MODEL 990 COMPUTER REFERENCE MANUAL

PRELIMINARY

TI CLASSIFIED

TI INTERNAL DATA

PROPERTY OF TEXAS INSTRUMENTS ONLY

TEXAS INSTRUMENTS
INCORPORATED

MODEL 990 COMPUTER REFERENCE MANUAL

PRELIMINARY

MANUAL NO. 943442-9701
ORIGINAL ISSUE 15 JUNE 1974
REVISED AND REISSUED 1 OCTOBER 1974

TEXAS INSTRUMENTS
INCORPORATED

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical bar in the outer margins of the page.

Model 990 Computer Reference Manual, Preliminary (943442-9701)

Original Issue . . . . . 15 June 1974
Revised and Reissued . . . . . 1 October 1974

Total number of pages in this publication is 355 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Title	0	App B Div	0	App G Div	0
ii - xiv	0	B-1 - B-4	0	G-1 - G-2	0
1-1 - 1-16	0	App C Div	0	App H Div	0
2-1 - 2-62	0	C-1 - C-4	0	H-1 - H-8	0
3-1 - 3-50	0	App D Div	0	App I Div	0
4-1 - 4-68	0	D-1 - D-12	0	I-1 - I-10	0
5-1 - 5-46	0	App E Div	0	User's Resp	0
6-1 - 6-4	0	E-1 - E-12	0	Bus. Reply	0
7-1 - 7-10	0	App F Div	0	Cover Blank	0
App A Div	0	F-1 - F-4	0	Cover	0
A-1 - A-16	0				

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

APPENDIXES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

SECTION I

SYSTEM INTRODUCTION

1.1 MODEL 990 COMPUTER

The Texas Instruments Model 990 Computer (figure 1-1) is a powerful processing unit featuring byte, bit, and word handling capability as an integral component of its comprehensive instruction set. Within its single chassis, the Model 990 houses its own power supply, a battery power pack to maintain data in memory during external power failures, a micro-programmable

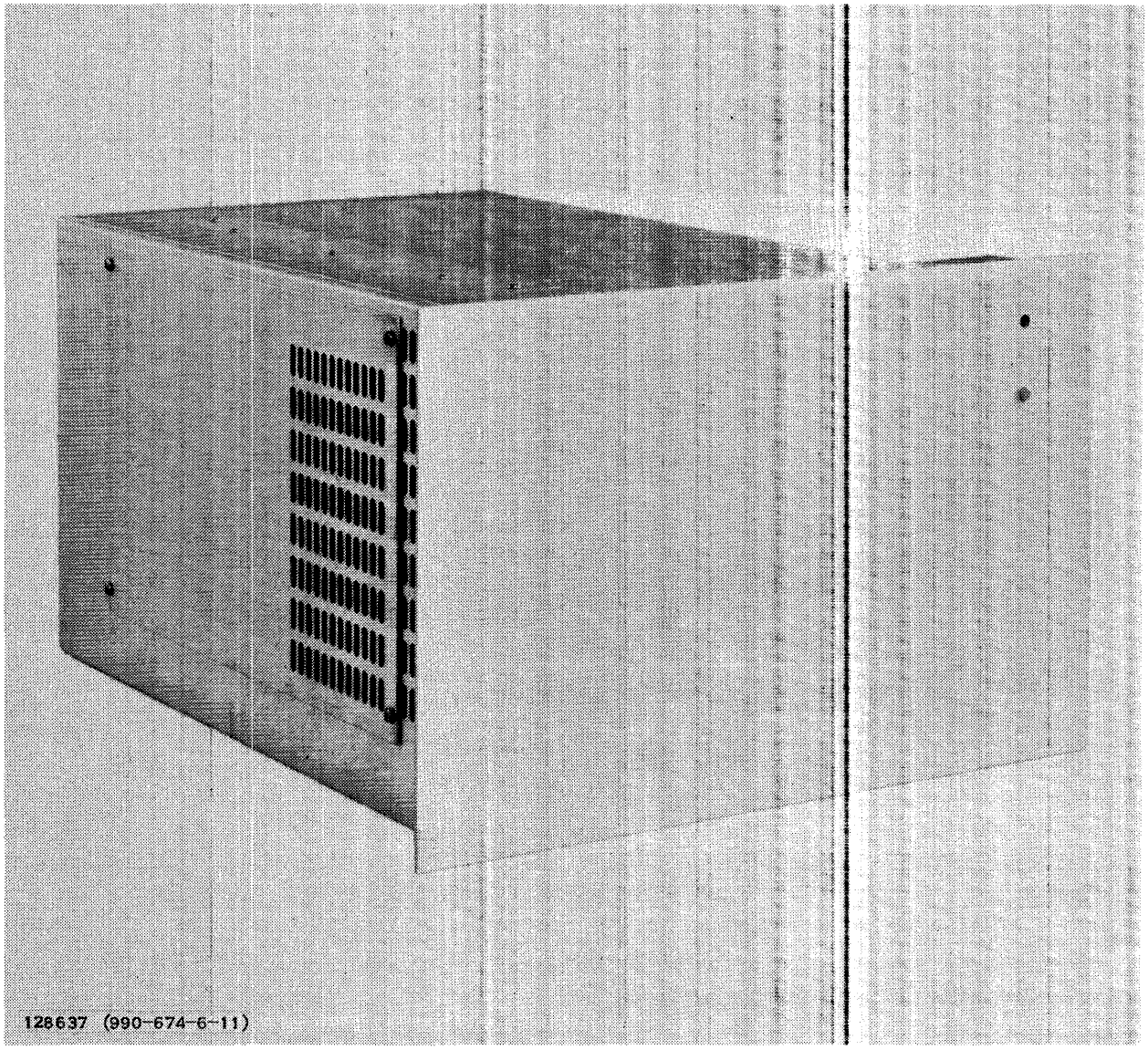


Figure 1-1. Texas Instruments Model 990 Computer

Arithmetic Unit, an error correcting memory controller with up to 32K (4K standard) words of MOS Random Access Memory, a 256-word Read Only Memory for use during start-up operations, plus interface connectors to accomodate a host of optional equipment controllers or interface circuit boards. The Model 990's speed, optional equipment, and competitive price adapt the computer to a wide range of mini-computer applications.

1.2 SYSTEM COMPONENTS

The Model 990 Computer system offers several components to satisfy exacting requirements of particular applications. Included in the standard Central Processing Unit (CPU) are the chassis and backpanel connections, a power supply and battery, the Arithmetic Unit, and a memory controller featuring error correction and 4K words of memory. Interface circuit boards and their corresponding peripheral equipment are offered as optional components, as are additional memory increments up to a total of 32K words. Figure 1-2 illustrates the components of the computer system. The following paragraphs outline the features of these components. Detailed explanation of these components and their operation within the system is supplied in Sections II and III of this manual.

1.2.1 TILINE INTERFACE

The direct memory access channel of the 990 computer is a 16-bit parallel data bus called the TILINE. TILINE links memory, the Arithmetic Unit (AU), and the peripheral devices with a bidirectional, asynchronous data bus whose speed is limited only by that of the devices involved in the transfer (3 million words per second maximum). All devices connected to the TILINE that respond to Read or Write commands are addressed using the same address lines as those used for memory, so that data input from a peripheral device is as simple as fetching data from memory. TILINE automatically resolves conflicts between controllers for access to the TILINE through a positional priority system.

1.2.2 CRU INTERFACE

The Communications Register Unit (CRU) interface provides a bit-addressable input/output channel between the AU and external equipment. The interface transfers serial data by individually sensing each input bit and individually sending each output bit. With this arrangement, the CRU communicates with peripheral I/O equipment through a series of bit transfers, and enables the AU to monitor and control digital processes by sampling individual status bits and generating discrete control signals under program control. All TILINE chassis locations in the computer are also wired for CRU interface cards. By using all of these locations to accept CRU modules containing 16 input and 16 output lines each, the chassis can accommodate up to 256 input and 256 output lines to external equipment. Further expansion of up to 4096 input and 4096 output lines is available through the use of additional

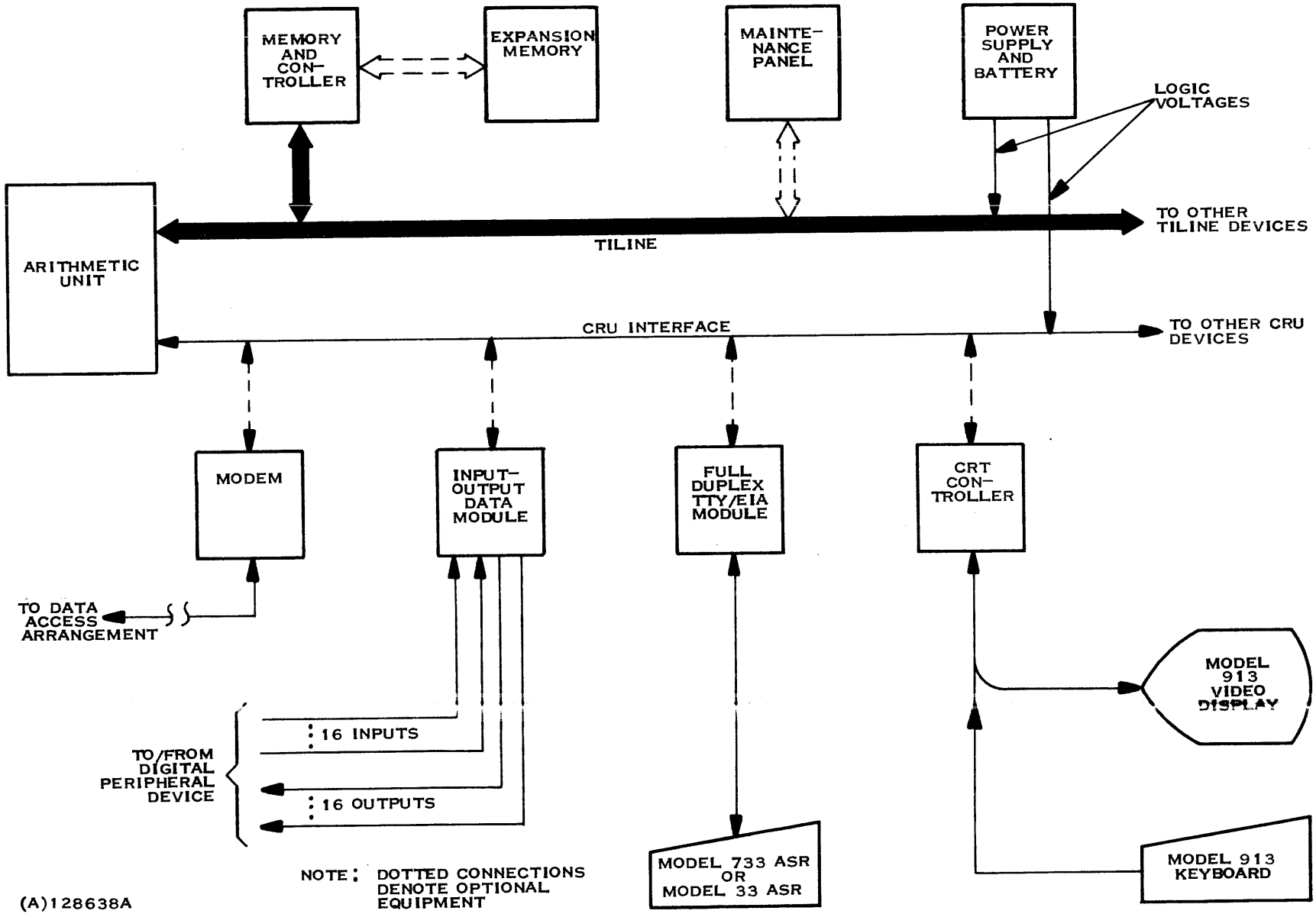


Figure 1-2. Model 990 Computer System Block Diagram

chassis equipment. The expansion capability of this interface and its bit-addressability, make the CRU interface adaptable to many applications.

1.2.3 ARITHMETIC UNIT

The AU is a single circuit board that is the main control and processing unit of the computer. The AU fetches instructions and operands from memory, operates on those instructions, coordinates input and output through either the TILINE or the CRU interface, performs basic arithmetic and logic functions, and stores results into memory for future recall. Primary features built into the AU include: an interface for capability expansion using an additional hardware module to process extended operations, a prioritized, vectored interrupt arrangement; byte addressing and processing; a 256-word ROM loader; and a workspace register concept that improves efficiency over conventional register file arrangements. The AU circuit board must be inserted into chassis location 2 (next to power supply circuit board).

1.2.4 MEMORY AND CONTROLLER

The memory and memory controller for the computer is a single circuit board containing control logic plus either 4K or 8K words of memory. The heart of the memory is a Texas Instruments Metal Oxide Semiconductor (MOS) memory that allows random access to any of 4096 bits within a single package. To ensure data accuracy, the memory controller employs a 6-bit error correcting and detecting code that corrects 1-bit errors, and detects multiple-bit errors. In addition, the controller contains interface logic for expansion memory, enabling the controller to access up to 32K words of memory. The memory circuit board is a TILINE device that responds to addresses generated by any TILINE master device. The memory circuit board can be placed in any TILINE chassis location.

1.2.5 EXPANSION MEMORY

The computer expansion memory is a single circuit board containing from 8K to 24K words (in 8K increments) of memory in addition to that contained on the memory controller circuit board. The expansion memory circuit board occupies the chassis location next to the memory controller circuit board and derives its logic and refresh voltages from the chassis connector. However, all data and control for the expansion memory enter the board through an interface connection with the memory controller circuit board. Like the memory controller, the expansion memory also employs the 4K Random Access Memory (RAM) integrated circuit as the building block of the memory system.

1.2.6 POWER SUPPLY

The power supply is implemented on a single, metal-encased circuit board (figure 1-3) that plugs into chassis connector A1. The module receives 115 Vac, 50 or 60 Hz, power and develops all the voltages required to operate the computer (±12 volts and ±5 volts). In addition, the power supply generates logic signals to inform the AU of an imminent power failure and to control operation of the computer during power transitions. The power supply also removes power from the computer if the temperature within the chassis becomes too high, and restarts the computer when the temperature returns to safe operating levels.

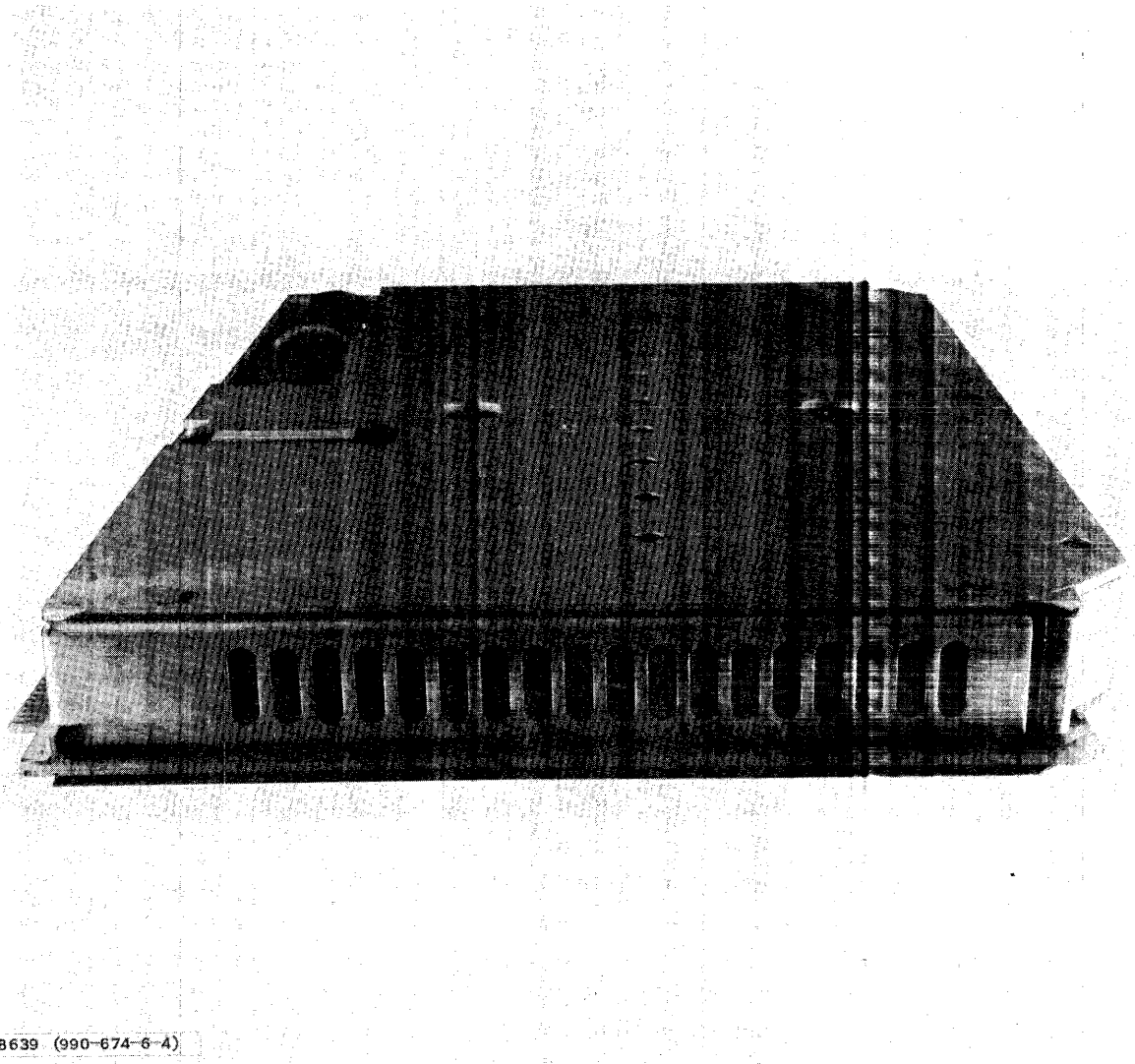


Figure 1-3. Power Supply Circuit Board

1.2.7 CHASSIS AND BACKPANEL

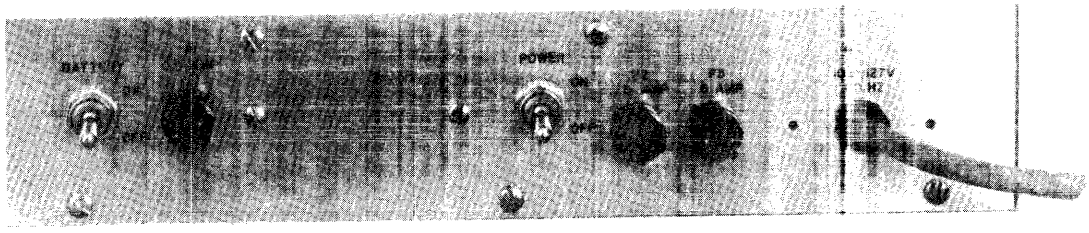
The chassis and backpanel assembly unifies the computer components into a processing system. The backpanel consists of a printed circuit board for fixed interconnections between the components, plus wire-wrap pins to allow individualized interconnections for specific system applications. Mounted to the backpanel are 22 female connectors that accept 80-contact printed circuit board edge connectors. The connectors are arranged in eleven pairs on the backpanel so that they may accept either single or double connector circuit boards. In addition to housing the backpanel, the chassis also contains the reserve power battery, a cooling fan, a fuse and switch panel, and the computer front panel.

1.2.7.1 STANDBY BATTERY. In the event of a momentary or prolonged loss of primary power, the computer switches power source from the power supply circuit board to a +6 volt battery within the chassis. The battery then supplies power to the memory and memory controller circuits to maintain data within memory for the duration of the power failure (to a maximum of 14 hours at room temperature with 4K words of memory). When primary power returns, the power supply recharges the battery pack so that it will be ready in case of another failure. The standby battery powers only the memory and associated circuitry, and is not used to operate the computer for processing.

1.2.7.2 FAN. Mounted behind the circuit boards within the computer chassis, a fan circulates ambient temperature air over the circuit boards to carry away excess heat generated during operation. A filter element behind the fan ensures that the circulated air is free of particles. The fan is powered directly from the main ac power source so that it is running whenever power is applied to the computer.

1.2.7.3 FUSE AND SWITCH PANEL. Mounted to the rear of the computer chassis are two toggle switches and three fuse holders, as illustrated in figure 1-4. When set to ON, the BATTERY toggle switch enables the standby battery to supply power to the unit during a primary power failure or to receive a charge during normal operation. The OFF position of this switch disables battery operation. When set to ON, the POWER toggle switch applies ac power to the computer. The OFF position of this switch removes ac power from the computer. Fuse F1 is a 0.5 amp fuse that protects the battery charging circuit from overload. Fuse F2 is a 5 amp fuse for the main ac power input line. Fuse F3 is not used.

1.2.7.4 FRONT PANEL. The computer front panel contains a power-on indicator and a Load switch, as illustrated in figure 1-5. When lighted, the Power indicator designates that ac power is on within the chassis. The Load switch is a pushbutton that when pressed loads the contents of the 256-word ROM into memory for system initialization. The Load switch is recessed behind the front panel to avoid accidental actuation.



128640 (990-674-6-17)

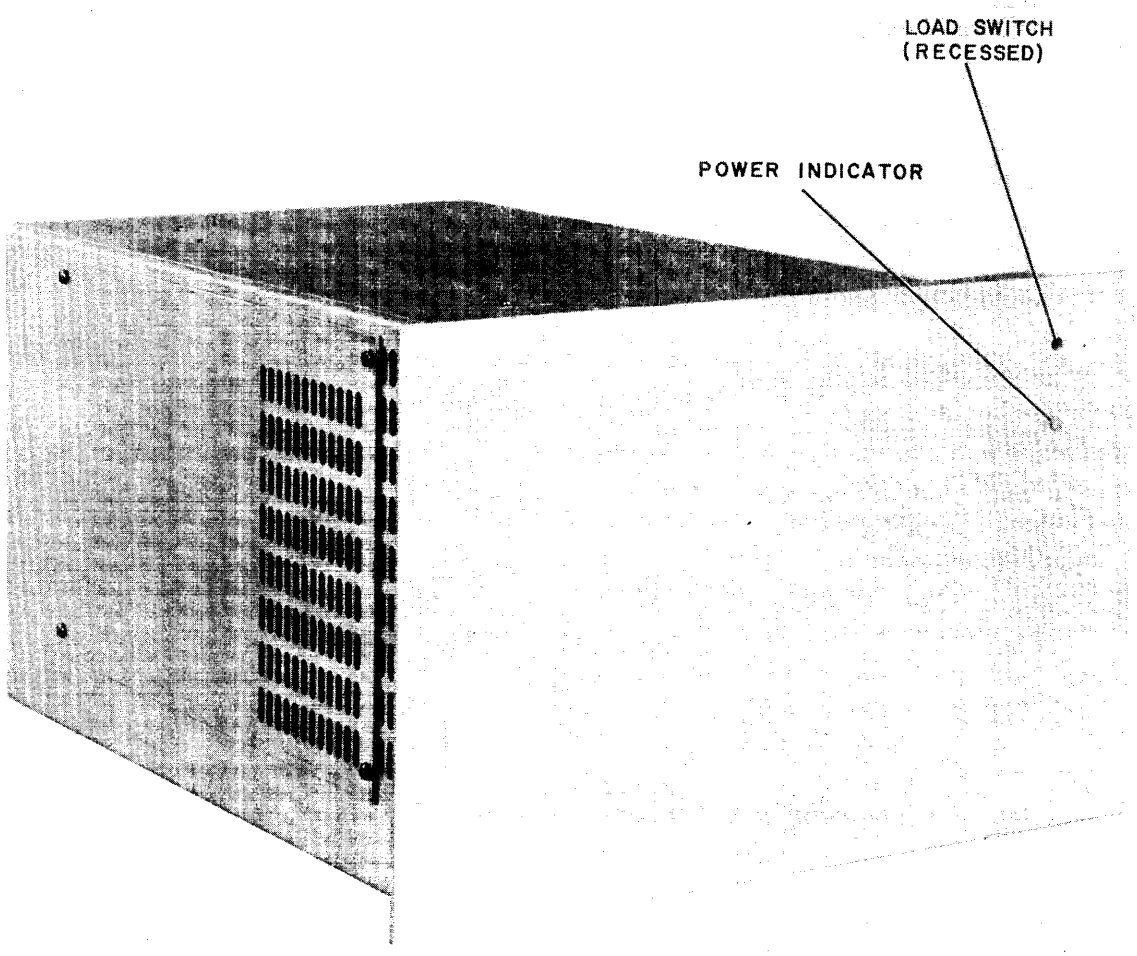
Figure 1-4. Fuse and Switch Panel

1.2.8 913 VIDEO DISPLAY AND KEYBOARD

For high-speed interaction between operator and computer system, the Model 990 computer offers the TI 913 Video Display package. The controller for the display unit is a double-connector circuit board that connects to any CRU chassis location within the computer. A cable attached to the top of the circuit board connects the controller to the display and keyboard units. The 913 is an adaptable display featuring completely programmable cursor positioning, an extra control bit that allows protected display fields, inverted video cursor, and its own refresh memory to relieve the computer from replenishing the contents of the display. The computer fills the refresh memory, and thereby the display screen, at CRU transfer speeds, replacing the contents of the entire screen in less than 20 milliseconds. The computer can also read the contents of the refresh memory at high speed. The display screen provides a legible display. The detachable keyboard features single-function, programmable keys.

1.2.9 TI MODEL 733 ASR

The TI Model 733 Automatic Send-Receive (ASR) Teleprinter provides operator keyboard entry capability as well as automatic input through prerecorded magnetic tape cassettes. The 733 ASR is a twin cassette I/O device that provides superior speed, low noise level and simplicity of operation in an



128641 (990-674-6-11)

Figure 1-5. Computer Front Panel

easily affordable package. Integrated with the cassette I/O is the TI "Silent 700" keyboard and electronic printer to provide absolutely silent operator interaction with the computer system. The 733 ASR expands the 990 Computer system into a powerful software development tool, as well as a versatile processing system.

1.2.10 TTY/EIA INTERFACE MODULE

The Teletypewriter (TTY)/Electronic Industries Association (EIA) Interface module provides a communication path for the 990, through the CRU interface, to peripheral devices that operate through an interface that conforms to EIA document RS232C. The TTY/EIA module may also be wired for TTY current loop interfaces. Typical devices that interface through this module

include: data sets for telephone line data transmission, video display terminals, and teleprinter terminals. The TI Model 733 ASR terminal, used for software loading and interchange with the computer system, interfaces with the 990 through this module. The interface logic is implemented on a single-connector circuit board that, with the accompanying adapter connector, plugs into any connector location that is wired for the CRU interface.

1.2.11 INPUT/OUTPUT DATA MODULE

The Input/Output Data module provides 16 individually addressable digital input bits and 16 individually addressable digital output bits as an extension of the CRU interface. The module is a single-connector circuit board that may be inserted into any connector in the computer chassis that has been wired for the CRU interface. An adapter card must be plugged into the chassis connector before the module is placed in the computer. An edge connector on the top of the module allows attachment of the interface cable to the external signal source. The external device may be a single, 16-bit oriented device such as a card reader, or may be several independent sense lines and output signals. The module also provides an interrupt option.

1.2.12 MODEM

The 990 Computer offers an optional half/full-duplex, 1200 Baud, asynchronous MOdulator-DEModulator (MODEM) with controller logic to interface with the computer CRU interface. The modem unit is compatible with a Bell System CBS 1001A Data Access Arrangement for data transmission over the switched telephone network. Also included is an automatic call, answer and termination circuit that can be compatible with either impulse or touch-tone dialing systems. The modem controller is mounted on a double-connector circuit board that plugs into any CRU chassis location. The modem and auto-call circuitry is implemented on a second, smaller circuit board that is fastened to the controller circuit board, and interfaces to the controller through a cable and edge-connector assembly. This "piggy-backed" arrangement prevents the adjacent chassis location from being used when the modem assembly is installed in the computer.

1.2.13 MAINTENANCE CONSOLE

To aid in fault isolation within the computer, a detachable maintenance console and interface board are offered as an option. The maintenance console allows the technician to exercise and display the internal registers of the computer or a specific memory location. The console is also a useful program debugging tool. The maintenance panel interface board plugs into the chassis location adjacent to the AU circuit board.

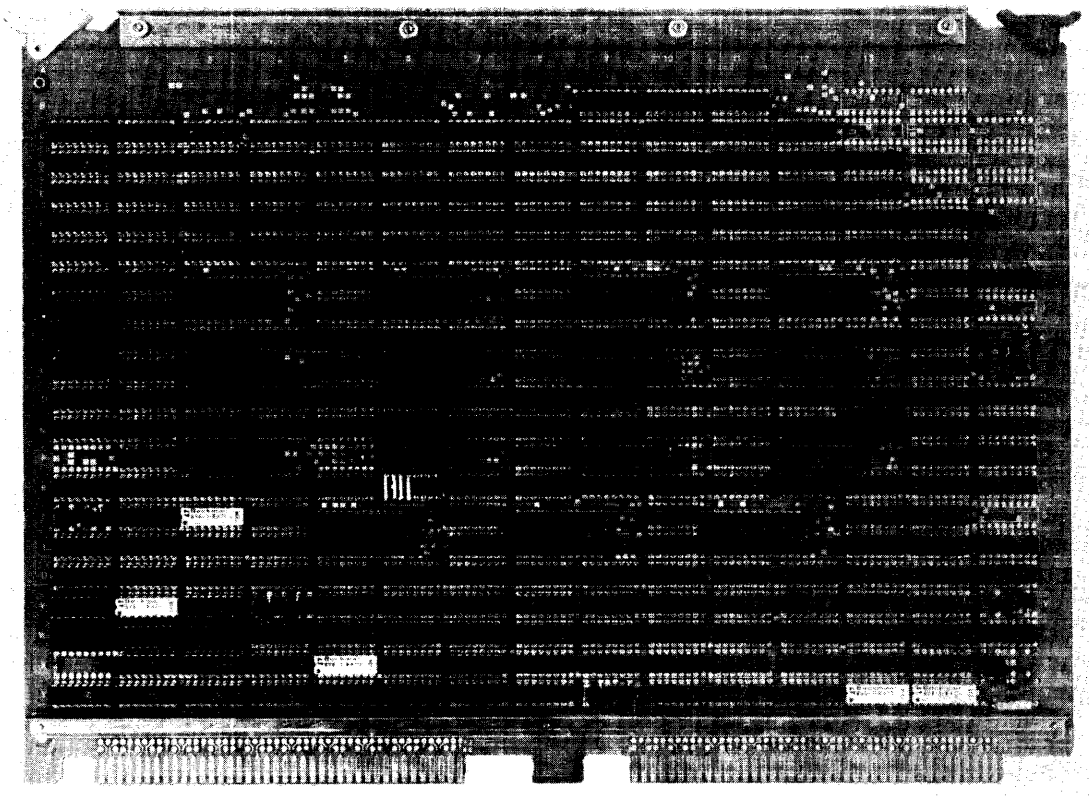
1.3 HARDWARE IMPLEMENTATION

The computer chassis contains eleven pairs of 80-contact connectors for insertion of logic modules with printed contact (card edge) connectors. Each

pair of chassis connectors is assigned a designator beginning with A1 (the bottom pair) through A11 (the top pair). The connectors receive double-connector circuit boards or, using an adapter, single-connector circuit boards that contain the functional modules of the computer system.

1.3.1 DOUBLE-CONNECTOR CIRCUIT BOARDS

Double-connector circuit boards are double-sided or multi-layer printed wiring boards that are approximately 14-3/8 inches wide and 10-1/4 inches high (see figure 1-6). Two connectors, P1 and P2, are formed along the bottom edge of the board by printed conductor contacts. The contacts are



128642 (990-674-6-5)

Figure 1-6. Double Connector Circuit Board

from 1 through 80 with the even numbered contacts on the component side of the circuit board. An additional tab, slightly offset from center along the bottom edge of the board, prevents the circuit board from being inserted into the chassis connectors when the board is backwards. As an additional reminder, one of the ejector tabs on the top edge of the card is colored, while the other tab is white. The colored tab should always be toward the front of the computer chassis. Included in those modules implemented on double-connector circuit boards are: the AU, memory and expansion memory, the power supply, and interface boards for the modem and CRT.

1.3.2 SINGLE CONNECTOR CIRCUIT BOARDS

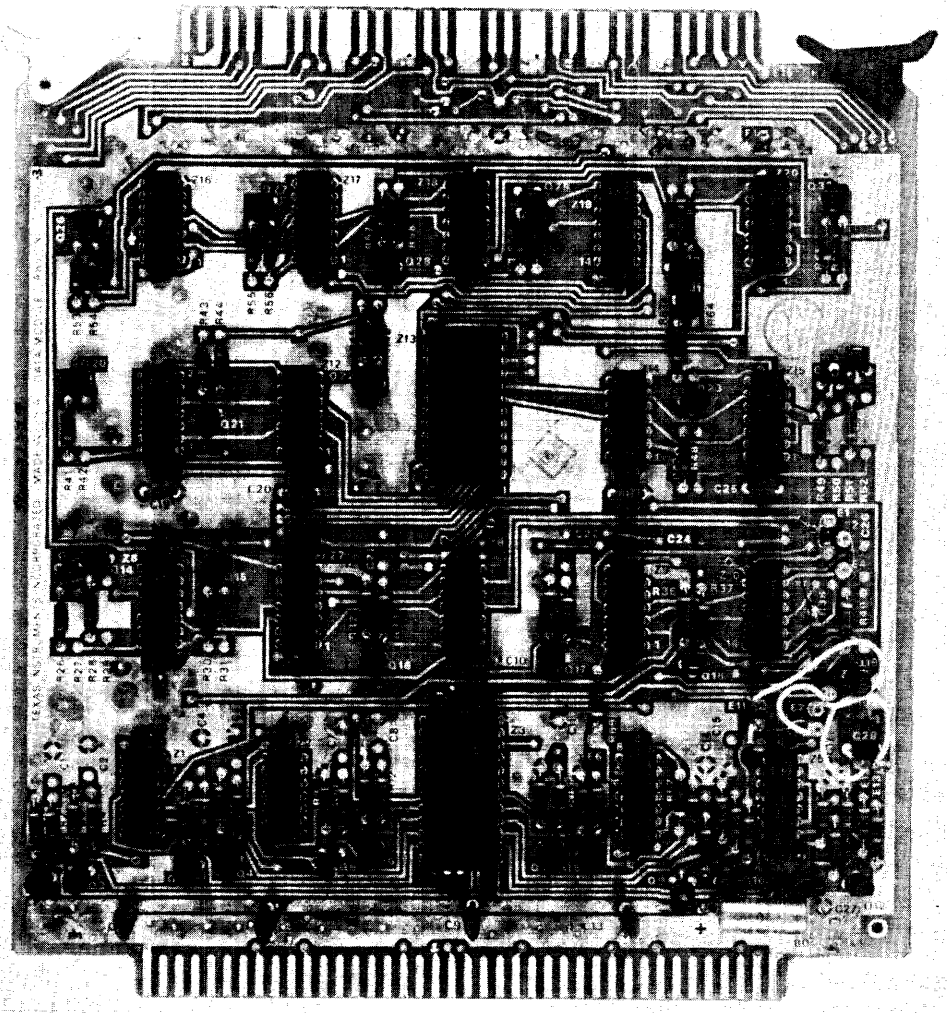
Single-connector circuit boards are doubled-sided printed wiring boards that are approximately seven inches wide and eight inches high (figure 1-7). These circuit boards mount CRU interface modules and typically contain approximately thirty integrated circuit packages plus associated discrete components. The single connector along the bottom edge is an 80-contact, printed conductor connector with the even-numbered contacts on the component side of the board. No provision is made to guarantee that the circuit board is inserted in the proper orientation. However, when inserted into the chassis, the colored ejector tab on the top edge of the card should be toward the front of the chassis.

1.4 SYSTEM APPLICATIONS

The Model 990 Computer's versatile CRU interface and powerful TILINE bus adapt the computer for use in systems of varying complexity from a basic software development processor, through order entry and communications network systems, to intricate systems involving processor redundancy or parallel operations. A variety of practical applications of the computer system is easily implemented. The following paragraphs describe some basic applications. These systems may be combined, altered and expanded to meet the requirements of a particular task.

1.4.1 PROCESSOR TERMINAL

Combining the Model 990 Computer with up to eight 913 CRT Display units together with the modem controller and interface transforms the computer into an intelligent terminal controller with the computation power and expansion capabilities of a larger computer (figure 1-8). Each display unit operates independently rather than in a slave mode to other displays, so that use of one display unit is not restricted by a process occurring at another unit. The use of a fully expanded memory system with this configuration and the addition of a bulk storage unit through the TILINE interface develop a polled terminal for on-site editing and batching of information to a central computer.

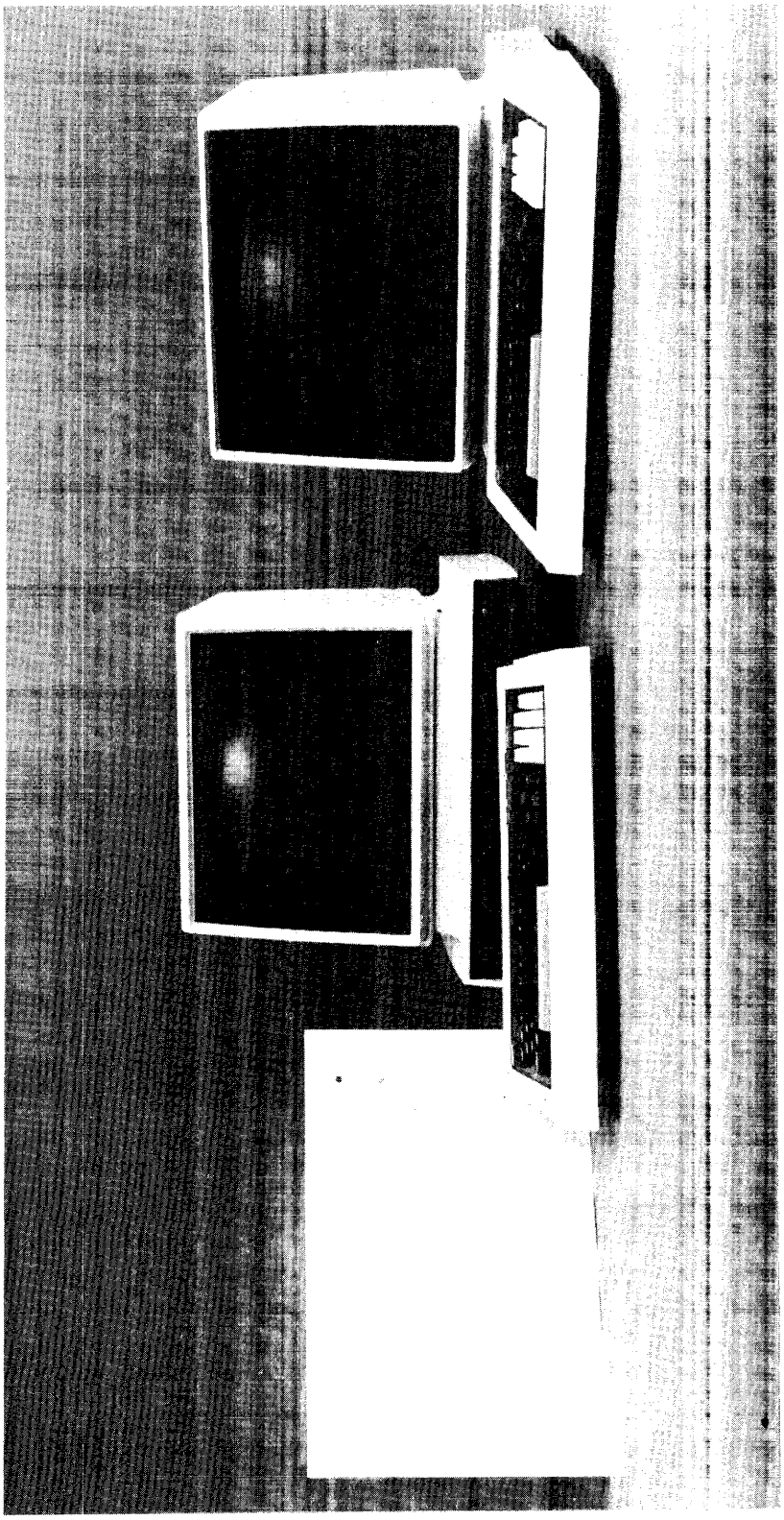


128643 (990-674-6-1)

Figure 1-7. Single Connector Circuit Board

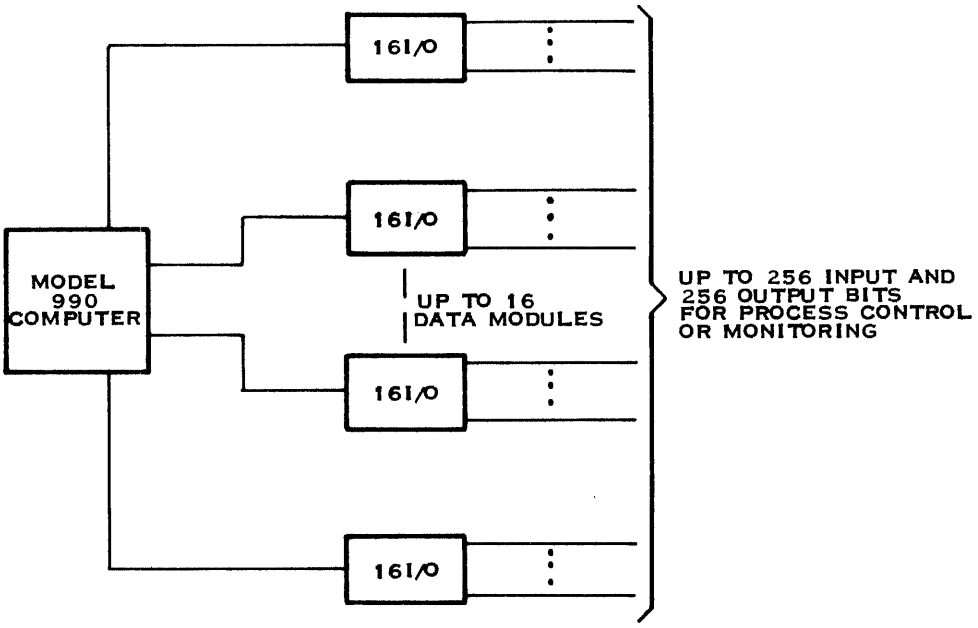
1.4.2 INDUSTRIAL PROCESS CONTROLLER

When fully implemented with 16 input/output data modules, the Model 990 Computer supplies 256 input lines and 256 output lines for industrial process monitoring and control (see figure 1-9). The input lines can feed to the computer status information such as temperature deviations, pressure levels, presence or absence of a piece-part at a particular station, or exact position of a tool bit. The computer can then process these inputs and respond by issuing control signals over the output lines to perform such functions as turning a valve on or off, initiating a machine process, or guiding a machine tool to the precise position required. Use of the computer in this mode ensures rapid response to changing conditions and therefore, more accurate



128644 (990-674-10-2)

Figure 1-8. Model 990 Processor Terminal Application



(A)128645

Figure 1-9. Model 990 Industrial Control Application

and dependable process results. For larger industrial applications, the Model 990 Computer CRU interface may be expanded through add-on chassis to provide up to 4096 input lines and 4096 output lines.

1.4.3 SOFTWARE DEVELOPMENT COMPUTER

Combining the Model 990 Computer with a Texas Instruments Model 733 ASR terminal creates a software development tool (figure 1-10) that leaves the larger portion of processor time for other operations. The 733 ASR allows easy loading of programs or recording of results using twin magnetic tape cassette input/output devices. Operating at 120 characters per second, the cassette system offers an easy-to-use alternative to bulky tape reel systems or inefficient paper tape input systems. Listings and printouts at a quick, silent 30 characters per second provide a convenient hard copy of resulting programs. The operator, using the 733 ASR keyboard, can easily enter any changes or corrections to his program, and respond in an interactive mode to events that occur during the program.

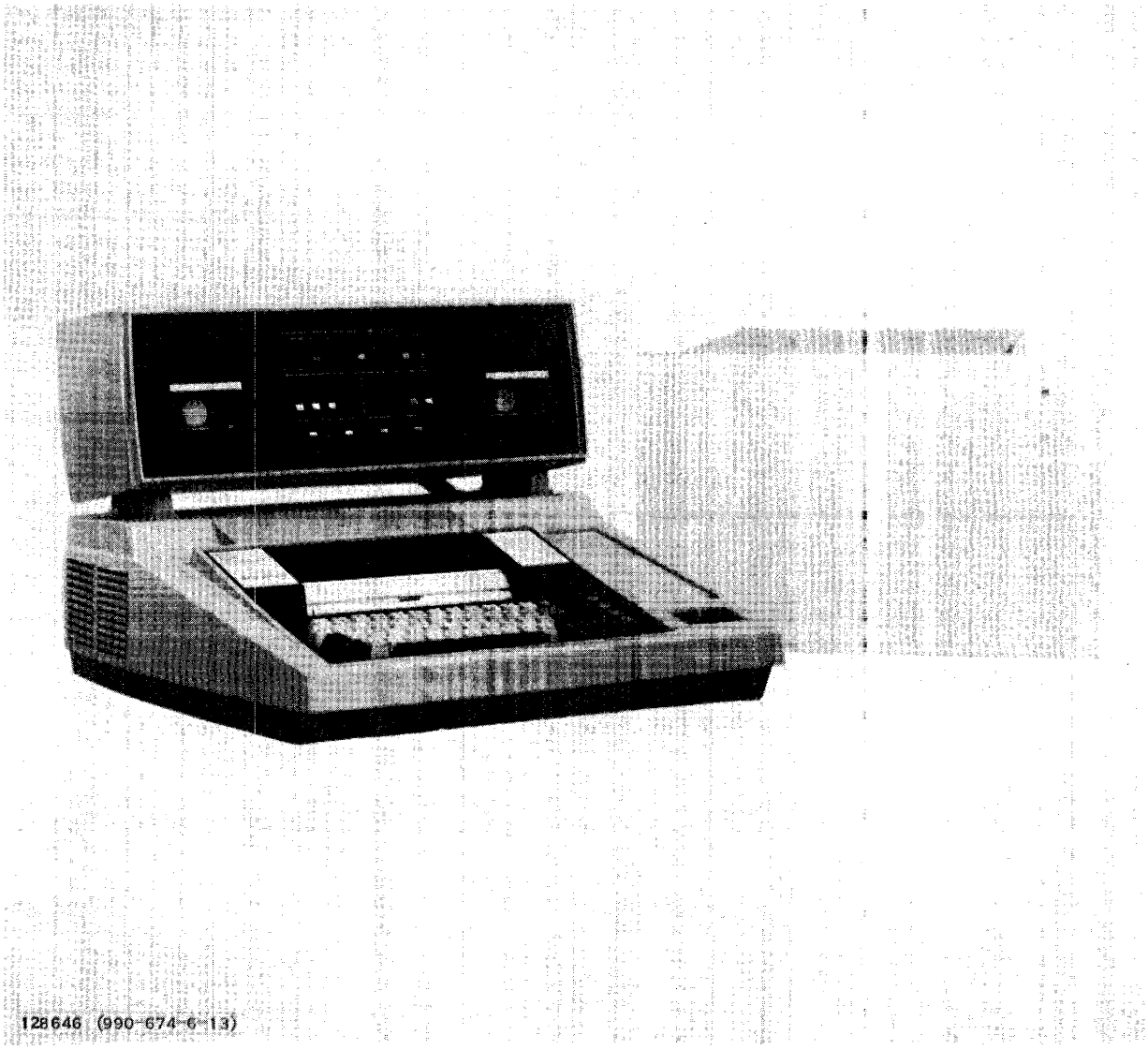


Figure 1-10. Model 990 Software Development Configuration

SECTION II

MAINFRAME HARDWARE

2.1 INTRODUCTION

This section discusses the theory of operation, design criteria, and interface specifications of the Model 990 Computer. It describes the operation of the Arithmetic Unit, memory, extended operation feature, TILINE and Communications Register Unit (CRU) interfaces, and the optional maintenance console. This information, together with the interface examples in Appendixes D and E of this manual, provides an essential background of the machine's capabilities and requirements.

2.2 ARITHMETIC UNIT

The Arithmetic Unit (AU) of the Model 990 Computer provides a wide range of computer functions in a compact, single circuit board configuration. It achieves its high function density through a mixture of multi-purpose registers, read-only-memory (ROM) control circuits and a workspace concept that substitutes memory locations for hardware register files. The single AU circuit board contains interface logic for both TILINE and CRU device, interrupt detection and priority logic, data handling registers, an arithmetic logic unit (ALU), plus three ROM networks. Figure 2-1 illustrates the interrelation of these internal components.

