction 6-12
branch on error (BER) instruction 6-15
branch on even (BEV) instruction 6-12
branch on logically greater than (BLGT) instruction 6-14
branch on logically greater than or equal (BLGE)
  instruction 6-14
branch on logically less than (BLLT) instruction 6-12
branch on logically less than or equal (BLE)
  instruction 6-12
branch on no carry (BNCY) instruction 6-14
branch on not condition (BNC) instruction 6-14
branch on not condition code (BNCC) instruction 6-15
branch on not equal (BNE) instruction 6-14
branch on not error (BNER) instruction 6-13
branch on not even (BNEV) instruction 6-14
branch on not mixed (BNMIX) instruction 6-14
branch on not negative (BNN) instruction 6-14
branch on not overflow (BNOV) instruction 6-16
branch on not positive (BNP) instruction 6-14
branch on not zero (BNZ) instruction 6-14
branch on overflow (BOV) instruction 6-16
branch on positive (BP) instruction 6-12
branch on zero (BZ) instruction 6-12
branch unconditional (B) instruction 6-10
burst mode 4-13
busy, condition code 4-19
busy after reset, condition code 4-19

carry indicator
  defined 2-8
  how used 2-8
  setting F-4, 2-10
  example, add operation F-1, F-3
  example, subtract operation F-2, F-4
chaining 4-5, 4-13
chaining flag bit in DCB 4-5, 4-13
character codes E-1
check indicator 5-4
check restart key, console 5-7

CIAR (see current instruction address register)
CIAR key, console 5-9
class interrupts 3-5
classing interrupts 3-8
execution of 3-5
machine check 3-6
power/thermal warning 3-7
priority of 3-5
program check 3-6
soft exception trap 3-9
summary of 3-8
supervisor call 3-7
trace 3-7
codes, character E-1
command field, IDC 4-3
command reject, condition code 4-19
commands, I/O, general
communications features, description 1-6
compare byte (CB) instruction
  register/storage format 6-18
  storage/storage format 6-18
compare byte field equal and decrement (CFED)
  instruction 6-21
compare byte field equal and increment (CFEN)
  instruction 6-21
compare byte field not equal and decrement
  (CFNED) instruction 6-22
compare byte field not equal and increment
  (CFNEN) instruction 6-22
compare byte immediate (CBI) instruction 6-19
compare double word (CD) instruction
  register/storage format 6-20
  storage/storage format 6-20
compare operation
  example 2-8
  indicator settings 2-8
  testing results 2-10
compare word (CW) instruction
  register/register format 6-28
  register/storage format 6-28
  storage/storage format 6-28
compare word immediate (CWI) instruction
  register immediate long format 6-29
  storage immediate format 6-29
compatibility 6-2
  no operation 6-2
  program check 6-2
  soft exception trap 6-2
complement register (CMR) instruction 6-23
condition codes, defined
  interrupt 4-19
  IO instruction 4-19
console 5-1
  basic 5-2
    indicators 5-2
    keys and switches 5-2
programmer 5-3
  combination keys/indicators 5-5
  display 5-3
  displaying main storage 5-12
  displaying registers 5-13
  indicators 5-4
console (continued)
programmer (continued)
keys and switches 5-8
storing into main storage 5-12
storing into registers 5-13
console data buffer 2-6, 5-3
console interrupt 3-8
console interrupt key 5-8
control command 4-7
controller busy condition code 4-19
controller end condition code 4-19
conversion tables, numbering systems and D-1
copy console data buffer (CPCON) instruction 6-24
copy current level (CPCL) instruction 6-23
copy in-process flags (CPIPF) instruction 6-25
copy interrupt mask register (CPIMR) instruction 6-24
copy level block (CPBLB) instruction 6-26
copy level status register (CPLSR) instruction 6-27
copy processor status and reset (CPPSR) instruction 6-27
count residual byte 4-12
restrictions for the start cycle steal status operation 4-12
word in DCB 4-6
CPU control check, bit in PSW 3-10
current-instruction address register (CIAR) 2-6
cycle steal
description 4-10
device options 4-13
interrupt status byte (ISB) 4-20
start cycle status operation 4-12
start operation 4-10
status words 4-13
termination conditions 4-15
cycle steal, typical operation 4-10
data buffer key, console 5-8
data display indicators 5-4
data entry keys (0-F), console 5-11
data stacking 2-24
example, allocating fixed storage areas 2-26
pop operation 2-25
push operation 2-25
DCB (see device control block) DCB specification check status bit 4-21
delayed command reject status bit 4-20
device address field, IDCB 4-3
device control block (DCB) 4-5
count word 4-5
data address word 4-6
device parameter word 3 4-5
device parameter word 4 4-6
device parameter word 5 4-6
device parameter words 1-2 4-5
for start cycle steal status command, summary of 4-12
specification check status bit 4-21
device cycle-steal-status word 1 4-13
device dependent status available status bit 4-20
device dependent status words 4-13
device end condition code 4-19
device ID word 4-6
device mask (I-bit) 3-13, 4-7
device not attached, condition code 4-19
device options, cycle steal 4-13
burst mode 4-13
chaining 4-13
program control interrupt (PCI) 4-13
suppress exception (SE) 4-13
device reset command 4-8
diagnose (DIAG) instruction 6-32
diagnostic mode, console switch 5-2
direct program control (DPC) operation 4-9
disable (DIS) instruction 6-33
displaying main storage 5-12
displaying registers 5-13
divide byte (DB) instruction 6-30
divide double word (DD) instruction 6-31
divide word (DW) instruction 6-34
DPC (direct program control) operation 4-9
EA (see effective address) effective address 2-13
effective address generation 2-13
base register storage address 2-18
base register word displacement 2-14
base register word displacement short 2-13
five-bit address argument 2-18
four-bit address argument 2-14
address mode (AM) 2-14
able (EN) instruction 6-35
end of chain bit 4-14
EOC (see end of chain bit) error conditions recovery from 3-8
that cause class interrupts 3-5
even indicator 2-8
exception condition code 4-19
exception conditions, during instruction execution program check basic instructions 6-1
soft exception trap basic instructions 6-1
exceptions, suppression of 4-14
exclusive OR byte (XB) instruction 6-97
exclusive OR double word (XD) instruction 6-98
exclusive OR word (XW) instruction register/register format 6-99
register/storage format 6-99
storage to register long format 6-100
exclusive OR word immediate (XWI) instruction 6-100
fill byte field and decrement (FFD) instruction 6-36
fill byte field and increment (FFN) instruction 6-36
general registers G-2, 2-6
halt I/O command 4-8
high limit address (HLA) 2-24
HLA (see high limit address)
I-bit, device mask 3-13
I-bit (device mask), field in IDC B 4-7
I/O check, bit in PSW 3-10
I/O commands, general
  control 4-7
  device reset 4-8
  halt I/O 4-8
  prepare 4-7
  read 4-6
  read ID 4-6
  read status 4-7
  start 4-8
  start cycle steal status 4-8
  summary chart 4-4
  write 4-7
I/O condition codes and status information, general 4-15
I/O status information 4-20
  interrupt status byte (IS B) 4-20
  interrupt condition codes 4-19
  interrupt information byte (IIB) 3-3
  interrupt status byte (IS B) 4-20
  IO instruction condition codes 4-19
  summary of 4-19
I/O interrupts 3-3
  prepare I/O device for 3-3
  present and accept 3-3
IAR (see instruction address register)
IAR key, console 5-10
IBM 4953 processor 1-3
ID word
  device 4-6
  interrupt 3-3, 4-20
IDCB (immediate device control block) 4-3
IIB (see interrupt information byte)
ILR (see incorrect length record)
immediate data field, IDC B 4-3
immediate device control block (IDCB) 4-3
in-process bit 2-11
  effect on program controlled level switching 3-10
incorrect length record (ILR)
  status bit 4-21
  suppression of reporting 4-14
indicator bits in LSR 2-7
indicators
  arithmetic 2-7
  basic console 5-2
  programmer console 5-4
  sequence 3-10
indicators, add and subtract operations (carry and overflow) F-1, 2-8
  signed numbers F-1
    add operation F-1
    subtract operation F-2
  unsigned numbers F-2
    add operation F-3
    subtract operation F-3
indicators, compare operations 2-8
indicators, condition code for I/O operations 2-8
indicators, multiple-word operands 2-9
indicators, result (even, negative, and zero) 2-8
indicators, shift operations (carry and overflow) 2-8
indicators, testing with branch and jump instructions 2-10
indirect address 2-16
inhibit trace (IT) bit
  effect on SELB instruction 6-75
  how used, programming note 3-7
initial program load (IPL) 2-22
  auto IPL, bit in PSW 3-12
  auto IPL mode, console switch 5-2
  source switch, console 5-2
input flag in DCB 4-5
input/output (see also I/O)
  commands (see I/O commands)
    condition codes and status information 4-15
  interrupt status byte (IS B) 4-20
  operate I/O (IO) instruction 4-2, 6-37
input/output operations 4-1
instruct step key/indicator 5-6
instruction
  exception conditions 6-1
  formats 2-11
    index of X-10
    names 2-13
    one word 2-11
    summary of B-1
    two word 2-12
    variable length 2-12, 2-19
    index of, by name X-13
    privileged 2-21
    termination or suppression 6-2
  instruction address register (IAR) 2-6
  instruction execution times A-1
    additional time for addressing mode A-2
    instruction formats B-1, 2-11
    interchange registers (IR) instruction 6-37
    interface data check, condition code 4-19
    interface data check status bit 4-21
  interrupt
    automatic branching 3-2
    class 3-5
    I/O 3-3
    masking facilities 3-13
      device mask (I-bit) 3-13
      mask register, interrupt level 3-13
      summary mask 3-13
    priority scheme 3-2
    interrupt ID word 3-3, 4-20
    interrupt information byte (IIB) 3-3
    interrupt level mask register 3-13
    interrupt scheme 3-1
    interrupt status byte (IS B) 4-20
      defined 4-20
      for cycle stealing devices 4-20
      for devices that do not cycle steal 4-20
    interrupts and level switching, introduction 3-1
    intervention required, condition code 4-19
    invalid function, bit in PSW 3-9
    invalid function, program check condition 6-1
    invalid function, soft exception trap condition 6-1
    invalid storage address 6-1
    invalid storage address, bit in PSW 3-9
    invalid storage address status bit 4-21
    invert register (VR) instruction 6-97
    IO (operate I/O) instruction 4-2, 6-37
    IPL (see initial program load)
IPL source switch 5-2
ISB (see interrupt status byte)
IT bit (see inhibit trace bit)