2.2.1 AU CONTROL

Much of the AU's functional density results from micro-programmed control circuits for data transfer and manipulation within the AU. Two ROM circuits generate all control signals required for the AU to respond to the 990 instruction set. Instructions from memory enter the basic function ROM (BFR), where they are decoded into a starting address for the instruction microsequence. The control ROM (256 x 64) receives the output from BFR. The control ROM, together with its associated next address selection circuitry, is the central control mechanism in the AU. Output bits from the control ROM select the function of the ALU, gate data through the AU data paths, and help select the next control word in the ROM to continue execution of the current instruction. New instructions in the program will produce a new starting address from BFR as they are read from memory.

2.2.2 INSTRUCTION REGISTER

Although instructions entering the AU are routed to the BFR for instruction decoding, the word is only available on the memory lines for one clock cycle. Since much of the instruction word is required during later clock cycles, a

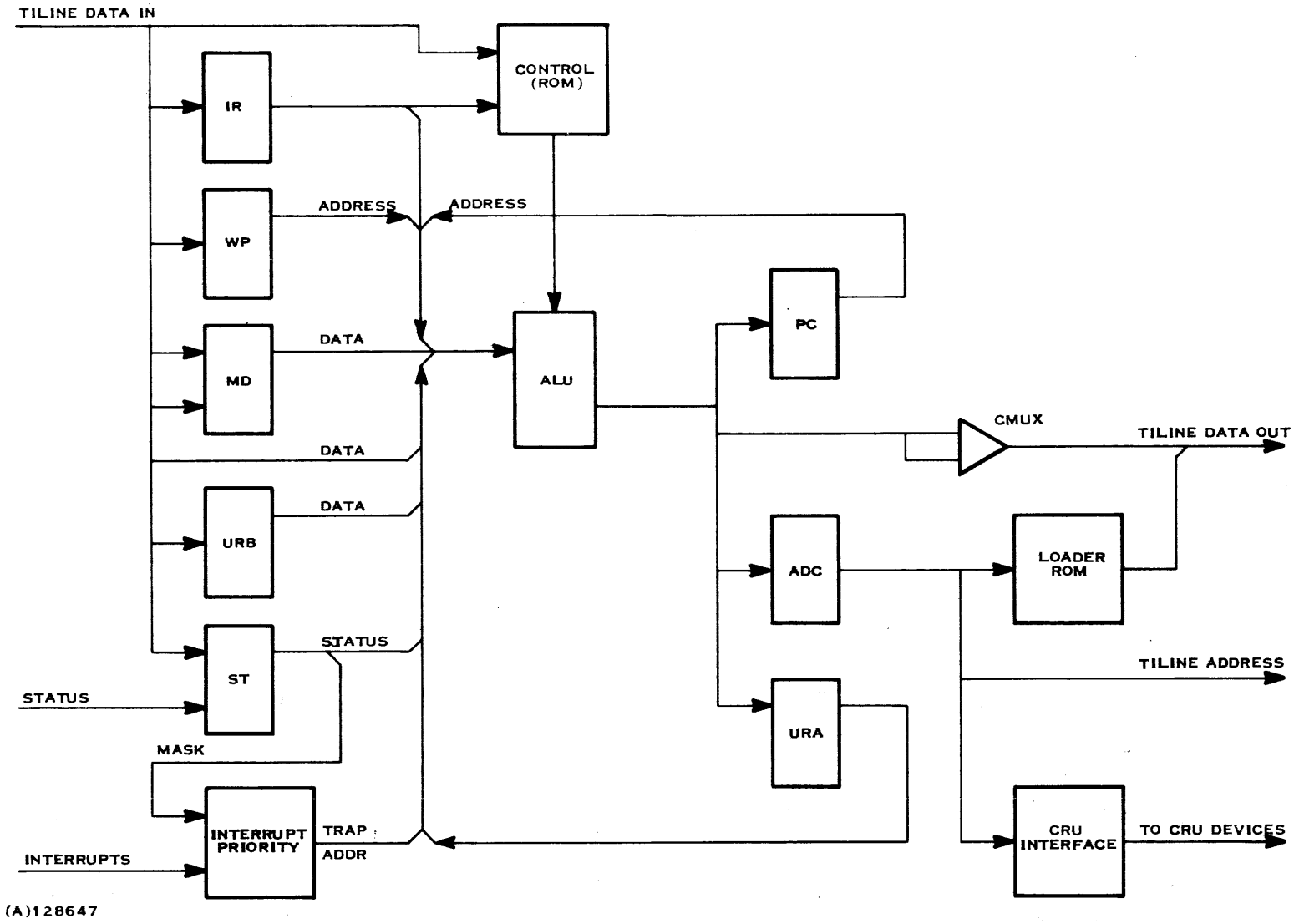


Figure 2-1. Arithmetic Unit Block Diagram

12-bit instruction register (IR) saves the most significant 12 bits of the instruction. The contents of IR then determine such parameters as addressing mode (TS/TD fields), operand length (byte indicator), shift lengths (C field), and result destination address (D field) of the instruction being executed (op code). The data remains in IR until a new instruction word enters from memory.

2.2.3 GENERAL REGISTERS

The AU contains three 16-bit general registers for data handling. Two of these registers, the memory data (MD) and universal register B (URB), receive inputs directly from memory and serve as the data staging area for input to the ALU. The MD register also has a selectable byte input for swap byte (SWPB) operations and for relocating the valid type to the most significant halfword position during byte operand functions. These operands return to their normal orientation before being stored into memory. The URB register also has a right or left shift capability for use during simple shift instructions, for shifting operands during multiply and divide instructions, and for parallel-to-serial conversion during CRU operations.

The third general register, universal register A (URA), receives results from the ALU after each operation. The contents of URA are then used for logical and arithmetic comparisons to control bits 0, 1, 2 and 5 of the status register. URA also has a selectable right or left shift capability that is used during multiply and divide instructions.

2.2.4 WORKSPACE

As an improvement over the undesirable consequences of a multi-register architecture, the 990 Computer uses a block of memory words, called a workspace, for instruction operand manipulation. The workspace occupies 16 contiguous memory words in any part of memory that is not reserved for other use. The individual workspace registers may contain data or addresses, and are used as operand registers, accumulators, address registers, or index registers. Some workspace registers take on special significance during execution of certain instructions. Table 2-1 lists each of these dedicated workspace registers and the instructions that use them.

2.2.5 WORKSPACE POINTER

To locate the workspace in memory, the AU has one hardware register called the workspace pointer (WP). The workspace pointer is a 15-bit register that contains the memory address of the first word in the workspace. The AU can then access any register in the workspace by adding the register number to the contents of the workspace pointer and initiating a memory request for that word. Figure 2-2 illustrates the relationship between the workspace pointer and its corresponding workspace in memory.

Table 2-1. Dedicated Workspace Registers

Register No.	Contents	Used During
0	Shift count (optional)	Shift instructions (SLA, SRA, SRC and SRL)
11	Return address	Branch and Link Instruction (BL)
	Effective address	Software implemented Extended Operation (XOP)
12	CRU base address	CRU instructions (SBO, SBZ, TB, LDCR and STCR)
13	WP register contents	Context switching (BLWP, RTWP, software XOP, and recognized interrupt)
14	Return address	Context switching (BLWP, RTWP, software XOP, and recognized interrupt)
15	ST register contents	Context switching (BLWP, RTWP, software XOP, and recognized interrupt)

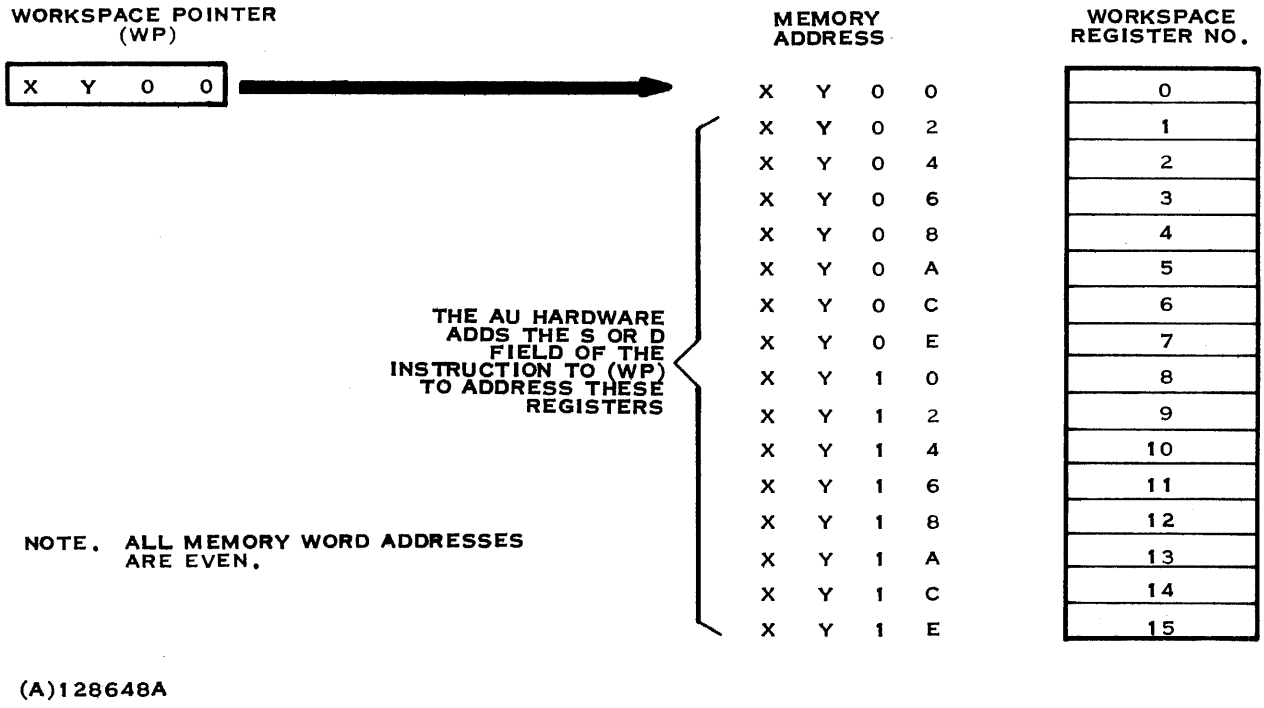


Figure 2-2. Workspace Pointer and Registers

2.2.6 CONTEXT SWITCHING

The workspace concept is particularly valuable during operations that require a context switch, or a change from one program to another or to a subroutine. Such an operation using a conventional multi-register arrangement requires that the entire contents of the register file, the program counter, and the status register be stored and reloaded using a memory cycle to store or fetch each word. The workspace concept accomplishes this operation in only three store cycles and three fetch cycles (program counter, status register and workspace pointer), producing a time savings of 15 store cycles and 15 fetch cycles. After the switch, the workspace pointer contains the starting address of a new 16-word workspace in memory for use in the new routine. A corresponding time saving occurs when the original context is restored. Not all context switching operations in the computer affect the status register. However, instructions that result in either a full or partial context switch include: Branch and Load Workspace Pointer (BLWP), Return from Interrupt Subroutine (RTWP), and an Extended Operation (XOP) instruction that is software implemented. Device interrupts also cause a context switch by forcing the AU to trap to a service subroutine.

2.2.7 ARITHMETIC LOGIC UNIT (ALU)

The arithmetic logic unit (ALU) is the computational component of the AU. It performs all arithmetic and logic functions required to execute 990 instructions. These functions include addition, subtraction, multiplication, division, AND, OR, exclusive OR, NAND, NOR and complement. A separate comparison circuit performs the logical and arithmetic comparisons to control bits 0 through 2 of the status register.

2.2.8 BYTE PROCESSING

The ALU is arranged in two halves to accommodate byte operations. Each half of the ALU operates on one byte of the operand. During word operand operations, both halves of the ALU function in conjunction with each other. However, during byte operand processing, the least significant half of the ALU operates in a passive mode, performing no operation on the data that it handles. The most significant half of the ALU performs all operations on byte operands so that the overflow circuitry used in word operations can also be used in byte operations. Because of this shift in functional halves of the ALU during byte operation, the AU inspects the address of each byte operand before transferring it from the TILINE bus to MD, the input register to the ALU. If bit 15 of the address is set, then the selected byte is in the least significant half of the memory word. The AU switches the position of the bytes within the incoming word as it enters MD. If bit 15 is not set, then the selected byte is in the most significant half of the memory word and no switch is required. If the bytes have been switched at the input to the ALU, they must be restored to their proper orientation before the word is returned to memory. A selectable gate (CMUX) allows the AU to perform any required

reorientation of bytes before placing the word on the TILINE bus to memory. Figure 2-3 illustrates the handling of an odd addressed (bit 15 set) byte operand as it passes through the AU.

2.2.9 PROGRAM COUNTER (PC)

The program counter is a 15-bit register-counter that contains an address that is one greater than the word address of the instruction currently executing in the AU. The AU references this address to fetch the next instruction from memory and increments the address in PC when the new instruction begins executing. PC does not, however, always contain the address of the next instruction to be processed. If the current instruction in the AU alters the contents of PC, then a program branch occurs to the location specified by the altered contents of PC. All context switching instructions plus simple branch instructions affect the contents of PC.

2.2.10 ADDRESS DEFINITION CODE REGISTER (ADC)

The address definition code (ADC) register is a 16-bit register that can be incremented by two without using the ALU. The 16 bits in the register represent an addressing capability to the byte level, so that an increment by two actually produces consecutive word addresses. The contents of the

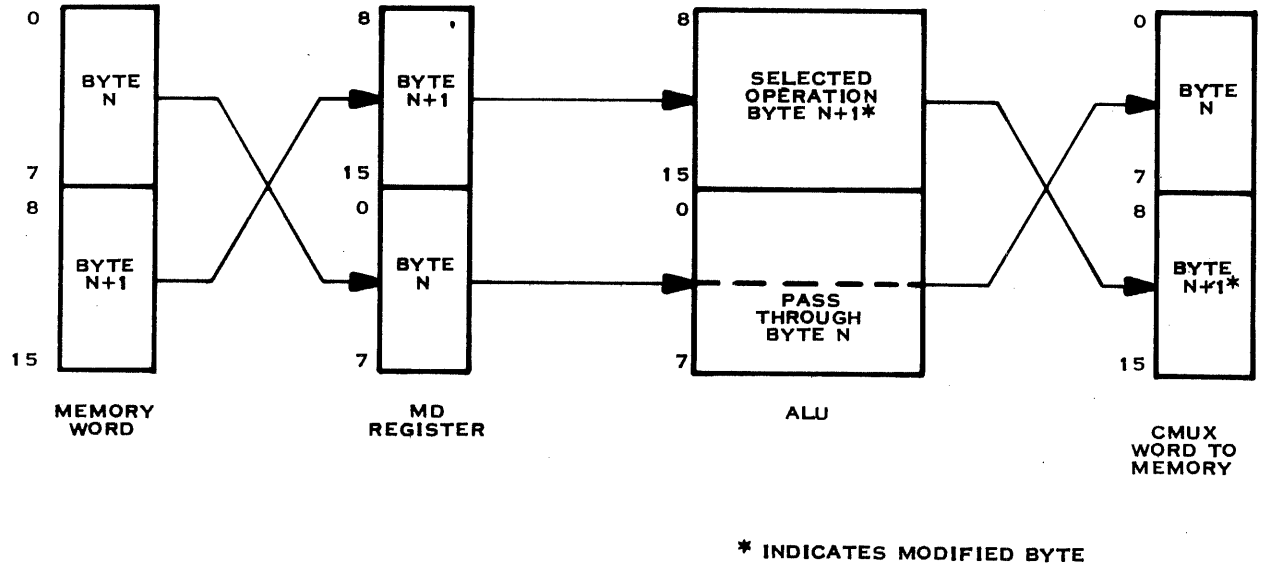


Figure 2-3. Odd Address Byte Switching

register are not used as a pure address, however. Portions of the code, combined with other parameters, form the actual addresses for computer usage. The ADC output performs three major functions in the AU: development of TILINE address, CRU bit selection, and loader ROM sequencing.

2.2.10.1 TILINE ADDRESS. The TILINE interface produces a 20-bit address to memory and other TILINE devices, providing the capability for addressing up to one million words of memory and TILINE device data. However, only 32K words of this addressing power are currently used. Since all TILINE addresses are word addresses, ADC bit 15 (byte address) is not required for TILINE addresses. This bit is used internal to the AU during byte operations. To produce the TILINE address, the AU transfers the 15-bit word address (most significant 15 bits) from ADC to the least significant 15 bits of the TILINE address (bits 5-19). The remaining 5 most significant bits of the TILINE address are forced to zeros except when the ADC bits exceed a value of approximately 31K. At that point, the AU forces the most significant 5 bits to ones, producing an address that memory cannot recognize. These last 1K words allow the AU to address TILINE device controllers without conflict with memory addresses.

2.2.10.2 CRU BIT SELECT. The output from ADC also selects a bit in the Communications Register Unit (CRU) network for CRU operations. The CRU base address is stored in workspace register 12. At the start of a CRU operation, the AU fetches the base address, modifies it as required, and loads it into ADC. Bits 3-14 of ADC then select a CRU bit based upon the address bit assignments illustrated in figure 2-4. The increment by two feature of ADC allows the AU to sequence through consecutive CRU bit addresses to manipulate a series of CRU bits (bit 15 is not used in the CRU interface address, so that an increment of ADC by two corresponds to an increment by one of the CRU address). The CRU interface is explained in detail later in this manual.

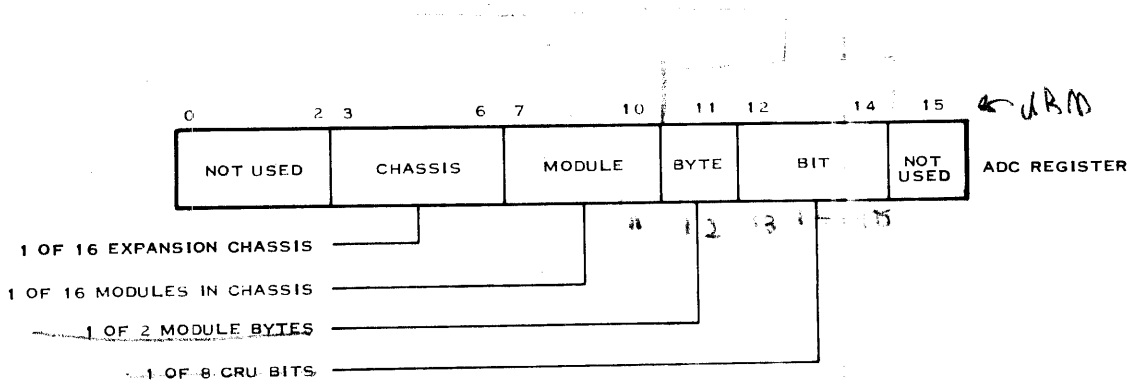


Figure 2-4. CRU Address Bit Assignments

2.2.11 LOADER ROM

Integral to the AU circuit board is a 256-word ROM that is preprogrammed with initialization data for the 990 Computer. Alternately, an external circuit board containing up to 1024 words of initialization data may be inserted into any TILINE slot in the 990 chassis. The presence of this external ROM board disables operation of the internal AU loader ROM. Pressing the Load switch on the front panel or execution of an LREX instruction transfers the contents of the ROM into active memory. Figure 2-5 illustrates the location of the four ROM packages on the AU circuit board. The packages are socket-mounted to the board for easy replacement.

2.2.11.1 DATA TRANSFERS. When the load operation begins, the AU examines the TILINE interface to determine if the external ROM board is present before enabling the internal ROM. Regardless of the number of initialization words in the ROM circuits, the AU then begins a sequential store operation into the first 4K words of memory. If the 256-word AU ROM provides the initialization data, this data is repeated sixteen times to fill the space in memory. An option is available for software development that only loads into the first 256 words of memory.

2.2.11.2 ROM SEQUENCING. The ADC register generates the addresses required to load the ROM data into memory. When the operation begins,

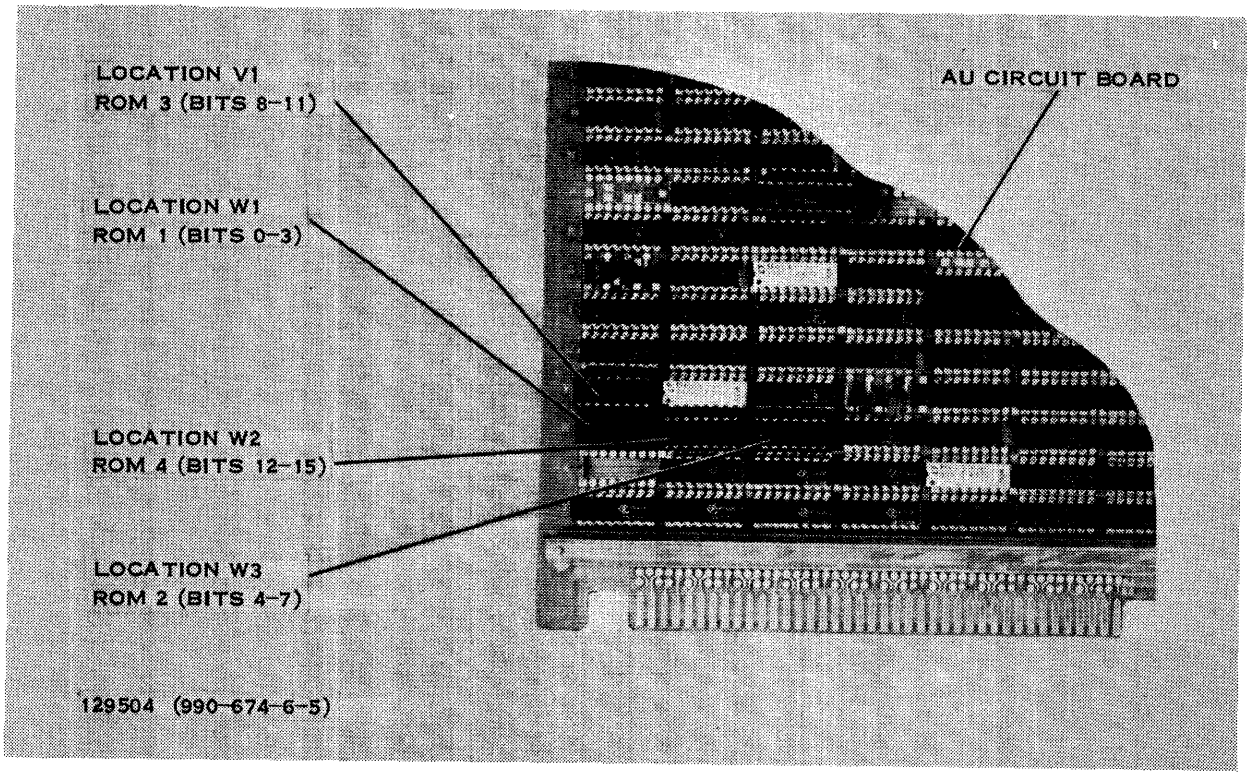


Figure 2-5. AU Loader ROM Chip Locations

ADC contains an address of zero. Each succeeding clock then increments the word address in ADC until it reaches a count of 4095 (255 with software development option). The next clock clears ADC and begins loading WP and PC with values from the initialization data in memory. As each address is generated during the load procedure, it is placed on the TILINE address bus and bits 7-14 are routed to the AU loader ROM. If enabled, the AU ROM will place the addressed word on the TILINE data bus to be stored in memory at the location indicated by the TILINE address. If the external ROM is used, the TILINE address bus defines both the memory storage address and the ROM address on the external board that will supply the data.

2.2.12 STATUS REGISTER

The status register is a 16-bit register that reports the results of program comparisons, indicates program status conditions, and supplies an interrupt mask level to the interrupt priority circuits. Each bit position in the register signifies a particular function or condition that exists in the AU. Figure 2-6 illustrates the bit position assignments. Some instructions use the status register to check for a prerequisite condition, others affect the values of the bits in the register, and others load the entire status register with a new set of parameters. The description of the instruction set later in this manual details the effect of each instruction on the status register.

2.2.12.1 LOGICAL GREATER THAN (BIT 0). When set, bit 0 indicates that an instruction produced a "logical greater than" result. When clear, bit 0 indicates that the instruction yielded a "not greater than" result.

2.2.12.2 ARITHMETIC GREATER THAN (BIT 1). When set, bit 1 indicates that an instruction produced an arithmetic "greater than" result. When clear, bit 1 indicates that the instruction yielded a "not greater than" result.

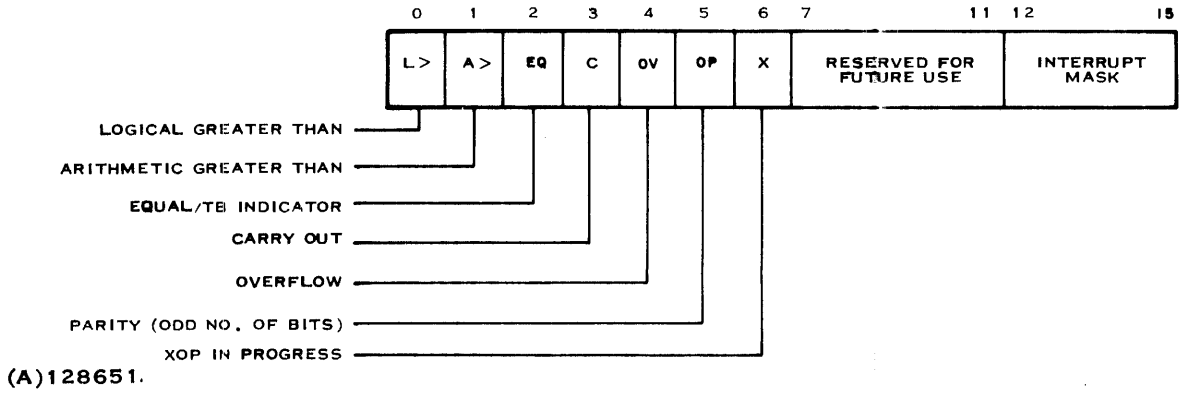


Figure 2-6. Status Register Bit Assignments

2.2.12.3 EQUAL (BIT 2). When set, bit 2 indicates that an instruction produced an "equal" result. When clear, the instruction yielded a "not equal" result. This bit is also used during test bit instructions to indicate the value of the CRU bit tested, and during compare corresponding instructions to indicate the outcome of that comparison.

2.2.12.4 CARRY (BIT 3). Bit 3 sets when a carry out from the most significant bit position occurs during an arithmetic operation. The bit resets when no carry out of the most significant bit position occurs. The arithmetic operations that affect the carry bit are addition, subtraction, increment, and decrement. The carry bit also stores the value of the last bit shifted out of the operand during shift instructions.

2.2.12.5 OVERFLOW (BIT 4). Bit 4 sets when the result of an arithmetic operation is too large or too small to be correctly represented in two's complement form within the number of bits used for the result. The bit resets when the result is correctly represented. The arithmetic operations that affect the overflow bit are addition, subtraction, increment, and decrement. A division operation sets the overflow bit when the most significant sixteen bits of the dividend are greater than or equal to the divisor. During an arithmetic left-shift, the overflow bit sets if the sign bit changes and clears if the sign bit does not change.

2.2.12.6 ODD PARITY (BIT 5). Bit 5 is affected only by byte operations. This bit sets to indicate that the number of one bits in the resultant byte is an odd number; this bit clears when the number of one bits in the resultant byte is an even number.

2.2.12.7 EXTENDED OPERATION IN PROGRESS (BIT 6). Bit 6 sets when a software implemented XOP instruction is encountered in the program sequence. If the XOP is to be executed by an external hardware module, bit 6 remains clear. If the XOP is implemented by a software subroutine, bit 6 sets until the subroutine completes and the resulting context switch restores the previous contents of the status register.

2.2.12.8 INTERRUPT MASK (BITS 12-15). The interrupt mask contains a 4-bit value that indicates which interrupt levels will be recognized by the AU interrupt priority logic. The value in the interrupt mask enables that interrupt level and all interrupt levels below it (those having higher priority). For example, a mask value of "4" (0100) enables interrupt level 4 (TILINE Time Out) and also levels 0 through 3.

2.2.13 INTERRUPTS

The 990 Computer employs sixteen interrupt levels. A priority ranking system assigns numbers from 0 (highest priority) to 15 (lowest priority) to the levels so that interrupt conflicts can be resolved. The six highest priority levels are used for internal interrupts and the remaining ten levels (6 through

15) are available for external device interrupts. The interrupt levels are vectored for rapid reaction to recognized interrupts. That is, corresponding to each interrupt level is a 2-word vector located in low-order memory (addresses 00 through 3F). When the AU recognizes an interrupt, it loads the vector for that level into WP (first vector word) and PC (second vector word) to define the new workspace and program starting point for the interrupt servicing routine. When the interrupt routine is complete, the AU returns to the program that was executing when the interrupt occurred.

2.2.13.1 MASKING. The AU uses a 4-bit field in the status register to determine the lowest priority interrupt that will be recognized during a program operation, and also to ensure that an interrupt service routine will not be halted due to another interrupt of equal or lower priority. At the start of a program the mask field in the status register is loaded with the mask value. The AU compares this value continuously with any interrupts that occur. If the level of the interrupt is equal to or less than the mask value (equal or greater priority), then the AU recognizes the interrupt and calls the service routine for that interrupt level. When the AU sets up the service routine, it loads a value into the mask field that is one less than the interrupt level being serviced, thereby disabling interrupts from devices of equal or less priority. Table 2-2 lists the interrupt levels, assignments, vector location and mask information.

2.2.13.2 LEVEL 0 - POWER ON. Whenever ac power is applied to the computer, it issues a level 0 interrupt, setting the interrupt mask to 0.

2.2.13.3 LEVEL 1 - POWER FAILING. When ac power begins to fail, a sensor in the power supply generates a level 1 interrupt. At that point the computer has one millisecond of program time before a power supply reset halts operation. This interrupt sets the interrupt mask to 0.

2.2.13.4 LEVEL 2 - MEMORY ERROR. When a non-recoverable memory error occurs, the memory controller generates a level 2 interrupt. This interrupt sets the interrupt mask to 1.

2.2.13.5 LEVEL 3 - ILLEGAL OPERATION. When the AU acquires an instruction from memory that cannot be executed, it generates a level 3 interrupt. If level 3 interrupts are disabled, the AU increments PC by two and attempts to execute the instruction at that address. When the interrupt is recognized, the AU sets the interrupt mask to 2. Illegal operation codes are within the following ranges:

0000 through 01FF

0780 through 07FF

0C00 through 0FFF

Table 2-2. Interrupt Level Data

Interrupt Level	Vector Location (Trap Address)	Device Assignment	Enabling Mask Values
0	00	Power on	0 through F
1	04	Power failing	1 through F
2	08	Memory error	2 through F
3	0C	Illegal operation	3 through F
4	10	TILINE time out	4 through F
5	14	Real time clock	5 through F
6	18	External device	6 through F
7	1C	External device	7 through F
8	20	External device	8 through F
9	24	External device	9 through F
10	28	External device	A through F
11	2C	External device	B through F
12	30	External device	C through F
13	34	External device	D through F
14	38	External device	E and F
15	3C	External device	F only

2.2.13.6 LEVEL 4 - TILINE TIME OUT. If the AU issues a request for data from memory or for communication with another TILINE device and fails to receive a response within 10 microseconds, the AU issues a level 4 interrupt. This interrupt sets the interrupt mask to 3.

2.2.13.7 LINE FREQUENCY CLOCK. The power supply contains a line frequency synchronized clock. The clock frequency is 120 Hz. A signal is generated every 8.33 ms to provide a level 5 interrupt request. The clock is started and stopped by execution of control instructions. The initial interrupt request following the starting of the clock may occur between 1 μs and 8.33 ms later. Timing by the clock is only as accurate as the power line frequency to which it is synchronized. When the interrupt is taken, the AU sets the interrupt mask to 4.

2.2.13.8 EXTERNAL INTERRUPTS. Interrupt levels 6 through 15 are available for assignment to CRU or TILINE devices. The external levels may be shared by several device interrupts depending upon system requirements. All interrupt requests must remain active until recognized by the interrupt service routine. The individual service routines must reset the interrupts before the routine is complete.

2.2.13.9 INTERRUPT TRANSFERS. The AU continuously compares the highest priority outstanding interrupt with the interrupt mask. When the level of the pending interrupt is less than or equal to the mask level (higher or equal priority), the AU recognizes the interrupt and initiates a context switch following completion of the currently executing instruction. The AU first fetches the interrupt vector corresponding to the interrupt level recognized, and loads the first word into WP and the second word into URB. Concurrently, the AU saves the previous WP value by transferring it to URA. The AU then stores the old program parameters, WP, (now in URA) PC and status register into registers 13, 14 and 15 of the new workspace. Having saved the old program, the AU enters the service routine starting point (now in URB) into PC and ADC to begin the service routine. No interrupts are permitted to disturb the initiation of the service routine until the first instruction has been executed, so that the program parameters are firmly established.

2.2.13.10 INTERRUPT ROUTINES. When the service routine begins, the AU forces the interrupt mask to a value that is one less than the level of the interrupt being serviced. This allows only interrupts of higher priority to interrupt a service routine. If a higher priority interrupt occurs, a second context switch ensues to begin servicing the higher priority interrupt. When that routine is complete, a return instruction (RTWP) restores the first service routine parameters to the AU to complete processing of the lower priority interrupt. All interrupt subroutines should terminate with some return instruction that restores program parameters to continue operation.

2.2.14 ARITHMETIC UNIT CLOCK

The AU clock signal is a 60 nanosecond, low active pulse that regulates all activities within the AU. When the AU is processing data internally, the clock pulse occurs with a period of 260 nanoseconds (3.8 MHz). However, when the AU is processing data using TILINE read cycles, the clock becomes dependent upon the response from the TILINE slave device (memory complete). Under these conditions, the period is never less than 260 nanoseconds, but it may be greater depending upon the speed of the memory device. This allows the AU to wait for the required instruction from memory through the asynchronous TILINE interface.

2.3 TILINE

The Model 990 Computer employs a high-speed, bidirectional data bus, termed the TILINE, for data exchange between the AU, memory and other rapid transfer peripheral devices connected to the computer. The TILINE bus operates asynchronously, so that the interaction rate of specific devices is the only factor limiting TILINE's transfer speed. Each TILINE device appears to the AU as if it were additional memory, since device interfaces are addressed using the same address lines as those used for memory. The similarity of function for each TILINE device allows standardization of the controller-to-TILINE interface to greatly reduce design effort for new interface controllers (refer to Appendix E of this manual for sample interface diagrams). TILINE, therefore, ensures maximum transfer rate with minimum interface complications.

2.3.1 MASTER-SLAVE CONCEPT

TILINE interfaces may assume one of two roles during a data transfer: master or slave. The master device controls the data transfer whether sending or receiving data. The slave interface performs the actions required by the master interface to effect the data transfer and acts only under the direction of some master interface. The AU is an example of a master device that generates addresses and the required control signals to perform both data storage and retrieval operations. Memory, however, is always a slave interface that is incapable of initiating a data transfer and can only respond to requests by the AU or other master devices. Peripheral device interfaces may be solely master, always slave, or more typically, both master and slave interfaces. In the latter case, the slave interface allows the AU to establish transfer parameters in the master interface registers. The AU then relinquishes control of the TILINE to the master interface, freeing the AU for other operations while the peripheral master interface performs a block transfer.

2.3.2 INTERFACE SIGNALS

Forty-seven signal lines perform the addressing, data transfer and control functions of the TILINE data bus. Each device connected to the bus monitors these signals and reacts when it receives a command addressed to it. Figure 2-7 illustrates and table 2-3 defines the TILINE interface signals and their assigned connector pin numbers within the computer backplane.

2.3.3 TILINE PRIORITY

The TILINE interface resolves conflicts for access to the data bus through a positional priority system. The circuit board in chassis location A11 receives the highest priority. Priority ranking decreases with each chassis location through A02 (the AU circuit board), which is the lowest priority position. The power supply circuit board (location A01) receives no ranking,

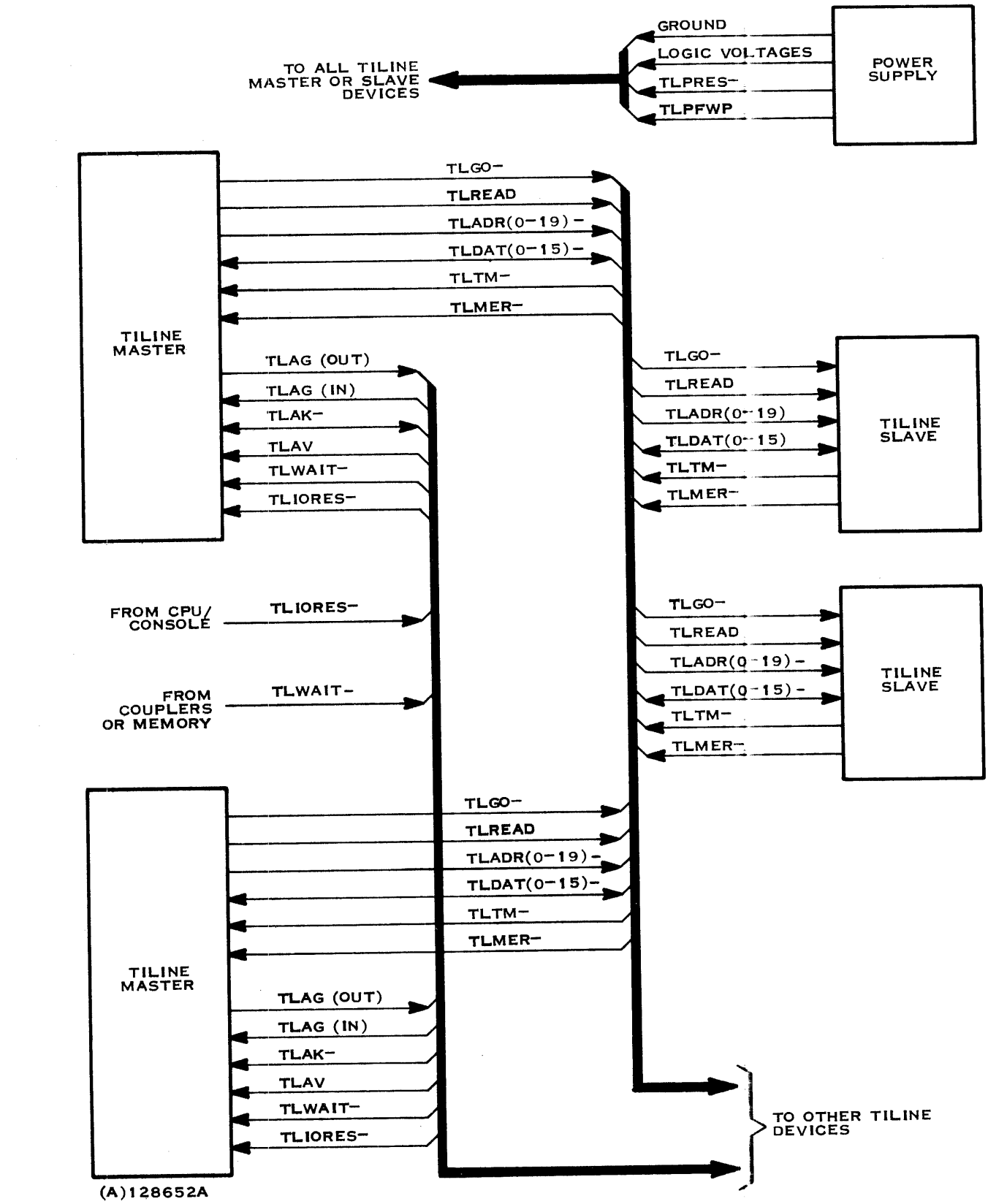


Figure 2-7. TILINE Interface Signals

Table 2-3. TILINE Signal Definitions

Signature	Pin No.	Definition
TLGO-	P1-25	TILINE Go: Initiates all data transfers when transition from high (3.0V) to low (1.0V) occurs.
TLREAD	P1-11	TILINE Read: When high (3.0V) designates a read from slave operation; when low (1.0V) designates a write to slave operation.
TLADR00-	P2-55	TILINE Address to define the location of data during a fetch or store operation. When high (3.0V) the corresponding address bit is a zero; when low (1.0V) the corresponding address bit is a one.
01-	P2-44	
02-	P2-51	
03-	P2-53	
04-	P2-57	
05-	P2-59	
06-	P2-47	
07-	P2-49	
08-	P2-17	
09-	P2-19	
10-	P2-10	
11-	P2-12	
12-	P2-11	
13-	P2-15	
14-	P2-8	
15-	P2-9	
16-	P2-29	
17-	P2-27	
18-	P2-25	
TLADR19-	P2-31	
TLDAT00-	P2-67	TILINE Data: Bidirectional data lines that when high (3.0V) represent zero data bits, and when low (1.0V) represent one data bits.
01-	P2-69	
02-	P2-35	
03-	P2-37	
04-	P2-61	
05-	P2-63	
06-	P2-43	
07-	P2-45	
08-	P2-21	
09-	P2-33	
10-	P2-23	
11-	P2-20	
12-	P1-27	
13-	P1-28	
14-	P1-30	
TLDAT15-	P1-31	

Table 2-3. TILINE Signal Definitions (Continued)

Signature	Pin No.	Definition
TLTM-	P1-20	TILINE Terminate: When low (1.0V) indicates that the slave device has completed the requested operation.
TLMER-	P1-55	TILINE Memory Error: When low (1.0V) indicates that a nonrecoverable error has occurred during a memory read operation.
TLAG (in)	P2-6	TILINE Access Granted: When high (2.4V), this signal indicates that no higher priority device has requested use of the TILINE. When low (0.4V), this signal prevents the receiving device from using the TILINE bus.
TLAG (out)	P2-5	TILINE Access Granted: When high (2.4V), this signal indicates that neither the sending device nor any higher priority device has requested use of the TILINE. When low (0.4V), this signal indicates that either the sending device or some higher priority device has requested use of the TILINE bus and prevents all lower priority devices from using the bus.
TLAK-	P1-71	TILINE Acknowledge: When high (3.0V), this signal indicates that no TILINE device has been recognized as the next device to use the TILINE. When low (1.0V), this signal indicates that some TILINE device has requested access, has been recognized, and is waiting for the bus to become available.
TLAV	P1-58	TILINE Available: When high (3.0V), this signal indicates that no TILINE device is using the bus. When low (1.0V), this signal indicates that the TILINE bus is busy.
TLWAIT-	P1-63	TILINE Wait: A normally high (3.0V) signal that when low (1.0V), temporarily suspends all TILINE master devices from using the TILINE bus. This signal is generated by bus couplers to allow them to use the bus as the highest priority user.

Table 2-3. TILINE Signal Definitions (Continued)

Signature	Pin No.	Definition
TLIORES-	P1-14 P2-14	TILINE I/O Reset. A normally high (3.0V) signal that when low (1.0V), halts and resets all TILINE I/O devices. This signal is a 250 nanosecond pulse generated by the RESET switch on the control console or by the execution of a Reset (RSET) instruction in the AU.
TLPRES-	P1-13 P2-13	TILINE Power Reset: A normally high (2.4V) signal that goes low (0.4V) to reset all TILINE devices and inhibit critical lines to external equipment. The signal is generated by the power supply at least 10 microseconds before dc voltages begin to fail during power-down, and until dc voltages are stable during power-up.
TLPFWP	P1-16 P2-16	TILINE Power Failure Warning Pulse: A 1.0 millisecond pulse preceding -TLPRES. When high (3.0V), this signal indicates that a power-down sequence is in progress, allowing the AU to perform its power failure interrupt subroutine.

since it is not a TILINE user. Only master devices can initiate TILINE requests. Therefore, only master devices participate in the priority circuits.

2.3.4 PRIORITY DETERMINATION

Figure 2-8 illustrates the connections between TILINE master controllers that establish the priority system. When a controller requires access to the bus, it examines Access Granted (TLAG) to determine if a higher priority device has also requested access. Access Granted establishes priority ranking within the chassis by passing through each circuit board in series. This arrangement allows a circuit board to reserve the TILINE bus by disabling TLAG to the lower priority circuits in the chassis if no higher priority device has already reserved it. Although slave interfaces do not participate in priority determination, they must transfer the TLAG signal from the input pin to the output pin in their connector location. Access Granted also enables the controller to issue a request. If Access Granted is high and the Acknowledge (TLAK-) signal is also high, the controller may reserve the next transfer period on the bus by pulling TLAK- low. The controller then has exclusive rights to the next transfer period and may not be removed from this position by a request of any priority. When the controller gains control of

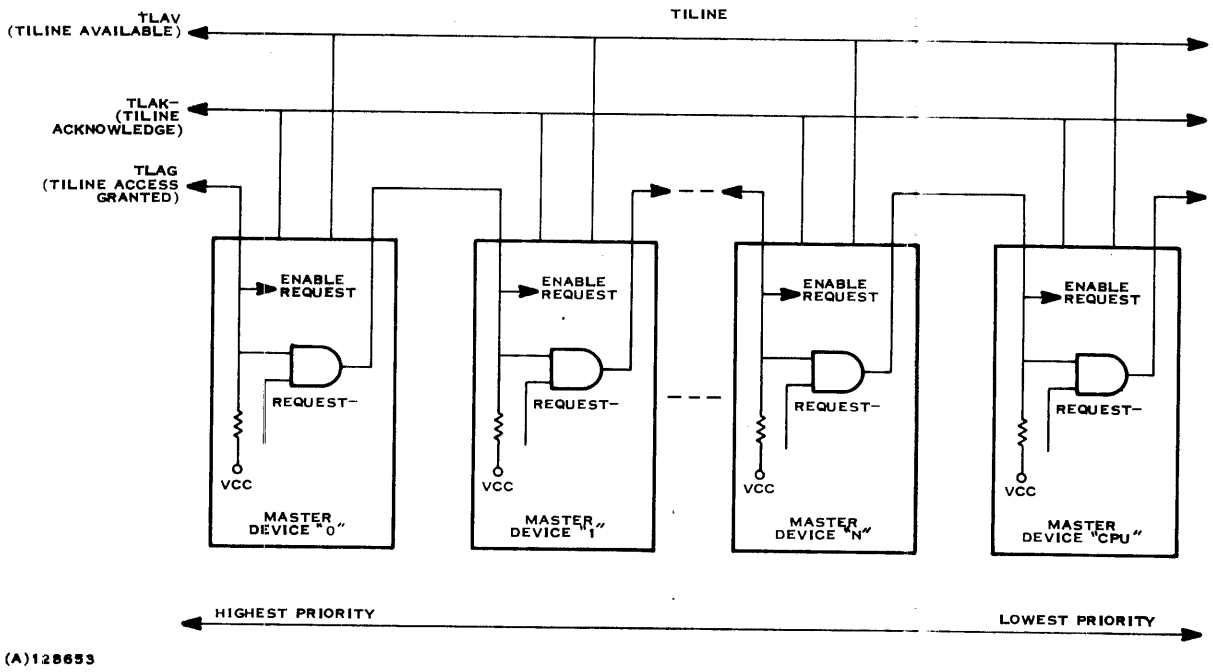


Figure 2-8. TILINE Priority Connections

the bus (after the current user relinquishes it), the controller pulls TLAV low to indicate a busy condition until it finishes its transfer. At that time it also enables TLAG to other master controllers to allow them to vie for the next access to the bus. TILINE masters other than the AU usually perform one memory cycle and then relinquish TILINE access. However, certain master interfaces may maintain TILINE access for several memory cycles. The AU maintains TILINE access unless another master device wants access. When Access Granted goes low at the AU input, the AU relinquishes TILINE access until Access Granted goes high following completion of the TLAK- and TLAV sequence. Figure 2-9 illustrates the timing relationships of the signals during a TILINE request.

2.3.5 TILINE WRITE CYCLE

A TILINE Write operation transfers 16 bits of data from a master device to a slave device. The timing of this operation is asynchronous and depends solely on the response time of the respective device controllers. Figure 2-10 illustrates the relationship of the signals involved in the write operation.