jump and link (JAL) instruction 6-38
jump if mixed (JMIX) instruction 6-39
jump if not off (JNOFF) instruction 6-41
jump if not on (JNON) instruction 6-41
jump if off (JOFF) instruction 6-39
jump if on (JON) instruction 6-39
jump on arithmetically greater than (JGT) instruction 6-41
jump on arithmetically greater than or equal (JGE) instruction 6-41
jump on arithmetically less than (JLT) instruction 6-39
jump on arithmetically less than or equal (JLE) instruction 6-39
jump on carry (JCY) instruction 6-39
jump on condition (JC) instruction 6-39
jump on count (JCT) instruction 6-40
jump on equal (JE) instruction 6-39
jump on even (JEV) instruction 6-39
jump on greater than (JGT) instruction 6-41
jump on logically greater than (JLGT) instruction 6-41
jump on logically greater than or equal (JLGE) instruction 6-41
jump on logically less than (JLLT) instruction 6-39
jump on logically less than or equal (JLLE) instruction 6-39
jump on negative (JN) instruction 6-39
jump on no carry (JNCY) instruction 6-41
jump on not condition (JNC) instruction 6-41
jump on not equal (JNE) instruction 6-41
jump on not even (JNEV) instruction 6-41
jump on not mixed (JNMIX) instruction 6-41
jump on not negative (JNN) instruction 6-41
jump on not positive (JNP) instruction 6-41
jump on not zero (JNZ) instruction 6-41
jump on zero (JZ) instruction 6-39
jump unconditional (J) instruction 6-38

level exit (LEX) instruction 6-41
level status block (LSB) 2-6
level status register (LSR) 2-6, 6-2
level switching
  priority interrupt 3-3
  program controlled 3-10
level 0 key/indicator 5-5
level 1 key/indicator 5-5
level 2 key/indicator 5-5
level 3 key/indicator 5-5
linkage stacking
  description 2-29
  example, reenterable subroutine 2-30
LLA (see low limit address)
load indicator 5-2
load key 5-2
load multiple and branch (LMP) instruction 6-42
load state 2-21
low limit address (LLA) 2-24
LSB (see level status block)
LSB pointer, class interrupts 3-5
LSR (see level status register)
LSR key, console 5-10

machine check conditions 3-6
machine check interrupt 3-6
main storage 2-1
  address boundaries, instruction operand 2-3
  addressing 2-3
main storage key, console 5-9
mask register 2-6, 3-13
mode switch 5-2
move address (MVA) instruction
  storage address to register format 6-45
  storage immediate format 6-45
move byte (MVB) instruction
  register/storage format 6-46
  storage/storage format 6-46
move byte and zero (MVBZ) instruction 6-47
move byte field and decrement (MVFDF) instruction 6-50
move byte field and increment (MVFN) instruction 6-50
move byte immediate (MVB) instruction 6-47
move double word (MVD) instruction
  register/storage format 6-48
  storage/storage format 6-48
move double word and zero (MVDZ) instruction 6-49
move word (MVW) instruction
  register/register format 6-51
  storage/storage format 6-51
move word long (MWL) instruction 6-53
move word register (MVR) instruction 6-53
move word short (MVS) instruction 6-54
multiple register/storage instructions
  load multiple and branch (LMB) 6-42
  store multiple (STM) 6-88
multiply byte (MB) instruction 6-43
multiply double word (MD) instruction 6-44
multiply word (MW) instruction 6-56

NE bit (see no exception bit)
negative indicator 2-8
no exception bit 4-14
no operation (NOP) instruction 6-57
normal mode, console switch 5-2
numbering representation 2-3
signed numbers 2-3
unsigned numbers 2-3
numbering systems and conversion tables D-1
op reg key, console 5-9  
operate I/O (IO) instruction 4-2, 6-37  
options, cycle steal device 4-13  
OR byte (OB) instruction  
register/storage format 6-58  
storage/storage format 6-58  
OR double word (OD) instruction  
register/storage format 6-59  
storage/storage format 6-59  
OR word (OW) instruction  
register/register format 6-60  
storage/storage format 6-60  
storage to register long format 6-61  
OR word immediate (OWI) instruction  
register immediate format 6-62  
storage/storage format 6-62  
overflow indicator  
how used 2-8  
setting F-1  

parametric instructions  
diagnose (DIAG)  6-32  
disable (DIS) 6-33  
enable (EN) 6-35  
level exit (LEX) 6-41  
stop (STOP) 6-89  

supervisor call (SVC) 6-89  

PCI (see program controlled interrupt)  
pop byte (PB) instruction 6-63  
pop doubleword (PD) instruction 6-63  
pop operation 2-25  
pop word (PW) instruction 6-65  
power on indicator 5-2  
power on/off switch 5-2  
power on reset, effects of 2-22  
power/thermal warning, bit in PSW 3-10  
power/thermal warning condition 3-7  
power/thermal warning interrupt 3-7  
prepare command 4-7  
primary IPL source, console switch 5-2  
privilege violate 6-1  
privilege violate, bit in PSW 3-9  
privileged instructions, list of 2-21  
problem state 2-21  
processing unit description 2-1  

processor  
data flow 2-2  
description 1-3  
features  
communications 1-6  
input/output units 1-6  
optional 1-3  
standard 1-3  
introduction 1-1  
models 1-4, 1-5  
options  

miscellaneous 1-6  
packaging and power 1-6  
sensor input/output 1-6  
processor, 4953 1-3  
processor state control 2-20  

processor status word (PSW) 2-6, 3-9  
program check 3-6  
program check conditions  

basic instructions 6-1  
program check conditions in PSW 3-9  
program check interrupt 3-6  
program controlled interrupt condition code 4-19  
program controlled level switching 3-10  
program execution 2-11  
jumping and branching 2-23  
level switching and interrupt 2-23  
sequential instructions 2-23  
programmed controlled interrupt (PCI) 4-5, 4-13  
programmer console 5-5  
PSW (see processor status word)  
PSW key, console 5-9  
push byte (PSB) instruction 6-64  
push double word (PSD) instruction 6-64  
push operation 2-25  
push word (PSW) instruction 6-65  

RB (see base register)  
read command 4-6  
read ID command 4-6  
read status command 4-7  
recovery from error conditions 3-8  
reference information G-1  
condition codes G-1  
I/O instruction G-1  
interrupt G-1  
general registers G-2  
cycle steal devices G-2  
DPC devices G-2  
interrupt status byte (ISB) G-2  
level status register (LSR) G-2  
processor status word (PSW) G-2  
register immediate instructions  
add byte immediate (ABI) 6-3  
add word immediate (AWI) 6-9  
AND word immediate (NWI) 6-57  
compare byte immediate (CBI) 6-19  
compare word immediate (CWI) 6-29  
exclusive OR word immediate (XWI) 6-99  
move byte immediate (MVBI) 6-47  
OR word immediate (OWI) 6-62  
reset bits word immediate (RBWI) 6-70  
subtract word immediate (SWI) 6-93  
test word immediate (TWI) 6-96  

register/register instructions  
add carry register (ACY) 6-4  
add word (AW) 6-6  
add word with carry (AWCY) 6-8  
compare word (CW) 6-28  
complement register (CMR) 6-23  
copy level status register (CPLSR) 6-27  
exclusive OR word (XW) 6-99  
interchange registers (IR) 6-37  
invert register (VR) 6-97  
move word (MVW) 6-51  
OR word (OW) 6-60  
reset bits word (RBTW) 6-68  
set indicators (SEIND) 6-74
register/register instructions (continued)
  subtract carry indicator (SCY) 6-71
  subtract word (SW) 6-90
  subtract word with carry (SWCY) 6-92
register/storage instructions
  add byte (AB) 6-3
  add double word (AD) 6-5
  add word (AW) 6-6
  compare byte (CB) 6-18
  compare double word (CD) 6-20
  compare word (CW) 6-28
  divide byte (DB) 6-30
  divide double word (DD) 6-31
  divide word (DW) 6-34
  exclusive OR byte (XB) 6-97
  exclusive OR double word (XD) 6-98
  exclusive OR word (XW) 6-99
  move address (MVA) 6-45
  move byte (MVB) 6-46
  move byte and zero (MVBZ) 6-47
  move double word (MVD) 6-48
  move double word and zero (MVDZ) 6-49
  move word (MVW) 6-51
  move word and zero (MVWZ) 6-55
  multiply byte (MB) 6-43
  multiply double word (MD) 6-44
  multiply word (MW) 6-56
  OR byte (OB) 6-58
  OR double word (OD) 6-59
  OR word (OW) 6-60
  pop byte (PB) 6-63
  pop double word (PD) 6-63
  pop word (PW) 6-65
  push byte (PSB) 6-64
  push double word (PSD) 6-64
  push word (PSW) 6-65
  reset bits byte (RBTB) 6-66
  reset bits double word (RBTD) 6-67
  reset bits word (RBTW) 6-68
  subtract byte (SB) 6-71
  subtract double word (SD) 6-72
  subtract word (SW) 6-90
register/storage long instructions
  add word (AW) 6-7
  exclusive OR word (XW) 6-100
  move word (MVW) 6-52
  operate I/O (IO) 6-37
  OR word (OW) 6-61
  reset bits word (RBTW) 6-69
  subtract word (SW) 6-91
register/storage short instruction
  move word short (MVWS) 6-54
registers
  console data buffer 2-6
  current-instruction address (CIAR) 2-6
  general 2-6
  instruction address (IAR) 2-6
  level status (LSR) 2-6
  mask 2-6
  processor status word (PSW) 2-6
  storage address (SAR) 2-6
reserved storage locations 3-2
reset 2-22
reset bits byte (RBTB) instruction
  register/storage format 6-66
  storage/storage format 6-66
reset bits double word (RBTD) instruction
  register/storage format 6-67
  storage/storage format 6-67
reset bits word (RBTW) instruction
  register/register format 6-68
  storage/storage format 6-69
storage to register long format 6-69
reset bits word immediate (RBTWI) instruction
  register immediate long format 6-70
storage immediate format 6-70
reset key, console 5-8
residual address 4-12
  after power-on reset 4-12
  updating 4-12
residual byte count 4-12
residual status block
  size of 4-14
  storing 4-14
restrictions
  instruction and operand address boundaries 2-3
  programming, DCB 4-6
  programming, DCB (start cycle steal status) 4-12
  when in problem state 2-21
result indicators (even, negative, and zero) 2-8
run indicator 5-2
run state 2-21
R0 key, console 5-10
R1 key, console 5-10
R2 key, console 5-10
R3 key, console 5-10
R4 key, console 5-10
R5 key, console 5-10
R6 key, console 5-10
R7 key, console 5-10
SAR key, console 5-9
satisfactory, condition code 4-19
scan byte field equal and decrement (SFED) instruction 6-76
scan byte field equal and increment (SFEN) instruction 6-76
scan byte field not equal and decrement (SFNED) instruction 6-77
scan byte field not equal and increment (SFNEN) instruction 6-77
sensor input/output options, description 1-6
sequence indicator, bit in PSW 3-10
set console data lights (SECON) instruction 6-73
set indicators (SEIND) instruction 6-74
set interrupt mask register (SEIMR) instruction 6-73
set level block (SELB) instruction 6-75
shift instructions
  shift left and test (SLT) 6-83
  shift left and test double (SLTD) 6-83
  shift left circular (SLC) 6-78
  shift left circular double (SCLD) 6-79
  shift left logical (SLL) 6-81