2.3.5.1 WRITE INITIATION. To begin a write operation, a master controller generates a 16-bit data word (TLDAT-) to be stored in a slave device, produces a 20-bit address (TLADR-) to select the proper slave device, drops the read signal (TLREAD) to indicate a write operation, and indicates to the slave devices that an operation is beginning by activating the go signal

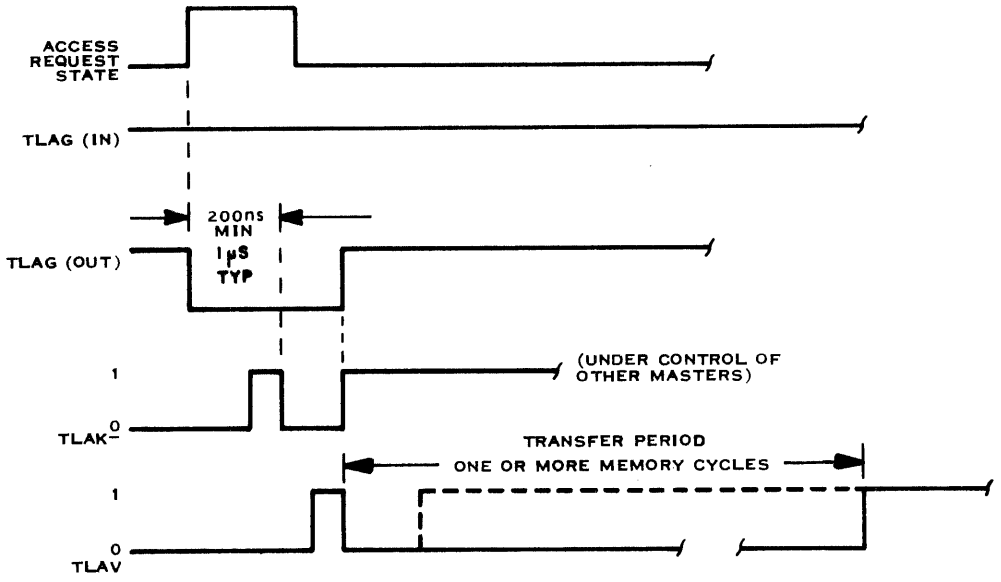


Figure 2-9. TILINE Access Request Timing

(TLGO-). All slave devices receive the go signal (after a transmission delay) and decode the address lines to determine which device is selected. To ensure that the address lines are valid, the slave device delays the effect of the go signal within its circuitry to allow for worst cases of address transmission skew (50 nanoseconds typical) and propagation delay during address decode (dependent on circuitry: 50 nanoseconds typical). The selected slave then transfers the data, address and write indication from the TILINE bus into registers, and generates a terminate signal (TLTM-) to the master controller.

2.3.5.2 WRITE TERMINATION. When the master controller receives the terminate signal from the slave, it releases those signals that it produced to initiate the transfer (TLGO-, TLREAD, TLADR-, and TLDAT-). Dropping TLGO- to the slave causes it to drop its terminate signal, which releases the master controller to begin another read or write cycle or allow access to another master controller.

2.3.6 TILINE READ CYCLE

A TILINE read operation transfers 16 bits of data from a slave device to a master device. The timing of this operation is asynchronous and depends solely on the response time of the respective device controllers. Figure 2-11 illustrates the relationship of the signals involved in the read operation.

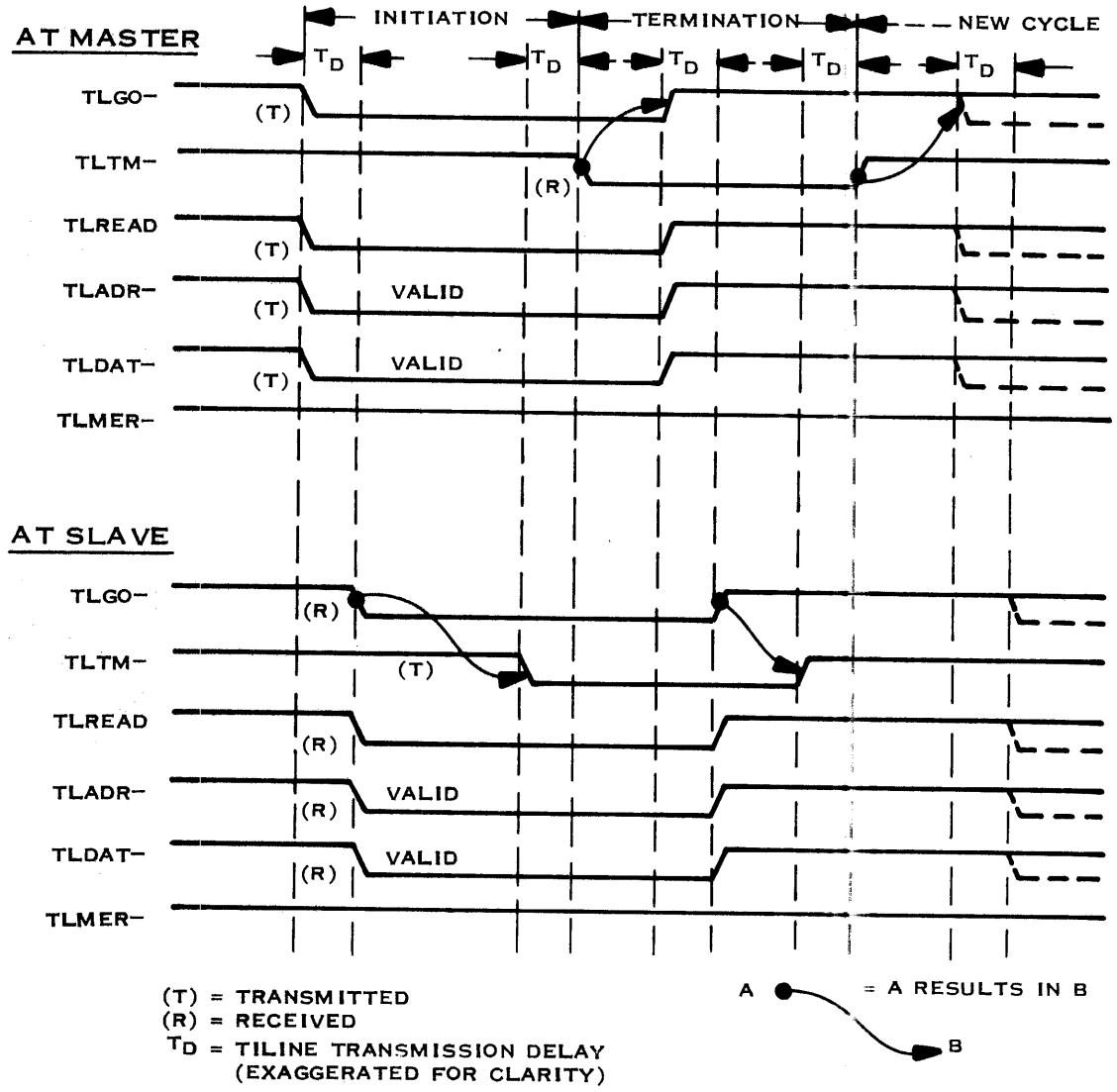


Figure 2-10. TILINE Write Cycle Timing

2.3.6.1 READ INITIATION. To begin a read operation, the master controller generates a 20-bit address to select the slave device, holds the read signal (TLREAD) high to indicate a read operation, and generates a go signal (TLGO-). All slave devices receive the go signal and decode the 20 address bits to determine which of the slave devices has been designated. To allow for transmission skew and address decoding time, the slave device delays the go signal internally so that the address is stable when the decoder results are checked. The selected slave device then initiates a read cycle to produce data for the master device.

2.3.6.2 READ TRANSFER. When the slave device has completed its read cycle and the data is stable on the TILINE, the slave controller issues a termination indication (TLTM-) to the master device. If a read error was

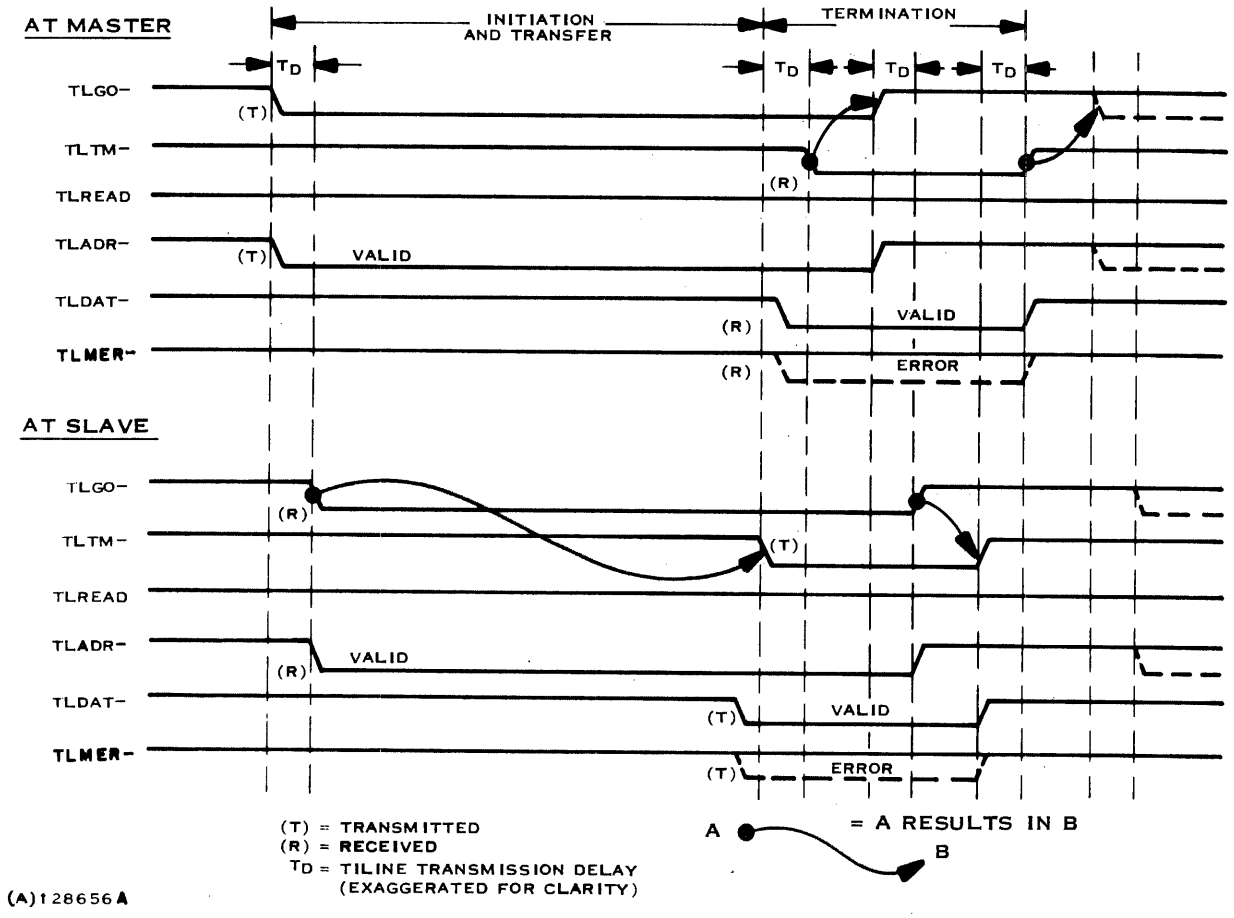


Figure 2-11. TILINE Read Cycle Timing

detected during the read cycle, the slave generates a read error signal (TLMER-) concurrent with the termination signal, and with the same timing that the read data lines would have.

2.3.6.3 READ TERMINATION. When the master device receives the termination signal from the slave device, it allows for worst case transmission skew. The master then captures read data into a register and disables the go indication and the address lines. The slave responds to the loss of TLGO by dropping the termination indication and the data on the read lines. Dropping terminate allows the master to issue another data transfer operation or relinquish the data bus to another master device.

2.3.7 TILINE TIME OUT

If a TILINE master has access to the bus and does not transfer data within 10 microseconds, or if the master addresses a nonexistent slave device, the master generates an internal time out signal. The time out function automatically returns the master to an idle state to initiate its access once more.

2.3.8 DESIGN CHARACTERISTICS

Figure 2-12 illustrates the termination circuits required to stabilize the signals on the TILINE. Table 2-4 presents other design characteristics for the bus. Refer to Appendix E of this manual for sample interface circuits of both a master and slave TILINE device.

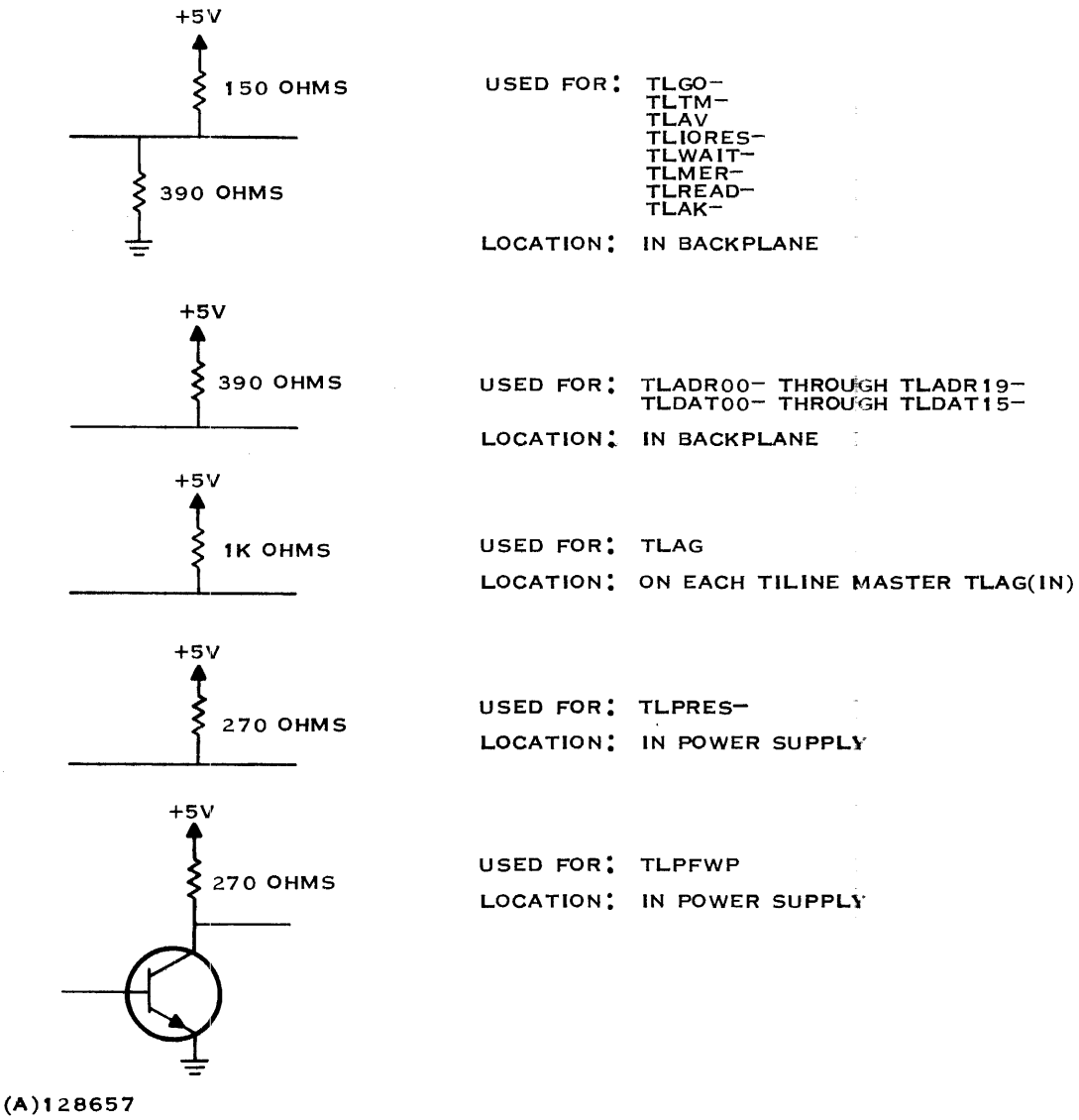


Figure 2-12. TILINE Termination Circuits

Table 2-4. TILINE Design Characteristics

Characteristic	Requirement
Length of signal path	20 inches (max) including routing on circuit boards.
Propagation delay	Less than signal rise and fall times.
Location of termination resistors.	Any place along the motherboard signal path.
Driver/receiver specification	
Circuit type	SN 75138 (typical)
Driver circuit	Open collector sinking 50 ma
Receiver circuit	Less than 50 microamps input current (typical) 2.0V to 3.0V input threshold
Control to data or address skew	50 nanoseconds maximum
Master access switching time	Approximately 60 nanoseconds

2.4 HARDWARE XOP INTERFACE

To reduce programming complexity, the Model 990 Computer offers an interface that allows more efficient hardware modules to perform complex arithmetic and logical operations while the AU waits for the result. This extended operation (XOP) feature is an integral component of the software XOP instruction. When the AU reads an XOP instruction, it checks first to determine if the hardware module for that function is included in the chassis. Only if the hardware module is not included does the AU call the software subroutine for that function from memory. The following paragraphs describe the operation of the XOP interface as an aid to the design of specialized modules and as a guide to the capabilities of the computer.

2.4.1 XOP INTERFACE SIGNALS

The hardware XOP interface employs six exclusive signals, plus five signal types used by TILINE and other devices, to control actions of an XOP circuit board. Figure 2-13 illustrates the interface lines connecting the XOP module to the AU. Table 2-5 defines these signals and lists the AU circuit board pin numbers for each signal. XOP signals are wired throughout the computer chassis, so that if an XOP circuit board is to be used, it may be inserted into any TILINE connector location in the chassis. Refer to the connector diagrams in Appendix G of this manual to determine the pin assignments of these signals.

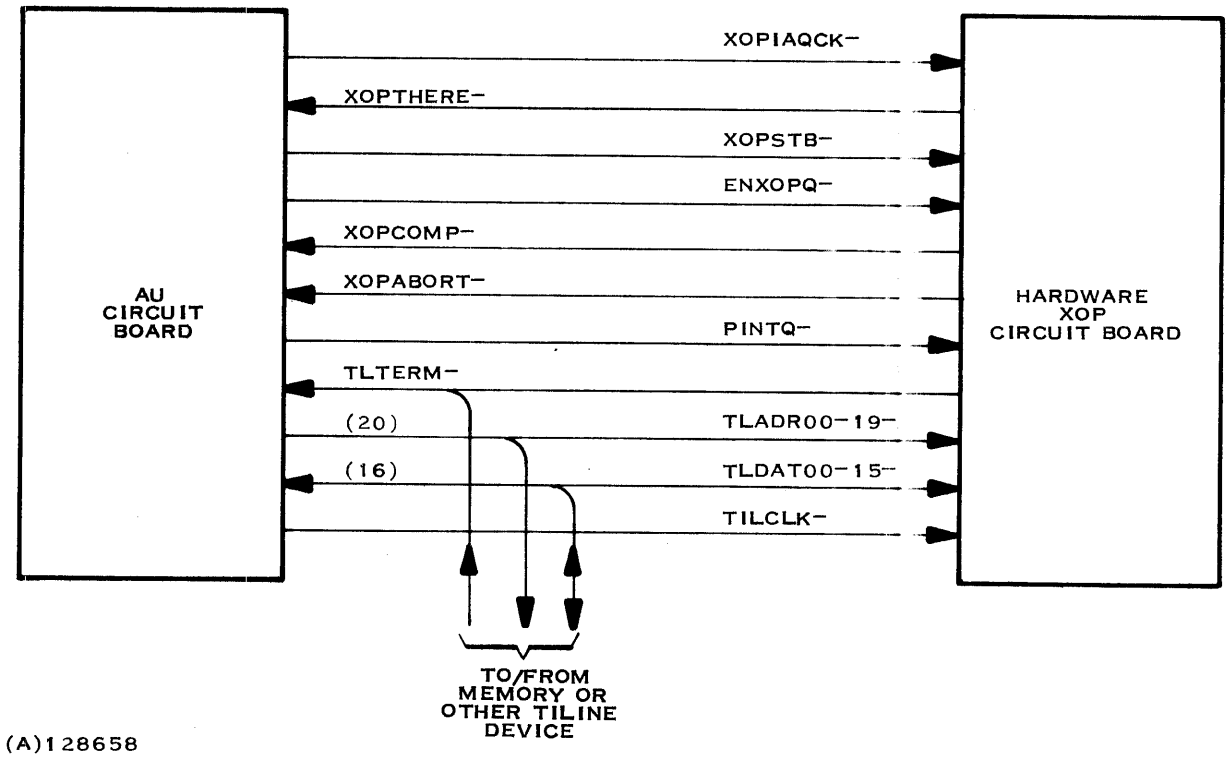


Figure 2-13. Hardware XOP Interface Signals

2.4.2 HARDWARE XOP OPERATION

Each XOP module processes a set of data to produce an end product that is either stored in memory as data, reported to the AU through status bits, or a combination of the two methods. In addition, the module continuously monitors the instructions sent to the AU when the module is not active. An extended operation cycle begins when the AU reads an XOP instruction from memory. Figure 2-14 illustrates the timing of interface signals during the entire XOP cycle using a hardware module. The following pargraphs describe the processes that occur during the operation.

2.4.2.1 INSTRUCTION MONITORING AND ACQUISITION. Each time that the AU fetches a new instruction from memory and receives the termination indication from memory, the AU produces a clock pulse (XOPIAQCK-) to the XOP module, allowing the module to capture the op code and D field portions of the instruction from the TILINE data bus. The module then decodes that portion of the instruction to determine if the module can process that instruction. If the module cannot perform the requested function, the module does nothing and waits for another instruction to be entered. However, if the module can perform the operation, it activates XOPTHERE- to indicate to the AU that the hardware module is present and active. This signal must be active within 200 nanoseconds of the instruction acquisition clock and must remain active until the AU recognizes its presence by transmitting the operand address to the XOP module.

Table 2-5. Hardware XOP Interface Signals

Signature	Pin No.	Definition
XOPIAQCK-	P2-43	XOP Instruction Acquisition Clock: a 60 nanosecond low active pulse that enables the XOP circuit boards to capture the data currently on the TILINE data lines. That data is an instruction to be inspected by the XOP module.
XOPTHERE-	P2-29	XOP Hardware Present: a low active signal generated by the XOP module within 200 nanoseconds following XOPIAQCK-. This signal indicates to the AU that a hardware module is available to perform the indicated XOP function and that no software subroutine need be started. This signal must be present until acknowledged by XOPSTB-.
XOPSTB-	P2-30	XOP Strobe: a low active enable signal generated by the AU to the XOP board at least 500 nanoseconds following XOPIAQCK-. This signal indicates to the XOP module that the effective address has been calculated and is available on the TLADR lines.
ENXOPQ-	P2-26	Enable XOP: a low active signal generated by the AU to allow the XOP module to begin processing. The signal remains active during the entire process and is cleared when the AU receives status at the completion of the XOP.
XOPCOMP-	P2-53	XOP Complete: a low active signal that is issued by the XOP module to indicate that it has successfully completed the required operation. While this signal is active, the TLDAT lines contain status bits from the completion of the operation for transfer to the AU status register.
XOPABORT-	P2-54	XOP Aborted: a low active signal from the XOP module to indicate that an interrupt from the AU has terminated the operation before the operation was complete. While this signal is active, the

Table 2-5. Hardware XOP Interface Signals (Continued)

Signature	Pin No.	Definition
XOPABORT- (Cont.)	P2-54	TLDAT lines contain status bits from the module to indicate the condition at time of termination.
PINTQ-	P2-25	Processor Interrupt: a low active signal from the AU that, if implemented, halts an extended operation in progress in an XOP module. The module then returns an XOPABORT- signal to the AU.
TLTERM-	P1-14	TILINE Terminate: a low active signal from any TILINE device or the XOP module to indicate the end of a particular data transfer. The XOP module uses TLTERM- to indicate receipt of the effective address from the AU.
TLADR00- to TLADR19-	See TILINE description	TILINE Address: a 20-bit address from the AU to the XOP module that designates the starting address of the data for the extended operation. This address transfers to the XOP module at the start of the XOP routine. The circuit board must store the address until completion of the routine. "1" = 1.0V; "0" = 3.0V
TLDAT00- to TLDAT15-	See TILINE description	TILINE Data: bidirectional data lines that carry the XOP instruction to the XOP module, the XOP result from the module, and the status bits from the module to the AU. "1" = 1.0V; "0" = 3.0V

2.4.2.2 ADDRESS TRANSFER. The XOP instruction may use the TS field to modify the source address for the XOP data. In this case, the S field of the instruction does not contain the true, effective address of the XOP data, and the AU must modify that value before it can be used by the XOP module. For this reason the XOP module does not capture the TS and S fields during the instruction acquisition phase. Instead, the module holds XOPTHERE- active until the AU has determined the effective address. At that point (500 nanoseconds minimum delay), the AU issues XOPSTB- and places the effective address on the TILINE address lines. The address remains on the lines

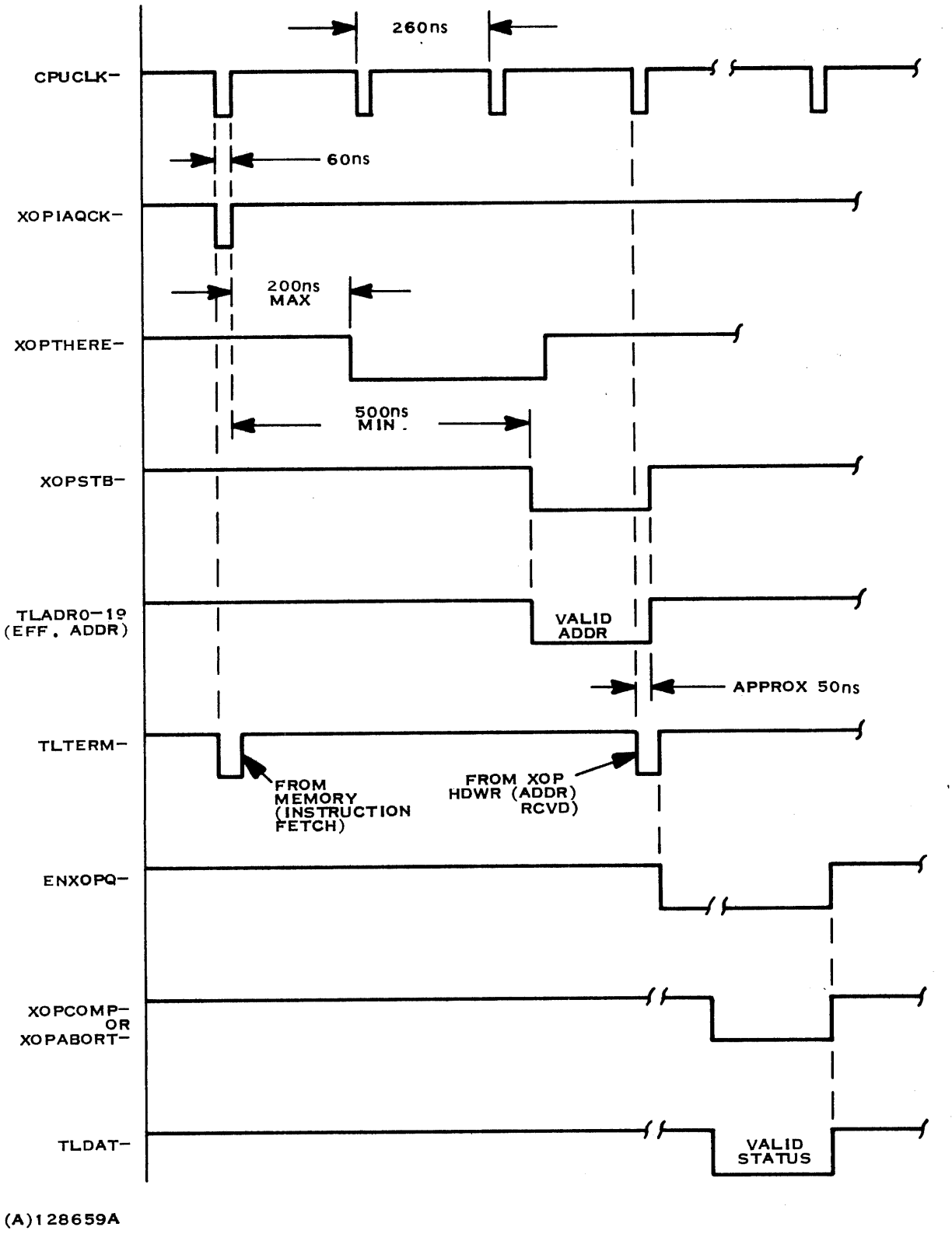


Figure 2-14. XOP Interface Timing Diagram

and the strobe remains active until the XOP module returns TLTERM- to indicate that it has captured the address in its address register. The AU's time out function clears the address and strobe lines if the XOP module fails to respond.

2.4.2.3 OPERATION EXECUTION. When the AU has received TLTERM- from the XOP module, it generates an enable signal to the module (ENXOPQ-) to initiate execution of the prescribed operation. This signal remains active until the module completes or terminates the operation. The hardware within the module must be able to function independently to fetch operands from memory, perform the required manipulations, and return the operands to memory in the location specified by the effective address stored in its address register. Operand size and length of operation are a function of the XOP module design.

2.4.2.4 OPERATION COMPLETION. When the XOP module has completed its operation and stored the results in the specified memory location, it issues a signal to the AU to indicate that the operation has completed normally (XOPCOMP-), and places status information on the TILINE data lines. When the AU has received the status information, it loads it into the status register for program monitoring, and responds to the XOP module by deactivating ENXOPQ-. The XOP module then removes the XOPCOMP- indicator and status information from the interface lines. The extended operation is complete.

2.4.2.5 OPERATION ABORTED. The XOP module may respond to an AU interrupt by terminating the extended operation before completion. The AU generates a processor interrupt signal (PINTQ-) when it receives an interrupt from one of the peripheral TILINE or CRU devices. Since the operation being performed in the XOP module may continue for a long time, the user may choose to terminate that operation so that the AU may service the interrupt. If the XOP module is designed to accept a processor interrupt, it terminates the operation in progress and returns XOPABORT- to the AU. Concurrent with the abort signal, the XOP module places its status information on the TILINE data bus for sampling by the AU. After the AU stores the status in the status register, it drops ENXOPQ- to the XOP module to end the operation. If the extended operation is terminated abnormally, the AU does not increment the address in the program counter, so that when the AU returns from servicing the interrupt, it initiates the extended operation again.

2.5 MEMORY

The 990 Computer offers high-speed metal oxide semiconductor (MOS) memory in increments of 4096 words from a minimum configuration containing 4096 words to a maximum capacity of 32,768 words. The memory controller interfaces with the TILINE as a slave device, and in addition to memory control logic contains logic that corrects single-bit errors in a word read from

memory, and detects multi-bit errors. The storage word is 22 bits long. Sixteen bits are data bits and the remaining six bits contain the error correcting code used to ensure data accuracy. Memory may be implemented on one or two double-connector circuit boards depending upon the size of memory required. The memory controller board can provide up to 8K words of memory. If larger configurations of memory are required, an expansion memory board, connected to the memory controller board through a top edge connector and cable, provides up to 24K additional words of memory. Both types of memory boards contain two light-emitting diodes (LED's) mounted on the outside edge of the circuit board to indicate an error on that board. In addition, the memory controller board contains two dual-in-line switch packages for disabling the memory correction feature and for selecting the starting address of the memory board.

2.5.1 MEMORY CHIP

The heart of the 990 memory is the TMS 4030 NL random access memory (RAM) integrated circuit. This device is a high-speed, MOS circuit providing a 1 x 4096 storage capacity in a package that is 1.2 inches long and 0.4 inches wide. Twelve address input lines allow selectability of any of the 4096 bits in the storage matrix for either a read or a write operation. The six most significant bits of the address are also used during refresh cycle to refresh 1/64th of the chip during a cycle. In addition to the standard 5 volt TTL logic levels, the chip requires +12 volts and -3 volts bias inputs and a +12 volt clock pulse to initiate the store or read operation. The circuitry within the chip decodes an address to select the storage location and executes its memory cycle in less than one microsecond.

2.5.2 CONTROLS AND INDICATORS

The 990 memory circuit board contains two LED indicators and two dual-in-line switch packages to aid in setting up and maintaining the system. Figure 2-15 illustrates the location of these components on the memory controller circuit board.

2.5.2.1 ERROR INDICATORS. Two light-emitting diodes, mounted next to the ejector tab on all memory circuit boards, light to indicate errors that occur during a memory cycle. The Correctable Error indicator lights when the error correcting logic detects and corrects an error in data read from memory. This indicator is not operational if the error correcting logic is disabled. The Non-correctable Error indicator lights when the error detecting logic senses a data error that either cannot be corrected (2 or more bits in error), or is not corrected because the correcting logic has been disabled.

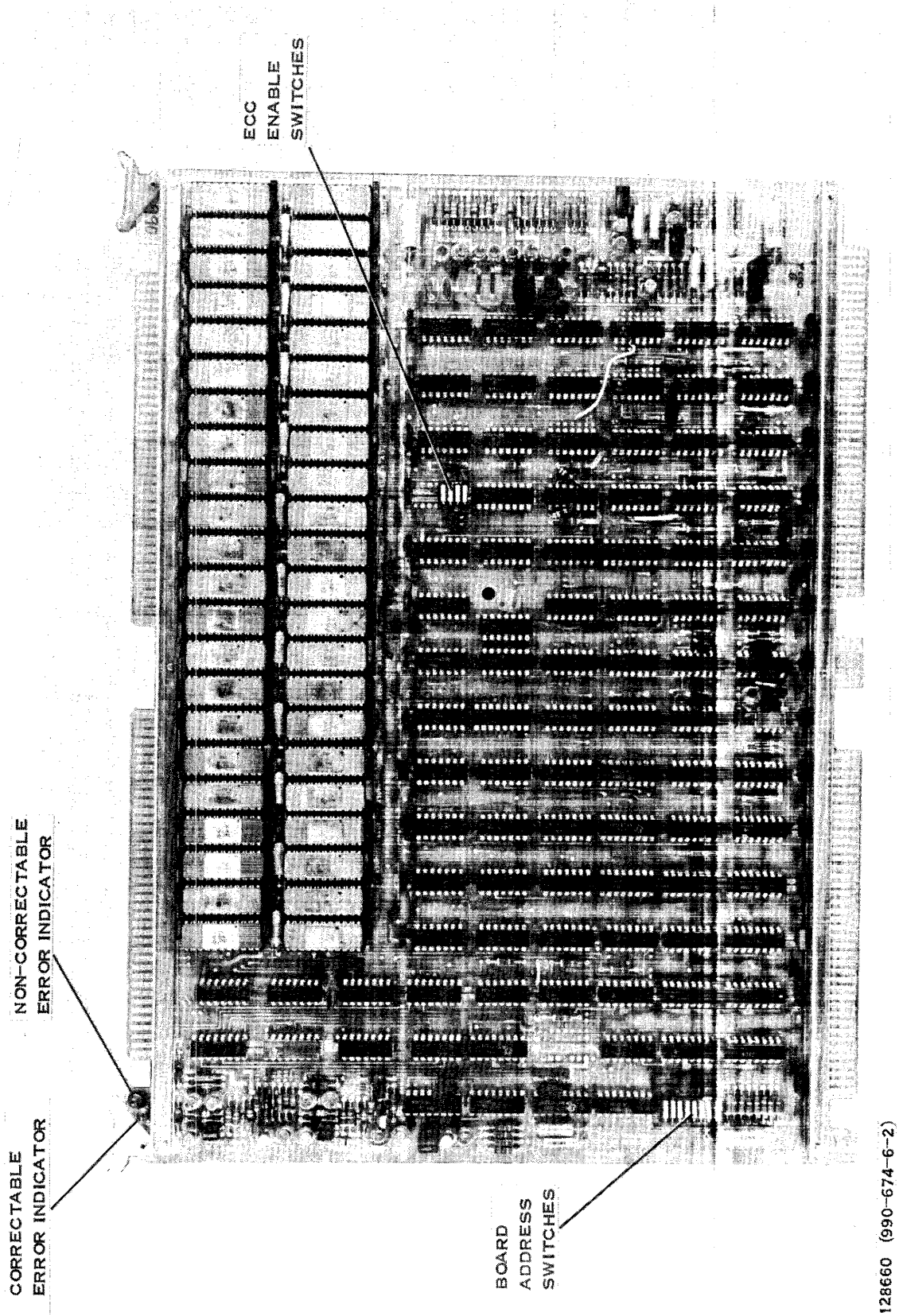


Figure 2-14. Memory Controller Controls and Indicators

2.5.2.2 ECC ENABLE SWITCHES. A dual-in-line package (DIP) containing four single-pole, single-throw switches allows the user to select or disable the error correcting logic. Switches 3 and 4 in this package perform no function. Switches 1 and 2 enable the error correcting logic when both switches are set to the ON position. These switches disable the error correcting logic when both switches are set to the OFF position. In no case should one switch be ON while the other switch is OFF, since this condition produces erroneous indications to the memory controller.

2.5.2.3 BOARD ADDRESS SWITCHES. A dual-in-line package containing eight single-pole, single-throw switches allows the user to set the starting address of the memory locations within the control of a particular memory controller board. The address switches correspond to the eight most significant bits of the 20-bit TILINE address, and allow board address selection in 4K-word increments. Switch 1 is the most significant bit of the address; switch 8 is the least significant bit of the address group. All Model 990 Computer systems must have memory locations from 0 to 4096, since the ROM loader for the computer is always loaded into the lowest 4K of memory. Therefore, in systems containing only one memory controller board, the board address switches must all be OFF. Table 2-6 lists the required switch settings for some memory board addresses up to 64K. Addresses greater than 64K can be represented in a similar manner using the eight switches to represent the binary number desired.

2.5.3 MEMORY INTERFACE

The memory controller interfaces with the TILINE as a slave device. As such it conforms to all signal specifications and timing requirements previously defined for the TILINE slave interface. In addition to the TILINE

Table 2-6. Memory Board Address Settings

TILINE Address$_{16}$	Address Switch Setting					Addresses on Board (4K)
	1-4	5	6	7	8	
00000	OFF	OFF	OFF	OFF	OFF	0 to 4095
01000	OFF	OFF	OFF	OFF	ON	4096 to 8191
02000	OFF	OFF	OFF	ON	OFF	8192 to 12,287
03000	OFF	OFF	OFF	ON	ON	12,288 to 16,383
04000	OFF	OFF	ON	OFF	OFF	16,384 to 20,479
05000	OFF	OFF	ON	OFF	ON	20,480 to 24,575
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
0F000	OFF	ON	ON	ON	ON	61,440 to 65,535

interface, the memory controller also interfaces with an additional memory expansion board if greater than 8K words of memory are required. The expansion board derives its voltage and ground levels from a 990 chassis connector, but receives data, address and control from the memory controller through a top edge connector and cable. Figure 2-16 illustrates the signals that interface the memory controller to the expansion board. Table 2-7 defines these signals and their connector pin assignments.

2.5.4 MEMORY CONTROLLER OPERATION

The memory controller monitors the TILINE, decodes addresses on the bus, and directs the memory to perform store, fetch and refresh cycles as required by a TILINE master device. Figure 2-17 illustrates the major components of the 990 Computer memory system. The following paragraphs describe the actions of the controller during major functions of the memory.

2.5.4.1 ADDRESS EXAMINATION. When the memory controller senses a TLGO- pulse on the TILINE, it inspects the address on the TLADR lines to determine if the operation is intended for a memory address under its control. To make this determination, the controller reads the lower bound address from the board address switches and compares that with the TILINE

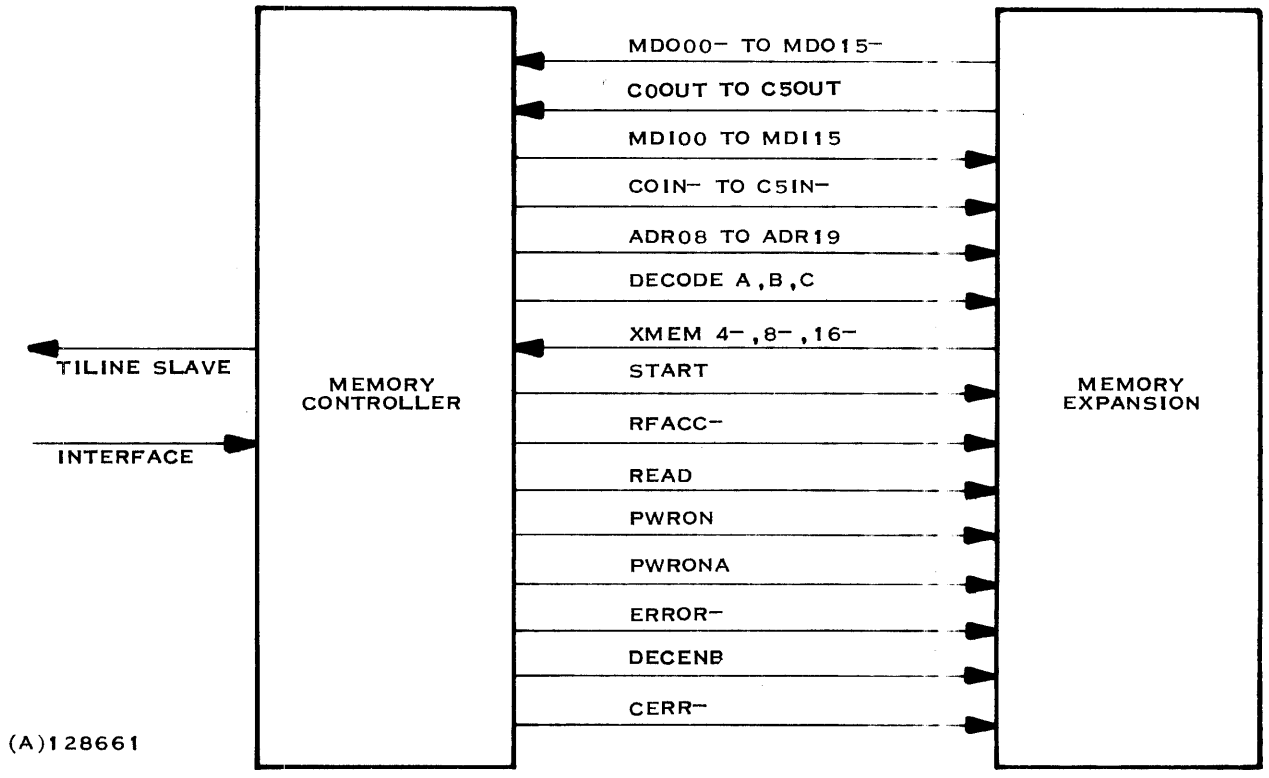


Figure 2-16. Memory Expansion to Memory Controller Interface

Table 2-7. Memory Controller to Memory Expansion Interface Signals

Signature	Pin No.	Definition
MD000-	P4-51	Memory read data output from memory expansion to memory controller.
MD001-	43	
MD002-	35	
MOD03-	27	
MD004-	19	
MD005-	13	
MD006-	09	
MD007-	05	
MD008-	P4-04	
MD009-	P3-75	
MD010-	71	
MD011-	63	
MD012-	55	
MD013-	47	
MD014-	39	
MD015-	P3-31	
C0OUT-	P3-23	Error correcting code outputs from memory expansion during read operation
C1OUT-	17	
C2OUT-	P3-13	
C3OUT-	09	
C4OUT-	05	
C5OUT-	P3-04	
MDI00	P4-53	Memory write data input from memory controller to memory expansion.
MDI01	45	
MDI02	37	
MDI03	29	
MDI04	21	
MDI05	15	
MDI06	11	
MDI07	07	
MDI08	P4-03	
MDI09	P3-77	
MDI10	73	
MDI11	65	
MDI12	57	
MDI13	49	
MDI14	41	
MDI15	P3-33	

Table 2-7. Memory Controller to Memory Expansion Interface Signals (Continued)

Signature	Pin No.	Definition
C0IN C1IN C2IN C3IN C4IN C5IN	P3-25 19 15 11 07 P3-03	Error correcting code inputs to memory expansion during write operation.
ADR08 ADR09 ADR10 ADR11 ADR12 ADR13 ADR14 ADR15 ADR16 ADR17 ADR18 ADR19	P4-67 61 69 73 77 75 57 55 59 65 63 P4-71	Least significant 12 bits of address from TILINE bus used as a store or fetch address when accessing a memory location. The most significant 6 bits of this address can also be generated internal to the controller for use during refresh cycles.
DECODEA DECODEB DECODEC	P4-78 P4-76 P4-74	A 3-bit code that indicates which bank (0-7) of memory chips is to be cycled. DECODEA is the least significant and DECODEC is the most significant bit of the code.
XMEM4- XMEM8- XMEM16-	P3-27 P3-29 P3-21	A 3-bit complement code, hardwired in the expansion board, that designates to the memory controller the size of the memory contained on the expansion memory board. Valid codes are: XMEM 16- 8- 4- 1 0 1 8K memory 0 1 1 16K memory 0 0 1 24K memory
START	P4-70	Initiates a memory cycle in the expansion board.
RFACC-	P4-68	When coincident with START, this signal initiates a refresh cycle in the memory expansion.
READ	P4-64	When a logic one, this signal indicates that a read cycle is to be performed from the address on the ADR__ lines. When this line is a logic zero, the expansion memory performs a write operation.

Table 2-7. Memory Controller to Memory Expansion Interface Signals (Continued)

Signature	Pin No.	Definition
PWRON	P4-64	When a logic one, this signal applies +5 Vdc to the memory expansion logic; when a logic zero, this signal removes power from the expansion logic.
PWRONA	P3-37	This signal is a logic zero during power on or off transitions and disables the clock input to the memory chips to prevent voltage spikes from affecting the memory chips. During normal power conditions, this signal is a logic one.
ERROR-	P3-67	A low active signal that indicates a non-correctable error in data from the memory expansion board and lights an LED on the memory expansion board to indicate that condition.
CERR-	P3-43	A low active signal that indicates a corrected error in data from the memory expansion board. This signal also lights an LED on the memory expansion board to indicate that condition.
DECENB	P4-72	A high active signal that enables memory expansion to decode the DECODEA, B, C lines to select a 4K bank of memory on the expansion board.

address to determine if the TILINE address is greater than the lower bound address. If the TILINE address is greater, the controller determines the size of its memory. The memory size is hard-wired into each memory board. If a memory expansion board is included, the controller must monitor the XMEM bits to determine the size of the expansion memory. The controller then adds these two figures to the lower bound address and compares the result with the TILINE address to determine if the TILINE address is less than the calculated upper bound address. If the TILINE address is within the specified bounds, the controller initiates the memory cycle requested. The 12 least significant bits of the TILINE address are sent directly to the memory chips as the address inputs. The controller decodes the eight most significant bits of the address to generate a bank select signal to designate one of the 4K-word banks of memory chips to receive the 12-bit address.

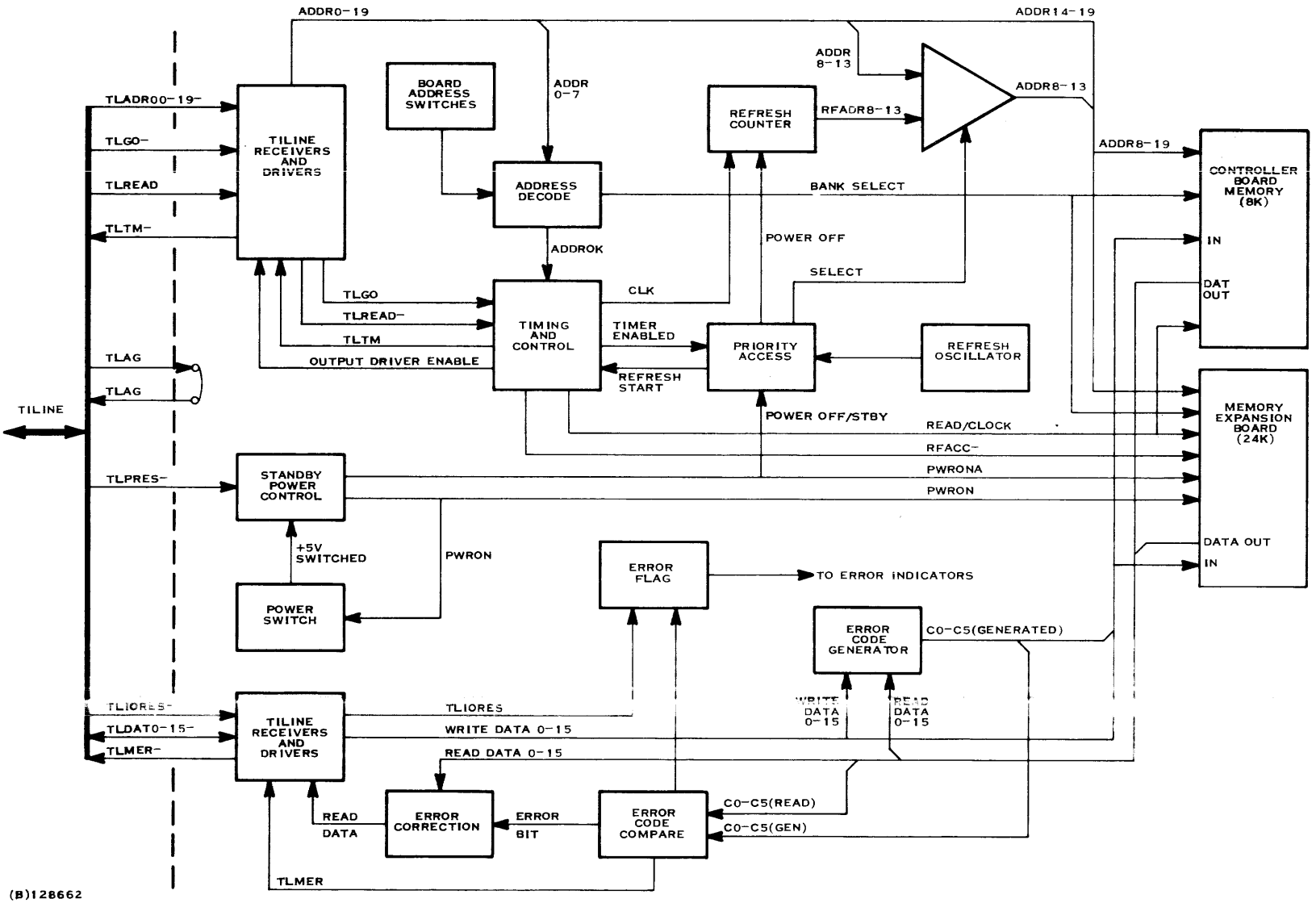


Figure 2-17. Memory System Block Diagram

2.5.4.2 WRITE CYCLE. If the TLREAD- signal line is high, the controller initiates a write cycle by holding the READ line low to the memory banks. The controller then passes the address from the TLADR lines to the inputs of the selected memory bank, and transfers the data bits from the TILINE to the memory chips for storage. The 16 data bits also pass through the error code generation logic. This circuit produces a 6-bit error detection and correction code that is stored in the six least significant bits of the 22-bit memory word.