Index X-7
shift instructions (continued)
shift left logical double (SLLD) 6-82
shift right arithmetic (SRA) 6-84
shift right arithmetic double (SRAD) 6-85
shift right logical (SRL) 6-86
shift right logical double (SRLD) 6-87
shift left and test (SLT) instruction 6-83
shift left and test double (SLTD) instruction 6-83
shift left circular (SLC) instruction
  count in register format 6-78
  immediate count format 6-78
shift left circular double (SLCD) instruction
  count in register format 6-80
  immediate count format 6-79
shift left logical (SLL) instruction
  count in register format 6-81
  immediate count format 6-81
shift left logical double (SLLD) instruction
  count in register format 6-82
  immediate count format 6-82
shift right arithmetic (SRA) instruction
  count in register format 6-84
  immediate count format 6-84
shift right arithmetic double (SRAD) instruction
  count in register format 6-85
  immediate count format 6-85
shift right logical (SRL) instruction
  count in register format 6-86
  immediate count format 6-86
shift right logical double (SRLD) instruction
  count in register format 6-87
  immediate count format 6-87
SIA (see start instruction address)
signed numbers
  defined 2-3
  examples 2-4
single bit manipulation instructions
  test bit (TBT) 6-94
  test bit and invert (TBTV) 6-95
  test bit and reset (TBTR) 6-94
  test bit and set (TBTS) 6-95
soft exception trap conditions 6-1
  basic instructions 6-1
soft exception trap conditions, in PSW 3-7
soft exception trap interrupt 3-9
specification check 6-1
specification check, bit in PSW 3-9
stack control block, relationship to data stack 2-24
stack exception 6-1
stack exception, bit in PSW 3-10
stack operations 2-23
stacking
  data, description 2-24
  linkage, description 2-29
start command 4-8, 4-8
start cycle steal command 4-8, 4-10
start cycle steal status operation 4-12
  DCB format 4-12
  DCB restrictions 4-12
  residual parameters (status) 4-12
start instruction address (SIA) 3-3
start key, console 5-8
states, processor
  following a reset 2-22
  load 2-21
  problem 2-21
  run 2-21
  stop 2-20
  supervisor 2-21
  wait 2-21
status address, DCB word 4  4-6
status after resets, processor 2-22
status block, residual 4-14
status flags, in PSW 3-9
status flags in residual status block 4-14
status information, I/O 4-20
status words, cycle steal 4-13
stop (STOP) instruction 6-89
stop key/indicator 5-6
stop on address key/indicator 5-6
stop on address mode 5-6
stop on error key, console 5-7
stop state 2-20
storage address register (SAR) 2-6
storage data check status bit 4-21
storage immediate instructions
  add word immediate (AWI) 6-9
  compare word immediate (CWI) 6-29
  move address (MVA) 6-45
  move word immediate (MVWI) 6-53
  OR word immediate (OWI) 6-62
  reset bits word immediate (RBTWI) 6-70
  subtract word immediate (SWI) 6-93
  test word immediate (TWI) 6-96
storage parity, bit in PSW 3-10
storage/storage instructions
  add double word (AD) 6-5
  add word (AW) 6-7
  compare byte (CB) 6-18
  compare byte field equal and decrement (CFED) 6-21
  compare byte field equal and increment (CFEN) 6-21
  compare byte field not equal and decrement (CFNED) 6-22
  compare byte field not equal and increment (CFNEN) 6-22
  compare double word (CD) 6-20
  compare word (CW) 6-28
  move byte (MVB) 6-46
  move byte field and decrement (MVFD) 6-50
  move byte field and increment (MVFN) 6-50
  move double word (MVD) 6-48
  move word (MVW) 6-52
  OR byte (OB) 6-58
  OR byte (OD) 6-59
  OR word (OW) 6-61
  reset bits byte (RBTB) 6-66
  reset bits double word (RBTD) 6-67
  reset bits word (RBTW) 6-69
  subtract double word (SD) 6-72
  subtract word (SW) 6-91
store key, console 5-8
store multiple (STM) instruction 6-88
storing into main storage 5-12
storing into registers 5-13
subtract byte (SB) instruction 6-71
subtract carry indicator (SCY) instruction 6-71
subtract double word (SD) instruction
  register/storage format 6-72
  storage/storage format 6-72
subtract word (SW) instruction
  register/register format 6-90
  register/storage format 6-90
  storage/storage format 6-91
  storage to register long format 6-91
subtract word immediate (SWI) instruction
  register immediate long format 6-93
  storage immediate format 6-93
subtract word with carry (SWCY) instruction 6-92
summary mask 3-13
summary of assembler syntax C-1
summary of character codes E-1
summary of instructions
  by format X-10
  by name X-13
supervisor call (SVC) instruction 6-89
supervisor call interrupt 3-7
supervisor state 2-21
supervisor state bit 2-11
suppress exception (SE) 4-5, 4-13
suppression of instructions 6-2
SXV chaining 4-13
syntax, assembler (summary of) C-1
system register/register instructions
  copy console data buffer (CPCON) 6-24
  copy current level (CPCL) 6-23
  set console data lights (SECON) 6-73
system register/storage instructions
  copy in-process flags (CPIPF) 6-25
  copy interrupt mask register (CPIMR) 6-24
  copy level block (CPLB) 6-26
  copy processor status and reset (CPPSR) 6-27
  set interrupt mask register (SEIMR) 6-73
  set level block (SELB) 6-75

TEA (see top element address)
termination of instructions 6-2
test bit (TBT) instruction 6-94
test bit and invert (TBTV) instruction 6-95
test bit and reset (TBTR) instruction 6-94
test bit and set (TBTS) instruction 6-95
test word immediate (TWI) instruction
  register immediate long format 6-96
  storage immediate format 6-96
testing indicators with conditional instructions 2-10
top element address (TEA) 2-24
trace bit 2-11
trace interrupt 3-7

unsigned numbers
  defined 2-3
  examples 2-4

variable field length byte instructions
  compare byte field equal and decrement (CFED) 6-21
  compare byte field equal and increment (CFEN) 6-21
  compare byte field not equal and decrement (CFNED) 6-22
  compare byte field not equal and increment (CFNEN) 6-22
  fill byte field and decrement (FFD) 6-36
  fill byte field and increment (FFN) 6-36
  move byte field and decrement (MVFD) 6-50
  move byte field and increment (MVFN) 6-50
  scan byte field equal and decrement (SFED) 6-76
  scan byte field equal and increment (SFEN) 6-76
  scan byte field not equal and decrement (SFNED) 6-77
  scan byte field not equal and increment (SFNEN) 6-77

wait indicator 5-2
wait state 2-21
WD (see word displacement) 2-13, 2-14
write command 4-7

zero indicator 2-8

Index X-9
Index of Instructions by Format

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch/jump instructions</td>
<td></td>
</tr>
<tr>
<td>branch and link (BAL)</td>
<td>6-10</td>
</tr>
<tr>
<td>branch and link external (BALX)</td>
<td></td>
</tr>
<tr>
<td>branch and link short (BALS)</td>
<td>6-11</td>
</tr>
<tr>
<td>branch indexed short (BXS)</td>
<td>6-17</td>
</tr>
<tr>
<td>branch on condition (BC)</td>
<td>6-12</td>
</tr>
<tr>
<td>branch if mixed (BMIX)</td>
<td></td>
</tr>
<tr>
<td>branch if negative (BN)</td>
<td></td>
</tr>
<tr>
<td>branch if off (BOFF)</td>
<td></td>
</tr>
<tr>
<td>branch if on (BON)</td>
<td></td>
</tr>
<tr>
<td>branch on arithmetically less than (BLT)</td>
<td></td>
</tr>
<tr>
<td>branch on arithmetically less than or equal (BLE)</td>
<td></td>
</tr>
<tr>
<td>branch on carry (BCY)</td>
<td></td>
</tr>
<tr>
<td>branch on equal (BE)</td>
<td></td>
</tr>
<tr>
<td>branch on even (BEV)</td>
<td></td>
</tr>
<tr>
<td>branch on logically less than (BLLT)</td>
<td></td>
</tr>
<tr>
<td>branch on logically less than or equal (BLLE)</td>
<td></td>
</tr>
<tr>
<td>branch on positive (BP)</td>
<td></td>
</tr>
<tr>
<td>branch on zero (BZ)</td>
<td></td>
</tr>
<tr>
<td>branch on condition code (BCC)</td>
<td>6-13</td>
</tr>
<tr>
<td>branch on not condition (BNC)</td>
<td>6-14</td>
</tr>
<tr>
<td>branch if not off (BNOFF)</td>
<td></td>
</tr>
<tr>
<td>branch if not on (BNON)</td>
<td></td>
</tr>
<tr>
<td>branch on arithmetically greater than (BGT)</td>
<td></td>
</tr>
<tr>
<td>branch on arithmetically greater than or equal (BGE)</td>
<td></td>
</tr>
<tr>
<td>branch on logically greater than (BLGT)</td>
<td></td>
</tr>
<tr>
<td>branch on logically greater than or equal (BLGE)</td>
<td></td>
</tr>
<tr>
<td>branch on no carry (BNCY)</td>
<td></td>
</tr>
<tr>
<td>branch on not equal (BNE)</td>
<td></td>
</tr>
<tr>
<td>branch on not even (BNEV)</td>
<td></td>
</tr>
<tr>
<td>branch on not mixed (BNMIX)</td>
<td></td>
</tr>
<tr>
<td>branch on not negative (BNN)</td>
<td></td>
</tr>
<tr>
<td>branch on not positive (BNP)</td>
<td></td>
</tr>
<tr>
<td>branch on not zero (BNZ)</td>
<td></td>
</tr>
<tr>
<td>branch on not condition code (BNCC)</td>
<td>6-15</td>
</tr>
<tr>
<td>branch on error (BER)</td>
<td></td>
</tr>
<tr>
<td>branch on not overflow (BNOV)</td>
<td>6-16</td>
</tr>
<tr>
<td>branch on overflow (BOV)</td>
<td>6-16</td>
</tr>
<tr>
<td>branch unconditional (B)</td>
<td>6-10</td>
</tr>
<tr>
<td>jump and link (JAL)</td>
<td>6-38</td>
</tr>
<tr>
<td>jump on condition (JC)</td>
<td>6-39</td>
</tr>
<tr>
<td>jump if mixed (JMX)</td>
<td></td>
</tr>
<tr>
<td>jump if off (JOFF)</td>
<td></td>
</tr>
<tr>
<td>jump if on (JON)</td>
<td></td>
</tr>
<tr>
<td>jump on arithmetically less than (JLT)</td>
<td></td>
</tr>
<tr>
<td>jump on arithmetically less than or equal (JLE)</td>
<td></td>
</tr>
<tr>
<td>jump on carry (JCY)</td>
<td></td>
</tr>
<tr>
<td>jump on equal (JE)</td>
<td></td>
</tr>
<tr>
<td>jump on even (JEV)</td>
<td></td>
</tr>
<tr>
<td>jump on logically less than (JLLT)</td>
<td></td>
</tr>
<tr>
<td>jump on logically less than or equal (JLLE)</td>
<td></td>
</tr>
<tr>
<td>jump on negative (JN)</td>
<td></td>
</tr>
<tr>
<td>jump on positive (JP)</td>
<td></td>
</tr>
<tr>
<td>jump on zero (JZ)</td>
<td></td>
</tr>
</tbody>
</table>

branch/jump instructions (continued)