2.5.4.3 ERROR CORRECTING CODE. The error code generation logic produces a 6-bit, modified hamming code that identifies the bit in error for 1-bit errors occurring in the 16-bit data word. Each bit in the code is an odd parity bit for selected bits within the data word, such that each bit in the data word participates in either three or five of the parity bits. If an error occurs in data read from memory, then the code generated from the read data will not match the code stored with the data. Examination of the bits that vary isolates the bit in error in the data word. Data errors produce three or five mismatched bits. If only one bit is mismatched, the code bit is in error. Figure 2-18 illustrates the bit patterns for each of the error code bits.

2.5.4.4 READ CYCLE. If the TLREAD signal line is high after the controller has determined the address, the controller initiates a read cycle by raising the READ line to the memory chips. The memory responds with a 22-bit word that enters the error checking and correcting (ECC) logic of the controller. The ECC logic generates a new error code for the 16 data bits of the memory word and compares it with the 6-bit error code that was stored with the data bits. If the two codes correspond, no error has occurred and the data is placed on the TILINE. If the codes do not compare, an error has occurred. The controller then corrects the inaccurate bit if it is a single-bit error and places the corrected data on the TILINE. The data in memory is not corrected. The controller also energizes the correctable error LED on the memory board to indicate the occurrence of the error. If

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DATA BIT / CHECK BIT
							X	X	X	X	X	X*	X	X	X	C0
X		X		X		X	X	X		X		X*		X		C1
X	X			X	X			X	X			X*	X			C2
X	X	X	X					X	X	X	X					C3
X	X	X	X	X	X	X	X								X	C4
X			X		X	X		X			X		X	X	X	C5

*BITS C0, C1, AND C2 MISMATCHED INDICATES DATA BIT 3 IN ERROR.

(A)128663

Figure 2-18. Error Correcting Code Bit Patterns

the error detected involved more than one bit, the logic cannot correct the error. For an even number of bits in error, the controller activates TLMER- to the TILINE device to indicate the error, and energizes the non-correctable error LED on the memory board. An odd number of errors greater than 1, though uncorrectable, is treated identically to a single bit error. If the ECC logic is disabled, any error generates TLMER- and energizes the non-correctable error LED.

2.5.4.5 REFRESH CYCLE. The memory controller initiates one refresh cycle every 31 microseconds to reinforce the data in 1/64th of the storage area in memory. This process ensures that the entire memory is refreshed within approximately 2 milliseconds. A refresh cycle takes precedence over memory requests from TILINE users, but does not interrupt a cycle in progress. A 6-bit counter within the controller increments each time a cycle is performed. The output of this counter is used during the refresh cycle as the six most significant bits of the 12-bit address sent to the memory chips. When clocked, these address bits enable the controller to refresh 1/64th of each memory chip.

2.5.4.6 STANDBY. If main power begins to fail, the memory controller resets the refresh counter and, following completion of a memory cycle in progress, begins a series of 64 refresh cycles in succession to ensure that memory data is maintained. When the last refresh cycle is complete, the controller sets a timer and turns off power to all memory control logic to conserve battery power. The timer is temperature-dependent such that at higher temperatures the time span is shorter than at lower (room) temperatures. When the timer times-out, it switches power back to the control circuits, resetting the refresh counter. The controller then initiates another 64 memory refresh cycles, sets the timer at the completion of the refresh period, and shuts power off to the control logic. This process continues until main power returns. When main power returns, the memory control logic is off and the controller is not aware of the power. When the timer times-out, the controller performs 64 refresh cycles. However, TLPRES- from the power supply is now high, indicating that power has been restored. This signal prevents the controller from switching logic power off, so that the controller remains active and ready for the first memory request. The temperature-sensitive timer spaces the refresh cycles during standby to ensure that the memory is maintained without needless refresh cycles (decay rate of the memory is also temperature-dependent in a ratio similar to the timer). The refresh rate decreases from approximately 2 milliseconds at 70°C to about 20 milliseconds at room temperature to conserve the standby battery.

2.6 COMMUNICATIONS REGISTER UNIT (CRU) INTERFACE

The Communications Register Unit (CRU) interface is the direct command driven input/output interface for the Model 990 Computer. The interface provides up to 4096 directly addressable input bits and up to 4096 directly addressable output bits. Input and output operations can address each of the bits individually or in fields of from one to sixteen bits. The computer instructions that drive the CRU interface can set, reset or test any bit in the CRU array, or move data between memory and the CRU data fields.

The AU circuit board controls the interface data and control lines. These lines are available to all computer chassis locations except those used for the power supply and the AU circuit board. The AU decodes sixteen module select signals and supplies them to eight chassis locations for CRU modules. Each chassis location accommodates one double-connector circuit board or two single-connector circuit boards. If all available chassis locations contain 16-bit data modules, the maximum internal CRU bit capacity is achieved (256 bits input, 256 bits output). Through the use of external chassis, the maximum expansion of the CRU can be realized (4096 bits input, 4096 bits output).

2.6.1 CRU APPLICATIONS

Because of its extremely flexible data format, the CRU interface can be used effectively for a wide range of control and data transaction operations. These applications can be divided into two broad categories: those involving a single control bit transfer, and those requiring input or output of several data or status bits.

2.6.1.1 SINGLE-BIT OPERATIONS. Single-bit operations typically involve the computer sampling a status bit. When the status bit sets, the computer responds by setting a control bit or by transferring to a different set of instructions. This operation is exemplified by a communications interface that generates a single interrupt for one of several reasons: output complete, input complete, or line status change. The output or input complete requires a transfer to instructions that perform another output or input operation. A line status change might require the setting of a control output or the transfer to instructions that handle the change in other ways.

2.6.1.2 MULTIPLE-BIT OPERATIONS. Multiple-bit operations typically involve a data input device such as a keyboard or card reader, or an output device such as a display or card punch. An interrupt from the device causes the AU to perform an STCR instruction to read data from the CRU device and store it into memory. Similarly, to output data to the device the AU executes an LDCR instruction to fetch data from memory and transfer it to the CRU device.

2.6.2 INTERFACE SIGNALS

Figure 2-19 illustrates the signals required to pass data to and from the CRU modules. Certain signals from the TILINE interface are common to the CRU modules since they pertain to power and master clear functions that affect the entire computer system. Table 2-8 defines each of the signals for the CRU interface and lists the connector pin numbers for these signals within the computer chassis. All CRU signals appear on the same pin number for both P1 and P2 of each chassis location in the computer except those locations occupied by the AU or a power supply. This duplicity allows two single-connector CRU modules to use a single chassis location if the module select and interrupt lines have been individually wired for that configuration. Since the small card adapter may be used when designing single-connector circuit boards, the table also includes the pin numbers of the CRU signals at the output of the small card adapter.

2.6.3 INTERFACE TIMING

Figure 2-20 illustrates the timing relationships between data and control signals on the CRU within the computer chassis. All CRU modules must respond within the times illustrated. To ensure reliability of data to the CRU device, the module should sample the CRUBITOUT on the positive-going edge of the STORECLK- pulse.

2.6.4 CRU ADDRESSING

The AU issues a 12-bit address (CRUBIT4-15) to address up to 4096 individual bits. Each address may be used for two purposes: once for an output bit operation and once for an input bit operation; therefore the AU can select a total of 8192 input and output lines using the 12-bit address. The main computer chassis receives addresses for 256 individual bits. The remaining address capability selects CRU bits from external expansion chassis and is not currently used.

2.6.4.1 ADDRESS FORMAT. Figure 2-21 illustrates the field assignments for the 12-bit CRU address. The four least significant bits select one of sixteen possible bits from a particular CRU module. The next four bits select one of sixteen possible modules from a particular chassis. The four most significant bits identify the desired chassis. The chassis select bits are always zeros when addressing a module within the computer chassis. The AU decodes the module select bits internally to produce sixteen individual select signals (IMODSEL0- through IMODSEL15-) that are routed to connectors within the computer chassis (either pin 46 or 48). Therefore, the bit select field (CRUBIT12-15) is the only portion of the CRU address that the module actually uses. When enabled by its hard-wired module select line, the module must decode the bit select field to determine which of its bits is affected by the operation. A particular CRU circuit board may employ more

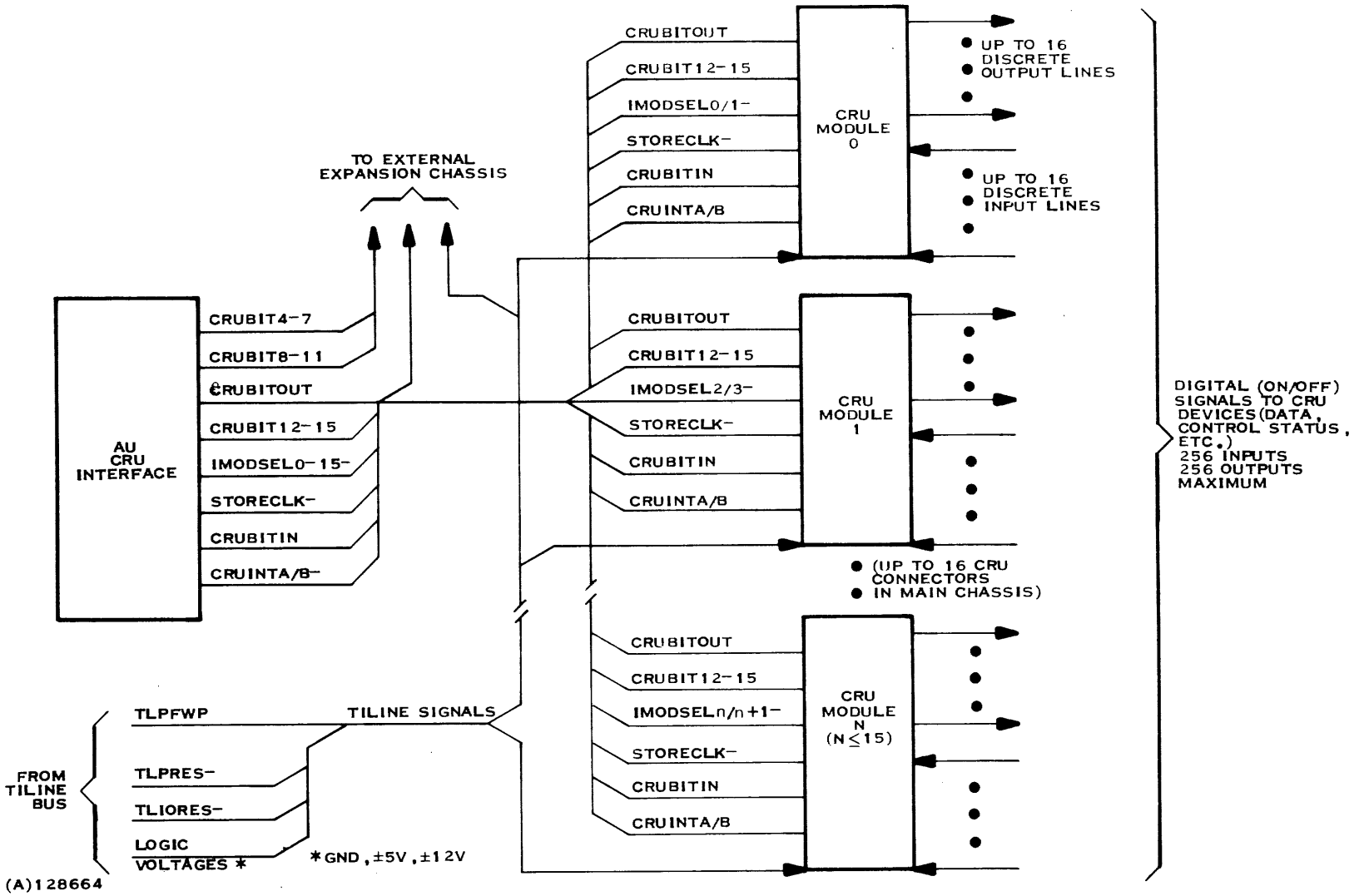


Figure 2-19. CRU Interface Signals

Table 2-8. CRU Interface Signals

Signature	Connector (P1/P2) Pin Number	Adapter (P1/P2) Pin Number	Definition
CRUBIT4	56*	-	Address bits generated by the AU to select a particular expansion chassis (bits 4-7), a 16-bit module within that chassis (bits 8-11), and a particular bit from that module (bits 12-15). Only bits 12-15 are used within the computer chassis. "1" = 2.4v; "0" = 0.4v.
CRUBIT5	54*	-	
CRUBIT6	52*	-	
CRUBIT7	50*	-	
CRUBIT8	62*	-	
CRUBIT9	64*	-	
CRUBIT10	68*	-	
CRUBIT11	70*	-	
CRUBIT12	36	42	
CRUBIT13	32	39,40	
CRUBIT14	38	37,38	
CRUBIT15	34	33,34	
CRUBITOUT	18	15,16	Serial data line for transfer of data from the AU to the addressed CRU bit(s). This line is active only when STORECLK- goes low. "1" = 2.4v; "0" - 0.4v.
CRUBITIN	60	59,60	Serial data line for transfer of data from the addressed CRU bit(s) to the AU. This line must be driven by an open collector gate and only when the module is selected. "1" = 2.4v; "0" = 0.4v.
IMODSEL0-	46 or 48	46 or 48	Module select signals generated by the AU from address bits 8-11 (CRUBIT8-11) for use within the main chassis. Each select signal is wired to a different CRU connector in the computer chassis to enable the circuitry of the circuit board in
IMODSEL1-	46 or 48	46 or 48	
IMODSEL2-	46 or 48	46 or 48	
IMODSEL3-	46 or 48	46 or 48	
IMODSEL4-	46 or 48	46 or 48	
IMODSEL5-	46 or 48	46 or 48	
IMODSEL6-	46 or 48	46 or 48	
IMODSEL7-	46 or 48	46 or 48	
IMODSEL8-	46 or 48	46 or 48	
IMODSEL9-	46 or 48	46 or 48	
IMODSEL10-	46 or 48	46 or 48	

*Connector P1 only.

Table 2-8. CRU Interface Signals (Continued)

Signature	Connector (P1/P2) Pin Number	Adapter (P1/P2) Pin Number	Definition
IMODSEL11- IMODSEL12- IMODSEL13- IMODSEL14- IMODSEL15-	46 or 48 46 or 48 46 or 48 46 or 48 46 or 48	46 or 48 46 or 48 46 or 48 46 or 48 46 or 48	that connector. When low (0.4v), the module is selected. Pin 48 is IMODSELA on CRU Circuit Boards; Pin 46 is IMODSELB.
STORECLK-	22	21,22	A 50 nanosecond low (0.4v) pulse that indicates to the selected CRU module that the operation is a write (Set Bit or LDCR) operation. This pulse transfers the data on the CRUBITOUT line into a holding flip-flop that is the CRU bit.
CRUINTB- CRUINTA-	65 66	- -	Low active (0.4v) interrupts generated by the particular CRU module to indicate to the AU that the module requires servicing. The priority of an interrupt is dependent upon the chassis location occupied by the module and the backpanel wiring of a particular unit. Interrupts must be open collector driven and must remain low until cleared by software.
TLPFWP	16	-	TILINE Power Failure Warning Pulse: A +5 volt, one millisecond pulse that indicates that a power failure is imminent.

Table 2-8. CRU Interface Signals (Continued)

Signature	Connector (P1/P2) Pin Number	Adapter (P1/P2) Pin Number	Definition
TLPRES-	13	11, 12*	TILINE Power Reset: A normally high (2.4v) signal that goes low (0.4v) to reset connected devices at least 10 microseconds before dc voltages begin to fail during power-down. During power-up, this signal is low until all power voltages are stable. CRU circuit boards may choose this pulse for a master reset signal.
TLIORES-	14	75, 76*	TILINE I/O Reset: A normally high (3.0v) signal that, when low (0.4v), resets all connected devices. This signal is a 250 nanosecond pulse that is generated by the RESET switch on the control console or by the execution of a reset (RSET) instruction in the AU. This signal is also low until dc power is stable. This signal is normally implemented as the master reset signal for CRU modules.

*NOTE: Adapter allows selection of either TLPRES- or TLIORES- to pins 75 and 76 for master reset. TLIORES- is standard.

Table 2-8. CRU Interface Signals (Continued)

Signature	Connector (P1/P2) Pin Number	Adapter (P1/P2) Pin Number	Definition
+12v	40	53, 54	Logic voltages. Provides up to 1 amp of ±12V and up to 10 amps of +5V total power for use by all modules in the chassis.
-12v	42	55, 56	
±12v Ground	-	57, 58	
+5v	3, 4, 77, 78	77, 78	
Ground	1, 2, 79, 80	79, 80	

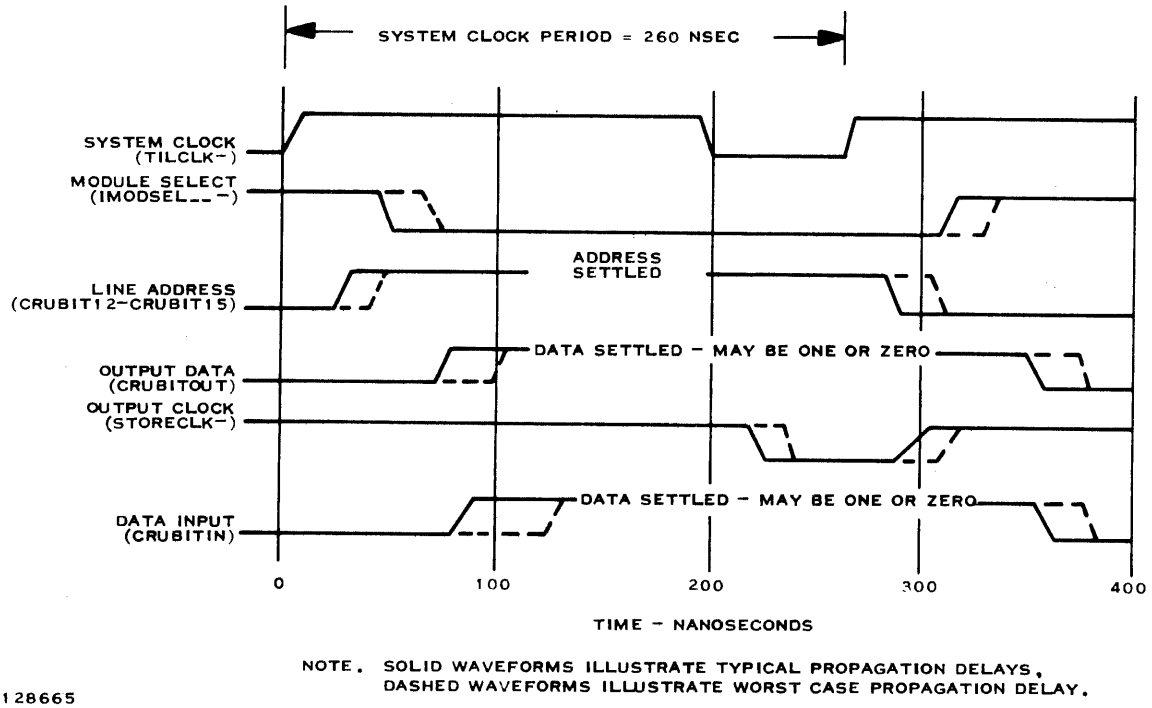


Figure 2-20. CRU Interface Timing

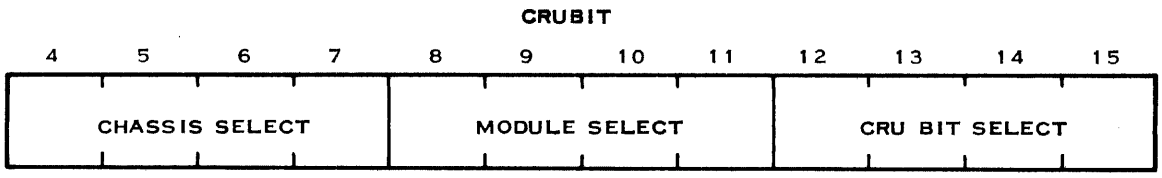


Figure 2-21. CRU Address Field Assignments

than 16 CRU addresses, but to enable a second group of 16 additional bits, a new IMODSEL signal must be assigned to that connector position. The number of selectable bits on one module is limited to 32.

2.6.4.2 BIT ADDRESS DEVELOPMENT. The AU develops a CRU bit address from the CRU base address contained in workspace register 12, and the signed displacement count contained in bits 8 through 15 of the Format 2 instruction. The displacement allows two's complement addressing from base minus 128 through base plus 127 (bit 8 is a sign bit). Figure 2-22 illustrates the AU development of the final CRU address. The base address is transferred from W12 to the MD register, added to the signed displacement contained in URB (after left-shifting the displacement one bit), and the result is loaded into ADC. The output of ADC supplies the address to the CRUBIT lines. Bit 15 of the address is not used, and the address extracted from ADC is right-justified so that bit 14 of ADC corresponds to bit 15 of the CRU address. This adjustment allows the CRU address to be incremented using the increment-by-two circuitry built into ADC. Therefore, the base address when entered into W12 must also allow for this shift, so that a base address of 1, for example, would be stored in W12 as an address of 2.

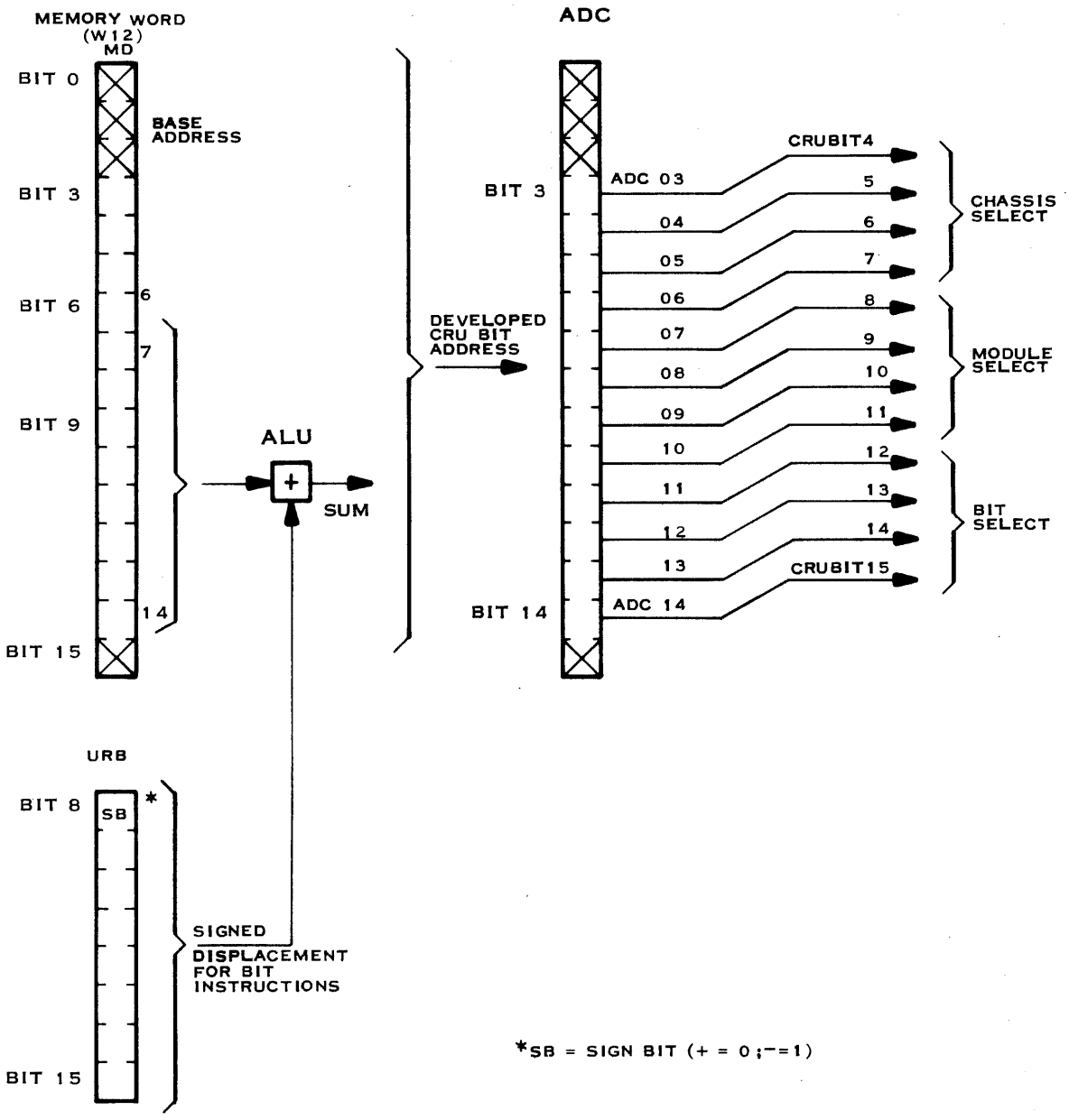
2.6.4.3 REGISTER ADDRESSING. CRU addresses for register operations (STCR, LDCR) are taken directly from W12 and loaded into ADC. The field length (C field) of the instruction is loaded into the SC counter. Each successive bit required for the operation increments the address in ADC and increments the count in SC. When SC equals a count of F_{16}, the CRU transfer is complete.

2.6.5 SINGLE-BIT CRU OPERATIONS

The AU performs three single-bit CRU functions: Test Bit (TB), Set Bit to One (SBO) and Set Bit to Zero (SBZ). To identify the bit to be operated upon, the AU develops the bit address as described previously and places it on the CRUBIT lines to the CRU devices. For the two write operations (SBO, SBZ), the AU also generates a STORECLK- pulse, indicating a write operation to the CRU device, and places bit 7 of the instruction word on the CRUBITOUT line to accomplish the specified operation (bit 7 is a 1 for SBO and a 0 for SBZ). If the transfer is a Test Bit instruction, the AU gates the addressed CRU bit from the CRUBITIN input line to bit 2 of the status register, and STORECLK- is not generated.

2.6.6 MULTIPLE-BIT CRU OPERATIONS

The AU performs two multiple-bit CRU operations: Store Communications Register (STCR) and Load Communications Register (LDCR). Both operations require a parallel-to-serial or serial-to-parallel conversion, as illustrated in figure 2-23. Although the figure illustrates a full 16-bit transfer



(A)128667A

Figure 2-22. CRU Bit Address Development

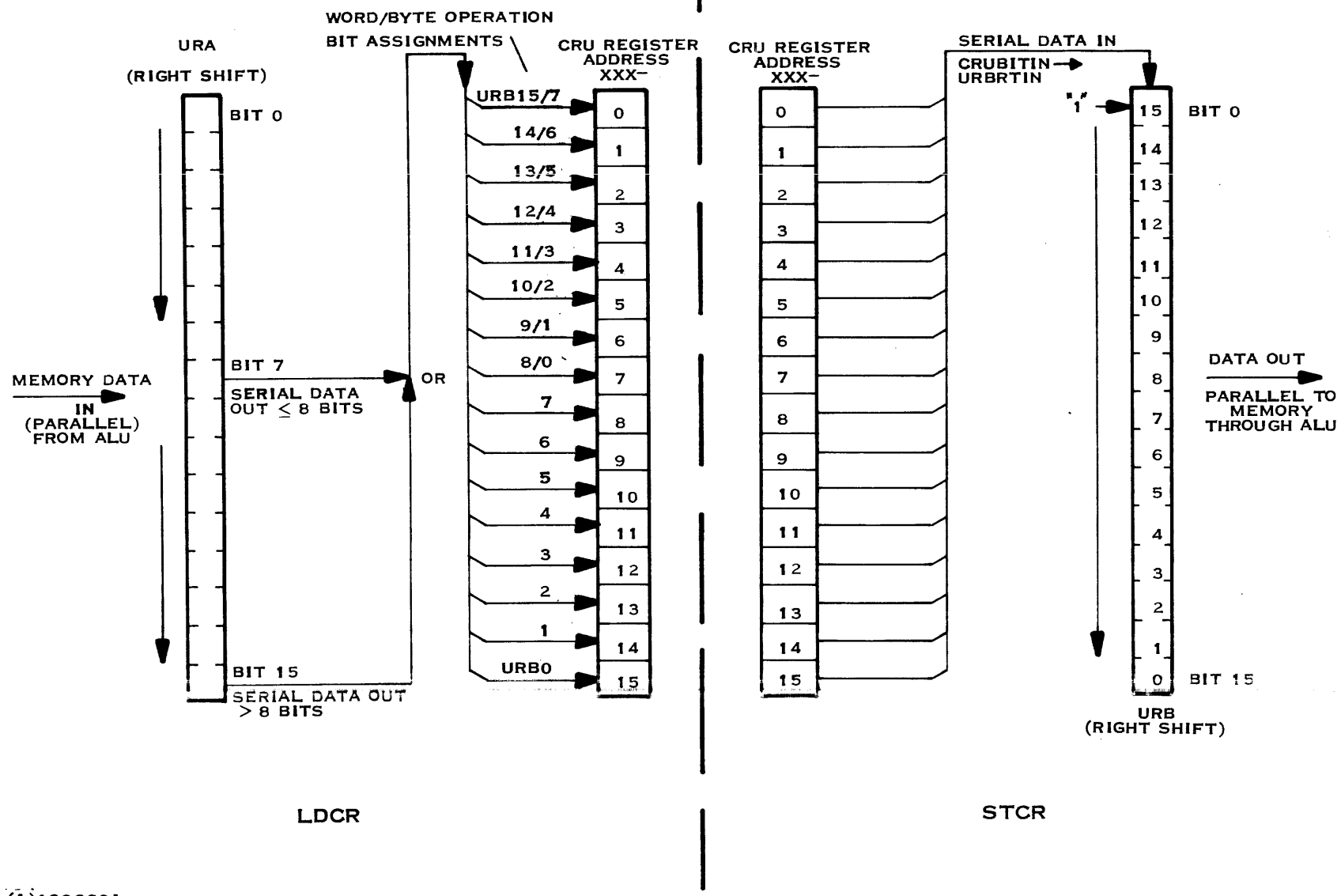


Figure 2-23. LDCR/STCR Data Handling

operation, any number of bits from 1 through 16 may be involved. The LDCR instruction fetches a word from memory, loads it into URA, and right-shifts URA to perform the parallel-to-serial conversion. If the load involves eight or fewer bits, those bits must be right-justified within the addressed byte of the memory word. This byte becomes the most significant byte of URA so that bit 7 of URA supplies the output bit to the CRU interface following each successive shift. If the load involves nine or more bits, those bits must be right-justified within the whole memory word. Bit 15 then supplies the output bit to the CRU interface following each successive shift. When transferred to the CRU interface, each successive bit from URA receives an address that is sequentially greater than the address for the previous bit. This addressing mechanism results in an order reversal of the bits; that is, bit 15 (or bit 7) becomes the lowest addressed bit in the CRU (most significant) and bit 0 of URA becomes the highest addressed bit in the CRU array (least significant).

An STCR instruction retrieves information from a CRU array and uses URB to perform the serial-to-parallel data conversion. If the operation involves a byte or less transfer, the valid information will be right-justified in the most significant byte of URB, with all leading bits set to zero. If the operation involves from 9 to 16 bits, the valid data is right-justified in the entire register with leading bits set to zero. When the input from the CRU device is complete, the first bit from the CRU is in bit 15 or bit 7 of URB, depending upon the number of bits transferred. A word operation result is stored directly into memory at the location specified by the instruction. To perform the byte store operation, the AU fetches the contents of the memory location, exchanges the new byte of information for the corresponding byte in the memory word, and stores the altered word into memory.

2.6.7 CRU MODULES

CRU modules are circuit boards that may be connected into any location in the computer chassis (either P1 or P2) to fan-in or branch-out the CRU data line from the AU to external control devices that operate from, or generate, two-level (on/off) electrical signals. The external devices may be process control devices such as valves (open or close), temperature or pressure monitors, photocell gates, or counters, or the external units may be low-speed digital input devices such as modems, card readers or keyboard devices. The line levels to and from these devices may be adjusted to correspond to the needs of the particular device being used. However, the components within each interface module, as illustrated in figure 2-24, remain constant with any end-use device.

2.6.7.1 OUTPUT. Output modules for CRU devices require a data input from the AU (CRUBITOUT), an address decoder to select the proper output line to receive the AU output (CRUBIT12-15), and a clocked flip-flop to receive and hold the signal from the CRUBITOUT line when STORECLK- becomes active. The flip-flop holds the signal for use by the external device

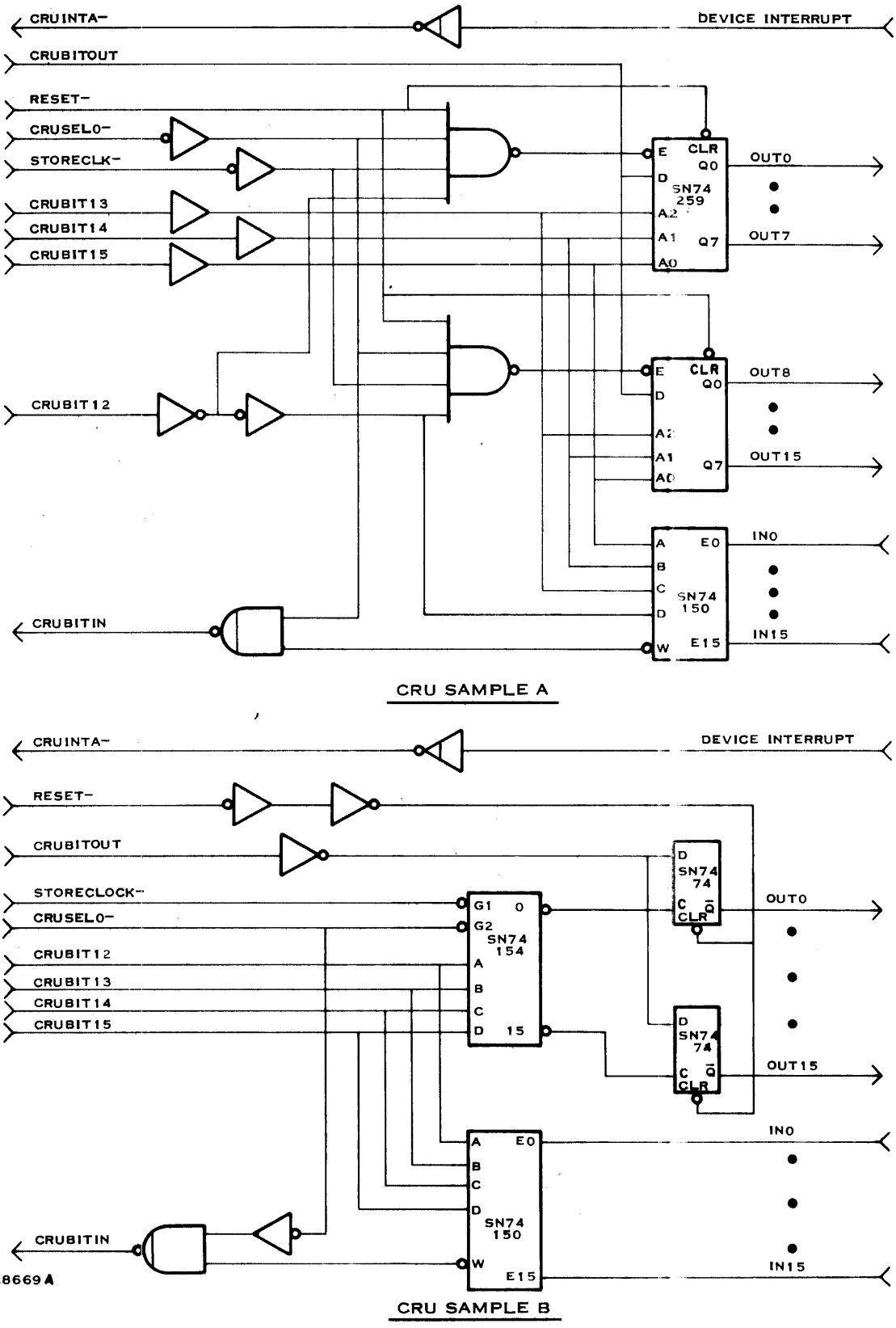


Figure 2-24. CRU Module Block Diagram

until the AU addresses that bit again. Level conversion circuits may be added to the output of the flip-flop to make the TTL signal compatible with the requirements of the external device.

2.6.7.2 INPUT. Input lines from the external CRU devices are fed into a selectable multiplexer so that the bit address and module select lines from the AU may choose the input line to be transferred to the AU over the CRUBITIN line. Whenever the AU generates an address, whether it is a read or a write operation, the selected module produces an input on the CRUBITIN line to the AU. The AU will ignore this input line unless it is performing a read. The CRUBITIN lines from all modules are bussed on one line to the AU. The module select signal from the AU, however, prevents more than one module from using the CRUBITIN line at one time.

2.6.7.3 AVAILABLE CRU MODULES. Four versatile CRU modules are offered as options with the computer to expand its capabilities. Other modules are under development and will be available shortly. In addition, modules can be specially designed for any stated purpose consistant with efficient use of the CRU interface. The four currently available CRU modules are outlined below. Refer to Section III of this manual for detailed descriptions of the available peripheral interfaces:

- Modem - A double-connector circuit board that provides a 1200 Baud, asynchronous interface for transmission and receipt of data over telephone lines. This circuit board enables the computer to function as a remote terminal processor to provide data to a larger computer system.
- 913 CRT Controller - A double-connector circuit board that interfaces the 913 display and keyboard with the computer. This circuit board offers its own refresh memory to the display, and enables the display to operate at computer speeds to fill the screen instantly with requested data.
- EIA Interface - A single-connector circuit board that provides a 16-bit input/output interface that is compatible with EIA Standard RS232C. This module requires an adapter card for insertion into either P1 or P2 connectors of any CRU slot.
- Data Module - A single-connector circuit board that interfaces 16 input bits and 16 output bits to the computer CRU interface. This module requires an adapter card for insertion into either P1 or P2 connectors of any CRU slot.

2.6.8 ELECTRICAL REQUIREMENTS

Since the CRU data and control paths are within one chassis and the line length is short, the signal lines do not need to be considered as transmission

lines. Instead they behave as if they were mounted on one printed circuit board. Other electrical considerations must be allowed for, however. All signals operate at TTL logic levels ("0" = 0.4v; "1" = 2.4v). Table 2-9 lists the interface signals and the electrical characteristics of the circuit on the CRU module that uses that signal.

Table 2-9. Electrical Interface Requirements

Signal	Active Level	Loading Allowed On Each Circuit Board	Driver Circuit Type
CRUBIT n	High true	2 TTL loads per bit	-
CRUBITOUT	High true	2 TTL loads	
CRUBITIN	High true	-	Open collector
IMODSEL n-	Low true	10 TTL loads per bit	-
STORECLK-	Low true	2 TTL loads	-
CRUINT-	Low true	-	Open collector
TLPFWP	High true	2 TTL loads	-
TLPRES-	Low true	10 TTL loads	-
TLIORES-	Low true	2 TTL loads	-

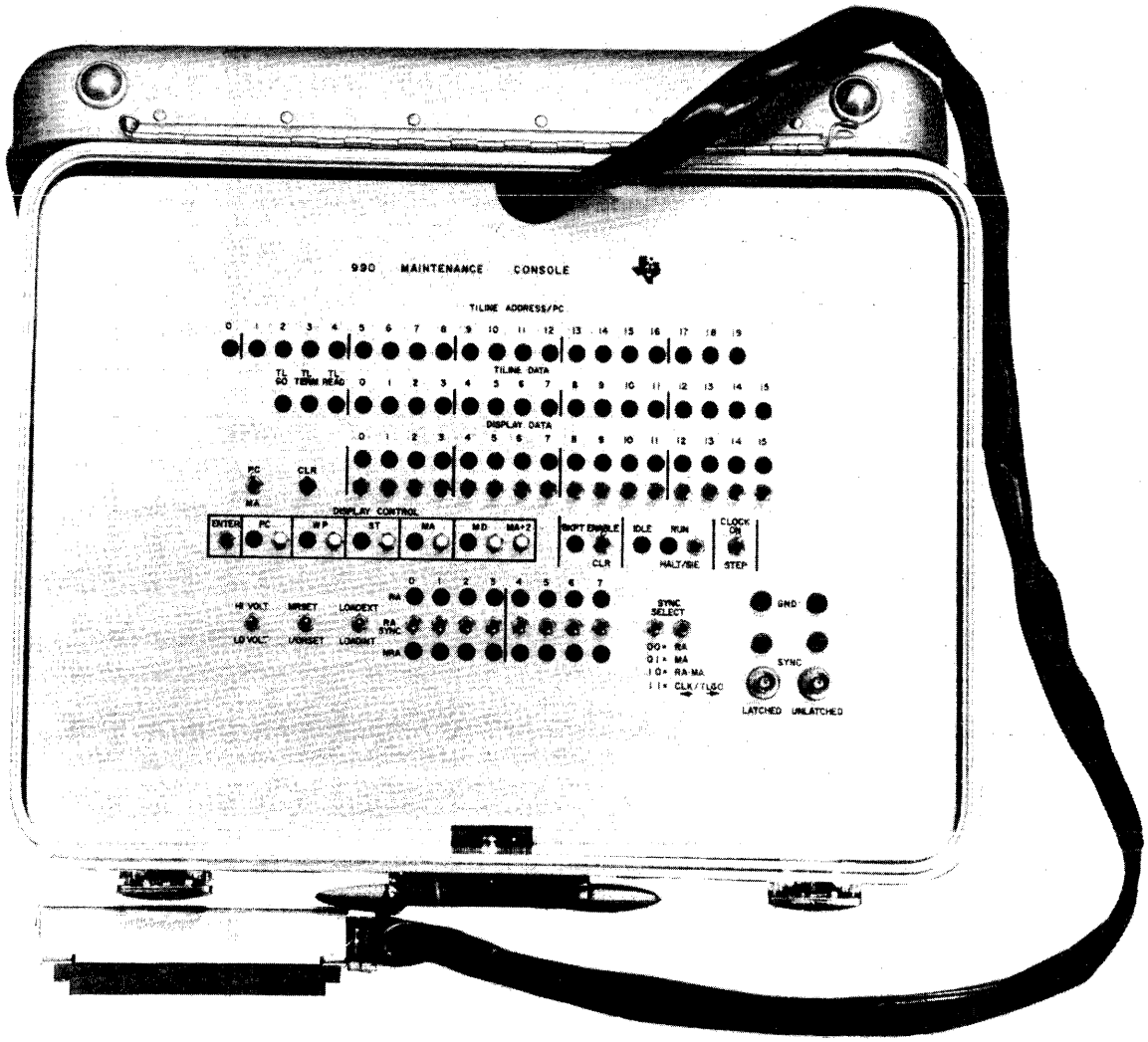
2.7 MAINTENANCE CONSOLE

The 990 Computer system offers an optional maintenance console to aid the maintenance or checkout technician in fault isolation. The console allows the operator to enter data or addresses into any of the 990 internal registers or memory, to step through instruction sequences one instruction at a time, to monitor the contents of the 990 internal registers, or to perform many other functions to exercise the computer for diagnosis of problem areas. The console connects to the computer through a cable and interface card assembly that inserts into the TILINE interface connector adjacent to the AU circuit board. Packaged in a rugged, lightweight, metal case, the 990 maintenance console can be easily and safely transported between sites for remote terminal repairs, or can be used in an OEM quality assurance function for performance verification on the computer. The 990 maintenance console is an extremely valuable maintenance and checkout tool.

2.7.1 CONTROLS AND INDICATORS

Light-emitting diodes mounted on the maintenance panel provide the light sources for the panel indicators. The operator controls are implemented with momentary pushbuttons, 2-position toggle switches, and three types of

3-position toggle switches (On-Off-On, On-Off-Momentary, and Momentary-Off-Momentary). Ground reference and sync pulse test points are also provided. Figure 2-25 illustrates the location and nomenclature of the controls and indicators on the maintenance panel. Table 2-10 defines the functions for each of these devices.



128670 (990-674-10-1)

Figure 2-24. 990 Maintenance Panel

Table 2-10. 990 Maintenance Console Controls and Indicators

Nomenclature	Device	Function
TILINE ADDRESS/PC 0-19	Indicators	These indicators display the binary contents of either the TILINE address bus or the program counter (PC), depending upon the position of the PC/MA toggle switch. A lighted indicator represents a binary "1"; an extinguished indicator represents a binary "0".
TILINE DATA 0-15	Indicators	These indicators display the binary value that is currently on the TILINE data bus. A lighted indicator represents a binary "1"; an extinguished indicator represents a binary "0".
DISPLAY DATA 0-15	Indicators	These indicators display either data entered by the operator through the corresponding DISPLAY DATA pushbuttons, or the contents of a selected area of the 990 AU as determined by the DISPLAY CONTROL switches. A lighted indicator represents a binary "1"; an extinguished indicator represents a binary "0".
	Pushbuttons (momentary)	These pushbuttons enter data into the DISPLAY DATA indicators. Pressing and releasing a pushbutton changes the state of the corresponding DISPLAY DATA indicator.
TLGO	Indicator	This indicator lights to indicate that the TILINE go pulse is active. The indicator is useful only during clock step mode since the pulse is too brief during normal operation.
TLTERM	Indicator	This indicator lights to indicate that the TILINE term pulse is active. The indicator is useful only during clock step mode since the pulse is too brief during normal operation.

Table 2-10. 990 Maintenance Console Controls and Indicators (Continued)

Nomenclature	Device	Function
TLREAD	Indicator	This indicator lights to indicate that the TILINE read line is active, designating that a read operation is being performed on some TILINE device, or that the bus is not performing a write operation.
PC/MA	Toggle switch (2-position)	This switch controls the function of the TILINE ADDRESS/PC indicators. When set to PC, this switch displays the contents of the program counter (PC), bits 0-14, in the TILINE ADDRESS/PC indicators 5 through 19. When set to MA, this switch displays the TILINE address bus, bits 0 through 19, in the TILINE ADDRESS/PC indicators.
CLR	Pushbutton (momentary)	Pressing this pushbutton resets the DISPLAY DATA indicators.
DISPLAY CONTROL		The Display Control switches operate only when the computer is halted.
PC	Pushbutton (momentary)	Pressing this pushbutton displays the contents of the program counter (PC) in the DISPLAY DATA indicators.
	Indicator	This indicator lights when the data in the DISPLAY DATA indicators represents the contents of the program counter.
WP	Pushbutton (momentary)	Pressing this pushbutton displays the contents of the workspace pointer (WP) register in the DISPLAY DATA indicators.
	Indicator	This indicator lights when the data in the DISPLAY DATA indicators represents the contents of the workspace pointer register.