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>jump on count (JCT)</td>
<td>6-40</td>
</tr>
<tr>
<td>jump on not condition (JNC)</td>
<td>6-41</td>
</tr>
<tr>
<td>jump if not off (JNOFF)</td>
<td></td>
</tr>
<tr>
<td>jump if not on (JNON)</td>
<td></td>
</tr>
<tr>
<td>jump on arithmetically greater than (JGT)</td>
<td></td>
</tr>
<tr>
<td>jump on arithmetically greater than or equal (JGE)</td>
<td></td>
</tr>
<tr>
<td>jump on logically greater than (JLGT)</td>
<td></td>
</tr>
<tr>
<td>jump on logically greater than or equal (JLGE)</td>
<td></td>
</tr>
<tr>
<td>jump on no carry (JNCY)</td>
<td></td>
</tr>
<tr>
<td>jump on not equal (JNE)</td>
<td></td>
</tr>
<tr>
<td>jump on not even (JNEV)</td>
<td></td>
</tr>
<tr>
<td>jump on not mixed (JNMIEx)</td>
<td></td>
</tr>
<tr>
<td>jump on not positive (JNP)</td>
<td></td>
</tr>
<tr>
<td>jump on not zero (JNZ)</td>
<td></td>
</tr>
<tr>
<td>jump unconditional (J)</td>
<td>6-38</td>
</tr>
<tr>
<td>no operation (NOP)</td>
<td>6-57</td>
</tr>
</tbody>
</table>

multiple register/storage instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>load multiple and branch (LMB)</td>
<td>6-42</td>
</tr>
<tr>
<td>store multiple (STM)</td>
<td>6-88</td>
</tr>
</tbody>
</table>

parametric instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>diagnose (DIAG)</td>
<td>6-32</td>
</tr>
<tr>
<td>disable (DIS)</td>
<td>6-33</td>
</tr>
<tr>
<td>enable (EN)</td>
<td>6-35</td>
</tr>
<tr>
<td>level exit (LEX)</td>
<td>6-41</td>
</tr>
<tr>
<td>stop (STOP)</td>
<td>6-89</td>
</tr>
<tr>
<td>supervisor call (SVC)</td>
<td>6-89</td>
</tr>
</tbody>
</table>

register immediate instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add byte immediate (ABI)</td>
<td>6-3</td>
</tr>
<tr>
<td>add word immediate (AWI)</td>
<td>6-9</td>
</tr>
<tr>
<td>AND word immediate (AWI)</td>
<td>6-57</td>
</tr>
<tr>
<td>compare byte immediate (CBI)</td>
<td>6-19</td>
</tr>
<tr>
<td>compare word immediate (CWI)</td>
<td>6-29</td>
</tr>
<tr>
<td>exclusive OR word immediate (XW)</td>
<td>6-100</td>
</tr>
<tr>
<td>move byte immediate (MVBI)</td>
<td>6-47</td>
</tr>
<tr>
<td>OR word immediate (OW)</td>
<td>6-62</td>
</tr>
<tr>
<td>reset bits word immediate (RBW)</td>
<td>6-70</td>
</tr>
<tr>
<td>subtract word immediate (SWI)</td>
<td>6-93</td>
</tr>
<tr>
<td>test word immediate (TWI)</td>
<td>6-96</td>
</tr>
</tbody>
</table>

register/register instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add carry register (ACY)</td>
<td>6-4</td>
</tr>
<tr>
<td>add word (AW)</td>
<td>6-6</td>
</tr>
<tr>
<td>add word with carry (AWCY)</td>
<td>6-8</td>
</tr>
<tr>
<td>compare word (CW)</td>
<td>6-28</td>
</tr>
<tr>
<td>complement register (CMR)</td>
<td>6-23</td>
</tr>
<tr>
<td>copy level status register (CPLSR)</td>
<td>6-27</td>
</tr>
<tr>
<td>exclusive OR word (XW)</td>
<td>6-99</td>
</tr>
<tr>
<td>interchange registers (IR)</td>
<td>6-37</td>
</tr>
<tr>
<td>invert register (VR)</td>
<td>6-97</td>
</tr>
<tr>
<td>move word (MVW)</td>
<td>6-51</td>
</tr>
</tbody>
</table>
register/register instructions (continued)

- OR word (OW) 6-60
- reset bits word (RBTW) 6-68
- set indicators (SEIND) 6-74
- subtract carry indicator (SCY) 6-71
- subtract word 6-90
- subtract word (SW)
- subtract word with carry (SWCY) 6-92

register/storage instructions

- add byte (AB) 6-3
- add double word (AD) 6-5
- add word (AW) 6-6
- compare byte (CB) 6-18
- compare double word (CD) 6-20
- compare word (CW) 6-28
- divide byte (DB) 6-30
- divide double word (DD) 6-34
- divide word (DW) 6-38
- exclusive OR byte (XB) 6-97
- exclusive OR double word (XD) 6-98
- exclusive OR word (XW) 6-99
- move address (MVA) 6-45
- move byte (MVB) 6-46
- move byte and zero (MVBZ) 6-47
- move double word (MVD) 6-48
- move double word and zero (MVDZ) 6-49
- move word (MVW) 6-51
- move word and zero (MVWZ) 6-55
- move word immediate (MVWI) 6-53
- multiply byte (MB) 6-43
- multiply double word (MD) 6-44
- multiply word (MW) 6-56
- OR byte (OB) 6-58
- OR double word (OD) 6-59
- OR word (OW) 6-60
- pop byte (PB) 6-63
- pop double word (PD) 6-63
- pop word (PW) 6-65
- push byte (PSB) 6-64
- push double word (PSD) 6-64
- push word (PSW) 6-65
- reset bits byte (RBTB) 6-66
- reset bits double word (RBTBD) 6-67
- reset bits word (RBTW) 6-69
- subtract byte (SB) 6-71
- subtract double word (SD) 6-72
- subtract word (SW) 6-90

register/storage long instructions

- add word (AW) 6-7
- exclusive OR word (XW) 6-100
- move word (MVW) 6-52
- operate I/O (IO) 6-37
- OR word (OW) 6-61
- reset bits word (RBTW) 6-69
- subtract word (SW) 6-91

register/storage short instructions

- move word short (MVWS) 6-54

shift instructions (continued)

- shift left circular double (SLCD) 6-79
- shift left logical (SLL) 6-81
- shift left logical double (SLLD) 6-82
- shift right arithmetic (SRA) 6-84
- shift right arithmetic double (SRAD) 6-85
- shift right logical (SRL) 6-86
- shift right logical double (SRLD) 6-87

single bit manipulation instructions

- test bit (TBT) 6-94
- test bit and invert (TBTV) 6-95
- test bit and reset (TBTR) 6-94
- test bit and set (TBTS) 6-95

storage immediate instructions

- add word immediate (AWI) 6-9
- compare word immediate (CWI) 6-29
- move address (MVA) 6-45
- move word immediate (MVWI) 6-53
- OR word immediate (OWI) 6-62
- reset bits word immediate (RBTWI) 6-70
- subtract word immediate (SWI) 6-93
- test word immediate (TWI) 6-96

storage/storage instructions

- add double word (AD) 6-5
- add word (AW) 6-7
- compare byte (CB) 6-18
- compare byte field equal and decrement (CFED) 6-21
- compare byte field equal and increment (CFEN) 6-21
- compare byte field not equal and decrement (CFNED) 6-22
- compare byte field not equal and increment (CFNEN) 6-22
- compare double word (CD) 6-20
- compare word (CW) 6-28
- move byte (MVB) 6-46
- move byte field and decrement (MVFD) 6-50
- move byte field and increment (MVFN) 6-50
- move double word (MVD) 6-48
- move word (MVW) 6-52
- OR byte (OB) 6-58
- OR double word (OD) 6-59
- OR word (OW) 6-61
- reset bits byte (RBTB) 6-66
- reset bits double word (RBTBD) 6-67
- reset bits word (RBTW) 6-69
- subtract byte (SB) 6-71
- subtract double word (SD) 6-72
- subtract word (SW) 6-91

system register/register instructions

- copy console data buffer (CPCON) 6-24
- copy current level (CPCL) 6-23
- set console data lights (SECON) 6-73

system register/storage instructions

- copy in-process flags (CPIPF) 6-25
- copy interrupt mask register (CPIMR) 6-24
- copy level block (CPLB) 6-26
- copy processor status and reset (CPPSR) 6-27
- set interrupt mask register (SEIMR) 6-73
- set level block (SELB) 6-75
variable field length byte instructions

- compare byte field equal and decrement (CFED) 6-21
- compare byte field equal and increment (CFEN) 6-21
- compare byte field not equal and decrement (CFNED) 6-22
- compare byte field not equal and increment (CFNEN) 6-22
- fill byte field and decrement (FFD) 6-36
- fill byte field and increment (FFN) 6-36
- move byte field and decrement (MVFD) 6-50
- move byte field and increment (MVFN) 6-50
- scan byte field equal and decrement (SFED) 6-76
- scan byte field equal and increment (SFEN) 6-76
- scan byte field not equal and decrement (SFNED) 6-77
- scan byte field not equal and increment (SFNEN) 6-77
Index of Instructions by Name

add byte (AB) instruction 6-3
add byte immediate (ABI) instruction 6-3
add carry register (ACY) instruction 6-4
add double word (AD) instruction
  register/storage format 6-5
  storage/storage format 6-5
add word (AW) instruction
  register/register 6-6
  register/storage format 6-6
  storage/storage format 6-7
  storage to register long format 6-7
add word immediate (AWI) instruction
  register immediate long format 6-9
  storage immediate format 6-9
add word with carry (AWCY) instruction 6-8
and word immediate (A WI) instruction 6-9
branch and link (BAL) instruction 6-10
branch and link external (BALX) instruction 6-10
branch and link short (BALS) instruction 6-11
branch external (BX) instruction 6-10
branch if mixed (B MIX) instruction 6-12
branch if negative (BN) instruction 6-12
branch if not off (BNOFF) instruction 6-14
branch if not on (BNON) instruction 6-14
branch if off (BOFF) instruction 6-12
branch if on (BON) instruction 6-12
branch indexed short (BXS) instruction 6-17
branch on arithmetically greater than (BGT)
  instruction 6-14
branch on arithmetically greater than or equal
  (BGE) instruction 6-14
branch on arithmetically less than (BLT)
  instruction 6-12
branch on arithmetically less than or equal (BLE)
  instruction 6-12
branch on carry (BCY) instruction 6-12
branch on condition (BC) instruction 6-12
branch on condition code (BCC) instruction 6-13
branch on equal (BE) instruction 6-12
branch on error (BER) instruction 6-15
branch on even (BEV) instruction 6-12
branch on logically greater than (BLGT) instruction 6-14
branch on logically greater than or equal (BLGE)
  instruction 6-14
branch on logically less than (BLLT) instruction 6-12
branch on logically less than or equal (BLLE)
  instruction 6-12
branch on no carry (BNCY) instruction 6-14
branch on not condition (BNC) instruction 6-14
branch on not condition code (BNCC) instruction 6-15
branch on not equal (BNE) instruction 6-14
branch on not error (BNER) instruction 6-13
branch on not even (BNEV) instruction 6-14
branch on not mixed (BNMIX) instruction 6-14
branch on not negative (BNNN) instruction 6-14
branch on not overflow (BNOV) instruction 6-16
branch on not positive (BNP) instruction 6-14
branch on not zero (BNZ) instruction 6-14
branch on overflow (BOV) instruction 6-16
branch on positive (BP) instruction 6-12
branch on zero (BZ) instruction 6-12
branch unconditional (B) instruction 6-10
branch and link ex ternal (BALX) instruction 6-10
branch on equal (BE) instruction 6-12
branch on error (BER) instruction 6-15
branch on not condition (BNC) instruction 6-14
branch on not condition code (BNCC) instruction 6-15
branch on not equal (BNE) instruction 6-14
branch on not error (BNER) instruction 6-13
branch on not even (BNEV) instruction 6-14
branch on not mixed (BNMIX) instruction 6-14
branch on not negative (BNNN) instruction 6-14
branch on not overflow (BNOV) instruction 6-16
branch on not positive (BNP) instruction 6-14
branch on not zero (BNZ) instruction 6-14
branch on overflow (BOV) instruction 6-16
branch on positive (BP) instruction 6-12
branch on zero (BZ) instruction 6-12
branch unconditional (B) instruction 6-10
compare byte (CB) instruction
  register/storage format 6-18
  storage/storage format 6-18
compare byte field equal and decrement (CFED)
  instruction 6-21
compare byte field equal and increment (CFEN)
  instruction 6-21
compare byte field not equal and decrement
  (CFNED) instruction 6-22
compare byte field not equal and increment
  (CFNEN) instruction 6-22
compare byte immediate (CBI) instruction 6-19
compare double word (CD) instruction
  register/storage format 6-20
  storage/storage format 6-20
compare word (CW) instruction
  register/register format 6-28
  register/storage format 6-28
  storage/storage format 6-28
compare word immediate (CWI) instruction
  register immediate long format 6-28
  register/register format 6-28
  register/storage format 6-28
  storage/storage format 6-28
compare word immediate (CWI) instruction
  register immediate long format 6-29
  storage immediate format 6-29
  storage immediate format 6-29
complement register (CMR) instruction 6-23
copy console data buffer (CPCON) instruction 6-24
copy current level (CPCL) instruction 6-23
copy in-process flags (CPIPF) instruction 6-25
copy interrupt mask register (CPIMR) instruction 6-24
copy level block (CPLB) instruction 6-26
copy level status register (CPLSR) instruction 6-27
copy processor status and reset (CPPSR) instruction 6-27
diagnose (DIAG) instruction 6-32
disable (DIS) instruction 6-33
divide byte (DB) instruction 6-30
divide double word (DD) instruction 6-31
divide word (DW) instruction 6-34
enable (EN) instruction 6-35
exclusive OR byte (XB) instruction 6-97
exclusive OR double word (XD) instruction 6-98
exclusive OR word (XW) instruction
  register/register format 6-99
  register/storage format 6-99
  storage to register long format 6-100
exclusive OR word immediate (XWI) instruction 6-100
fill byte field and decrement (FFD) instruction 6-36
fill byte field and increment (FFN) instruction 6-36