Table 2-10. 990 Maintenance Console Controls and Indicators (Continued)

Nomenclature	Device	Function
DISPLAY CONTROL (Continued)		
ST	Pushbutton (momentary)	Pressing this pushbutton displays the contents of the status register (ST) in the DISPLAY DATA indicators.
	Indicator	This indicator lights when the data in the DISPLAY DATA indicators represents the contents of the status register.
MA	Pushbutton (momentary)	Pressing this pushbutton displays the 16 least significant bits of the address currently being requested from memory in the DISPLAY DATA indicators.
	Indicator	This indicator lights when the data in the DISPLAY DATA indicators represents the memory address.
MD	Pushbutton (momentary)	Pressing this pushbutton displays the contents of the memory data (MD) register in the DISPLAY DATA indicators.
	Indicator	This indicator lights when the data in the DISPLAY DATA indicators represents the contents of the memory data register.
MA+2	Pushbutton (momentary)	Pressing this pushbutton increments the current memory address by two, displays the contents of that memory location in the DISPLAY DATA indicators, and lights the MD indicator.
ENTER	Pushbutton (momentary)	Pressing this pushbutton concurrently with pressing one of the other DISPLAY CONTROL pushbuttons enters data from the DISPLAY DATA indicators into the computer area indicated by the concurrently pressed pushbutton. For example, pressing ENTER and WP transfers the data from

Table 2-10. 990 Maintenance Console Controls and Indicators (Continued)

Nomenclature	Device	Function
DISPLAY CONTROL (Continued)		
ENTER (Continued)	Pushbutton (momentary)	the DISPLAY DATA indicators to the workspace pointer register, and lights the WP indicator.
BKPT	Indicator	When lighted, this indicator designates that the current value on the TILINE ADDRESS/PC indicators matches the value set in the DISPLAY DATA indicators. When extinguished, this indicator designates that no breakpoint compare has been reached since the ENABLE/CLR switch was last moved to CLR or since the computer entered the run mode (RUN/HALT/SIE switch in RUN position).
ENABLE/CLR	Toggle switch 3-position (On-Off-Moment)	The center (rest) position of this switch is the normal operation position for the switch, and enables the computer to operate without stopping at a breakpoint. When set to the ENABLE (up) position, this switch stops the computer when the value on the TILINE ADDRESS/PC indicators matches the value in the DISPLAY DATA indicators (stop on breakpoint). When moved to the CLR (momentary) position, the switch clears the BKPT indicator.
IDLE	Indicator	This indicator lights to indicate that the computer is executing an IDLE instruction. The computer remains in the run mode but will not continue processing until the RUN/HALT/SIE switch is set to HALT and then set back to the RUN position, or until a recognized interrupt places the computer into a service routine. When extinguished, this indicator designates that the computer is not executing an IDLE instruction.

Table 2-10. 990 Maintenance Console Controls and Indicators (Continued)

Nomenclature	Device	Function
RUN/HALT/SIE	Indicator	This indicator lights to indicate that the computer is in the run mode. When this indicator is extinguished, the computer is in the halt mode.
RUN/HALT/SIE	Toggle switch 3-position (Mom-Off-Mom)	When in the center (rest) position, this switch allows the CPU to operate in its currently selected mode. When moved to RUN (up-momentary), this switch places the CPU in the run mode, allowing continuous instruction execution, and lights the RUN/HALT/SIE indicator. When this switch is moved to HALT/SIE (down-momentary) while in the run mode, the CPU executes the current instruction and terminates program execution. When this switch is moved to HALT/SIE while in the halt mode, the CPU executes one instruction from the program sequence (Single Instruction Execution).
CLOCK	Toggle switch 3-position (On-Off-Moment)	When set to the ON position (up), this switch enables normal system operation with a free-running clock signal. When set to the center position (off), this switch stops the system clock. When moved to the STEP position (down-momentary), this switch enables the system to generate a single clock pulse.
HI VOLT/LO VOLT	Toggle switch 3-position (On-Off-On)	When set to HI VOLT (up), this switch raises the 12 volt supply line to memory to its upper margin (12.6 volts). When set to LO VOLT (down), this switch lowers the 12 volt supply line to memory to its lower margin (11.4 volts). When set to the center position, this switch allows the memory supply voltage to operate at its normal level.

Table 2-10. 990 Maintenance Console Controls and Indicators (Continued)

Nomenclature	Device	Function
MRSET/I/ORSET	Toggle switch 3-position (Mom-Off-Mom)	When in the center (rest) position, this switch is inactive. When moved to the MRSET position (up-momentary) while the CLOCK switch is set to ON, this switch produces a master reset that clears the interrupt mask, pending interrupts and the 120 Hz clock, places the computer in the run mode, and initiates a power-up sequence. When moved to I/ORSET (down-momentary) while the CPU is in the halt mode, this switch issues an I/O Reset signal that masks all interrupts, clears pending interrupts, and resets the 120 Hz clock.
LOADEXT - LOADINT	Toggle switch 3-position (Mom-Off-Mom)	When in the center (rest) position, this switch is inactive. When moved to LOADEXT (up-momentary), this switch initiates an operation that loads 256 words from a ROM in the maintenance panel repeatedly into the first 4K words of memory. When moved to LOADINT (down-momentary), this switch loads the 256 ROM loader from the AU circuit board repeatedly into the first 4K words of memory.
RA 0 through 7	Indicators	These indicators display the current control ROM address. A lighted indicator represents a "1" bit; an extinguished indicator represents a "0" bit.
NRA 0 through 7	Indicators	These indicators display the next control ROM address to be executed in the control sequence. A lighted indicator represents a "1" bit; an extinguished indicator represents a "0" bit.
RA SYNC 0 through 7	Toggle switches 2-position	These switches allow the operator to set an 8-bit value that generates a sync pulse when the current ROM address corresponds to that value. (Up = "1"; Down = "0"). The operation of these switches is controlled by the SYNC SELECT toggle switches.

Table 2-10. 990 Maintenance Console Controls and Indicators (Continued)

Nomenclature	Device	Function
SYNC SELECT	Toggle switches 2-position	These switches select the function that will generate a sync pulse for monitoring on the BNC connector and test points. The function code for the switches is as follows (Up = "1"; Down = "0"): 00 - Sync pulse generated when the current ROM address matches the value set in the RA SYNC switches. 01 - Sync pulse generated when the current address on the TILINE ADDRESS/PC indicators matches the value set in the DISPLAY DATA switches. 10 - Sync pulse generated when both codes "00" and "01" are satisfied simultaneously. 11 - Sync pulse on test point and BNC connector labeled LATCHED is the system clock pulse; sync pulse on the UNLATCHED test point and connector is the TLGO- pulse.
GND - LATCHED	Test point	Provides a reference ground potential for use with the SYNC - LATCHED test point.
GND - UNLATCHED	Test point	Provides a reference ground potential for use with the SYNC - UNLATCHED test point.
SYNC - LATCHED	Test point or BNC connector	Provides a sync pulse as defined by the SYNC SELECT switches after the generating signal has been clocked through a synchronizing flip-flop to isolate the signal from transient pulses.
SYNC - UNLATCHED	Test point or BNC connector	Provides a sync pulse that is the raw output of the comparison circuit defined by the SYNC SELECT switches, and is therefore, not delayed as is the LATCHED output.

SECTION III

990 COMPUTER PERIPHERAL DEVICES

3.1 INTRODUCTION

This section contains pertinent information about the peripherals available with the Model 990 Computer. These peripherals are:

- TI Model 913 CRT Display Terminal
- TI Model 733 ASR Data Terminal*
- Model 33 ASR Teletypewriter Data Terminal*
- Modem Controller Communication I/O Module
- Asynchronous TTY/EIA Communications Interface Module
- 16 I/O Data Module
- Prototype Development Cards

Descriptions of these peripherals include use operations, programming requirements, interconnect information, and power requirements. Where these peripherals are offered as kits, the various kit options are described.

3.2 TI MODEL 913 CRT DISPLAY TERMINAL

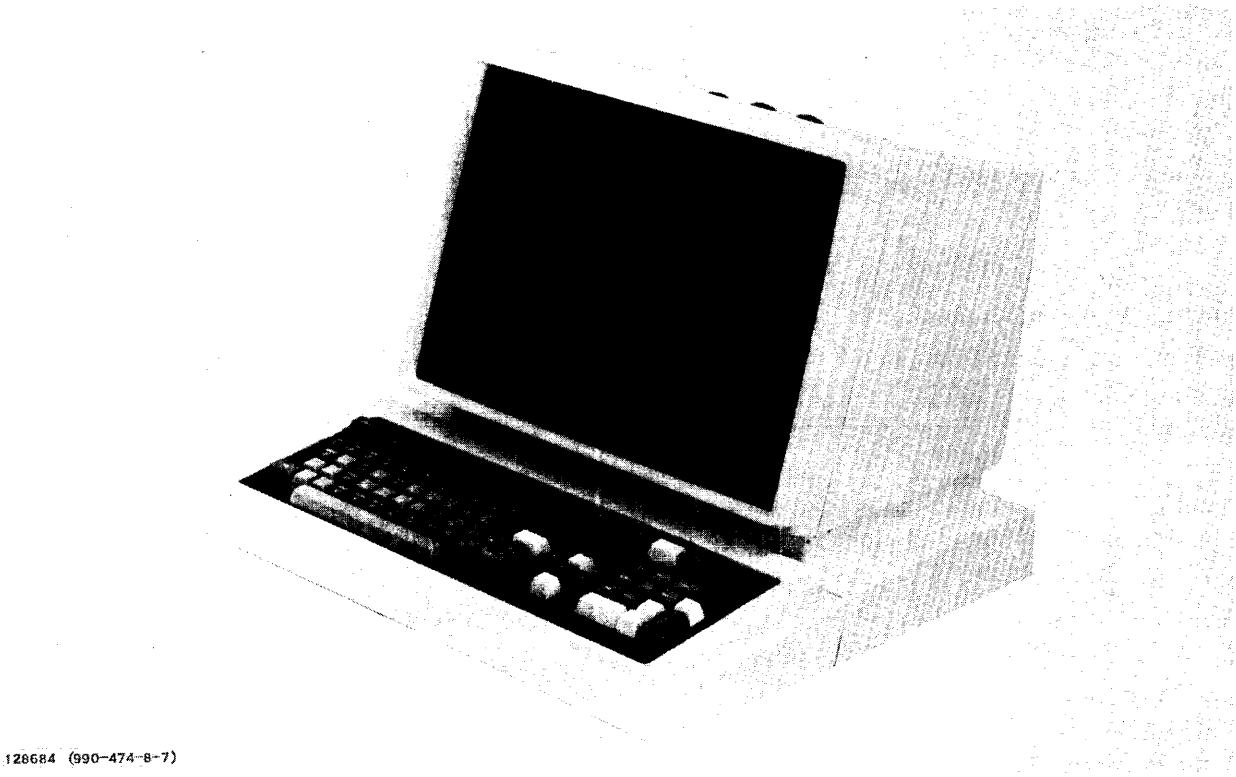
3.2.1 GENERAL

The 913 CRT display terminal is an interactive data terminal connected to the CRU in the Model 990 Computer to provide a fully-programmable data terminal to satisfy a dynamic-interface user requirement. Hard copies of the display may be made on an optional print-only printer attached to the CRT.

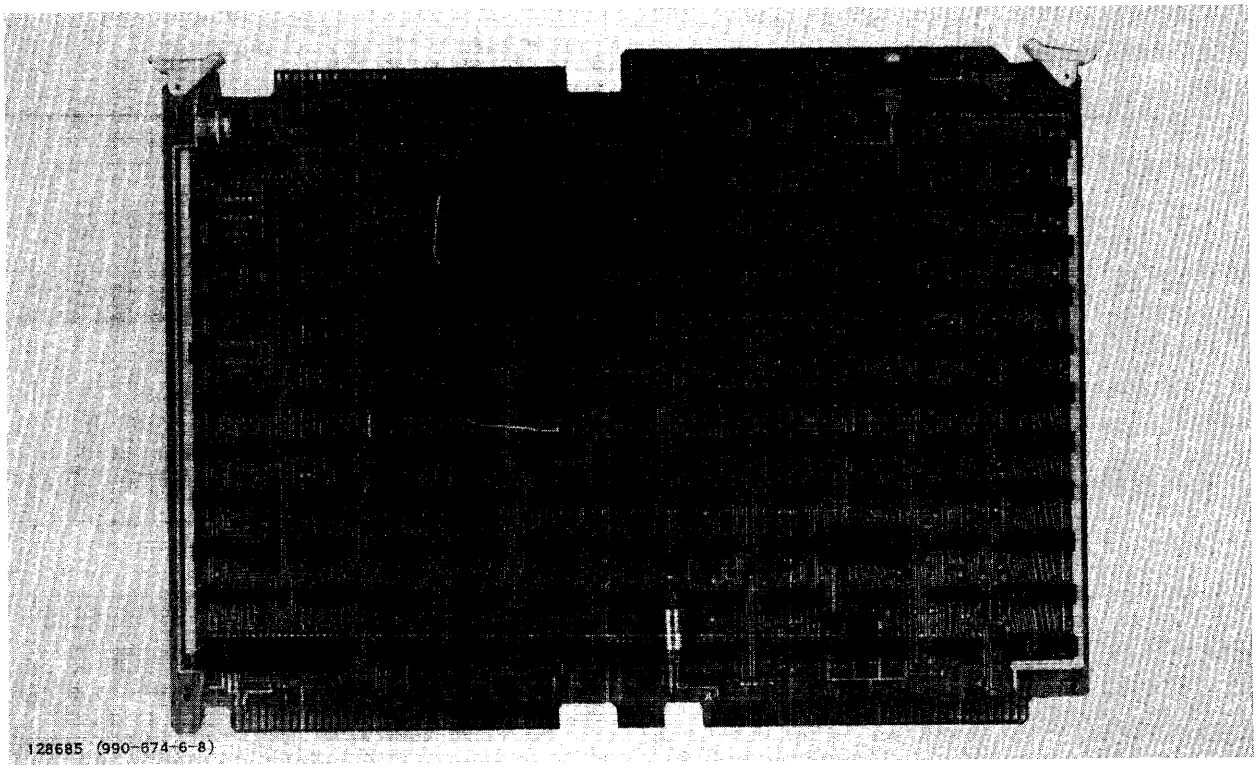
3.2.2 DESCRIPTION

The 913 CRT display (figure 3-1) is a stand-alone 12-inch (diagonal measure) television monitor with a non-reflective, high-resolution screen. CRT driving circuitry consists of all solid-state components mounted on a single printed-circuit (PC) board next to the CRT tube. Character generation, video generation, buffer memory, and other control circuitry consists of all solid-state components mounted on the CRU interface board, which resides in the CPU chassis. The keyboard is remote from the CRT chassis and consists of the keys shown below, made of "double shot" molded, non-glare-finish plastic,

*Either of these two data terminals may be used for input/output for software development systems.



128684 (990-474-8-7)



128685 (990-674-6-8)

Figure 3-1. CRT Display and Controller

mounted in the keyboard case. Each key of the keyboard is a single-function key designed for single-handed operation. The numeric keys are arranged in a 10-key numeric pad for ease of operation.

3.2.3 OPERATING CONTROLS, DISPLAY, AND KEYBOARD

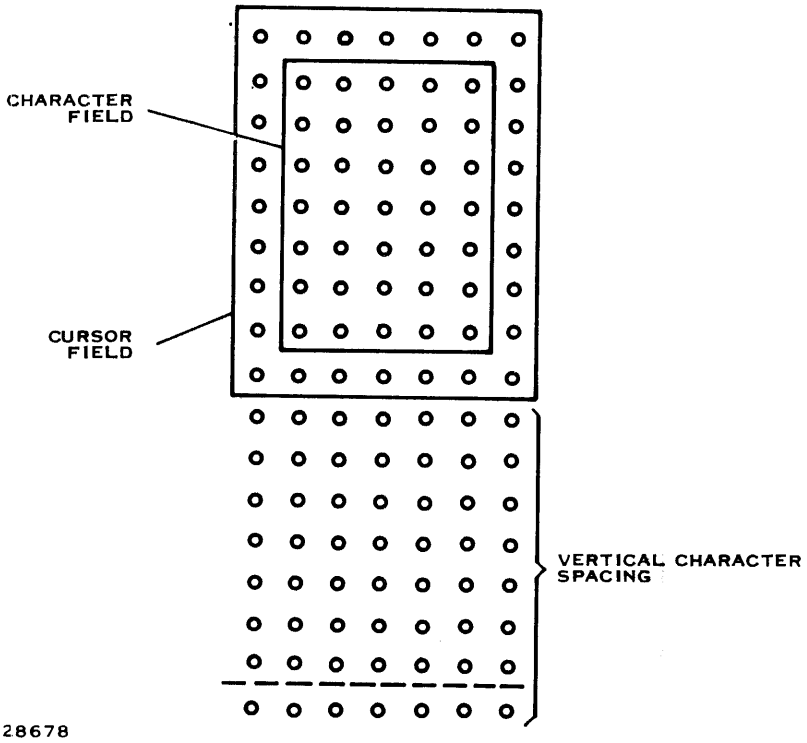
The following items are used to input data from the CRT and display output from the 990 Computer.

3.2.3.1 CONTROLS. The controls for the CRT display terminal are:

- Display - Brightness, Contrast, Horizontal/Vertical Hold
- Cursor - ↑, ↓, ←, →, Home
- Format - Tab, Space, Repeat
- Control - 14 Special User defined keys.

3.2.3.2 DISPLAY. The display characters are formed as follows:

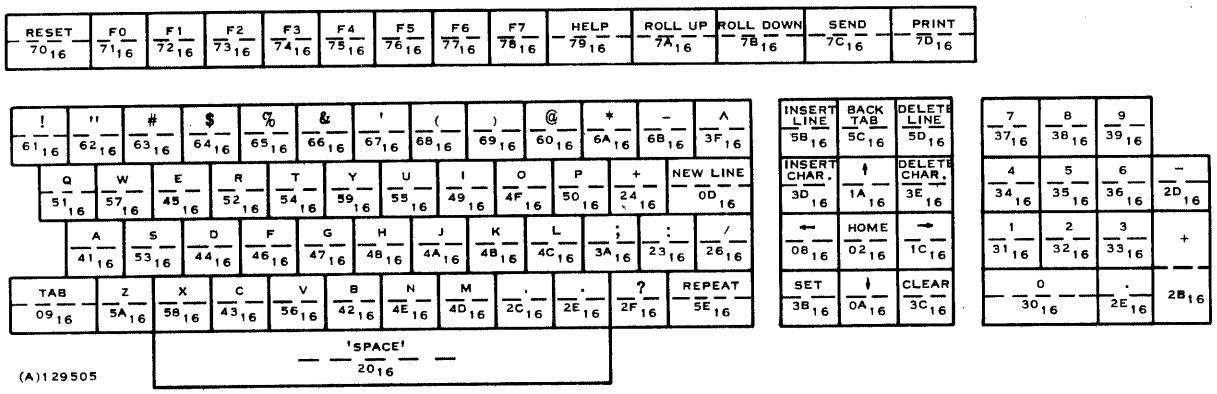
- Character shape - 5 x 7 dot matrix
- Cursor field - 7 x 9 dot matrix
- Character field - 7 x 16 dot matrix



- Screen capacity - 12 horizontal lines and 80 characters per line.

3.2.3.3 KEYBOARD CHARACTERS. The following characters are available on the keyboard of the Model 913 CRT Display terminal:

- Upper Case USASCII Keyboard - 57 display characters and 32 control keys in the following arrangement (Note that the USASCII hexadecimal value generated by each key is shown below the dotted line and the character is indicated above the dotted line):



3.2.4 CAPABILITIES

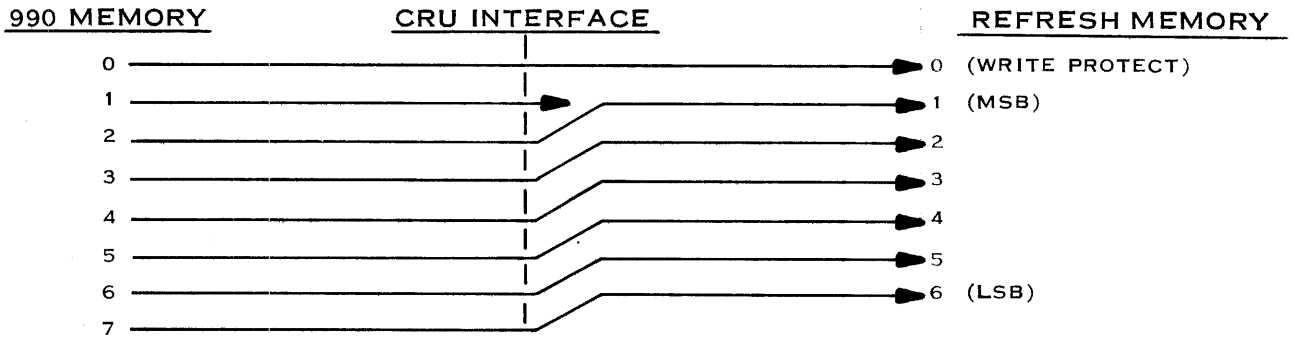
The 913 CRT display has the following capabilities:

- Instant Display - Screen filled in less than 20 milliseconds.
- Programmable Cursor Positioning
- Processor Controlled Functions
- Screen Refresh at 60 frames per second from CRT controller memory
- Programmable Editing
- Protected Display Fields - Programmable.

3.2.5 CRT DISPLAY SPECIFICATIONS

3.2.5.1 TRANSMISSION CODE AND CHARACTER SET. The transmission code and character set is shown in the keyboard illustration.

The CRU output byte from the 990 memory is stored in the CRT refresh memory as follows:



(A)129506

The second most significant bit (CRU Bit 1) is dropped in the refresh memory store operation. When this byte is read from the refresh memory for either CRT refresh or a CRU read operation (STCR machine instruction), bit 1 is restored as the complement of bit 2. This results in the USASCII hexadecimal codes of 00_{16} through $0F_{16}$, 10_{16} through $1F_{16}$, 60_{16} through $6F_{16}$, and 70_{16} through $7F_{16}$ being changed to 40_{16} through $4F_{16}$, 50_{16} through $5F_{16}$, 20_{16} through $2F_{16}$, and 30_{16} through $3F_{16}$, respectively. This character alteration is shown in table 3-1 for each keytop of the CRT keyboard.

Table 3-1. Character Set as Read from Refresh Memory and Displayed on the CRT Screen

Keyboard		Refresh Memory Character Read	Display Character
USASCII Code	Character		
02	HOME	42	B
08	←	48	H
09	TAB	49	I
0A	↓	4A	J
0D	NEW LINE	4D	M
1A	↑	5A	Z
1C	→	5C	\
20	space	20	space
23	:	23	#
24	+	24	$

Table 3-1. Character Set as Read from Refresh Memory and Displayed on the CRT Screen (Continued)

Keyboard		Refresh Memory Character Read	Display Character
USASCII Code	Character		
26	/	26	&
2B	+	2B	+
2C	,	2C	,
2D'	-	2D	-
2E	.	2E	.
2F	?	2F	/
30	0	30	0
31	1	31	1
32	2	32	2
33	3	33	3
34	4	34	4
35	5	35	5
36	6	36	6
37	7	37	7
38	8	38	8
39	9	39	9
3A	;	3A	:
3B	SET	3B	;
3C	CLEAR	3C	<
3D	INSERT CHAR	3D	=
3E	DELETE CHAR	3E	>
3F	^	3F	?
41	A	41	A
42	B	42	B
43	C	43	C
44	D	44	D

Table 3-1. Character Set as Read from Refresh Memory and Displayed on the CRT Screen (Continued)

Keyboard		Refresh Memory Character Read	Display Character
USASCII Code	Character		
45	E	45	E
46	F	46	F
47	G	47	G
48	H	48	H
49	I	49	I
4A	J	4A	J
4B	K	4B	K
4C	L	4C	L
4D	M	4D	M
4E	N	4E	N
4F	O	4F	O
50	P	50	P
51	Q	51	Q
52	R	52	R
53	S	53	S
54	T	54	T
55	U	55	U
56	V	56	V
57	W	57	W
58	X	58	X
59	Y	59	Y
5A	Z	5A	Z
5B	INSERT LINE	5B	[
5C	BACK TAB	5C	\
5D	DELETE LINE	5D	]
5E	REPEAT	5E	^

Table 3-1. Character Set as Read from Refresh Memory and Displayed on the CRT Screen (Continued)

Keyboard		Refresh Memory Character Read	Display Character
USASCII Code	Character		
60	@	20	space
61	!	21	!
62	"	22	"
63	#	23	#
64	$	24	$
65	%	25	%
66	&	26	&
67	'	27	'
68	(	28	(
69	)	29	)
6A	*	2A	*
6B	-	2B	+
70	RESET	30	0
71	F0	31	1
72	F1	32	2
73	F2	33	3
74	F3	34	4
75	F4	35	5
76	F5	36	6
77	F6	37	7
78	F7	38	8
79	HELP	39	9
7A	ROLL UP	3A	:
7B	ROLL DOWN	3B	;
7C	SEND	3C	<
7D	PRINT	3D	=

3.2.5.2 DISPLAY RATE. Each character requires 6 microseconds to be displayed and less than 20 milliseconds are required to fill the screen.

3.2.6 INSTALLATION

3.2.6.1 CONNECTION INFORMATION. The CRT controller mounts in any chassis CRU slot. A cable supplied as part of the peripheral kit connects the controller to the CRT monitor and another cable connects the keyboard to the CRT monitor.

3.2.6.2 PROGRAMMING INFORMATION. Figure 3-2 shows the CRU input as read from the refresh memory of the CRT controller when enabled by the IMODSELA signal from the CRU. Figure 3-3 shows the CRU input from the cursor position circuits of the CRT controller. This input is enabled by the IMODSELB signal from the CRU. Bits 0 through 3 indicate cursor position in a row as follows:

BIT 0 1 2 3	Hexadecimal Value	Row
0 0 0 0	0	0 (Top)
0 0 0 1	1	1
0 0 1 0	2	2
⋮	⋮	⋮
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	A	10
1 0 1 1	B	11 (Bottom)

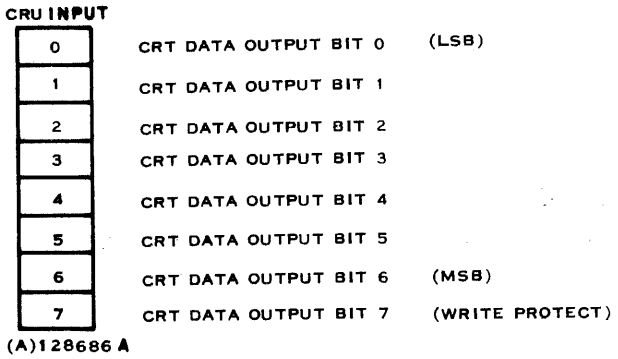


Figure 3-2. CRT Data Output to CRU

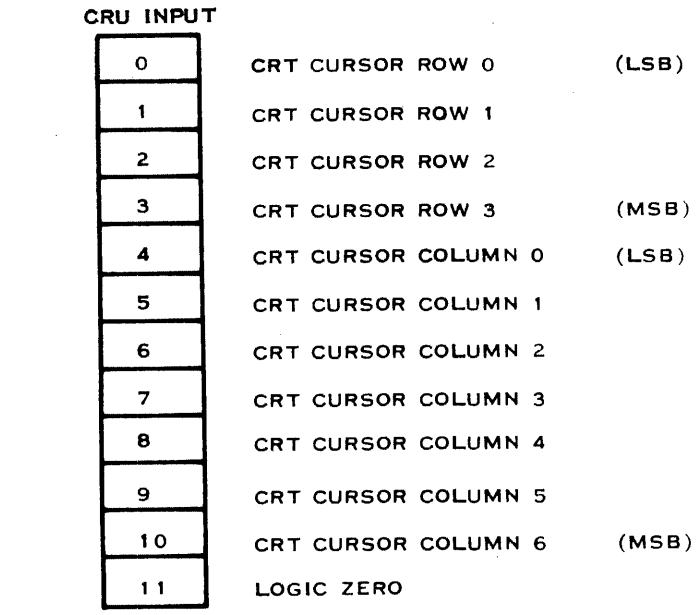


Figure 3-3. CRT Cursor Position

Bits 4 through 10 indicate cursor column position as follows:

BIT 4	5	6	7	8	9	10	Hexadecimal Value (Leading zero assumed)	Row
0	0	0	0	0	0	0	00	0 (Left)
0	0	0	0	0	0	1	01	1
0	0	0	0	0	1	0	02	2
			⋮				⋮	⋮
1	0	0	1	1	0	1	4D	77
1	0	0	1	1	1	0	4E	78
1	0	0	1	1	1	1	4F	79 (Right)

Figure 3-4 shows the output from the CRU which is CRT controller input. Bits 0 through 7 are the data bits that are stored in the refresh memory. (Note that table 3-1 shows the relationship of the data stored in the refresh memory and the data read from the refresh memory.) Bit 8 is the line that when strobed, indicates the data in bits 0 through 7 is the data byte to store in the refresh memory. Bits 9 through 14 are the cursor control signals that control cursor position according to the program being executed. Figure 3-5 shows the arrangement of the bits that represent the data placed on the CRU when a key on the keyboard is pressed. The USASCII hexadecimal character generated when a key is pressed is shown in table 3-1 and on the standard keyboard diagram. Figure 3-6 shows the keyboard signals output by the CRU which are used to acknowledge a keyboard entry and to enable the keyboard interrupt feature.

3.2.6.3 POWER REQUIREMENTS. The power requirements for the CRT display and keyboard are as follows:

- CRT - 115 VAC, 50-60 Hz, 130 watts
- Keyboard - +5 Vdc @ 0.5A
- CRU Interface Board - ±12 Vdc
  +5 Vdc @ 2.7A

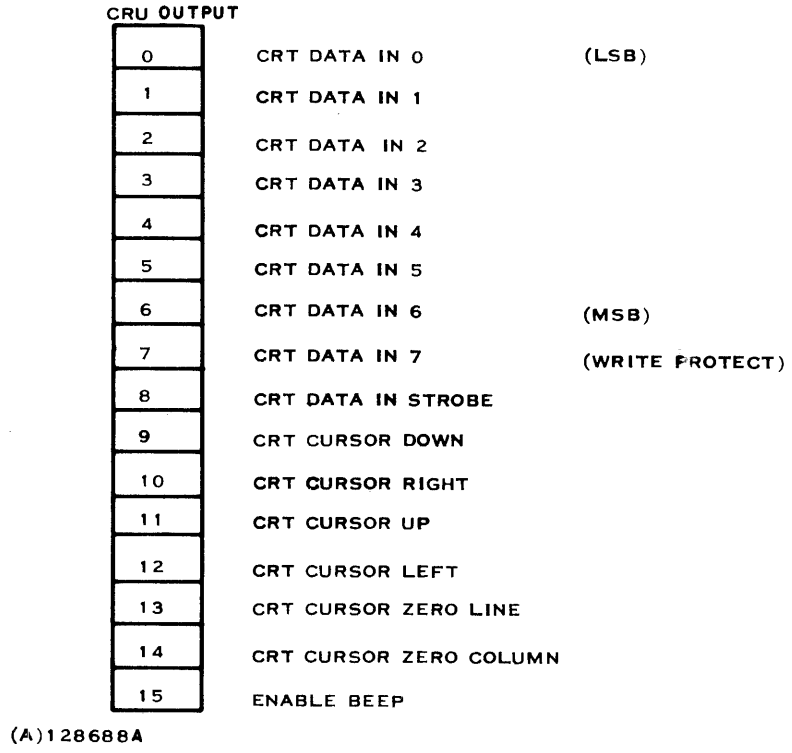


Figure 3-4. CRT Input

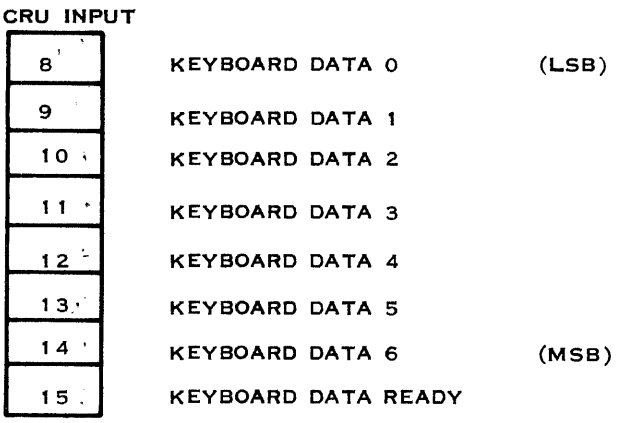


Figure 3-5. Keyboard Data Output

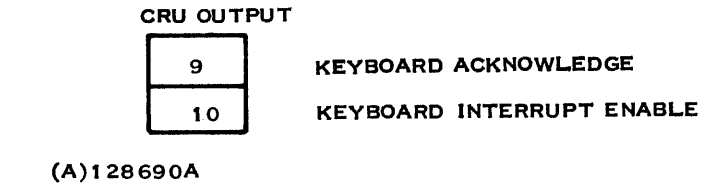


Figure 3-6. Keyboard Data Input

3.2.6.4 SPACE REQUIREMENTS. Space requirements for the CRT display and keyboard are as follows (note, dimensions are in inches):

- Display - 15.5(H) x 12.8(D) x 19.0(W)
- Keyboard - 4.0(H) x 9.0(D) x 18.75(W)

3.2.7 PERIPHERAL KIT OPTIONS (Kit part number 974708)

The following list identifies the various options available with the 913 CRT Display peripheral kit:

Kit Part Number	Description of Kit Contents
974708-0001	CRT Peripheral Kit, Model 913, 960-character This kit includes the 913 CRT, CRT controller, 15-foot cable, and standard keyboard.
974708-0002	CRT Peripheral Kit, Model 913, 960-character, Long Cable This kit contains the same items as the -0001 kit except the cable is 100 feet in length.
974708-0003	CRT Peripheral Kit, Model 913, 960-character, CRT This kit contains only the CRT display and controller board with a 15 foot cable.
974708-0991	CRT Peripheral Kit, Model 913, 960-character, Documentation, Cassette This kit contains the kit documentation and peripheral PAT on cassette.
974708-0992	CRT Peripheral Kit, Model 913, 960-character, Documentation, Paper Tape This kit contains the same items as does the -0991 kit except the PAT is on paper tape.

3.3 TI MODEL 733 ASR DATA TERMINAL

3.3.1 GENERAL

The MODEL 733 ASR Data Terminal may be used as the system keyboard/printer and principal input/output device for the 990 Computer. Printed output from the CPU appears on this data terminal. Input/output media is magnetic cassette for this data terminal.

3.3.2 DESCRIPTION

The TI Model 733 ASR Data Terminal (figure 3-7) is a USASCII data terminal and can be used as a system input/output device for the 990 Computer. The 733 is modular in design with a keyboard, printer mechanism, transmit/receive electronics, record/playback units, and associated controls.

- The standard 733 USASCII keyboard permits manual typing operations and transmission of printable characters and operational codes in 7-level USASCII code. Full uppercase and lowercase capability is available as an option.

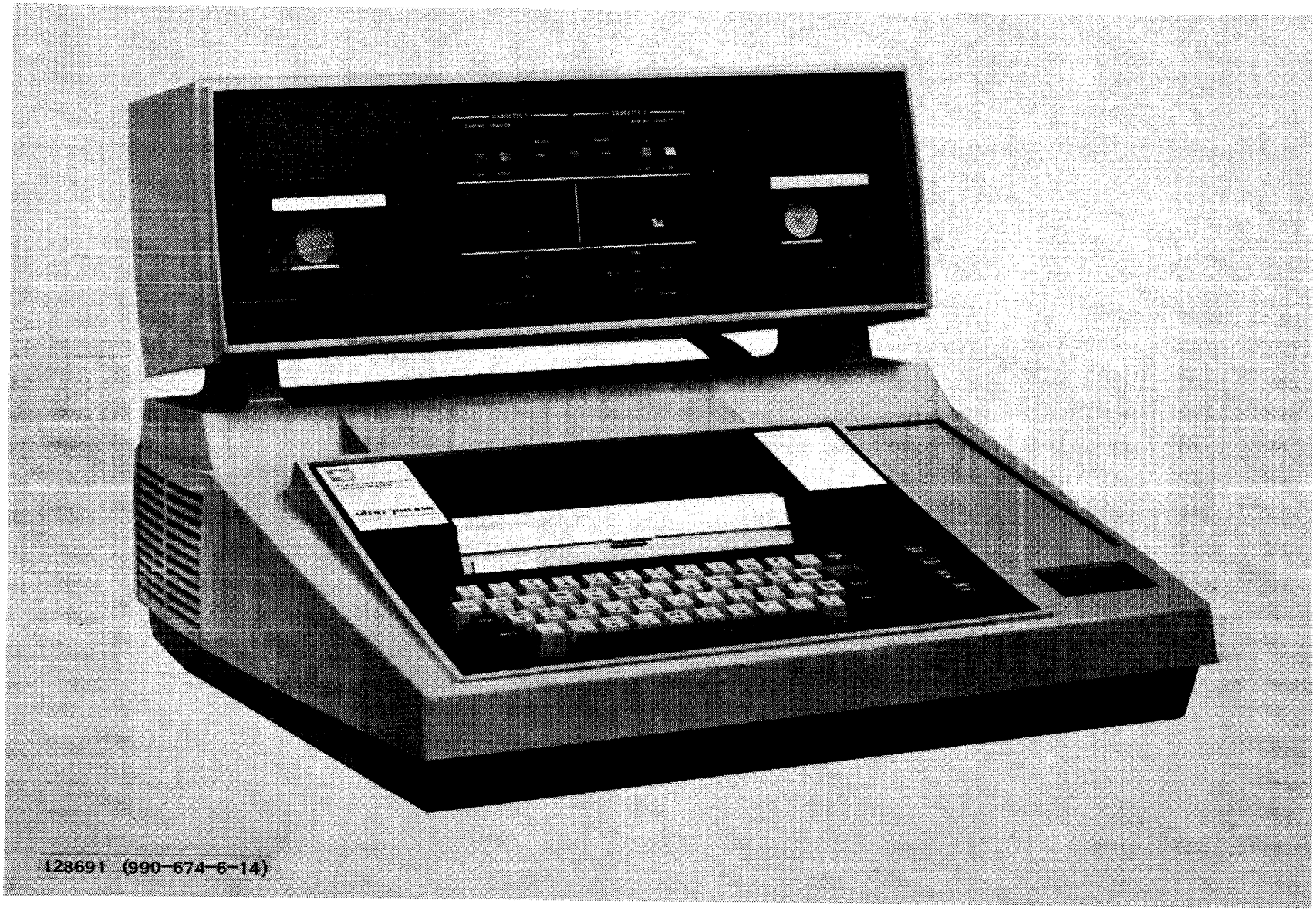


Figure 3-7. 733 ASR Data Terminal

- The 733 printer mechanism features a solid-state printhead that prints characters through the use of a five-by-seven dot matrix of heating elements, paper handling mechanics, and printhead movement devices.
- The transmit/receive electronics control communications with the 990 Computer.
- The record section controls recording of local (from the keyboard) or remote (from the computer) messages on magnetic tape.
- The playback section controls playback of messages that were previously recorded on magnetic cassettes for local use (printed by the printing mechanism) and/or transmission to the computer.

3.3.3 OPERATING CONTROLS, INDICATORS, AND KEYBOARD CHARACTERS

The following controls and indicators are used to control the 733 ASR Data Terminal and the 990 Computer. For more detailed information, refer to The Silent 700* Electronic Data Terminals manual.

3.3.3.1 CONTROLS. The controls for the data terminal are:

- Terminal - Power-On/Off, Speed-Low/Medium/High, Transmission Mode-Half/Full Duplex, Parity Select-Odd/Even/Mark, Terminal Status-On-Line/Off-Line, and Speed Status-Low/High (with 1200 baud option)
- Printer - Print Contrast, Line Space-Single/Double, Control-Line/Off/Local
- Cassette 1/Cassette 2 - Control-Record/Playback, Tape-Rewind/Stop and Load/Fast Forward/Stop
- Record Control - Control-Line/Off/Local and On/Off, Tape-Erase/Print, Tape Format-Line/Continuous
- Playback Control - Control-Line/Off/Local, Tape-Continuous/Start/Stop, Block Forward/Reverse, and Character Forward
- Keyboard - REPEAT, PAPER ADVANCE, LINE FEED, HERE IS, BREAK, CARRIAGE RETURN, TAPE, TAPE Keys, Control-Line/Off/Local

*Trademark of Texas Instruments Incorporated

3.3.3.2 INDICATORS. The following indicators show the status of the 733 Data Terminal:

- Terminal - "Power On" light
- Cassette 1/Cassette 2 - "READY", "END", "RECORD", and "PLAYBACK" lights.
- Record Control - "RECORD ON" light and CHARACTER display
- Playback Control - "PLAYBACK ON" light and "PLAYBACK ERROR" light

3.3.3.3 KEYBOARD CHARACTERS. The following characters are available on the keyboard of the Model 733 ASR Data Terminal (see table 3-2 for USASCII code set):

Table 3-2. USASCII Code Systems and Character Set

b_4 b_3 b_2 b_1	b_7→0 b_6→0 b_5→0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 0 0	NUL	DLE	SPACE	0	@	P	`	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	¦
1 1 0 1	CR	GS	-	=	M	]	m	}
1 1 1 0	SO	RS	.	>	N	^	n	~
1 1 1 1	SI	US	/	?	O	_	o	DEL

PRINTABLE CHARACTER

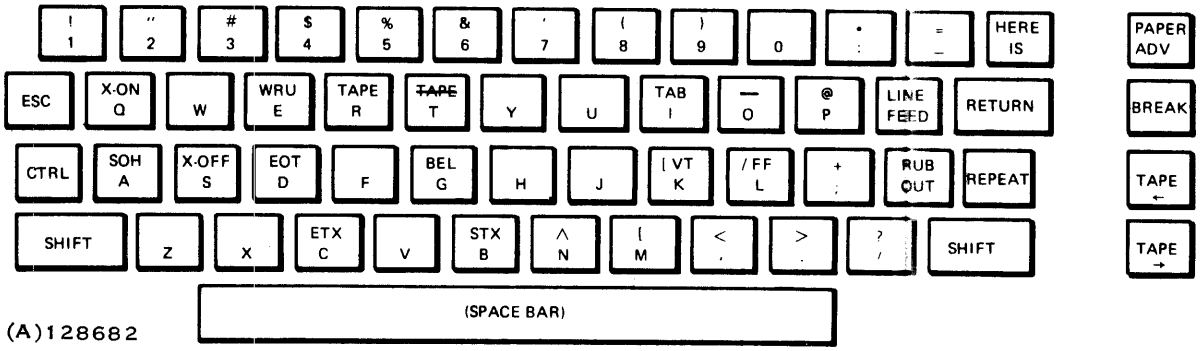
PRINTER CONTROL CHARACTER

AUXILIARY DEVICE CONTROL CHARACTER

CODES GENERATED BY KEYBOARD, BUT NO ACTION TAKEN

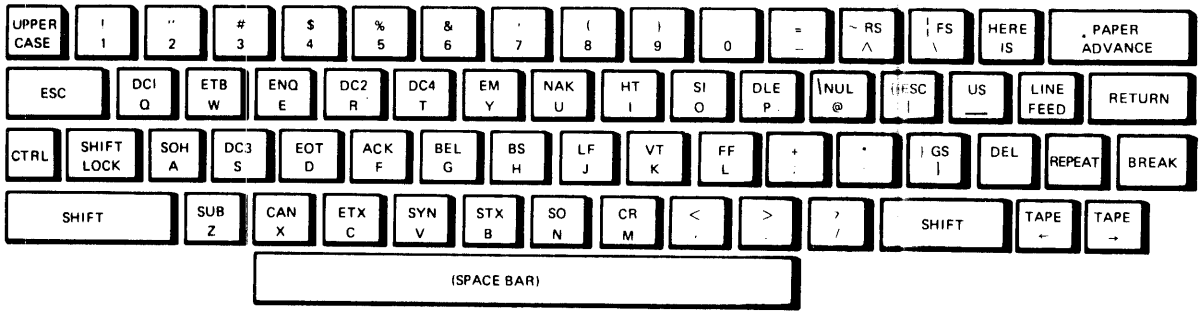
(A)128680

- Standard USASCII Keyboard - 68 printable characters and 33 control characters in the following arrangement:



(A)128682

- Optional Full USASCII Keyboard - 95 printable characters and 33 control characters in the following arrangement (see table 3-2 for USASCII code set):



LEGEND:

= Control Character
= Alphabetic character (SHIFT for uppercase)
= Shifted character
= Unshifted character
= Shifted character, control character
= Graphic unshifted

(A)128681

3.3.4 CAPABILITIES

The 733 ASR Data Terminal permits the programmer to input/output system development software, application programs, and control information for the 990 Computer.

3.3.5 TERMINAL SPECIFICATIONS

3.3.5.1 TRANSMISSION CODE AND CHARACTER SET. The transmission code is USASCII, 7-level, 10 bits per character that include 7 data bits, a parity bit, a start bit, and a stop bit. The 733 has 95 printable characters, four print control characters, one terminal control character, and four additional terminal control characters, DC1 through DC4, which are optional. Communication between the 990 Computer and the 733 ASR Data Terminal occurs at 1200 bits per second. When operating the printer, the computer sends the delete (DEL) character between printable characters to effectively match the 300-bit-per-second printer rate of the data terminal.

3.3.5.2 DATA FORMAT AND TRANSMISSION. Data is routed within the terminal via a single data bus. The data is sent serially by bit with eight bits per character. The eight bits include a 7-bit USASCII character and an end-of-block indicator in the ASR unit. Transmission speed is 1200 Baud.

3.3.5.3 PRINTER. Refer to table 3-3 for the printer specifications.

Table 3-3. Printer Specifications

Specification	Value
Printing method	5 x 7 dot matrix, electronically heated, on heat-sensitive paper
Line length	7.9 inches, 80 characters
Character spacing	0.1 inch, character center to center
Line spacing	Six or three lines per inch (single or double spaced)
Paper (TI part number 213714-0001 or 953167-0001)	Roll, 8.5 inches wide by 3.625 inches maximum diameter (300 feet), heat-sensitive
Platen	Friction feed
Carriage return time	195 milliseconds maximum
Line feed time	33 milliseconds maximum (single space), 66 milliseconds maximum (double space)
Audible alarm time	250 (±50) milliseconds on receipt of the BEL character
Printable characters	95
Carriage return and line feed (CR/LF)	Automatic at column 81, no code is transmitted
Visibility of printed lines	At least 50 previous lines of print (including line and character being printed) are visible and unobstructed
Print contrast	Operator adjustable

3.3.5.4 COMMUNICATION LINE INTERFACE. The standard line interface conforms to the EIA standard RS232C. The terminal can receive, without error, signals with mark and space distortion of up to 45 percent. The minimum stop bit time for error-free reception at any speed is 0.6 of a normal bit time.

3.3.5.5 TAPE TRANSPORT. Refer to table 3-4 for the tape transport specifications.

Table 3-4. Tape Transport Specifications

Specification	Description
Recording speed	8 inches per second
Recording method	Phase-encoding
Recording density	800 bits per inch (1600 flux changes per inch)
Rewind time	60 seconds maximum
Playback speed	120 characters per second (to communication line or printer) or 250 characters per second (duplication)
Tape drive	Capstan drive for recording or playback
Error rate	One in 10^{16} maximum, using certified cassette tapes and proper head cleaning procedures; one in 10^{7} typical
Interchangeability	Any tape recorded on any 733 ASR transport operating within specifications may be read on any other 733 ASR of the same model operating within specifications
Sensors	EOT, BOT, cassette in place, write tab, and transport door closed
Media	Improved Philips type cassette containing 275 to 300 feet of digital grade magnetic tape with approximately 20 inches of transparent tape joined to each end.

3.3.6 INSTALLATION

3.3.6.1 ADAPTER CARD INFORMATION. The 733 ASR data terminal requires a TTY/EIA Module (TI part number 961642-0003) and an adapter card (TI part number 975200-0001) for interface connections on the CRU of the 990 Computer.

3.3.6.2 POWER REQUIREMENTS. The power requirements for the 733 ASR are as follows:

- Frequency - Normal operation with primary input frequencies in the band of 48 to 62 Hz.
- Voltage - 115 (+10%, -15%) volts RMS
- Power - Required primary input power at maximum rated voltage is 200 VA maximum

3.3.6.3 SPACE REQUIREMENTS. Space requirements for the 733 ASR are shown in figure 3-8.

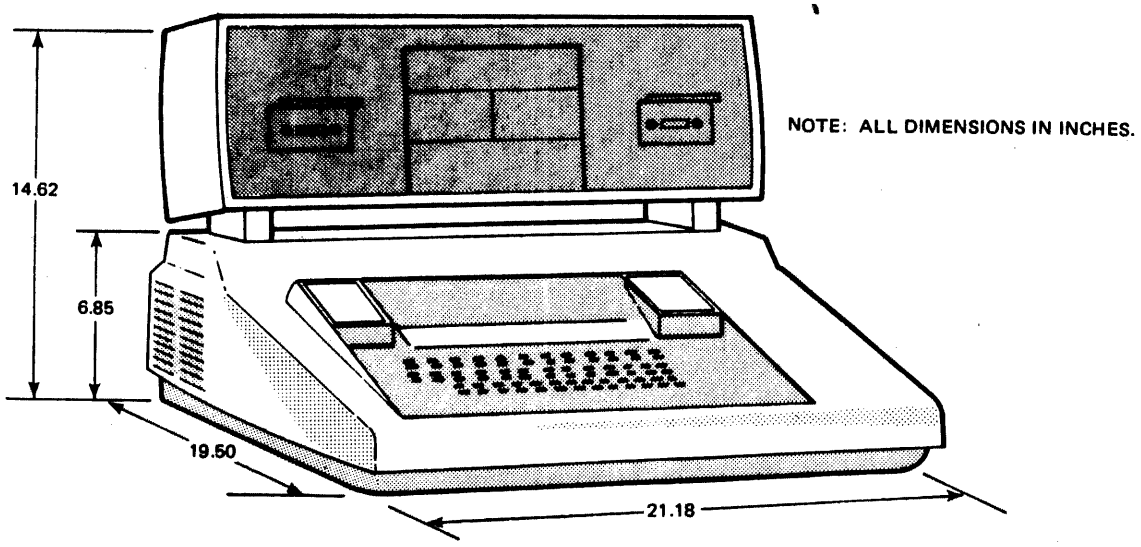


Figure 3-8. Space Requirements

3.3.7 PERIPHERAL KIT OPTIONS (Kit part number 974707)

The following list identifies the various options available with the 733 ASR peripheral kit:

Kit Part Number	Description of Kit Contents
974707-0001	733 ASR Peripheral Kit, 110 Vac, 60 Hz This kit includes the 733 ASR, Model 960 TTY/EIA card, interface cable, and adapter card
974707-0002	733 ASR Peripheral Kit, Interface This kit includes the Model 960 TTY/EIA card interface cable, and adapter card
974707-0003	733 ASR Peripheral Kit, ASR This kit contains only the 733 ASR.
974707-0990	733 ASR Peripheral Kit, Documentation This kit includes pertinent documentation, including PAT, for the 733 ASR.

3.4 MODEL 33 ASR TELETYPEWRITER DATA TERMINAL

3.4.1 GENERAL

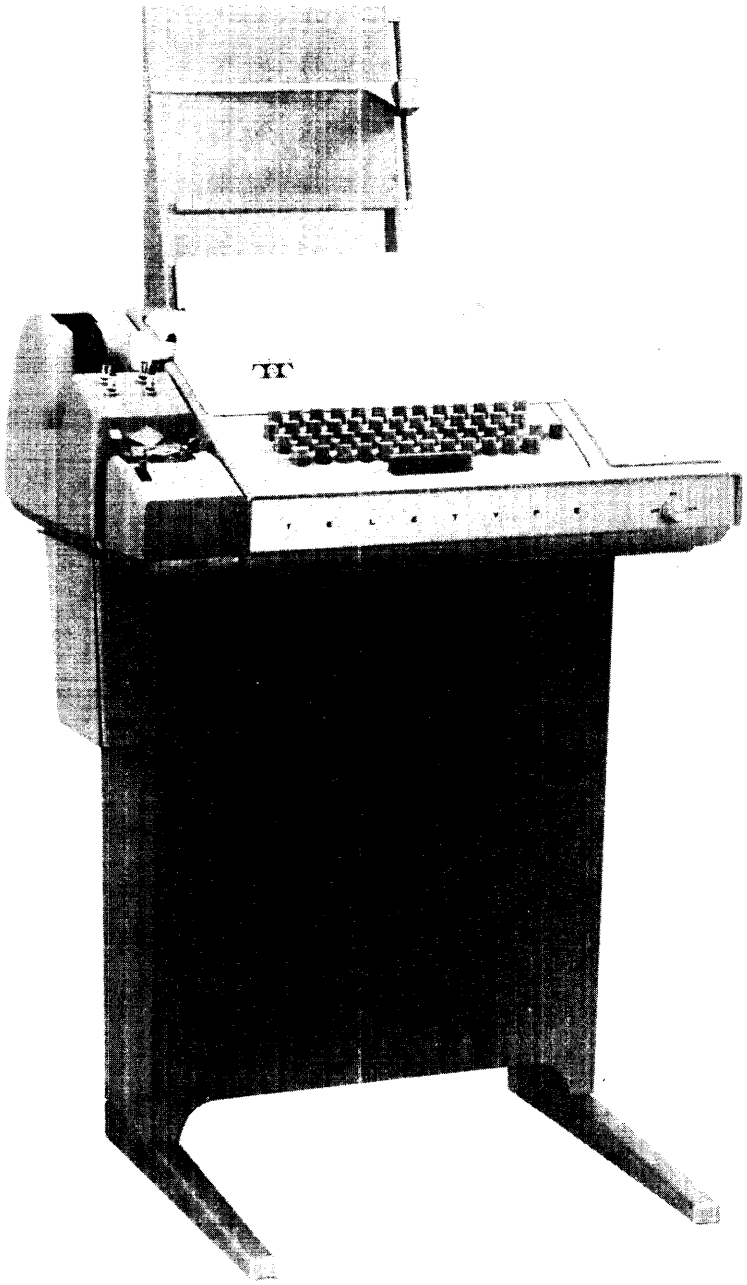
The Model 33 ASR Teletypewriter Data Terminal may be used as system keyboard/printer and principal input/output device for the 990 Computer. Input/output media for this configuration is paper tape. Printed output from the 990 Computer appears on this terminal.

3.4.2 DESCRIPTION

The Model 33 ASR Teletypewriter (figure 3-9) is a USASCII data terminal used as an I/O device for the 990 Computer. The functions of the 33 ASR teletypewriter are:

- The 33 ASR keyboard provides manual typing operations and transmission of printable characters and operational codes in 7-level USASCII code. Only uppercase alphanumeric characters are printed.
- The printer mechanism uses impact printing of all printable characters on teletypewriter paper or on multiple copy business forms.
- Transmission from the data terminal to the 990 Computer is from either the keyboard or the paper tape reader.
- Reception from the 990 Computer is routed to the printing mechanism and to the paper tape punch. Tape is punched only when the punch is on.

Note that in the off-line mode, paper tape may be read by the reader or punched on the punch, or paper tape may be printed by the printing mechanism.



128693 (990-674-6-16)

Figure 3-9. 33 ASR Teletypewriter

3.4.3 OPERATING CONTROLS

The controls and indicators described in the following paragraphs are used to control the 33 ASR Teletypewriter and the 990 Computer. For more detailed information on the Model 33 ASR Teletypewriter, refer to the supplied vendor manual.

3.4.3.1 CONTROLS. The controls for the data terminal are:

- Terminal - LOCAL/OFF/LINE
- Punch - ON/OFF/B.SP./REL (figure 3-10)
- Reader - MANUAL START/AUTO/MANUAL STOP/FREE (figure 3-11)
- Keyboard - HERE IS/LINE FEED/REPT/BREAK

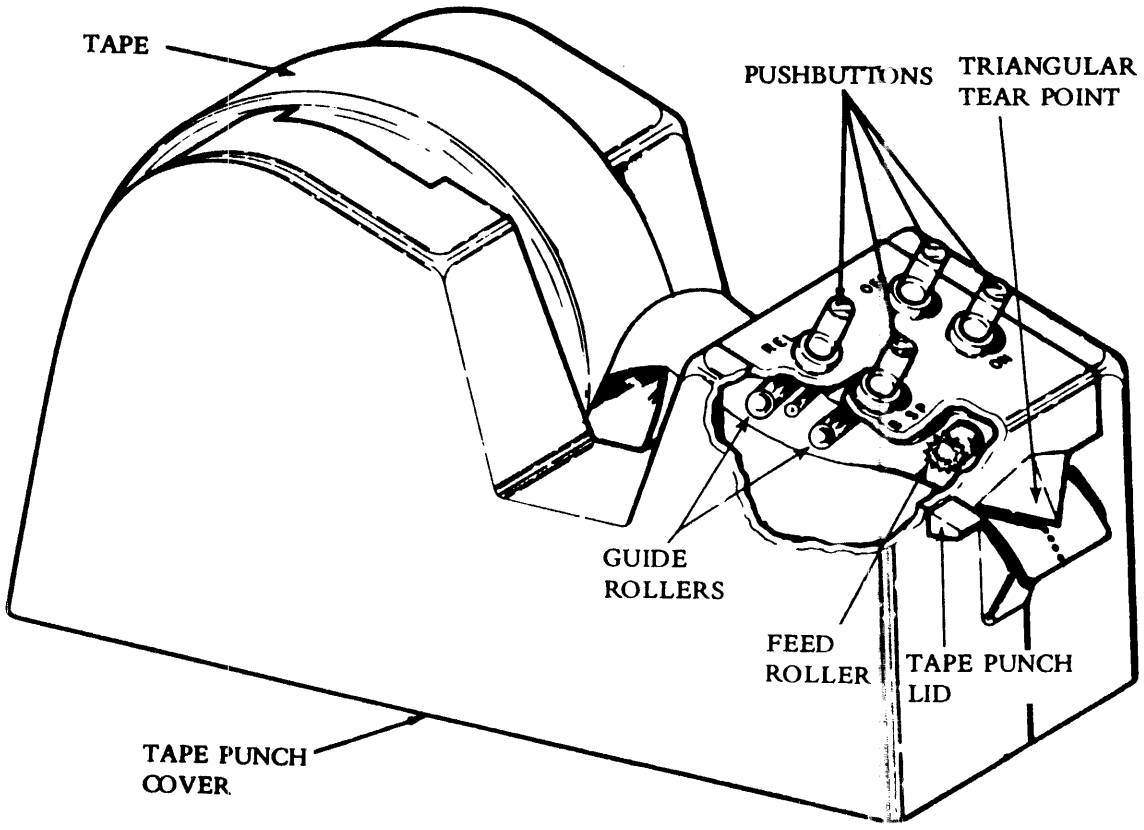


Figure 3-10. Paper Tape Punch Controls

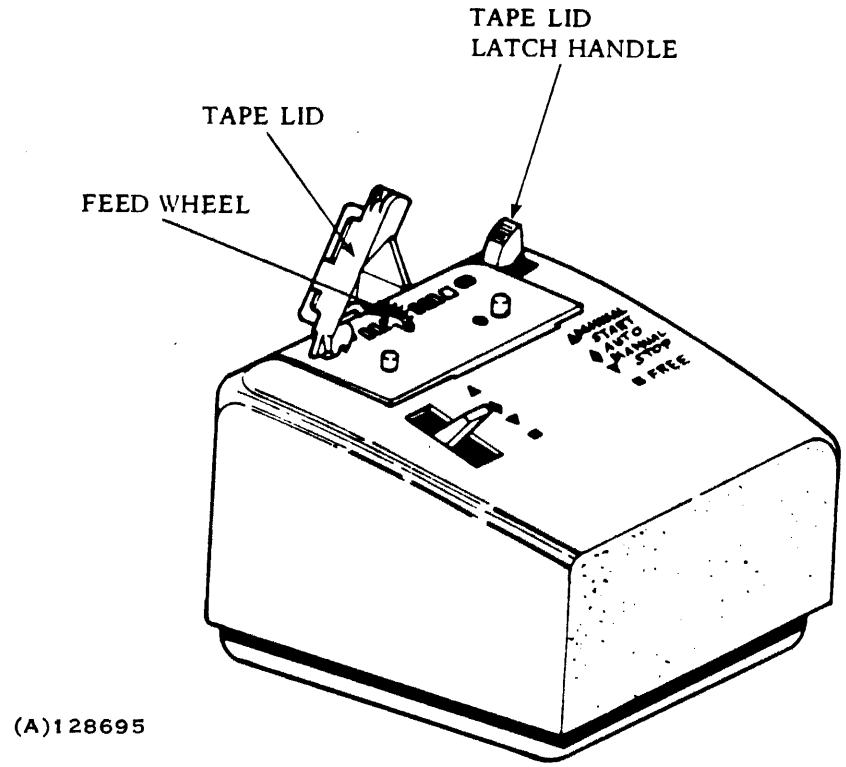
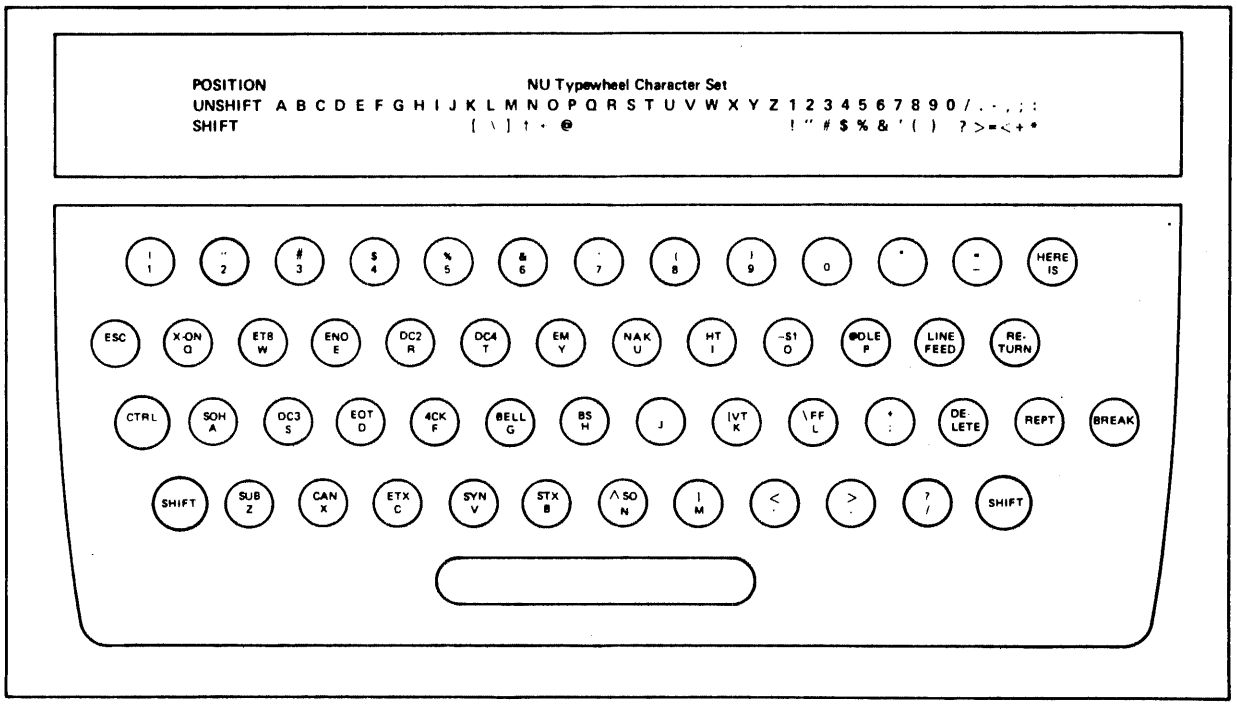


Figure 3-11. Paper Tape Reader Controls

3.4.3.2 KEYBOARD. The following characters are available on the keyboard for USASCII Teletype models (see table 3-2 for the USASCII code set):



3.4.4 CAPABILITIES

The 33 ASR Data Terminal permits the programmer to input/output system development software, application programs, and control information for the 990 Computer.

3.4.5 TERMINAL SPECIFICATIONS

3.4.5.1 TRANSMISSION CODE AND CHARACTER SET. The transmission code and character set is the same as shown in paragraph 3.3.5.

3.4.5.2 DATA FORMAT AND TRANSMISSION. The 7-level USASCII code is transmitted to the 990 Computer interface in the full duplex mode. The Baud rate available with the Model 33 ASR Teletypewriter is 110.

3.4.5.3 PRINTER. Refer to table 3-5 for the printer specifications.

Table 3-5. Teletypewriter Specifications

Item	Description
Page copy	
Rolled paper	
Outside diameter	5 in. maximum
Page width	8.453 ±0.031 in.
Length per roll (approx.)	400 ft.
Core diameter I.D.	1 in. + 0.1 - 0.05 in.
Ribbon	Ink-impregnated nylon
Paper tape	
Type paper	Oiled stock, rolled
Width	1 in.
Thickness	0.004 in.
Maximum roll diameter	8 in.
Length per roll (approx.)	1000 ft
Roll core diameter	2 in.
Environmental requirements	
Operating	40° to 110° F ambient measured outside of terminal cover
Storage	40° to 150° F
Relative humidity	90% maximum at 100° F maximum
Maintenance interval	Initial lubrication is required after 100 to 200 hours of operation. Thereafter, lubricate every 750 operating hours or every 6 months, whichever occurs first.

3.4.5.4 COMMUNICATIONS LINE INTERFACE. The 33 ASR Teletypewriter interfaces via a current loop that is a wirable option on the EIA communications module.

3.4.6 INSTALLATION

3.4.6.1 CONNECTION INFORMATION. The 33 ASR Teletypewriter requires a TTY/EIA module (TI part number 961642-0003) and a small card adapter (TI part number 975200-0001) for interface to the 990 CRU. Cables are supplied with the peripheral kits to interconnect the TTY/EIA card and the 33 ASR Teletypewriter.

3.4.6.2 POWER REQUIREMENTS. The power requirements for the Model 33 ASR Teletypewriter are:

- Frequency - Normal operation with primary input power at either 50 or 60 Hz.
- Voltage - 115 ± 10% Vac single phase.
- Power - 250 Watts (nominal), starting surge current 15A maximum, operating current 3A nominal (5A maximum).

3.4.6.3 TELETYPEWRITER MODIFICATION (Model 3320/5JE). If the data terminal is a teletypewriter purchased from Texas Instruments Incorporated, the teletypewriter is modified for improved noise suppression, full duplex communications, 20 ma neutral signaling, removal of answer back functions, parity generation disabled, and automatic carriage return and line feed enabled. The following procedures completely define the modifications for user-supplied terminals.

1. Use the following procedures to remove the top cover from the teletypewriter for access to the area for the remaining modifications.
   a. Remove the paper roll and paper tape (if installed).
   b. Remove the paper advance (platen) knob.
   c. Remove the knob from the LINE/OFF/LOCAL switch.
   d. Remove the teletypewriter nameplate strip by pulling it down and out.
   e. Remove the 4 screws uncovered by the removal of the nameplate strip.
   f. Remove the 3 thumbscrews from the rear of the cover.
   g. Remove the screw located on the left rear corner of the tape reader cover.
   h. Lift off the cover top.

2. Use the following procedure to modify the teletypewriter for full duplex operation.

   a. Locate the "X" terminal board at the left rear (viewed from the rear) of the machine.

   b. Move the white/blue wire from terminal 4 to terminal 5 on the 'X' terminal block.

   c. Move the brown/yellow wire from terminal 3 to terminal 5 on the 'X' terminal block.

3. Use the following procedures to modify the teletypewriter for 20 ma neutral signaling.

   a. Move the purple wire from terminal 8 to terminal 9 on the 'X' terminal block.

   b. Move the blue wire from terminal 3 to terminal 4 on the 'X' terminal block.

4. Use the following procedure to modify the teletypewriter for improved noise suppression.

   a. Remove the green/black wire from terminal 8 of the 'X' terminal block, insulate the wire end to prevent inadvertant electrical connections, and tie the wire end back out of the way.

   b. Move the 2 black wires from terminal 2 to terminal 8 of the 'X' terminal block.

   c. Move the two white wires from terminal 1 to terminal 2 of the 'X' terminal block.

   d. Install the capacitor and inductor as shown in figure 3-12.

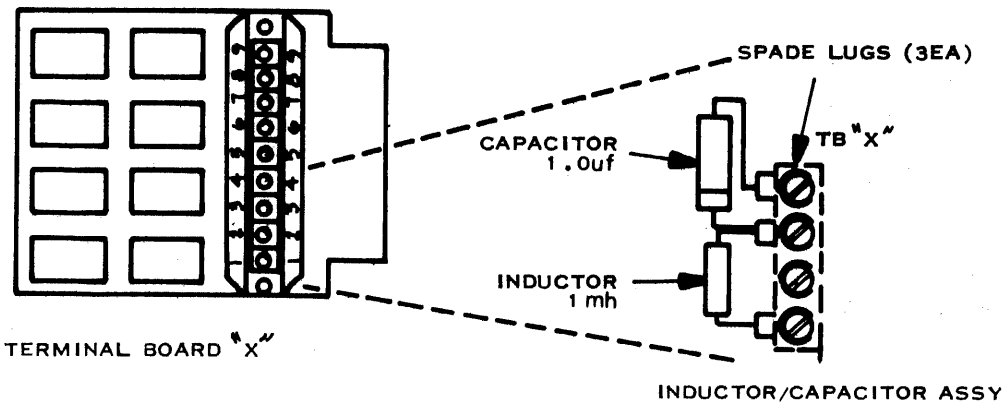
5. Use figure 3-13 to remove the answer back and WRU function bars from the teletypewriter.

6. Use the following procedure and figure 3-13 to disable the parity generation function and enable the automatic carriage return and line feed functions.

   a. Remove the white/blue wire from the left-hand terminal of the terminal block located on the right side of the teletypewriter below the keyboard, insulate the end of the wire, and tie back out of the way.

   b. Remove the copper-colored clip from the "A" position on the code bar. The clip is located below the print mechanism.

7. Re-install the carrier top by performing the procedure in step 1 in reverse order (installing where removed and in reverse order, h through a).



(A)128696

Figure 3-12. Capacitor and Inductor Installation

3.4.6.4 DIMENSIONS AND WEIGHT.

Terminal:

Width . . . . . 22 inches

Depth . . . . . 18-1/2 inches

Height . . . . . 8-3/8 inches

Weight . . . . . 44 pounds

Stand:

Width . . . . . 17-3/4 inches

Height . . . . . 24-1/2 inches

Depth (at top of enclosure) . . . . . 6-1/2 inches

Length of Feet . . . . . 17-3/4 inches

Weight . . . . . 12 pounds

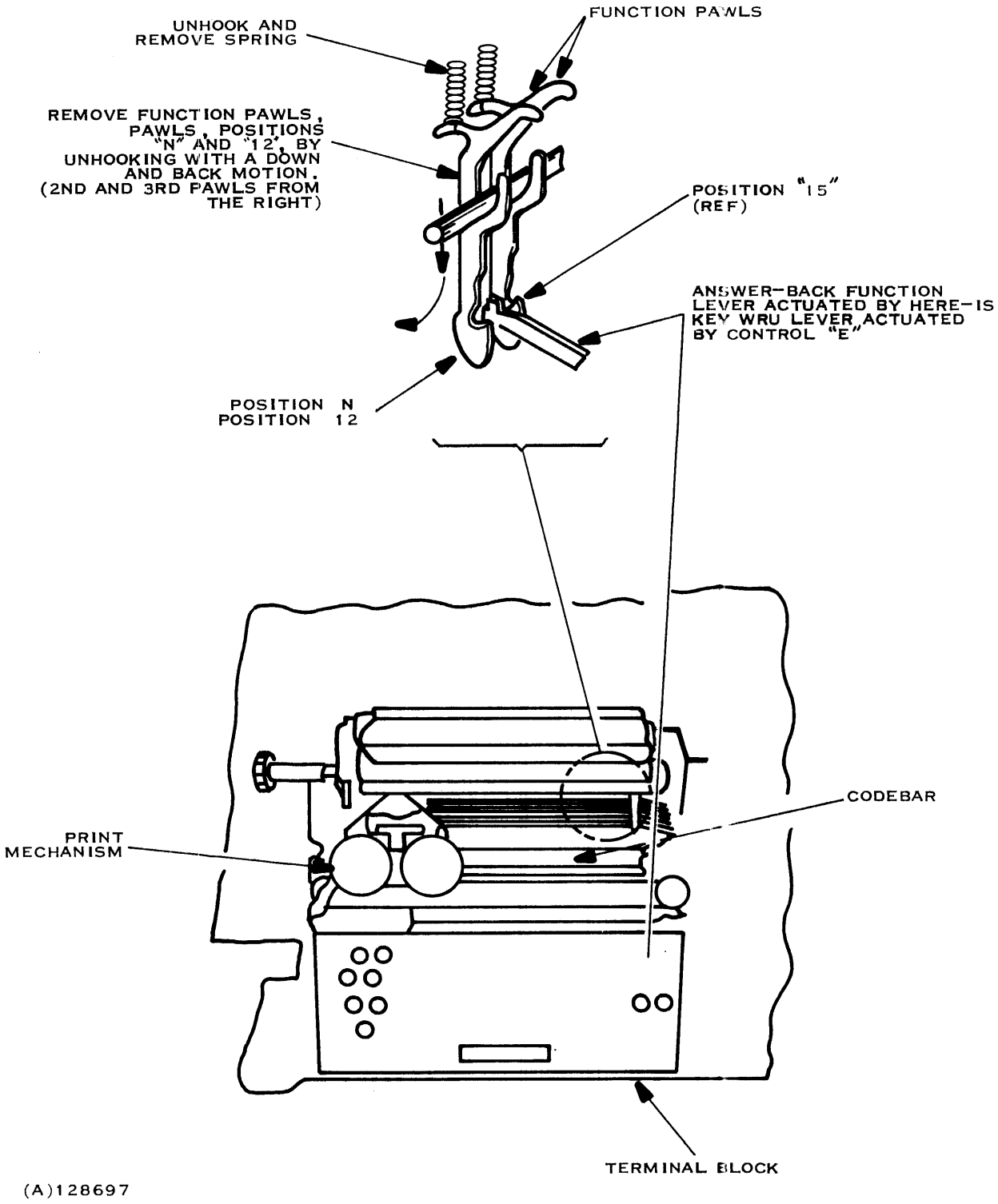


Figure 3-13. Answer Back and WRU Function Bar Removal

3.4.7 PERIPHERAL KIT OPTIONS (Kit part number 974704)

The following list identifies the various options available with the 33 ASR peripheral kit:

Kit Part Number	Description of Kit Contents
974704-0001	Teletype Peripheral Kit, 33 ASR - 3320/5JE, 60 Hz This kit includes a modified teletype, a TTY/EIA card, interface cable, test procedure, a documentation kit (for reference only), and an adapter card.
974704-0002	Teletype Peripheral Kit, ASR 3320/5JE, Interface This kit includes the TTY/EIA card, interface cable, and adapter card.
974704-0003	Teletype peripheral Kit, ASR 3320/5JE, TTY This kit includes only the modified TTY.
974704-0990	Teletype Peripheral Kit, ASR-3320/5JE, Documentation This kit includes pertinent documentation, including PAT, for the 33 ASR.

3.5 MODEM CONTROLLER COMMUNICATION I/O MODULE

3.5.1 GENERAL

The modem controller interfaces the 990 Computer to the switched telephone (DDD) network via a Bell Systems CBS 1001A Data Access Arrangement (DAA) in order to provide automatic call origination, termination, and answer functions. The modem can also function with a 4-wire full duplex leased line network.

3.5.2 DESCRIPTION

There are three functional sections of the modem controller (figure 3-14) that provide the functions listed above; modem, automatic call unit (ACU), and the cyclic redundancy check (CRC).

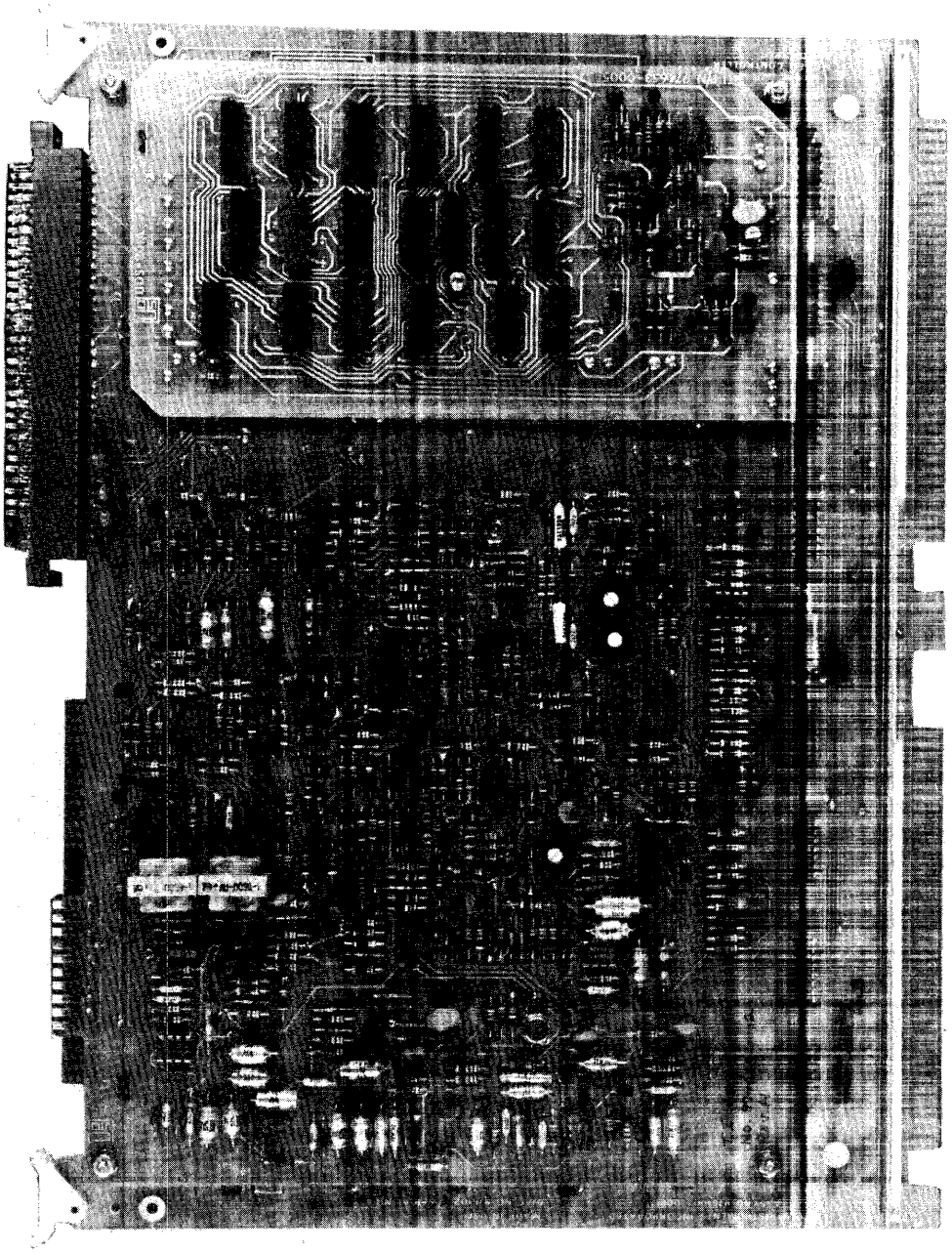


Figure 3-14. Modem Controller and Modem

3.5.2.1 MODEM. The modem provides data transmission capabilities that are functionally compatible with Bell System 202 type modems. The functions provided are:

- Auto answer with CBS 1001A DAA
- Call turnaround delay
- Receive mark-hold with carrier off
- Enable local copy
- 200 milliseconds clear-to-send delay

In order for these functions to operate correctly, the 990 Computer must be programmed to handle the following:

- Auto answer control via DTR
- Maintaining mark-hold with clear-to-send off
- Ignoring local copy except for diagnostic purposes

3.5.2.2 AUTOMATIC CALL UNIT (ACU). If the terminal is intended for automatic call origination via the Bell CBS DAA, the ACU control circuits incorporated as part of the modem provide the following functions:

- Dial tone detection
- BCD - pulse or BCD - touch tone transmission
- Interdigit interval timing
- Busy tone detection
- Line monitor control
- Tandem dial control

To provide these functions, the 990 Computer must be programmed to accomplish the following:

- Proper control sequences
- Digit presentation in BCD form
- Abandon call and retry timing
- Successive call separation

3.5.2.3 CYCLIC REDUNDANCY CHECK (CRC). The CRC module permits the generation of the transmitting, receiving, and decoding of a 16-bit block check character when enabled.

3.5.3 CAPABILITIES

The modem controller is capable of automatic call origination, call termination, and automatic call answer functions when the 990 Computer is programmed to accomplish these requirements.

3.5.4 MODEM CONTROLLER SPECIFICATIONS

The specification of the three parts of the modem controller are shown in table 3-6.

3.5.5 INSTALLATION

3.5.5.1 CABLE CONNECTION INFORMATION. The modem controller interconnections to the CRU in the 990 Computer are shown in figure 3-15, those to the DAA interface are contained in table 3-7, and those from the controller to the modem/ACU are contained in table 3-8.

3.5.5.2 POWER REQUIREMENTS. The power requirements for the modem controller are as follows:

- +5Vdc @ 1.0 A, ± 3%
- +12Vdc @ 60 ma, ± 5%
- -12Vdc @ 110 ma, ± 5%

Table 3-6. Modem Controller Specifications

Module	Item	Specification
Modem	Data	Binary, serial, asynchronous
	Data rate	0 to 1200 bits per second
	Transmitter	Frequency: Stability: ±5% Transmit Level: 0 to -12 dbm in 2 db steps Transmit impedence: 600 + 10% ohms resistive between 500 - 3500 Hz
	Timing	REQUEST-TO-SEND/CLEAR-TO-SEND: 200 ms ± 20 ms
	Delay	Turnaround: Receiver squelched for 110 ms ± 25 ms after REQUEST-TO-SEND goes off
	Receiver sensitivity	-45 dbm ± 3 db
	Carrier detection	40 ms ± 10 ms of received carrier to turn DATA-CARRIER-DETECT on 10 ms ± 2 ms of no carrier to turn DATA-CARRIER-DETECT off
	Modulation	Phase coherent, amplitude stable FSK Data frequencies: Mark = 1200 Hz; Space = 2200 Hz accurate to ±0.5%
Control	Auto answer tone	2025 Hz ±25 Hz
	Answer tone transmit time	3.0 ± 1 second
	Auto answer control	Data terminal ready
	Data set ready	Follows CCT in answer, requires at least 100 ms of answer back tone in originator
	Data coupler required	Bell CBS (1001A) or equivalent
Automatic Call unit	Line monitor	Turned off and on by dialing control digits, also turned off when data terminal ready is turned off
	Dial tone detect	200 to 800 Hz, 0 to -25 dbm
	Tone detect delay	700 ± 200 ms for standard dial tone between -10 and -25 dbm

Table 3-6. Modem Controller Specifications (Continued)

Module	Item	Specification
	Busy Signals	7 ± standard line busy tones or 14 ± standard line busy tones at -32 dbm
	Pulse dial requirements	Pulse width: 100 ms ± 10% Duty cycle: 61% ± 3% Interdigit interval: 600 to 1700 ms
	Touch tone dial rqmts	Tones on: At least 50 ms Tones off: At least 45 ms Cycle time: 100 ms minimum Transmit level: within +1, -2 db of the average data signal power
Cyclic Redundancy Check	Block check character	16 bits
	Polynomial generator	$X^{16} + X^{15} + X^2 + 1$
	Control	Enabled by CRU bits

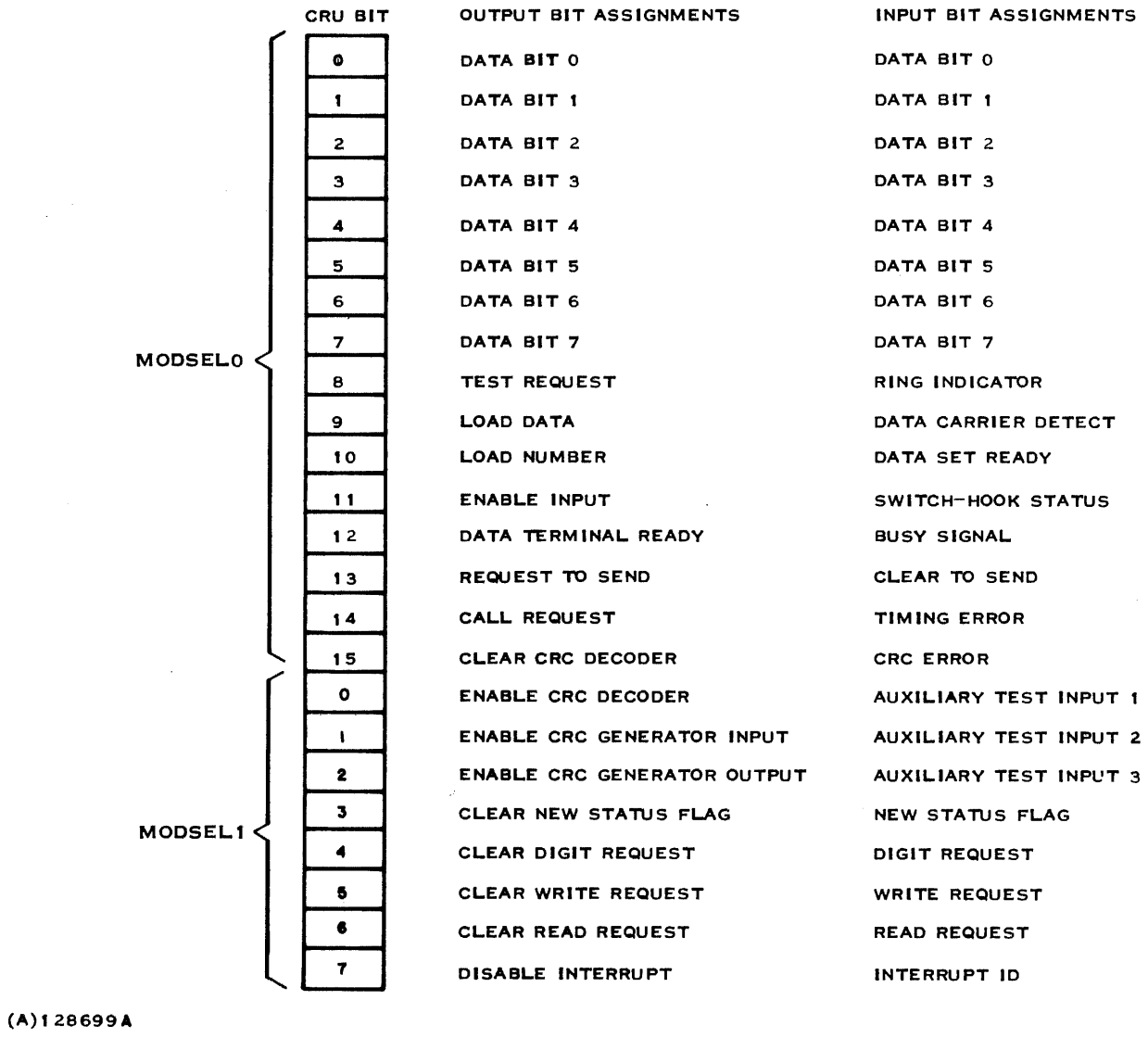


Figure 3-15. CRU Input/Output Bit Assignments

Table 3-7. Modem/ACU to DAA Line Connections

Signal	Mnemonic	Level On	Level Off	Pin Number
Off Hook	OH	+8	-8	P2-1, A
Data Transmission	DA	+8	-8	P2-2, B
Coupler Cut Through	CCT	+8	-8	P2-7, H
Ring Indicator	RI	+8	-8	P2-8, J
Data Tip	DT			P2-5, E

Table 3-7. Modem/ACU to DAA Line Connections (Continued)

Signal	Mnemonic	Level On	Level Off	Pin Number
Data Ring	DR			P2-9, K
Switch Hook	SH	+8	-8	P2-6, F
Signal Ground	SG			P2-3, C
Transmit Line (4-wire)	H			P2-10
	L			P2-L
Receive Line (4-wire)	H			P2-4
	L			P2-D

Table 3-8. Controller to Modem/ACU Interface

Signal	Mnemonic	Connector Pin Number
Transmit Data	TDATA	P1-h, 29
Call Request	CRQ	P1-E, 5
Request To Send	RTS	P1-b, 24
Test Request	TRQ	P1-D, 4
Data Terminal Ready	DTR	P1-Y, 21
+5v		P1-j, 30
Digit Present	DPR	P1-J, 8
+12v		P1-1, 32
Digit Number Bit 1	NB1	P1-U, 17
-12v		P1-m, 33
Digit Number Bit 2	NB2	P1-L, 10
Common	GND	P1-K, 31
Digit Number Bit 4	NB4	P1-R, 14
Digit Number Bit 8	NB8	P1-k, 9
Received Data	RDATA	P1-F, 6
Line Monitor Output	MH	P1-M, 11

Table 3-8. Controller to Modem/ACU Interface (Continued)

Signal	Mnemonic	Connector Pin Number
Clear-To-Send	CTS	P1-f, 28
Switch-Hook Status	SH	P1-c, 25
Data Carrier Detect	DCD	P1-V, 18
Data Set Ready	DSR	P1-Z, 22
Ring Indicator	RI	P1-d, 26
Present Next Digit	PND	P1-p, 13
Busy Signal	BSY	P1-w, 19

3.5.6 PERIPHERAL KIT OPTIONS (Kit Part Number 974709)

The following list identifies the various options available with the Modem Controller Peripheral Kit:

Kit Part Number	Description of Kit Contents
974709-0001	Modem Peripheral Kit, Asynchronous, 1200 Baud Pulse Dialer, 2-Wire. This kit contains a pulse dialer, 2-wire modem, controller, and the DAA interface cable.
974709-0002	Modem Peripheral Kit, Asynchronous, 1200 Baud, Touch Tone Dialer, 2-Wire. This kit contains the same part as does the -0001 kit but contains a touch tone dialer instead of a pulse dialer.
974709-0003	Modem Peripheral Kit, Asynchronous, 1200 Baud, 4-Wire Leased Line. This kit contains a 4-wire leased line modem, controller, and DAA interface cable.
974709-0990	Modem Peripheral Kit, Documentation This kit contains pertinent documentation for the modem controller peripheral kit.

3.6 ASYNCHRONOUS TTY/EIA COMMUNICATIONS INTERFACE MODULE

3.6.1 GENERAL

The full duplex asynchronous TTY/EIA communications interface module interfaces the 990 Computer to a variety of RS232C compatible equipment via the CRU.

3.6.2 DESCRIPTION

The TTY/EIA module (figure 3-16) is operated under program control to send and receive character data and data terminal control signals. Full duplex communications is via a voltage interface compatible with EIA standard

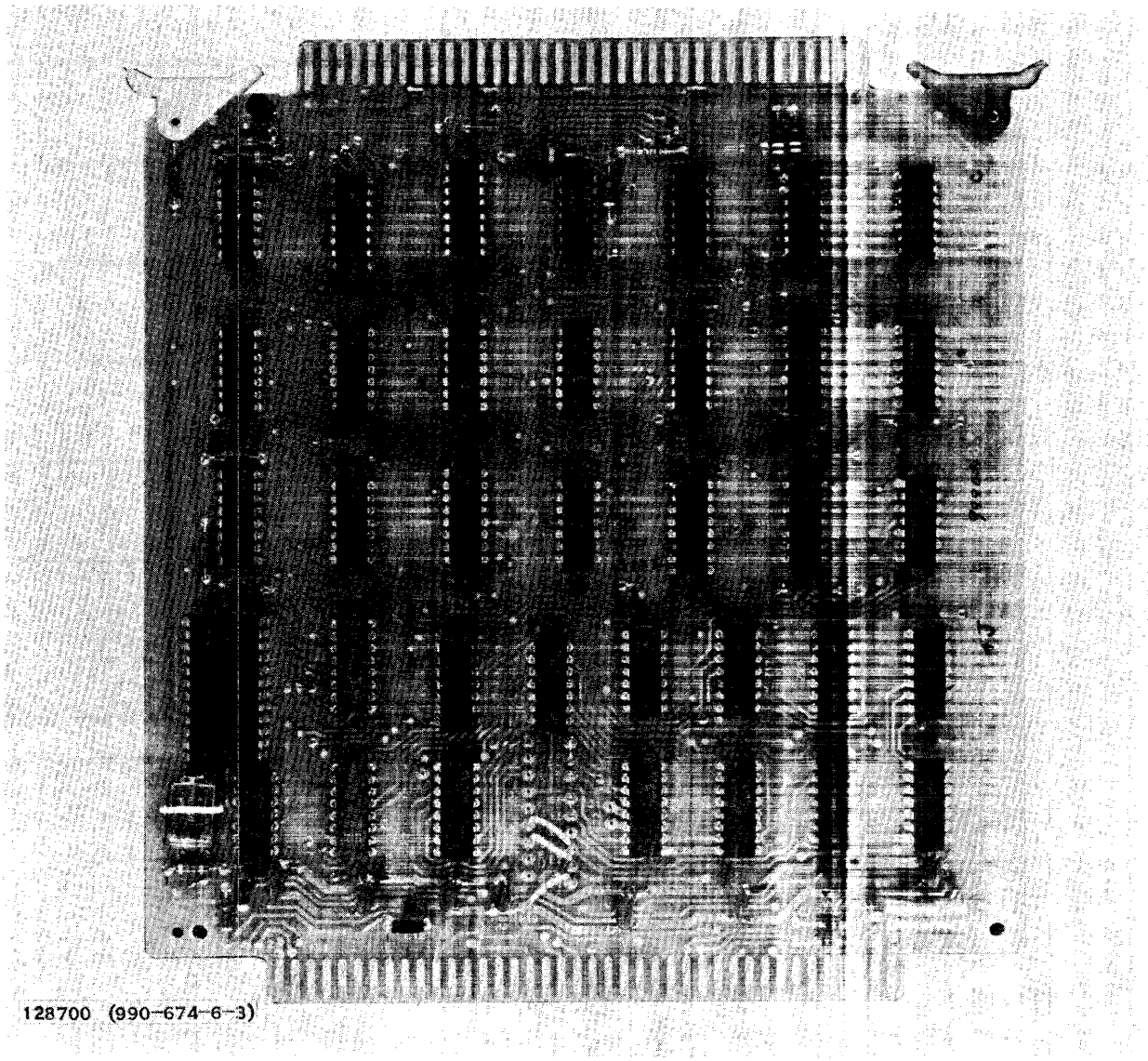


Figure 3-16. Asynchronous TTY/EIA Communications Interface Module

RS232C. Available as an option is 20 ma neutral signaling. Input from the data terminal is converted to CRU interface signals by the module and the signals are used by the 990 Computer to determine terminal status, activity requirements, and user input requirements of the data terminal. The module may be wired to generate interrupts to the 990 Computer. Interrupts are generated for each character sent, each character received, and when (RS232C only) the status of the attached device changes.

3.6.3 OPERATION

Operation of the TTY/EIA module is programmed to accommodate the device actually connected to the module.

3.6.4 CAPABILITIES

The TTY/EIA module permits the connection of TTY compatible or EIA compatible devices to the 990 Computer CRU for communication interaction between the attached devices and the computer.

3.6.5 MODULE SPECIFICATIONS

Wirable options on the module permit the selection of the Baud rate of the device and the selection of the type of interface from either TTY compatible or EIA compatible specifications. The Baud rate may be selected from several rates that begin at 110 Baud and extend through 9600 Baud. An 11-bit code is used with the 110 Baud rate while 10-bit codes are used with higher Baud rates.

3.6.6 INSTALLATION

3.6.6.1 PROGRAMMING INFORMATION. Figure 3-17 shows the module-to-CRU signal requirements and table 3-9 defines these input signals. Figure 3-18 shows the CRU-to-module signals and table 3-10 defines these output signals.