interchange registers (IR) instruction 6-37
invert register (VR) instruction 6-97

jump and link (JAL) instruction 6-38
jump if mixed (JMIX) instruction 6-39
jump if not off (JNOFF) instruction 6-41
jump if off (JOFF) instruction 6-39
jump if on (JON) instruction 6-39
jump on arithmetically greater than or equal (JGE) instruction 6-41
jump on arithmetically greater than (JGT) instruction 6-41
jump on arithmetically less than (JLT) instruction 6-39
jump on arithmetically less than or equal (JLE) instruction 6-39
jump on carry (JCY) instruction 6-39
jump on condition (JC) instruction 6-39
jump on count (JCT) instruction 6-40
jump on equal (JE) instruction 6-39
jump on even (JNEV) instruction 6-39
jump on logically greater than (JLGT) instruction 6-41
jump on logically greater than or equal (JLGE) instruction 6-41
jump on logically less than or equal (JLLT) instruction 6-39
jump on logically less than (JLLE) instruction 6-39
jump on negative (IN) instruction 6-39
jump on not carry (JNCY) instruction 6-41
jump on not condition (JNC) instruction 6-41
jump on not equal (JNE) instruction 6-41
jump on not even (JNEV) instruction 6-41
jump on not mixed (JNMIX) instruction 6-41
jump on not negative (JNN) instruction 6-41
jump on not positive (JNP) instruction 6-41
jump on not zero (JNZ) instruction 6-41
jump on positive (JP) instruction 6-39
jump on zero (JZ) instruction 6-39
jump unconditional (J) instruction 6-38

level exit (LEX) instruction 6-41
load multiple and branch (LMB) instruction 6-42

move address (MVA) instruction
storage address to register format 6-45
storage immediate format 6-45
move byte (MVB) instruction
register/storage format 6-46
storage/storage format 6-46
move byte and zero (MVBZ) instruction 6-47
move byte field and decrement (MVFD) instruction 6-50
move byte field and increment (MVFN) instruction 6-50
move byte immediate (MVB) instruction 6-47
move double word (MVD) instruction
register/storage format 6-48
storage/storage format 6-48
move double word and zero (MVDZ) instruction 6-49
move word (MVW) instruction
register/register format 6-51
register/storage format 6-51
register to storage long format 6-51
storage/storage format 6-52
storage to register long format 6-52
move word and zero (MVWZ) instruction 6-55
move word immediate (MVWI) storage immediate format 6-53
storage to register format 6-53
move word short (MVWS) instruction register to storage format 6-54
storage to register format 6-54
multiply byte (MB) instruction 6-43
multiply double word (MD) instruction 6-44
multiply word (MW) instruction 6-56

no operation (NOP) instruction 6-57

operate I/O (IO) instruction 6-37
OR byte (OB) instruction register/storage format 6-58
storage/storage format 6-58
OR double word (OD) instruction register/storage format 6-59
storage/storage format 6-59
OR word (OW) instruction register/register format 6-60
register/storage format 6-60
storage/storage format 6-61
storage to register long format 6-61
OR word immediate (OWI) instruction register immediate format 6-62
storage immediate format 6-62

pop byte (PB) instruction 6-63
pop doubleword (PD) instruction 6-63
pop word (PW) instruction 6-65
push byte (PSB) instruction 6-64
push double word (PSD) instruction 6-64
push word (PSW) instruction 6-65

reset bits byte (RBBT) instruction register/storage format 6-66
storage/storage format 6-66
reset bits double word (RBTD) instruction register/storage format 6-67
storage/storage format 6-67
reset bits word (RBWT) instruction register/register format 6-68
register/storage format 6-68
storage/storage format 6-69
storage to register long format 6-69
reset bits word immediate (RBTWI) instruction register immediate long format 6-70
storage immediate format 6-70
scan byte field equal and decrement (SFED) instruction 6-76
scan byte field equal and increment (SFEN) instruction 6-76
scan byte field not equal and decrement (SFNED) instruction 6-77
scan byte field not equal and increment (SFNEN) instruction 6-77
set console data lights (SECON) instruction 6-73
set indicators (SEIND) instruction 6-74
set interrupt mask register (SEIMR) instruction 6-73
set level block (SELB) instruction 6-75
shift left and test (SLT) instruction 6-83
shift left and test double (SLTD) instruction 6-83
shift left circular (SLC) instruction
  count in register format 6-78
  immediate count format 6-78
shift left circular double (SLCD) instruction
  count in register format 6-80
  immediate count format 6-79
shift left logical (SLL) instruction
  count in register format 6-81
  immediate count format 6-81
shift left logical double (SLLD) instruction
  count in register format 6-82
  immediate count format 6-82
shift right arithmetic (SRA) instruction
  count in register format 6-84
  immediate count format 6-84
shift right arithmetic double (SRAD) instruction
  count in register format 6-85
  immediate count format 6-85
shift right logical (SRL) instruction
  count in register format 6-86
  immediate count format 6-86
shift right logical double (SRLD) instruction
  count in register format 6-87
  immediate count format 6-87
stop (STOP) instruction 6-89
store multiple (STM) instruction 6-88
subtract byte (SB) instruction 6-71
subtract carry indicator (SCY) instruction 6-71
subtract doubleword (SD) instruction
  register/storage format 6-72
  storage/storage format 6-72
subtract word (SW) instruction
  register/register format 6-90
  register/storage format 6-90
  storage/storage format 6-91
  storage to register long format 6-91
subtract word immediate (SWI) instruction
  register immediate long format 6-93
  storage immediate format 6-93
subtract word with carry (SWCY) instruction 6-92
supervisor call (SVC) instruction 6-89

test bit (TBT) instruction 6-94
test bit and invert (TBTV) instruction 6-95
test bit and reset (TBTR) instruction 6-94
test bit and set (TBTS) instruction 6-95
test word immediate (TWD) instruction
  register immediate long format 6-96
  storage immediate format 6-96
YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

What is your occupation? ____________________________________________________________

Number of latest Technical Newsletter (if any) concerning this publication: _______________________________________

Please indicate your name and address in the space below if you wish a reply.

___________________________________________________________________________

___________________________________________________________________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)
Your comments, please . . .

This manual is part of a library that serves as a reference source for IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.