3.6.6.2 POWER REQUIREMENTS. The EIA module requires the following dc power:

- +5 Vdc 0.7 amp
- ±12 Vdc 0.06 amp (EIA option, line driver)

3.6.6.3 ADAPTER CARD INFORMATION. The EIA module requires the small card adapter, TI part number 975200-0001.

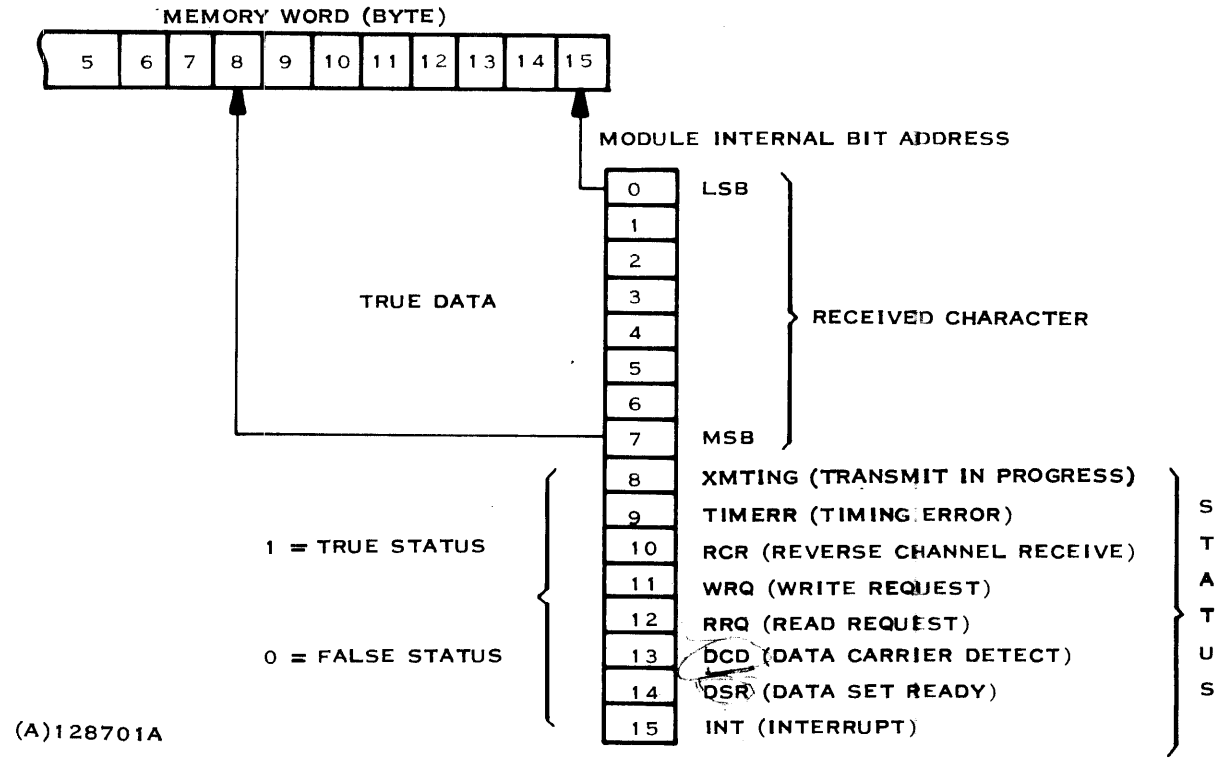


Figure 3-17. Module to CRU Inputs

Table 3-9. Input Signals

Signal	Description
True Data In	Eight bits of data from teleprinter forming the last character received from teleprinter. A STCR (bits 0 through 7) of teleprinter interface places character in memory.
Transmit In Progress	Status bit from interface to CPU indicating that interface is transmitting a character to teleprinter. This bit is set to a 1 when interface receives the eight data bit from CPU and is reset to a 0 when interface has completed transmission of the character. This bit may be tested by CPU by a TB instruction addressing bit 8 of the teleprinter interface.

Table 3-9. Input Signals (Continued)

Signal	Description
Timing Error	Status bit from interface indicating that a timing error occurred while receiving last character from teleprinter. It indicates that the true data in above may be in error. This bit may be tested by a TB instruction addressing bit 9 of the teleprinter interface.
Reverse Channel Receive	Bit 10 of teleprinter interface. It is used with modems equipped with a reverse channel.
Write Request flag	Status bit from interface indicating that the interface is ready to transmit another character. This bit is set to a 1 and generates an interrupt when the interface has finished its last transmission to teleprinter. This bit may be tested by CPU by a TB instruction addressing bit 11 of the teleprinter interface. The write request flag and the interrupt generated by write request flag may be cleared by issuing a Clear Write Request.
Read Request flag	Status bit from interface indicating that a character has been received from teleprinter and is ready to be read by CPU. This signal and the interrupt line go to a 1 and remain until Clear Read Request is issued by the CPU. This bit may be tested by a TB instruction addressing bit 12 of the teleprinter interface.
Data Carrier	Used with modem equipment.
Data Set Ready	Used with modem equipment.
Interrupt	CRU interrupt bit from the data terminal interface. The interrupt line will be a 1 when Write Request or Read Request is a 1, or when Data Carrier Detect or Data Set Ready change states (new status flag). The interrupt will be reset to a 0 by clearing the flag (or flags) that generated the interrupt.

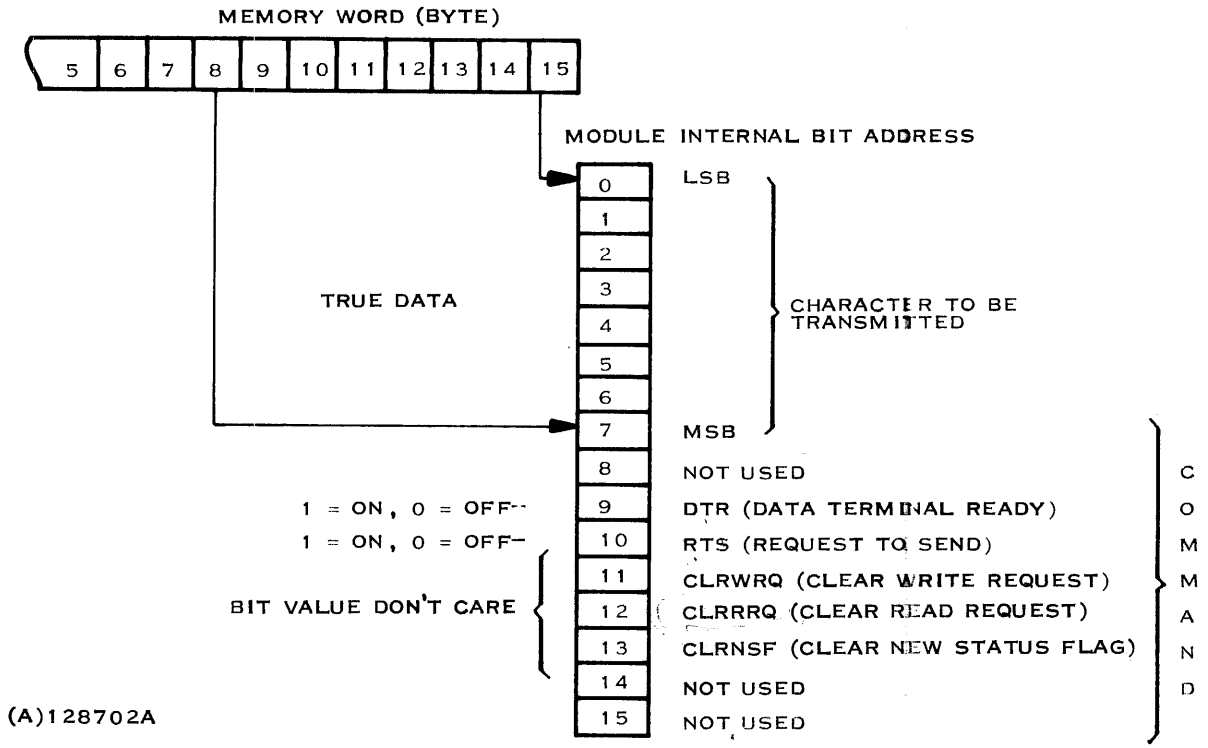


Figure 3-18. CRU to Module Output Signals

Table 3-10. Signals from CRU to Module

Signal	Description
True Data Out	Eight bits of data from the CPU. CRU output bits 0 through 7 form the character to be sent to the teleprinter. A LDCR (bits 0 through 7) loads the character into the interface and initiates transmission to the teleprinter.
Data Terminal Ready	Used with modems.
Request To Send	Used with modems.
Clear Write Request	CPU command to clear write request flag. Flag is cleared by a LDCR or SBZ instruction that addresses CRU bit 11 of teleprinter interface.
Clear Read Request	CPU command to clear read request flag. Flag is cleared by a LDCR or SBZ instruction that addresses CRU bit 12 of teleprinter interface.

Table 3-10. Signals from CRU to Module (Continued)

Signal	Description
Clear New Status Flag	CPU command to clear new status flag interrupt (Data Set Ready and Data Carrier Detect). This command is not normally used with teleprinter except to clear possible interrupt at power turn on. Flag is cleared by SB or LDCR instruction that addresses bit 13 of teleprinter interface.

3.7 DATA MODULE, 16 INPUT/16 OUTPUT

3.7.1 GENERAL

The 16 I/O data module connects low-speed single/multiple bit peripherals to the 990 Computer.

3.7.2 DESCRIPTION

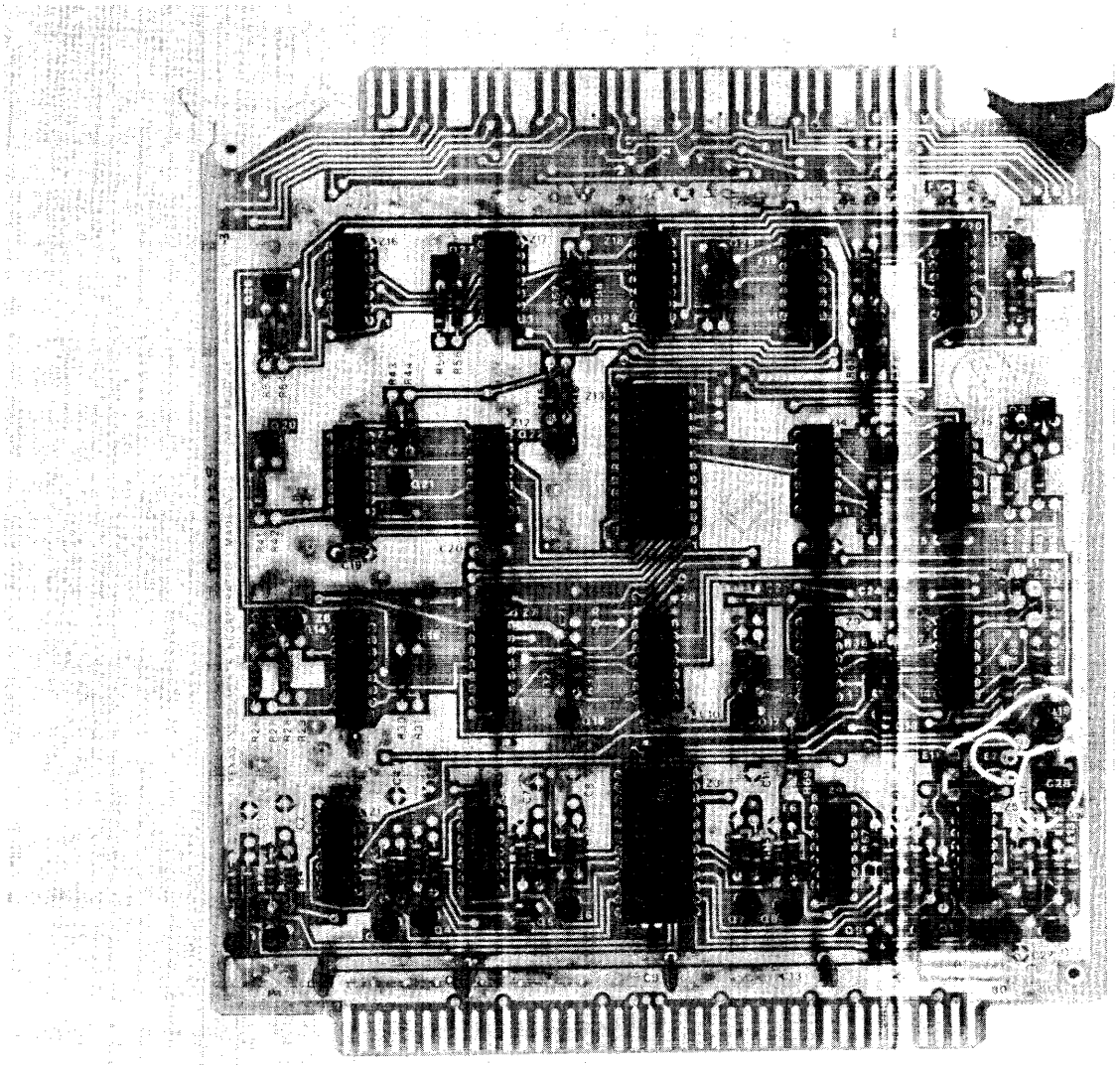
A data module (figure 3-19) provides 16 inputs and 16 outputs. Each line (input or output) can be addressed as a single independent line or as a member of a group of lines from 2 to 16 total. Each output circuit contains a storage flip-flop that maintains the output until changed by the 990 Computer. An alternate version of the data module permits 15 normal inputs, 14 normal outputs, and 3 interrupt lines for interrupt input, mask, and clear. In addition, each output contains an open-collector output transistor that may be attached to an external pull-up resistor and power supply for signal levels up to 30 Vdc @50 ma.

3.7.3 OPERATION

The operation of the 16 I/O module depends entirely upon what peripheral is connected to the interface. Output lines may be connected to non-computer or computer peripherals for control applications and input lines may be used for data input from these peripherals. The only restriction to the type of peripheral connected to the 16 I/O module is that the input/output signal levels must be representative of two discrete signal levels for a logic one or a logic zero. These levels are mutually exclusive.

3.7.4 CAPABILITIES

The 16 I/O data module may be used to interface low-speed non-computer peripherals, or slow to medium speed computer peripherals to the 990 Computer.



128703 (990-674-6-1)

Figure 3-19. 16 I/O Data Module

3.7.5 MODULE SPECIFICATIONS

The input and output specifications are as follows:

- 16 inputs - Emitter follower transistor inputs at TTL levels.
  0v to 1.0v = Logic "1", 3.0v to 5.0v = Logic "0"

- 16 outputs - Transistor collector with optional external pull up.
  0v to 0.4v = Logic "1", open = Logic "0"

- Optional interrupt - 15 inputs, 1 interrupt input.
  14 outputs, 1 interrupt mask and 1 interrupt clear.
  Interrupt input will operate on either positive or negative transition of the input.

3.7.6 INSTALLATION

3.7.6.1 CONNECTION INFORMATION. Connections to the input circuits are contained in table 3-11. Connection to the output circuits are contained in table 3-12.

3.7.6.2 POWER REQUIREMENTS. The 16 I/O module requires +5 Vdc @ 0.5 amp maximum.

3.7.6.3 ADAPTER CARD REQUIREMENTS. The 16 I/O module requires the card adapter, TI part number 975200-0001.

Table 3-11. 16 INPUT Circuit Connections

Function	Mnemonic	Pin Number
Bit 0 Input	IN0	P2-$\overline{M}$
Bit 1 Input	IN1	P2-$\overline{H}$
Bit 2 Input	IN2	P2-$\overline{C}$
Bit 3 Input	IN3	P2-Y
Bit 4 Input	IN4	P2-U
Bit 5 Input	IN5	P2-P
Bit 6 Input	IN6	P2-K
Bit 7 Input	IN7	P2-E
Bit 8 Input	IN8	P2-33
Bit 9 Input	IN9	P2-29
Bit 10 Input	IN10	P2-25
Bit 11 Input	IN11	P2-21
Bit 12 Input	IN12	P2-17
Bit 13 Input	IN13	P2-13
Bit 14 Input	IN14	P2-9
Bit 15 Input	IN15	P2-5

Table 3-12. 16 OUTPUT Circuit Connections

Function	Menmonic	Pin Number
Bit 0 Output	OUT0	P2-$\overline{L}$
Bit 1 Output	OUT1	P2-$\overline{F}$
Bit 2 Output	OUT2	P2-$\overline{B}$
Bit 3 Output	OUT3	P2-X
Bit 4 Output	OUT4	P2-T
Bit 5 Output	OUT5	P2-N
Bit 6 Output	OUT6	P2-J
Bit 7 Output	OUT7	P2-D
Bit 8 Output	OUT8	P2-32
Bit 9 Output	OUT9	P2-28
Bit 10 Output	OUT10	P2-24
Bit 11 Output	OUT11	P2-20
Bit 12 Output	OUT12	P2-16
Bit 13 Output	OUT13	P2-12
Bit 14 Output	OUT14	P2-8
Bit 15 Output	OUT15	P2-4

3.8 PROTOTYPE DEVELOPMENT CARDS

As an aid to design of new interface circuit boards for either the TILINE or CRU interface, two types of blank circuit boards are available for the Model 990 Computer. Each type of circuit board uses wire-wrap connections to tie together the logic circuits mounted on them. Printed circuit ground and VCC planes are supplied on the board.

3.8.1 SINGLE-CONNECTOR DEVELOPMENT CARD

The single-connector development card, part number 217863-0001, (figure 3-20) conforms to the circuit board size specifications for a single-connector circuit board described in Section I of this manual. The board contains four rows of seven locations for mounting 14 pin dual-in-line package (DIP) integrated circuits (ICs), four rows of six locations for mounting 14 or 16 pin ICs, and two rows of four locations for mounting 24 pin ICs. Wire-wrap pins on the reverse side of the circuit board allow interconnection of the circuits in any required configuration. In order to gain cable access to the circuit

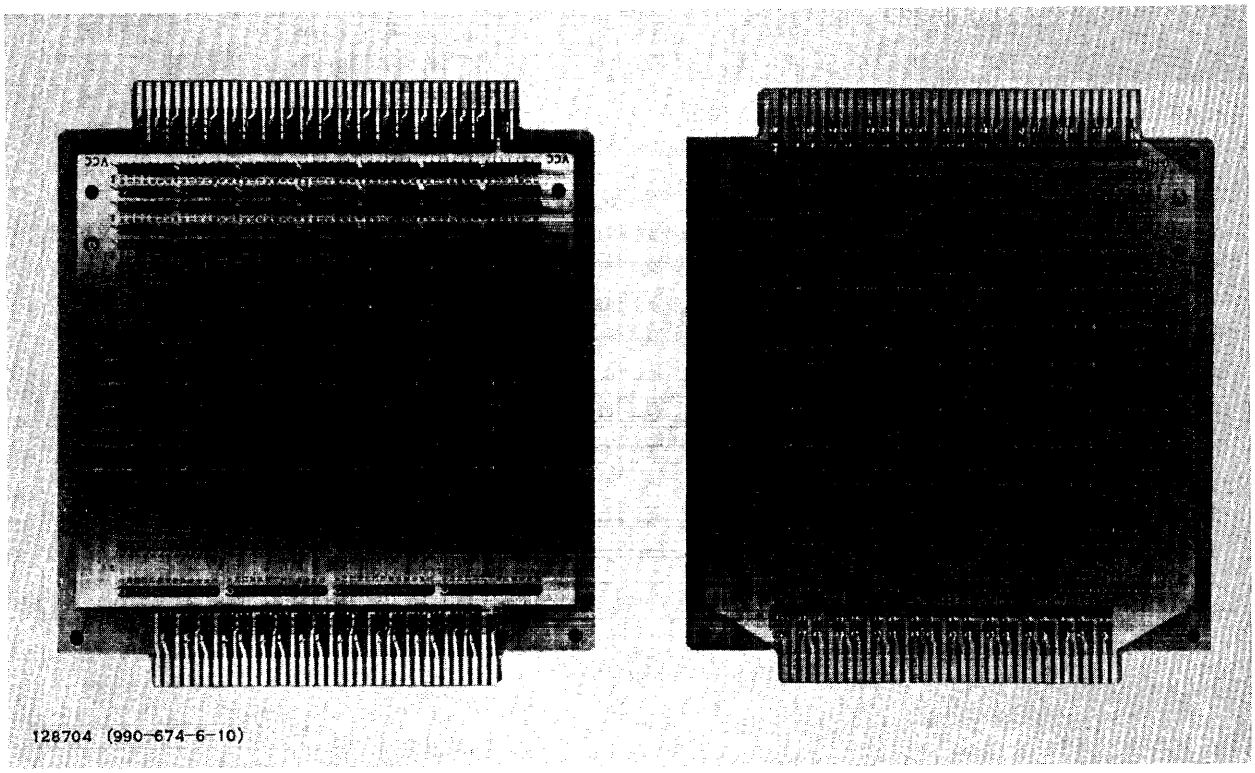


Figure 3-20. Single-Connector Development Board

board and to support the center of the board when mounted in the chassis, the single-connector development board must be used with the small card adapter.

3.8.2 DOUBLE-CONNECTOR DEVELOPMENT CARD

The double-connector development card, part number 974651-0001, (figure 3-21) conforms to the circuit board size specifications for a double-connector circuit board described in Section I of this manual. The board contains thirteen rows of fifteen locations for mounting either 14 or 16 pin ICs, two rows of ten locations for mounting 24 pin ICs, and ample space for associated discrete components. Other DIP formats on 0.3 or 0.5 inch wide packages can also be accommodated. Wire-wrap pins on the reverse side of the circuit board allow interconnection of the circuits in any required configuration.

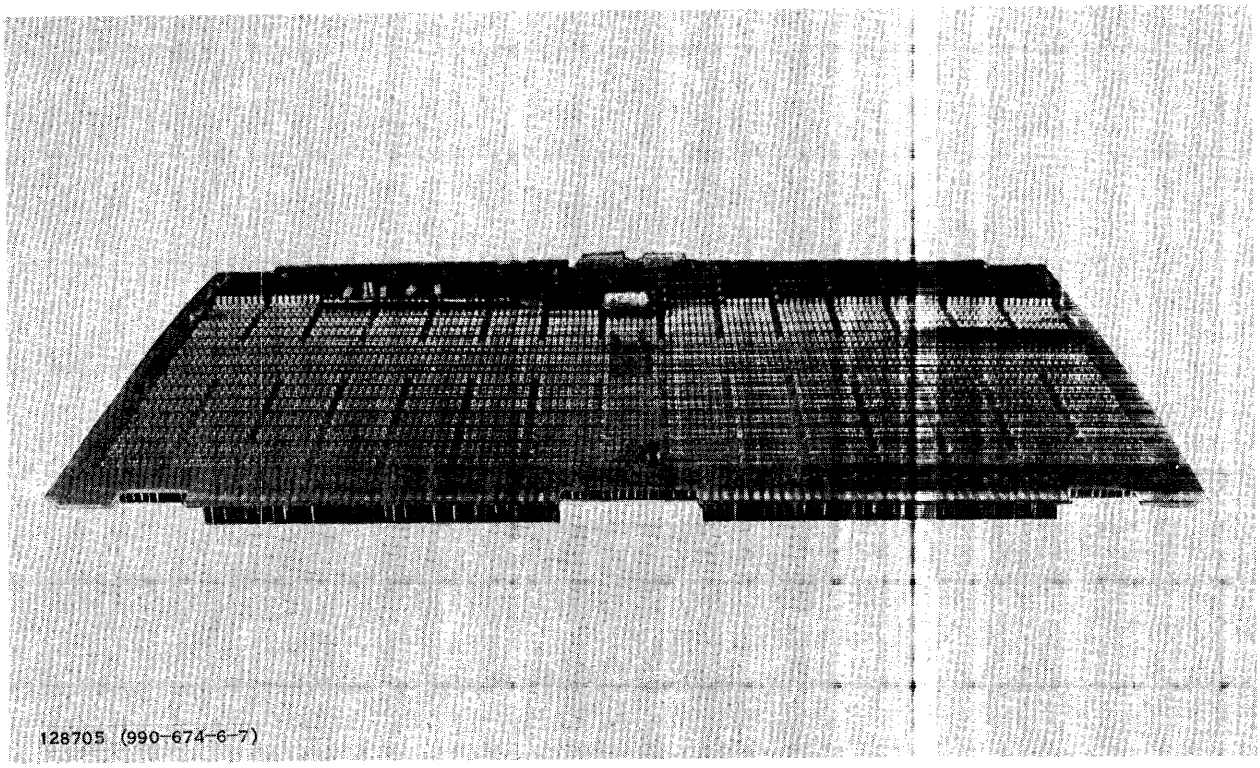


Figure 3-21. Double-Connector Development Board

SECTION IV

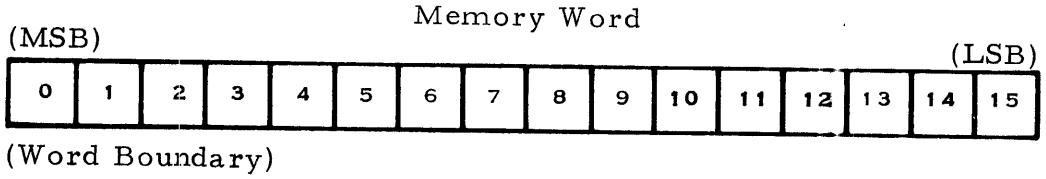
ASSEMBLY LANGUAGE MACHINE INSTRUCTIONS

4.1 GENERAL

This section of the manual contains information about the set of machine instructions available with the 990 Computer. These instructions are grouped according to function in the paragraphs that follow this introduction.

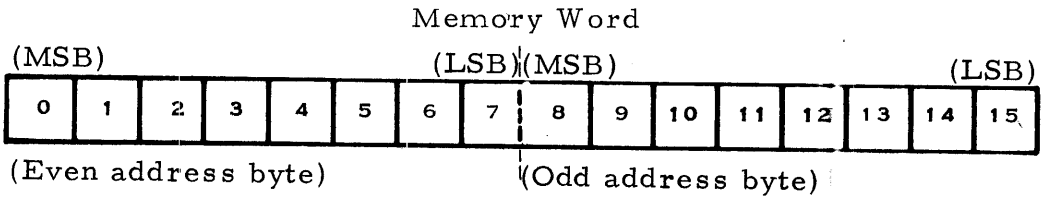
4.1.1 WORD AND BYTE DESCRIPTIONS

A word in the 990 Computer consists of 16 bits. The most significant bit (MSB) is bit zero of the memory word and the least significant bit (LSB) is bit 15. The following diagram illustrates a memory word.



Word boundaries are assigned to even-numbered addresses in memory. The even address byte is bits 0 through 7 and the odd address byte is bits 8 through 15. A word cannot begin on an odd byte address. If word instructions address an odd byte, the word operand is taken from the memory word that contains the addressed byte. This is the memory address that corresponds to the odd byte address minus one. For example: A memory address of 1023_{16} is effectively an address of 1022_{16} when used in a word instruction.

A byte consists of eight bits of memory that may be either bits 0 through 7 or bits 8 through 15 of the memory word. The following diagram illustrates the byte address concept.



Byte instructions may address either byte as necessary. Byte instructions that address a workspace register operate on the most significant byte (even address) of the workspace register and the least significant byte (odd address) is not changed. Since the workspace is also addressable as a memory address, the least significant byte may be directly addressed if desired.

CRU instructions may operate on a single bit of a word if necessary. If the CRU instruction operates on eight or less bits in a word, the byte may be addressed as either even or odd. If the CRU instruction operates nine or more bits in a word, the specified address is a word address.

The MSB of the byte or word is the sign bit when the word/byte contains a value. The remaining bits in the word/byte represent the magnitude of the value. The seven remaining bits of the byte may represent a number from -128 to +127. The value of the remaining 15 bits of a word may represent from -32,768 to +32,767. Multiply and divide instructions operate on a 16-bit word as an unsigned magnitude value from 0 to 65,535. If the MSB is equal to one, the word/byte is a negative value. To determine the magnitude of that value, invert the individual bits in the word and add a value of one to the word. This converted value is the magnitude represented by the two's complement word/byte contents.

4.1.2 MEMORY MAP AND MEMORY ALLOCATION

Refer to Section II for a description of the memory hardware. As previously described, memory consists of one or two printed circuit cards, which together may contain from 4096 to 32,768 words, each having 16 bits and a parity bit. A group of 128 bytes is dedicated for use by the priority interrupts and the software implemented extended operations, as shown in figure 4-1. Another group of words in memory is dedicated for use by the CPU as TILINE addresses. (Refer to Section II for a description of the TILINE address generation.) The memory map (figure 4-1) shows the locations of interrupt vectors, extended operation vectors, and TILINE addresses.

4.1.3 HARDWARE REGISTERS

There are three dedicated hardware registers in the 990 Computer that determine the programming environment. These registers are the workspace pointer (WP), the program counter (PC), and the status register (ST).

The contents of the WP register locate the 16-word area of memory that is the active workspace, and each word in the workspace is called a workspace register. A program may use more than one workspace. The WP is a 15-bit register whose contents represent the 16-bit (unsigned integer) word address of the first word of the workspace, with the right-most bit (the least significant bit) truncated. The PC contains the address of the next instruction to be executed. Instructions always reside on word boundaries. The PC is changed to the address of the next instruction during the execution of the present instruction so that the PC contains the address of the next instruction at all times. The PC is a 15-bit register that represents a 16-bit (unsigned integer) word address, with the right-most bit truncated.

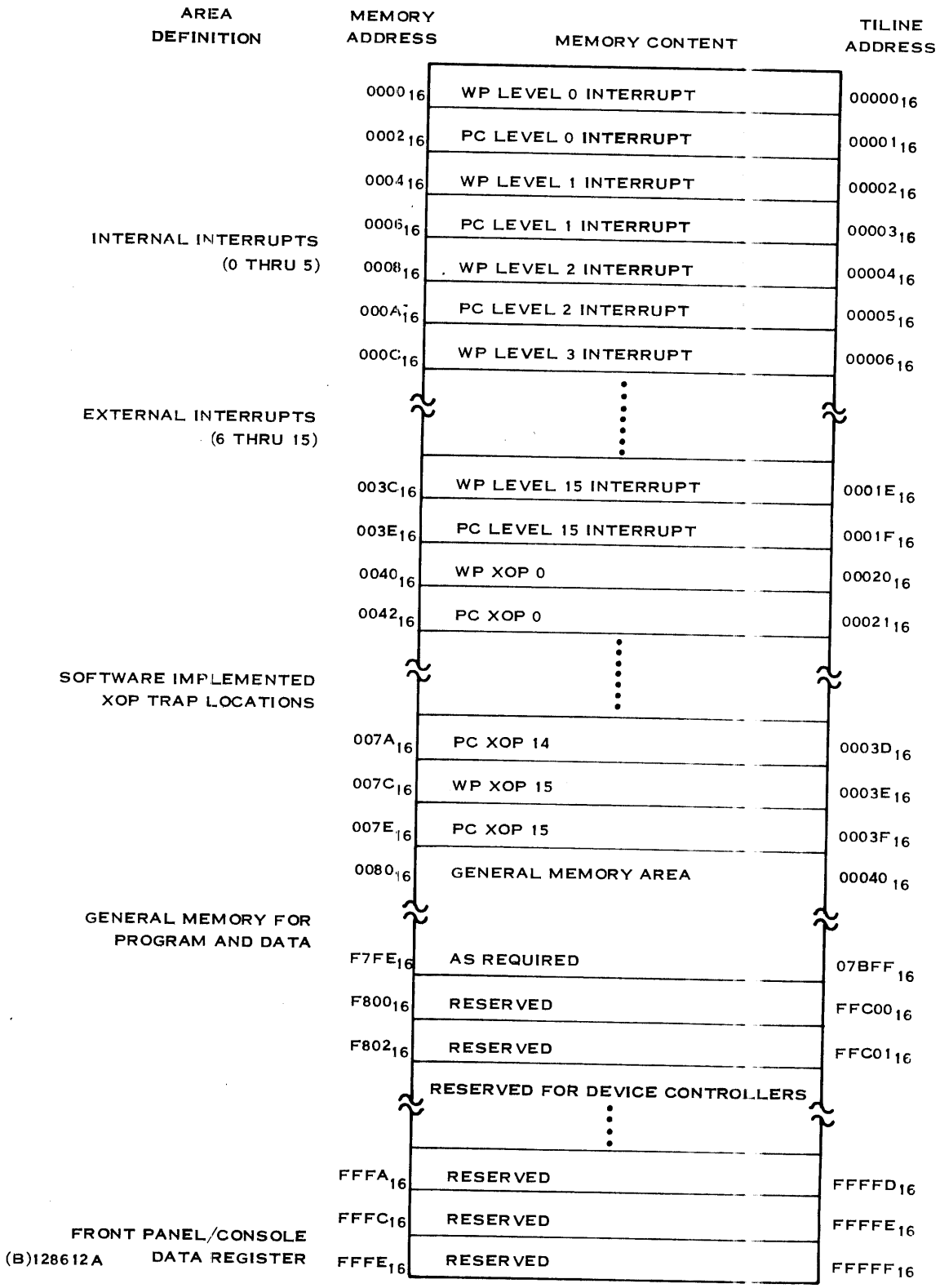


Figure 4-1. Memory Map and Assignments

The ST register is a 16-bit register that contains three parts:

- Bits zero through six - Status
- Bits seven through eleven - Reserved
- Bits twelve through fifteen - Interrupt mask

During and immediately after instruction execution, the status of execution is determined. This status is maintained in the first seven bits of the status register.

4.1.4 WORKSPACE REGISTERS

The Model 990 Computer uses a workspace that consists of 16 memory words called workspace registers. The address contained in the workspace pointer (WP) register is the address of a memory word addressed as workspace register 0. Workspace registers 1 through 15 are the 15 memory words following the memory word addressed as workspace register 0. The Branch and Load Workspace Pointer instruction, the Return from Interrupt Subroutine instructions, and the Load Workspace Pointer Immediate instruction alter the contents of the WP register. Occurrence of an interrupt and initiation of a software-implemented extended operation also alter the contents of the WP register. This results in activation of a new workspace, which is referred to as a context switch. The active workspace is addressed by those instructions that specify workspace registers.

Workspace registers may contain data or addresses. They are used as operand registers, accumulators, address registers, or index registers. Several workspace registers are used for special purposes by certain instructions. Figure 4-2 shows a workspace map, and table 4-1 lists the utilized workspace registers, with appropriate instruction mnemonics that alter the workspace. The indicated workspace utilization is only true for the instruction listed. When that instruction is used, the contents of the workspace must be appropriate for the application. When that instruction is not used, the workspace may be used for any other application shown.

4.1.5 MACHINE INSTRUCTION DESCRIPTIONS

Each description contains the following information about the instruction:

- Instruction mnemonic with the mnemonic definition
- Instruction op code
- Instruction word format as stored in memory
- Instruction word addressing mode
- Status bits affected by the execution of the instruction
- Execution results of the machine instruction

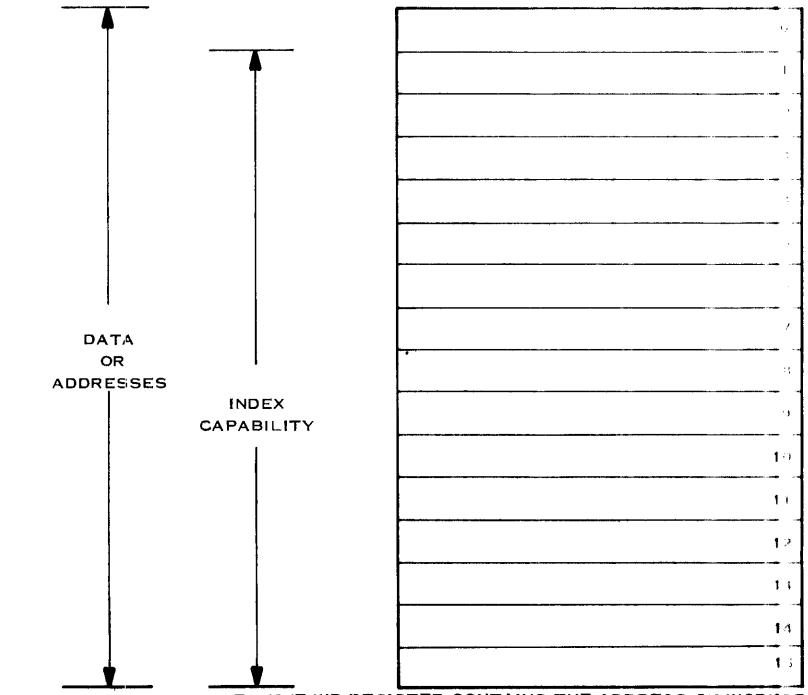


Figure 4-2. Workspace Map

Table 4-1. Workspace Register Utilization

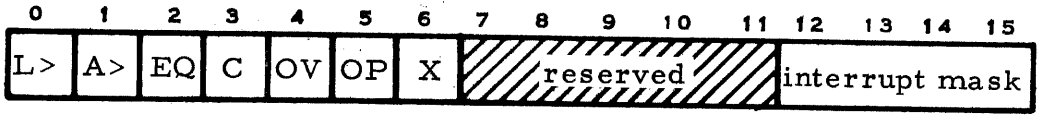
Workspace Register	Instruction Mnemonic Operation Code	Purpose
0	SLA, SRA, SRC, SRL	Shift count (optional)
11	BL	Return address
11	XOP (Software)	Effective address
12	SBO, SBZ, TB, LDCR, STCR	CRU base address
13	BLWP, RTWP, XOP (Software)	WP register contents
14	BLWP, RTWP, XOP (Software)	Return address
15	BLWP, RTWP, XOP (Software)	ST register contents

NOTE

Execution of an interrupt uses workspace registers 13, 14, and 15 for the same purposes as a BLWP instruction. (Refer to the remainder of Section IV for the instructions that utilize WR.)

- Application notes that include appropriate examples and the results of execution of the example
- Assembly language format of the machine instruction

The status bits affected by the execution of an instruction refer to the first seven bits of the status register, bits zero through six. The contents of the status register are arranged as follows:



If the status bit is set by the instruction execution, the bit is equal to a logic one. If the status bit is reset, it is equal to a logic zero. The definitions of the status bits are as follows:

- L> - Logical greater than
- A> - Arithmetic greater than
- EQ - Equal
- C - Carry
- OV - Overflow
- OP - Odd parity
- X - XOP (Extended operation, software-implemented)

Note that the OP status bit sets/resets for byte instructions only. OP sets when the bits in the byte affected by the instruction establish odd parity. Odd parity is established when the sum of the logic one bits is an odd number. The combinations of bits that establish odd or even parity are as follows:

Byte Contents	OP Status Bit
0 0 0 0 0 0 0 1 (Any of the 8 positions)	1
0 0 0 0 0 0 1 1 (Any 2 of the 8 positions)	0
0 0 0 0 0 1 1 1 (Any 3 of the 8 positions)	1
0 0 0 0 1 1 1 1 (Any 4 of the 8 positions)	0
0 0 0 1 1 1 1 1 (Any 5 of the 8 positions)	1
0 0 1 1 1 1 1 1 (Any 6 of the 8 positions)	0
0 1 1 1 1 1 1 1 (Any 7 of the 8 positions)	1
1 1 1 1 1 1 1 1 (All 8 positions)	0

The execution results of the various instructions are presented with the convention shown in table 4-2. Refer to this table to determine what occurs during instruction execution.

Table 4-2. Assembly Language Format and Execution Result Conventions

Symbol	Definition	Description of Use
[ ]	Brackets	Indicates that the item enclosed may be used at the option of the programmer.
< >	Angle Brackets	Indicates that the item enclosed is to be supplied by the user.
lower case alphabetic characters	-	Indicates the location of the item supplied by the user.
⏨	Blank	Indicates the only area within the assembly language statement where a blank may be inserted. Furthermore, when one ⏨ is shown in the general format, at least one blank (or other terminating character) must be included in the statement.
ga_s	General address source	Indicates that the user must supply a general address for this operand. Refer to paragraph 4.1.6.
ga_d	General address	Indicates that the user must supply a general address for the destination operand.
$wa_{s/d}$	Workspace register address	Indicates that the user must supply an input that represents a workspace register from 0 through 15.
iop	Immediate operand	Indicates that the user must supply an input for the immediate operand.
disp	Displacement	Indicates that the user must supply a displacement* value in the range of -128 to +127.
cnt	Count	Indicates that the bit count supplied by the user must be in the range of -128 to +127.
scnt	Shift Count	Indicates that the user supplied shift count must be in the range of 0 to +15.

*For jump instructions, the displacement is in the form of a byte address that occurs within the range limits relative to the present location +2.

Table 4-2. Assembly Language Format and Execution Result Conventions (Continued)

Symbol	Definition	Description of Use
→	Replaces	Indicates the replacement of an operand by the results of the operation.
( )	Contents of	Indicates that this represents a value contained in a memory word/byte.
| |	Absolute value	Indicates that the value within the two vertical lines is the absolute value operand.

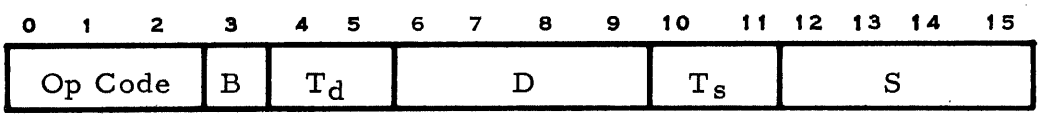
The assembly language formats of the various machine instructions are presented with the conventions shown in table 4-2. Refer to this table to determine what is required to code the instructions for input to the assembler.

When reading the application examples (where provided), pay particular attention to the methods of addressing and operand definition. These examples will provide the user with additional information about the actual methods involved in coding in assembly language.

Since the instructions are grouped according to function, the formats of the instructions as they appear in memory are shown in the following paragraphs. Each instruction contains a reference, by format type, to one of the paragraphs titled Format I through Format IX.

4.1.6 FORMAT I INSTRUCTIONS

Format I instructions are two address instructions in which either address may appear in one of five addressing modes. These instructions may operate on bytes or words, and the available operations include arithmetic and bit manipulation. The format of these instructions is as follows:



Where

Op Code - Indicates the bits defining the operation code

B - Byte indicator when equal to one and word indicator when equal to zero

T_d - Addressing mode of the destination operand

D - Workspace register of the destination operand

T_s - Addressing mode of the source operand

S - Workspace register of the source operand

The five addressing modes of the operands are:

T Field	Description
00	Workspace register
01	Workspace register indirect
10	Indexed memory
10	Symbolic memory
11	Workspace register indirect auto-increment

A workspace register address is written as a term having a value in the range of 0 to 15. An indirect workspace register address is written as a term preceded by an asterisk (*). A symbolic memory address is written as an expression preceded by an at sign (@). An indexed memory address is written as an expression preceded by an at sign (@) and followed by a term enclosed in parentheses. The workspace register specified by the term in parentheses is the index register. Workspace register 0 may not be specified as an index register. A workspace register autoincrement address is written as a term preceded by an asterisk (*) and followed by a plus sign (+). Refer to Appendix H for definitions of term and expression. The following examples show the various methods of addressing using a MOV (move word) machine instruction:

```
MOV   4, 8      (workspace register)
MOV   *2, *7    (workspace register indirect)
MOV   @TABLE, @LIST   (symbolic memory)
MOV   @TABLE(3), @LIST(4)   (indexed memory)
MOV   *2+, *7+   (workspace register indirect autoincrement)
```

In the workspace register mode, the contents of the workspace register specified in either the D or S operand is the destination or source operand, respectively. This address may be specified by a value from zero to fifteen.

NOTE

When byte operations are performed in the workspace register mode, the left-most byte (bits 0-7) is the operand and the right-most byte (bits 8-15) is unchanged.

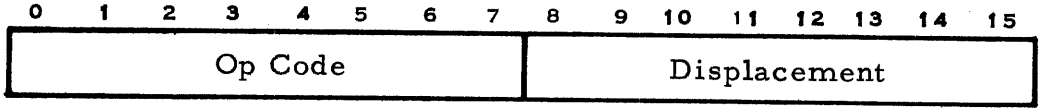
In the indirect workspace register mode, the contents of the workspace register specified in the D or S field is a memory address for the destination or source operand respectively.

In the indexed memory addressing modes, the specified workspace register in the D or S field is used as an index register. An additional word is required for each operand in the indexed memory or symbolic memory addressing modes. When the indexed mode is required for either operand (one only), the instruction requires two words in memory, and the contents of the second word are added to the workspace register contents to obtain a memory address. When the D or S field contains zero, the contents of the second word are used without modification as a memory address. The contents of the specified memory address are the operand. When the indexed mode is required for both operands, the instruction occupies three words of memory. The second word is used with the workspace register specified in the S field and the contents of the third word are used with the workspace register specified in the D field. When the address is used as a symbolic memory address, no index is specified and the contents of the symbolic memory address are the operand.

The indirect workspace register auto-increment mode is similar to the indirect workspace register mode in that the workspace register contains the operand address. After the operand has been accessed, the workspace register is incremented and the incremented value is placed in the workspace register. The workspace register increment is two when the B field contains zero (word operands) or one when the B field contains one (byte operand).

4.1.7 FORMAT II INSTRUCTIONS

Format II instructions derive an address from a signed displacement that is algebraically added to the contents of the program counter (jump instructions) or to the contents of workspace register 12, bits 3 - 14 (CRU instructions). The format of the instruction word for Format II instructions is:



Where

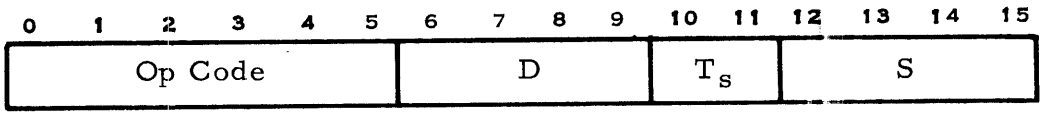
Op Code - Indicates the bits defining the operation code

Displacement - Defines the signed displacement value

The signed displacement is an 8-bit, two's complement value representing words in jump instructions and representing bits for bit instructions. This set of eight bits provides a range of -128 to +127. This range is relative to the program counter for jump instructions and relative to the Communication Register Unit (CRU) base address in workspace register 12 for CRU bit instructions.

4.1.8 FORMAT III INSTRUCTIONS

Format III instructions perform logical operations on two operands. The format of the instruction word for Format III instructions is as follows:



Where

Op Code - Indicates the bits defining the operation code

D - Workspace register of the destination operand

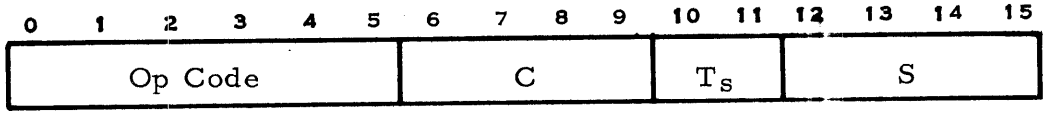
T_s - Addressing mode of the source operand

S - Workspace register of the source operand

The five addressing modes defined for Format I instructions are permitted for the source operand of this format. The workspace register mode of addressing is the only mode permitted for the destination operand. When the auto-increment mode is used for the source operand, the increment value is two.

4.1.9 FORMAT IV INSTRUCTIONS

Format IV instructions transfer data between memory and the Communication Register Unit (CRU). One to sixteen bits may be transferred by these instructions. When fewer than nine bits are specified (1-8), the memory address of the source operand is effectively a byte address. When more than eight bits are specified (9-16, a zero specifies 16), the memory address is effectively a word address. Bits three through fourteen of workspace register 12 contain the CRU base address. The format of the instruction word for a Format IV instruction is as follows:



Where

Op Code - Indicates the bits defining the operation code

C - Indicates the bits defining the bit count

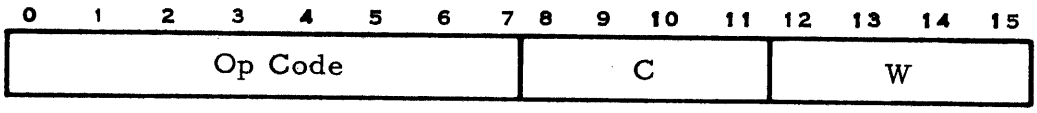
T_s - Addressing mode of the source operand

S - Workspace register that contains the source operand

The C field contains the number of bits to be transferred, specified by a value from 0 to 15. The value 0 specifies 16 bits. The five addressing modes defined for Format I instructions apply to the source operand. The value in the C field determines whether the source operand is a byte or a word. When the C field contains 1 through 8, the source address is a byte address and the workspace register increment for the indirect workspace register auto-increment mode is one. When the C field contains 9 through 15, or 0, the source address is a word address and the workspace register increment is two. The source operand for Format IV instructions is the memory operand. It is called the source operand for uniformity, but is the destination of the Store Communication Register operation.

4.1.10 FORMAT V INSTRUCTIONS

Format V instructions perform shifting operations on the contents of a workspace register. The format of the instruction word for Format V instructions is as follows:



Where

Op Code - Indicates the bits defining the operation code

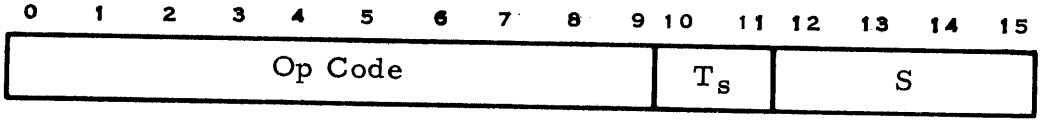
C - Indicates the bits defining the shift count

W - Indicates the bits defining the workspace register to be shifted

The workspace register may only be addressed in the workspace register mode. The C field contains the shift count, which must be an integer value from 0 to 15. When the C field contains zero, the shift count is in bits 12 through 15 of workspace register zero. When the C field contains zero and bits 12 through 15 of workspace register zero are equal to zero, the shift count is 16.

4.1.11 FORMAT VI INSTRUCTIONS

Format VI instructions are single address instructions, where the address may be in any of the five addressing modes specified for Format I instructions. The format of the instruction word for Format VI instructions is as follows:



Where

Op Code - Indicates the bits defining the operations code

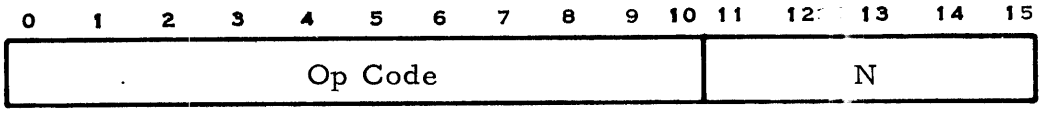
T_s - Indicates the bits defining the addressing mode of the source operand

S - Indicates the bits defining the source operand workspace register

The five addressing modes defined for Format I instructions apply to the source operand of Format VI instructions. The workspace register increment for the auto-increment mode is two. This single operand is called the source operand for uniformity and may also be the destination operand for Format VI instructions.

4.1.12 FORMAT VII INSTRUCTIONS

Format VII instructions are control instructions that require no operands. The format of the instruction word for Format VII instructions is as follows:



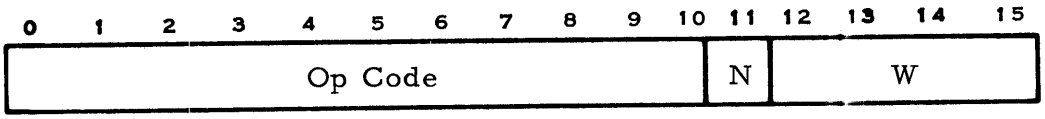
Where

Op Code - Indicates the bits defining the operation code

N - Indicates an unused field of the instruction word (Programmers should set this field to zero)

4.1.13 FORMAT VIII INSTRUCTIONS

Format VIII instructions are immediate instructions that may use the word of memory immediately following the instruction word as an operand. On instructions that require another operand, the operand is specified as a workspace register. The format of the instruction word for Format VIII instructions is as follows:



Where

Op Code - Indicates the bits defining the operation code

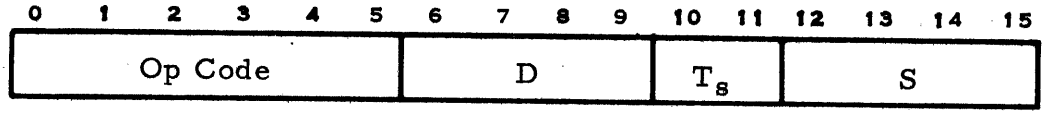
N - Unused bit of the instruction word, may be either 0 or 1 (Programmers should set this bit to zero)

W - Indicates the bits defining the workspace register for the second operand

The immediate operand is the contents of the word immediately following the instruction in memory. When the workspace register operand is not required by the instruction, bits 12 through 15 of the instruction word may contain any value. The store immediate instructions (store status, store workspace pointer) do not use the immediate operand.

4.1.14 FORMAT IX INSTRUCTIONS

Format IX instructions are the extended instructions; extended operation, multiply, divide. The extended operation instruction is a means of extending the instruction set to include additional instructions that may be either software or hardware implemented. The multiply and divide instructions use the same instruction word format. The source operand may be addressed in any of the five addressing modes defined for Format I instructions, and the destination operand is an adjacent pair of workspace registers. The format of Format IX instructions is as follows:



Where

Op Code - Indicates the bits defining the operation code

D - The workspace register address of two adjacent workspace registers for the destination operand of multiply and divide instructions or specifies a predefined operation for the extended operation instruction

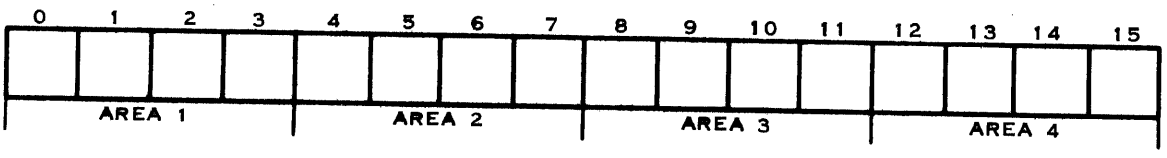
T_s - The addressing mode of the source operand

S - The workspace register of the source operand

The five addressing modes defined for Format I instructions apply to the source operand of Format IX instructions. The workspace register increment for the auto-increment function is two. For multiply and divide instructions, when the workspace register of the D field is specified as 15, the word in memory immediately following workspace register 15 is the second word of the destination operand.

4.1.15 DETERMINING OP CODES

To determine the op code of an instruction, arbitrarily divide the entire memory word into four bit areas as follows:



(A)129507

Fill in any bit position not indicated as either a logic zero or logic one with a logic zero (trailing zeros). Convert each four bit area to a hexadecimal value. The value obtained is the instruction op code. For example: The A (Add words) machine instruction has the following specified bits for bit positions 0 through 3:

1 0 1 0

Add trailing zeros to fill in all 16 bit positions as follows:

1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

Divide this into four bit areas as follows:

```
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
      /       /       /
```

Determine the hexadecimal value for each area to determine the op code for the instruction as follows:

```
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
      /       /       /
  A       0       0       0      -- Op Code
```

4.2 ARITHMETIC INSTRUCTIONS

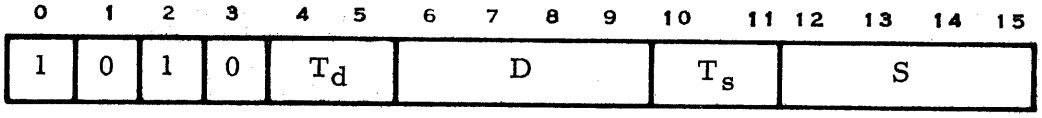
Arithmetic instructions included are:

Instruction	Mnemonic	Paragraph
Add Words	A	4.2.1
Add Bytes	AB	4.2.2
Absolute Value	ABS	4.2.3
Add Immediate	AI	4.2.4
Decrement	DEC	4.2.5
Decrement By Two	DECT	4.2.6
Divide	DIV	4.2.7
Increment	INC	4.2.8
Increment By Two	INCT	4.2.9
Multiply	MPY	4.2.10
Negate	NEG	4.2.11
Subtract Words	S	4.2.12
Subtract Bytes	SB	4.2.13

4.2.1 A (ADD WORDS)

Op Code: A000

Format:



Definition: Add a copy of the source operand (word) to a copy of the destination operand (word) and replace the destination operand with the sum. The AU compares the sum to zero and sets/resets the status bits to indicate the result of the comparison. When there is a carry out of bit zero, the carry status bit sets. When there is an overflow (the sum cannot be represented as a 16-bit, two's complement value), the overflow status bit sets.

Addressing Mode: Format I instructions

Status Affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

Execution results: $(ga_s) + (ga_d) \rightarrow (ga_d)$

Application Notes: A is used to add signed integer words. For example, if the address labeled TABLE contains 3124_{16} and workspace register 5 contains 8_{16}, then the instruction

```
A       5,@TABLE
```

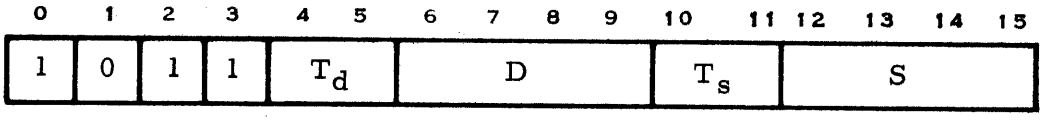
results in the contents of TABLE changing to $312C_{16}$ and the contents of workspace register 5 not changing. The logical and arithmetic greater than status bits set and the equal, carry, and overflow status bits reset.

Assembly language format: [<label>] ⌀ A ⌀ <ga_s>, <ga_d> [⌀ <comment>]

4.2.2 AB (ADD BYTES)

Op Code: B000

Format:



Definition: Add a copy of the source operand (byte) to the destination operand (byte), and replace the destination operand with the sum. When the destination operand is addressed in the workspace register mode, the rightmost byte (bits 8-15) of the addressed workspace register is unchanged. The

AU compares the sum to zero and sets/resets the status bits to indicate the results of the comparison. When there is a carry out of the most significant bit of the byte, the carry status bit sets. When there is an overflow (the sum cannot be represented within a byte as an 8-bit two's complement value), the overflow status bit sets. The odd parity bit sets when the bits in the sum (destination operand) establish odd parity and resets when the bits in the sum establish even parity.

Addressing mode: Format I instructions

Status affected: Logical greater than, arithmetic greater than, equal, carry, overflow, and odd parity.

Execution results: $(ga_s) + (ga_d) \longrightarrow (ga_d)$

Application notes: AB is used to add signed integer bytes. For example, if the contents of workspace register 3 is 7400_{16}, the contents of memory location 2122_{16} is $F318_{16}$, and the contents of workspace register 2 is 2123_{16}, then the instruction

```
AB      3,*2+
```

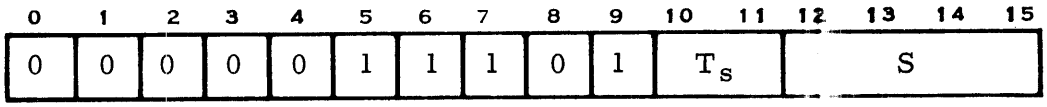
changes the contents of memory location 2122_{16} to $F38C_{16}$ and the contents of workspace register 2 to 2124_{16}, while the contents of workspace register 3 remain unchanged. The logical greater than, overflow, and odd parity status bits set, while the arithmetic greater than, equal, and carry status bits reset.

Assembly language format: [< label >] ⍽ AB ⍽ < ga_s >,< ga_d > [⍽ < comment >]

4.2.3 ABS (ABSOLUTE VALUE)

Op code: 0740

Format:



Definition: Compute the absolute value of the source operand and replace the source operand with the result. The absolute value is the two's complement of the source operand when the sign bit (bit zero) is equal to one. When the sign bit is equal to zero, the source operand is unchanged. The AU compares the original source operand to zero and sets/resets the status bits to indicate the results of the comparison.

Addressing mode: Format VI instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: $|(ga_s)| \longrightarrow (ga_s)$

Application notes: Use the ABS instruction to take the absolute value of an operand. For example, if the third word in array LIST contains the value $FF3C_{16}$ and workspace register seven contains the value 4_{16}, then the instruction

```
ABS     @LIST(7)
```

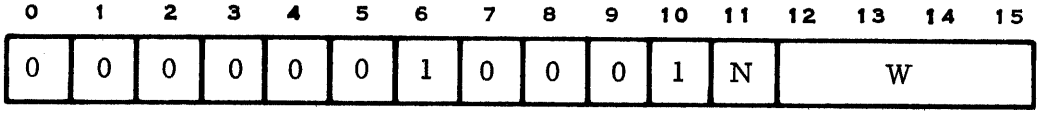
changes the contents of the third word in array LIST to $00C4_{16}$. The logical greater than status bit sets while the arithmetic greater than and equal status bits reset. Refer to Section V for additional application notes.

Assembly language format: [< label >] ⊬ ABS ⊬ < ga_s > [⊬ < comment >]

4.2.4 AI (ADD IMMEDIATE)

Op code: 0240

Format:



Definition: Add the immediate operand, the contents of the word following the instruction word in memory, to the contents of the workspace register specified in the W field and replace the contents of the workspace register with the results. The AU compares the sum to zero and sets/resets the status bits to indicate the result of the comparison. When there is a carry out of bit zero, the carry status bit sets. When there is an overflow (the result cannot be represented within a word as a two's complement value), the overflow status bit sets.

Addressing mode: Format VIII instructions

Status affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

Execution results: (wa) + iop $\longrightarrow$ (wa)

Application notes: Use the AI instruction to add an immediate value to the contents of a workspace register. For example, if workspace register 6 contains a zero, then the instruction

```
AI      6,>C
```

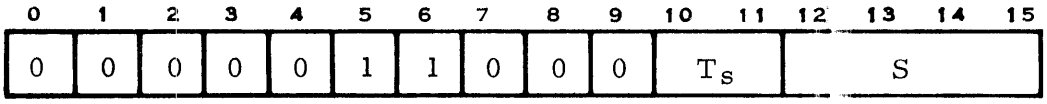
changes the contents of workspace register 6 to $000C_{16}$. The logical greater than and arithmetic greater than status bits are set while the equal, carry, and overflow status bits reset.

Assembly language format: [< label >] ⊬ AI ⊬ < wa >,< iop > [⊬ < comment >]

4.2.5 DEC (DECREMENT)

Op code: 0600

Format:



Definition: Subtract a value of one from the source operand and replace the source operand with the result. The AU compares the result to zero and sets/resets the status bits to indicate the result of the comparison. When there is a carry out of bit zero, the carry status bit sets. When there is an overflow (the difference cannot be represented in a word as a two's complement value), the overflow status bit sets.

Addressing mode: Format VI instructions

Execution results: $(ga_s) - 1 \longrightarrow (ga_s)$

Status affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

Application notes: Use the DEC instruction to subtract a value of one from any addressable operand. The DEC instruction is also useful in counting and indexing byte arrays. For example, if COUNT contains a value of 1_{16}, then

```
DEC    @COUNT
```

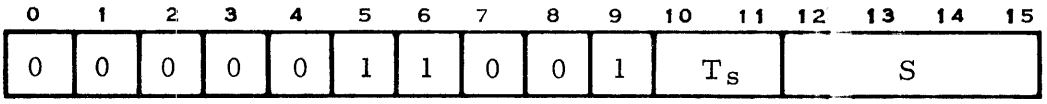
results in a value of zero in location COUNT and sets the equal and carry status bits while reseting the logical greater than, arithmetic greater than, and overflow status bits. Refer to Section V for additional application notes.

Assembly language format: [<label>] ⊬ DEC ⊬ < ga_s > [⊬ <comment>]

4.2.6 DECT (DECREMENT BY TWO)

Op code: 0640

Format:



Definition: Subtract two from the source operand and replace the source operand with the result. The AU compares the result to zero and sets/resets the status bits to indicate the result of the comparison. When there is a carry out of bit zero, the carry status bit sets. When there is an overflow (the result cannot be represented in a word as a two's complement value), the overflow status bit sets.

Addressing mode: Format VI instructions

Status affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

Execution results: $(ga_s) - 2 \longrightarrow (ga_s)$

Application notes: The DECT instruction is useful in counting and indexing word arrays. Also, use the DECT instruction to subtract a value of two from any addressable operand. For example, if workspace register PRT (PRT equals 3) contains a value of $2C10_{16}$, then the instruction

```
DECT    PRT
```

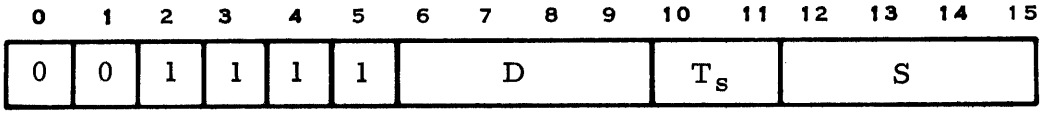
changes the contents of workspace register 3 to $2C0E_{16}$. The logical greater than, arithmetic greater than and carry status bits set while the equal and overflow status bits reset. Refer to Section V for additional application notes.

Assembly language format: [< label >] ⊬ DECT ⊬ < ga_s > [⊬ < comment >]

4.2.7 DIV (DIVIDE)

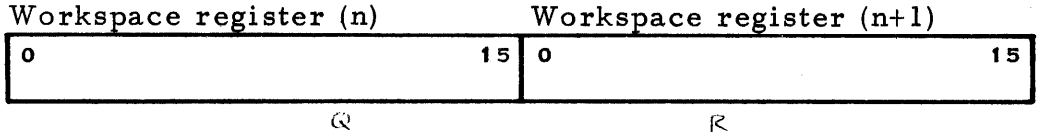
Op code: 3C00

Format:

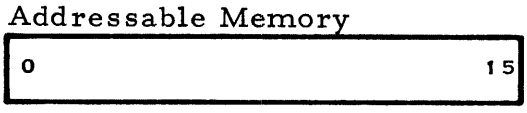


Definition: Divide the destination operand (a consecutive 2-word area of workspace) by the source operand (one word), using integer rules, and place the quotient in the first of the 2-word destination operand area and place the remainder in the second word of that same area. This division is graphically represented as follows:

Destination operand workspace registers



Source operand



The first of the destination operand workspace registers, shown above, is addressed by the contents of the D field. The dividend is located right-justified in this 2-word area. When the division is complete, the quotient (result) is placed in the first workspace register of the destination operand (represented by n above) and the remainder is placed in the second word of the destination operand (represented by n+1 above).

When the source operand is greater than the first word of the destination operand, normal division occurs. If the source operand is less than or equal to the first word of the destination operand, normal division will result in a quotient that cannot be represented in a 16-bit word. In this case, the AU sets the overflow status bit, leaves the destination operand unchanged, and aborts the division operation.

If the destination operand is specified as workspace register 15, the first word of the destination operand is workspace register 15 and the second word of the destination operand is the word in memory immediately following the workspace area.

Addressing modes: Format IX instructions

Status affected: Overflow

Execution results: $((wa(n \text{ and } n+1)_d) \div (ga_s) \longrightarrow (wa(n)_d)$ quotient
$\longrightarrow (wa(n+1)_d)$ remainder

Application notes: Use the DIV instruction to perform a magnitude division. For example, if workspace register 2 contains a zero and workspace register 3 contains $000C_{16}$, and the contents of LOC is 0005_{16}, then the instruction

```
DIV    @LOC, 2
```

results in a 0002_{16} in workspace register 2 and a 0002_{16} in workspace register 3. The overflow status bit resets. If workspace register 2 contained the value 0005_{16}, the magnitude contained in the destination operand would equal 327, 692 and division by the value 5 would result in a quotient of 65, 538, which cannot be represented in a 16-bit word. This attempted division would set the overflow status bit and the AU would abort the operation.

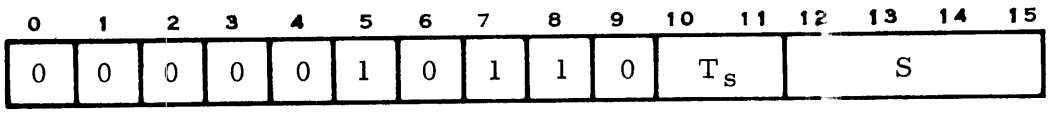
Assembly language format:

```
[<label>] ⊬ DIV ⊬<gas>,<wad> [⊬<comment>]
```

4.2.8 INC (INCREMENT BY ONE)

Op code: 0580

Format:



Definition: Add one to the source operand and replace the source operand with the result. The AU compares the sum to zero and sets/resets the status bits to indicate the result of the comparison. When there is a carry out of bit zero, the carry status bit sets. When there is an overflow (the sum cannot be represented in a 16-bit, two's complement value), the overflow status bit sets.

Addressing mode: Format VI instructions

Status affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

Execution results: $(ga_s) + 1 \longrightarrow (ga_s)$

Application notes: Use the INC instruction to count and index byte arrays, add a value of one to an addressable memory location, or occasionally set flags. For example, if COUNT contains a zero, the instruction

```
INC     @COUNT
```

places a 0001_{16} in COUNT and sets the logical greater than and arithmetic greater than status bits, while the equal, carry, and overflow status bits reset. Refer to Section V for additional application notes.

Assembly language format: [< label >] ⊬ INC ⊬ < ga_s > [⊬ < comment >]

4.2.9 INCT (INCREMENT BY TWO)

Op code: 05C0

Format:

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

Definition: Add a value of two to the source operand and replace the source operand with the sum. The AU compares the sum to zero and sets/resets the status bits to indicate the result of the comparison. When there is a carry out of bit zero, the carry status bit sets. When there is an overflow, (the sum cannot be represented in a 16-bit word as a two's complement value), the overflow status bit sets.

Addressing mode: Format VI instructions

Status affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

Execution results: $(ga_s) + 2 \longrightarrow (ga_s)$

Application notes: Use the INCT instruction to count and index word arrays, and add the value of two to an addressable memory location. For example, if workspace register 5 contains the address of the fifteenth word of array FIX located at address 2100_{16}, the instruction

```
INCT    5
```

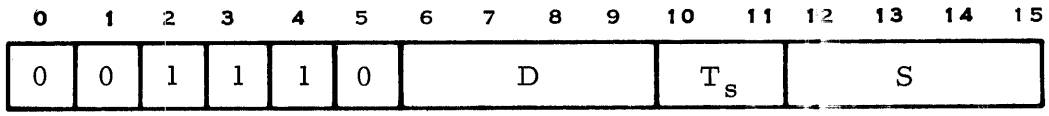
changes workspace register 5 to 2102_{16}, which points to the sixteenth word of the array. The logical greater than and arithmetic greater than status bits are set while the equal, carry, and overflow status bits are reset. Refer to Section V for additional application notes.

Assembly language format: [<label>] ⌀ INCT ⌀ < ga_s > [⌀ <comment>]

4.2.10 MPY (MULTIPLY)

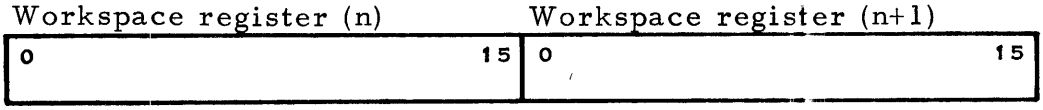
Op code: 3800

Format:

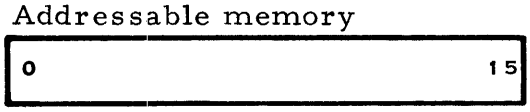


Definition: Multiply the destination operand (a consecutive 2-word area in workspace) by the source operand and replace the destination operand with the result. The multiplication operation may be graphically represented as follows:

Destination operand workspace registers



Source operand



The first word of the destination operand shown above is addressed by the contents of the D field. The multiplicand (unsigned magnitude value of 16 bits) is located right-justified in the workspace register addressed by the D field (represented by workspace n above). The 16-bit, unsigned multiplier

is located in the source operand. When the multiplication operation is complete, the product appears right-justified in the entire 2-word area addressed by the D field as a 32-bit unsigned magnitude value. The maximum value of either input operand is $FFFF_{16}$ and the maximum value of the unsigned product is $(16^8 - 2(16^4) + 1)$ or $FFFE\ 0001_{16}$.

If the destination operand is specified as workspace register 15, the first word of the destination operand is workspace register 15 and the second word of the destination operand is the memory word immediately following the workspace memory area.

Addressing modes: Format IX instructions

Status affected: None

Execution results: $(wa(n)_d) * (ga_s) \longrightarrow (wa(n \text{ and } n+1)_d)$

Application notes: Use the MPY instruction to perform a magnitude multiplication. For example, if workspace register 5 contains 0012_{16}, workspace register 6 contains $1B31_{16}$, and memory location NEW contains 0005_{16}, then the instruction

```
MPY     @NEW,5
```

changes the contents of workspace register 5 to 0000_{16} and workspace register 6 to $005A_{16}$. The source operand is unchanged. The status register is not affected by this instruction.

Assembly language format:

[< label >] ⊬ MPY ⊬< ga_s >,< wa_d > [⊬< comment >]

4.2.11 NEG (NEGATE)

Op code: 0500

Format:

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

Definition: Replace the source operand with the two's complement of the source operand. The AU determines the two's complement value by inverting all bits of the source operand and adding one to the resulting word. The AU then compares the result to zero and sets/resets the status bits to indicate the result of the comparison.

Addressing mode: Format VI instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: $-(ga_s) \longrightarrow (ga_s)$

Application notes: Use the NEG instruction to make the contents of an addressable memory location its additive inverse. For example, if workspace register 5 contains the value $A342_{16}$, then the instruction

```
NEG   5
```

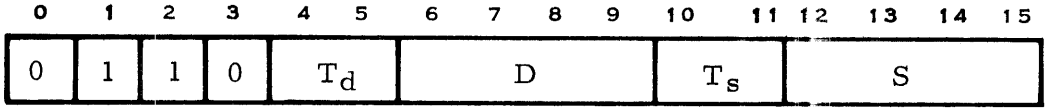
changes the contents of workspace register 5 to $5CBE_{16}$. The logical greater than and arithmetic greater than status bits set while the equal status bit resets.

Assembly language format: [<label>] ⊬ NEG ⊬ < ga_s > [⊬ <comment>]

4.2.12 S (SUBTRACT)

Op code: 6000

Format:



Definition: Subtract a copy of the source operand from the destination operand and place the difference in the destination operand. The AU compares the difference to zero and sets/resets the status bits to indicate the result of the comparison. When there is a carry out of bit zero, the carry status bit sets. When there is an overflow (the difference cannot be represented within a word as a two's complement value), the overflow status bit sets. The source operand remains unchanged.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

Execution results: $(ga_d) - (ga_s) \longrightarrow (ga_d)$

Application notes: Use the S instruction to subtract signed integer values. For example, if memory location OLDVAL contains a value of 1225_{16} and memory location NEWVAL contains a value of 8223_{16}, then the instruction

```
S     @OLDVAL,@NEWVAL
```

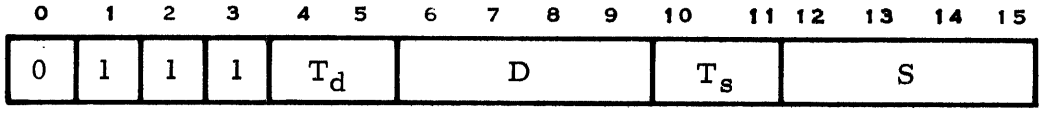
results in the contents of NEWVAL changing to $6FFE_{16}$. The logical greater than, arithmetic greater than, carry, and overflow status bits set while the equal status bit resets.

Assembly language format: [<label>] ⊬ S ⊬ < ga_s >,< ga_d > [⊬ <comment>]

4.2.13 SB (SUBTRACT BYTE)

Op code: 7000

Format:



Definition: Subtract a copy of the source operand (byte) from the destination operand (byte) and replace the destination operand byte with the difference. When the destination operand byte is addressed in the workspace register mode, the right-most byte (bits 8-15) in the workspace register is unchanged. The AU compares the result byte to zero and sets/resets the status bits accordingly. When there is a carry out of the most significant bit of the byte, the carry status bit sets. When there is an overflow (the difference cannot be represented as an 8-bit, two's complement value in a byte), the overflow status bit sets. If the result byte establishes odd parity (an odd number of logic one bits in the byte), the odd parity status bit sets.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, equal, carry, overflow, and odd parity.

Execution results: $(ga_d) - (ga_s) \longrightarrow (ga_d)$

Application notes: Use the SB instruction to subtract signed integer bytes. For example, if workspace register 6 contains the value $121C_{16}$, memory location $121C_{16}$ contains the value 2331_{16}, and workspace register 1 contains the value 1344_{16}, then the instruction

```
SB    *6+,1
```

results in the contents of workspace register 6 changing to $121D_{16}$ and the contents of workspace register 1 changing to $F044_{16}$. The logical greater than status bit sets while the other status bits affected by this instruction reset.

Assembly language format: [<label>] ⊬ SB ⊬ < ga_s >,< ga_d> [⊬ <comment>]

4.3 BRANCH (TRANSFER OF CONTROL) INSTRUCTIONS

Branch instructions transfer control either unconditionally or conditionally, according to the state of one or more status bits of the status register. The instructions included in this paragraph are:

Instruction	Mnemonic	Paragraph
Branch	B	4.3.1
Branch and Link	BL	4.3.2
Branch and Load WP	BLWP	4.3.3
Jump if Equal	JEQ	4.3.4
Jump if Greater Than	JGT	4.3.5
Jump if High or Equal	JHE	4.3.6
Jump if Logical High	JH	4.3.7
Jump if Logical Low	JL	4.3.8
Jump if Low or Equal	JLE	4.3.9
Jump if Less Than	JLT	4.3.10
Unconditional Jump	JMP	4.3.11
Jump if No Carry	JNC	4.3.12
Jump if Not Equal	JNE	4.3.13
Jump if No Overflow	JNO	4.3.14
Jump if Odd Parity	JOP	4.3.15
Jump On Carry	JOC	4.3.16
Return WP	RTWP	4.3.17
Execute	X	4.3.18

Note that for all jump instructions, if the displacement is equal to zero, the transfer is equal to a (PC) + 2⟶(PC) execution result.

4.3.1 B (BRANCH)

Op code: 0440

Format:

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

Definition: Replace the PC contents with the source address and transfer control to the instruction at that location.

Addressing mode: Format VI instructions

Status affected: None

Execution results: $ga_s \longrightarrow (PC)$

Application notes: Use the B instruction to transfer control to another section of code to change the linear flow of the program. For example, if the contents of workspace register 3 is $21CC_{16}$, then the instruction

```
B     *3
```

causes the word at location $21CC_{16}$ to be used as the next instruction, because this value replaces the contents of the PC when this instruction is executed.

Assembly language format: [< label >] ⌀ B ⌀ < ga_s > [⌀ < comment >]

4.3.2 BL (BRANCH AND LINK)

Op code: 0680

Format:

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

Definition: Place the source address in the program counter, place the address of the instruction following the BL instruction (in memory) in workspace register 11, and transfer control to the new PC contents.

Addressing modes: Format VI instructions

Status affected: None

Workspace registers affected: 11

Execution results: (PC)⟶(WR11) and $ga_s \longrightarrow (PC)$

Application notes: Use the BL instruction when return linkage is required. For example, if the instruction

```
BL    @TRAN
```

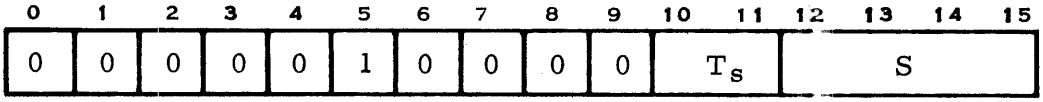
occurs at memory location (PC count) $04BC_{16}$, then this instruction has the effect of placing memory location TRAN in the PC and placing the value $04C0_{16}$ in workspace register 11. Refer to Section V for additional application notes.

Assembly language format: [< label >] ⌀ BL ⌀ < ga_s > [⌀ < comment >]

4.3.3 BLWP (BRANCH AND LOAD WORKSPACE POINTER)

Op code: 0400

Format:



Definition: Place the source operand in the WP and the word immediately following the source operand in the PC. Place the previous contents of the WP in the new workspace register 13, place the previous contents of the PC (address of the instruction following BLWP) in the new workspace register 14, and place the contents of the ST register in the new workspace register 15. When all store operations are complete, the AU transfers control to the new PC.

Addressing modes: Format VI instructions

Status affected: None

Workspace registers affected: WR13, WR14, and WR15

Execution results: $(ga_s) \longrightarrow (WP)$
$(ga_s + 2) \longrightarrow (PC)$
(original WP) $\longrightarrow$ (WR13)
(original PC) $\longrightarrow$ (WR14)
(original ST) $\longrightarrow$ (WR15)

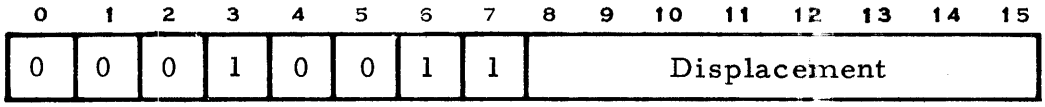
Application notes: Use the BLWP instruction for linkage to subroutines, program modules, or other program that do not necessarily share the calling program workspace. Refer to Section V for a detailed explanation and example.

Assembly language format: [< label >] ⌀ BLWP ⌀ < ga_s > [⌀ < comment >]

4.3.4 JEQ (JUMP IF EQUAL)

Op code: 1300

Format:



Definition: When the equal status bit is set, transfer control by adding the signed displacement in the instruction word to the program counter and then place the sum in the PC to transfer control.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement) ⟶ (PC) if EQ = 1
(PC) + 2 ⟶ (PC) if EQ = 0

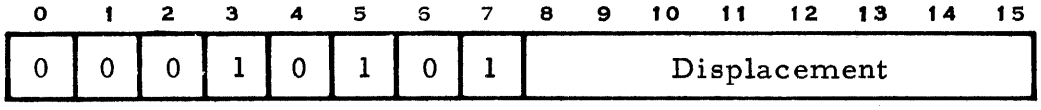
Application notes: Use the JEQ instruction to transfer control when the equal status bit is set and to test CRU bits.

Assembly language format: [< label >] ⊬ JEQ ⊬ < disp > [⊬ < comment >]

4.3.5 JGT (JUMP IF GREATER THAN)

Op code: 1500

Format:



Definition: When the arithmetic greater than status bit is set, add the signed displacement in the instruction word to the PC and place the sum in the PC. Transfer control to the new PC location.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement) ⟶ (PC) if A> = 1
(PC) + 2 (PC) if A> = 0

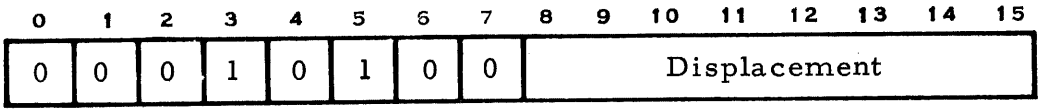
Application notes: Transfers control if the arithmetic greater than status bit is set.

Assembly language format: [< label >] ⊬ JGT ⊬ < disp > [⊬ < comment >]

4.3.6 JHE (JUMP IF HIGH OR EQUAL)

Op code: 1300

Format:



Definition: When the equal status bit or the logical greater than status bit is set, add the signed displacement in the instruction word to the PC and replace the contents of the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement) ⟶ (PC) if L > or EQ = 1
(PC) + 2 ⟶ (PC) if L > and EQ = 0

Application notes: Use the JHE instruction to transfer control when either the logical greater than or equal status bit is set.

Assembly language format: [<label>] ␣ JHE ␣ <disp> [␣ <comment>]

4.3.7 JH (JUMP IF LOGICAL HIGH)

Op code: 1B00

Format:

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

Definition: When the equal status bit is reset and the logical greater than status bit is set, add the signed displacement in the instruction word to the contents of the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement) ⟶ (PC) if L > = 1 and EQ = 0
(PC) + 2 ⟶ (PC) if L > = 0 or EQ = 1

Application notes: Use the JH instruction to transfer control when the equal status bit is reset and the logical status bit is set.

Assembly language format: [<label>] ␣ JH ␣ <disp> [␣ <comment>]

4.3.8 JL (JUMP IF LOGICAL LOW)

Op code: 1A00

Format:

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

Definition: When the equal and logical greater than status bits are reset, add the signed displacement in the instruction word to the PC contents and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement) ⟶ (PC) if L > and EQ = 0
(PC) + 2 ⟶ (PC) if L > or EQ = 1

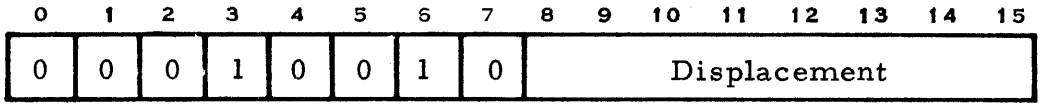
Application notes: Use the JL instruction to transfer control when the equal and logical greater than status bits are reset.

Assembly language format: [< label >] ␣ JL ␣ < disp > [␣ < comment >]

4.3.9 JLE (JUMP IF LOW OR EQUAL)

Op code: 1200

Format:



Definition: When the equal status bit is set or the logical greater than status bit is reset, add the signed displacement in the instruction word to the contents of the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (Displacement) ⟶ (PC) if L > = 0 or EQ = 1
(PC) + 2 ⟶ (PC) if L > = 1 and EQ = 0

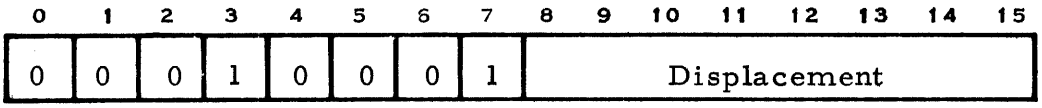
Application notes: Use the JLE instruction to transfer control when the equal status bit is set or the logical greater than status bit is reset.

Assembly language format: [< label >] ␣ JLE ␣ < disp > [␣ < comment >]

4.3.10 JLT (JUMP IF LESS THAN)

Op code: 1100

Format:



Definition: When the equal and arithmetic greater than status bits are reset, add the signed displacement in the instruction word to the PC and replace the PC contents with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement) ⟶ (PC) if A > and EQ = 0
(PC) + 2 ⟶ (PC) if A > or EQ = 1

Application notes: Use the JLT instruction to transfer control when the equal and arithmetic greater than status bits are reset.

Assembly language format: [< label >] ♭ JLT ♭ < disp > [♭ < comment >]

4.3.11 JMP (JUMP UNCONDITIONAL)

Op code: 1000

Format:

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

Definition: Add the signed displacement in the instruction word to the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement) ⟶ (PC)

Application notes: Use the JMP instruction to transfer control to another section of the program module.

Assembly language format: [< label >] ♭ JMP ♭ < disp > [♭ < comment >]

4.3.12 JNC (JUMP IF NO CARRY)

Op code: 1700

Format:

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

Definition: When the carry status bit is reset, add the signed displacement in the instruction word to the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement)⟶(PC) if C = 0
(PC) + 2⟶(PC) if C = 1

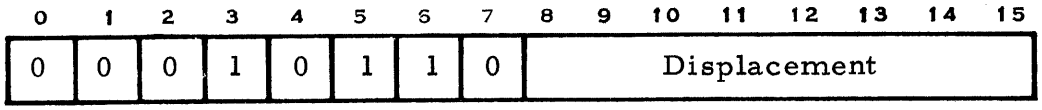
Application notes: Use the JNC instruction to transfer control when the carry status bit is reset.

Assembly language format: [< label >] ⌀ JNC ⌀ < disp > [⌀ < comment >]

4.3.13 JNE (JUMP IF NOT EQUAL)

Op code: 1600

Format:



Definition: When the equal status bit is reset, add the signed displacement in the instruction word to the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (displacement)⟶(PC) if EQ = 0
(PC) + 2 ⟶(PC) if EQ = 1

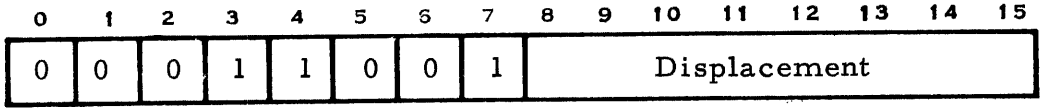
Application notes: Use the JNE instruction to transfer control when the equal status bit is reset. The JNE instruction is also useful in testing CRU bits.

Assembly language format: [< label >] ⌀ JNE ⌀ < disp > [⌀ < comment >]

4.3.14 JNO (JUMP IF NO OVERFLOW)

Op code: 1900

Format:



Definition: When the overflow status bit is reset, add the signed displacement in the instruction word to the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (Displacement)⟶(PC) if OV = 0
(PC) + 2 ⟶(PC) if OV = 1

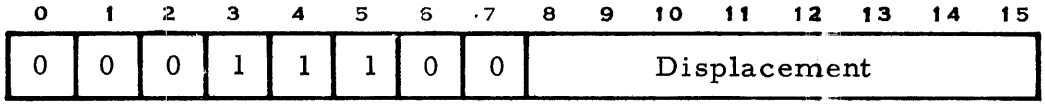
Application notes: Use the JNO instruction to transfer control when the overflow status bit is reset. Refer to Section V for additional application notes.

Assembly language format: [< label >] ⌀ JNO ⌀ < disp > [⌀ < comment >]

4.3.15 JOP (JUMP IF ODD PARITY)

Op code: 1C00

Format:



Definition: When the odd parity status bit is set, add the signed displacement in the instruction word to the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (Displacement) ⟶ (PC) if OP = 1
(PC) + 2 ⟶ (PC) if OP = 0

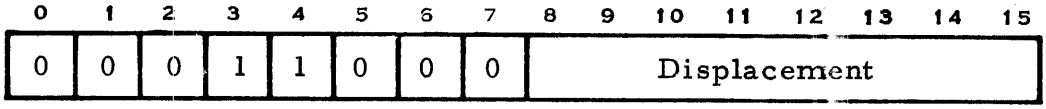
Application notes: Use the JOP instruction to transfer control when there is odd parity. Odd parity indicates that there is an odd number of logic one bits in the byte tested. Refer to Section V for additional application notes.

Assembly language format: [< label >] ⌀ JOP ⌀ < disp > [⌀ < comment >]

4.3.16 JOC (JUMP ON CARRY)

Op code: 1800

Format:



Definition: When the carry status bit is set, add the signed displacement in the instruction word to the PC and replace the PC with the sum.

Addressing mode: Format II instructions

Status affected: None

Execution results: (PC) + (Displacement) ⟶ (PC) if C = 1
(PC) + 2 ⟶ (PC) if C = 0

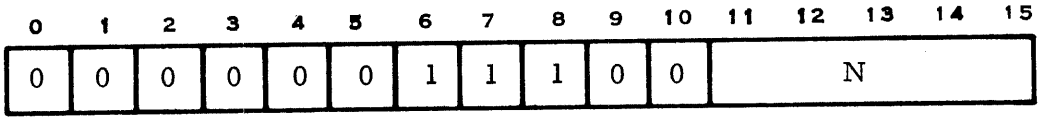
Application notes: Use the JOC instruction to transfer control when the carry status bit is set.

Assembly language format: [< label >] ⊬ JOC ⊬ < disp > [⊬ < comment >]

4.3.17 RTWP (RETURN WITH WORKSPACE POINTER)

Op code: 0380

Format:



Definition: Replace the contents of the WP register with the contents of the current workspace register 13. Replace the contents of the PC with the contents of the current workspace register 14. Replace the contents of the ST register with the contents of the current workspace register 15. The effect of this instruction is to restore the execution environment that existed prior to an interrupt, a BLWP instruction, or an XOP instruction.

Addressing mode: Format VII instructions

Status affected: Restores all status bits to the value contained in workspace register 15.

Execution results: (WR13) ⟶ (WP)
(WR14) ⟶ (PC)
(WR15) ⟶ (ST)

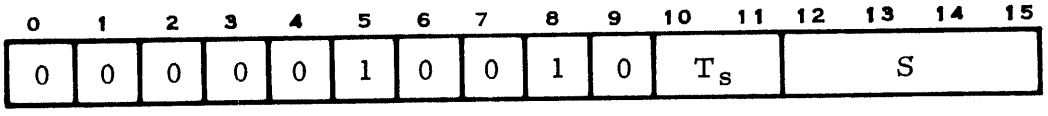
Application notes: Use the RTWP instruction to restore the execution environment after the completion of execution of an interrupt, a BLWP instruction, or an XOP instruction. Refer to Section V for additional information.

Assembly language format: [< label >] ⊬ RTWP [⊬ < comment >]

4.3.18 X (EXECUTE)

Op code: 0480

Format:



Definition: Execute the source operand as an instruction. When the source operand is not a singleword instruction, the word or words following the execute instruction are used with the source operand as a 2-word or 3-word

instruction. The source operand, when executed as an instruction, may affect the contents of the status register. The PC increments by either one, two, or three words depending upon the source operand.

Addressing modes: Format VI instructions

Status affected: None (execute instruction only)

Execution results: Dependent upon the source operand.

Application notes: Use the X instruction to execute the source operand as an instruction. This is primarily useful when the instruction to be executed is dependent upon a variable factor. Refer to Section V for additional application notes.

Assembly language format: [< label >] ⍽ X ⍽ < ga_s > [⍽ < comment >]

4.4 COMPARE INSTRUCTIONS

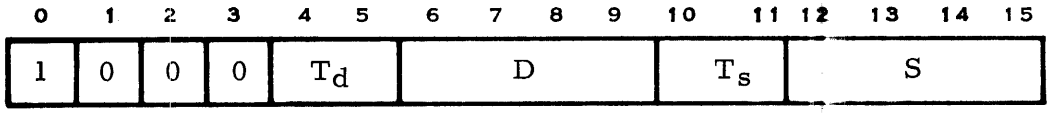
Compare instructions have no effect other than the setting/resetting of appropriate status bits in the status register. The compare instructions perform both arithmetic and logic comparisons. The arithmetic comparison is on the two operands as two's complement values and the logical comparison is on the two operands as unsigned magnitude values. The instructions included are:

Instruction	Mnemonic	Paragraph
Compare Words	C	4.4.1
Compare Bytes	CB	4.4.2
Compare Immediate	CI	4.4.3
Compare Ones Corresponding	COC	4.4.4
Compare Zeros Corresponding	CZC	4.4.5

4.4.1 C (COMPARE WORDS)

Op code: 8000

Format:



Definition: Compare the source operand (word) with the destination operand (word) and set/reset the status bits to indicate the results of the comparison. The arithmetic and equal comparisons compare the operand as signed, two's complement values. The logical comparison compares the two operands as unsigned, 16-bit magnitude values.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: Comparison and status bits set/reset.

Application notes: C compares the two operands as signed, two's complement values and as unsigned integers. Some examples are:

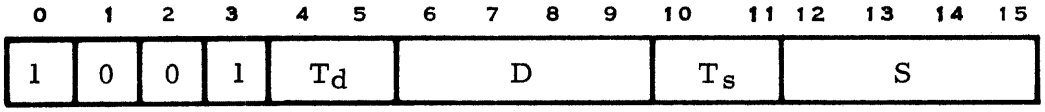
Source	Destination	Logical	Arithmetic	Equal
FFFF	0000	1	0	0
7FFF	0000	1	1	0
8000	0000	1	0	0
8000	7FFF	1	0	0
7FFF	7FFF	0	0	1
7FFF	8000	0	1	0

Assembly language format: [<label>] ♭ C ♭ <ga_s>,<ga_d> [♭ <comment>]

4.4.2 CB (COMPARE BYTES)

Op code: 9000

Format:



Definition: Compare the source operand (byte) with the destination operand (byte) and set/reset the status bits according to the result of the comparison. CB uses the same comparison basis as does C. If the source operand contains an odd number of logic one bits, the odd parity status bit sets. The operands remain unchanged.

Addressing modes: Format I instructions

Status affected: Logic greater than, arithmetic greater than, equal, and odd parity.

Execution results: Comparison and set/reset status bits.

Application notes: CB compares the two operands as signed, two's complement values or as unsigned integers. Some examples are:

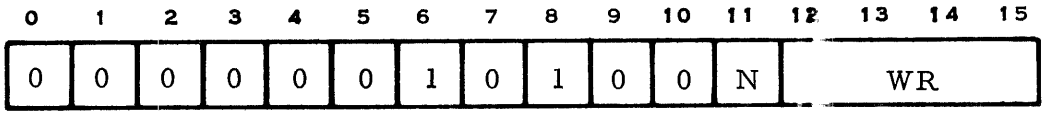
Source	Destination	Logical	Arithmetic	Equal	Odd Parity
FF	00	1	0	0	0
7F	00	1	1	0	1
80	00	1	0	0	1
80	7F	1	0	0	1
7F	7F	0	0	1	1
7F	80	0	1	0	1

Assembly language format: [< label >] ⊬ CB ⊬ < ga_s >, < ga_d > [⊬ < comment >]

4.4.3 CI (COMPARE IMMEDIATE)

Op code: 0280

Format:



Definition: Compare the contents of the specified workspace register with the word in memory immediately following the instruction. Set/reset the status bits according to the comparison. CI makes the same type of comparison as does C.

Addressing mode: Format VIII instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: Comparison and set/reset status bits.

Application notes: Use the CI instruction to compare the workspace register to an immediate operand. For example, if the contents of workspace register 9 is 2183_{16}, then the instruction

```
CI    9,>F330
```

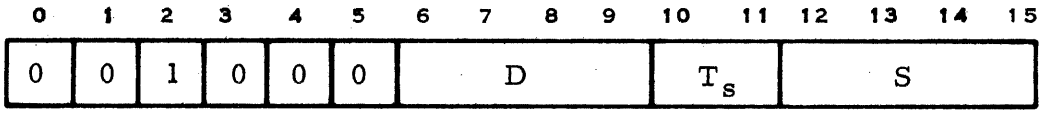
results in the arithmetic greater than status bit set and the logical greater than and equal status bits reset.

Assembly language format: [< label >] ⊬ CI ⊬ < wa >, < iop > [⊬ < comment >]

4.4.4 COC (COMPARE ONES CORRESPONDING)

Op code: 2000

Format:



Definition: When the bits in the destination operand workspace register that correspond to the logic one bits in the source operand are equal to logic one, set the equal status bit. The source and destination operands are unchanged.

Addressing modes: Format III instructions

Status affected: Equal

Execution results: Compare logic one and set/reset the equal status bit.

Application notes: Use the COC instruction to test single/multiple bits within a word in a workspace register. For example, if TESTBI contains the word $C102_{16}$ and workspace register 8 contains the value $E306_{16}$, then the instruction

```
COC     @TESTBITS,8
```

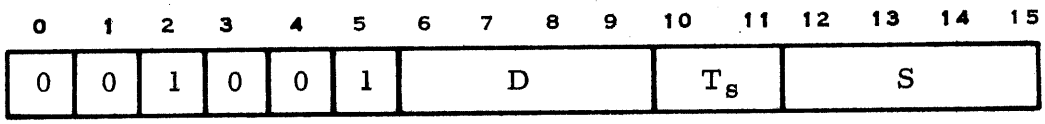
results in setting the equal status bit. If workspace register 8 were to contain $E301_{16}$, the equal status bit would reset.

Assembly language format: [< label >] ⍽ COC ⍽ <ga_s>,<wa_d>[⍽ < comment >]

4.4.5 CZC (COMPARE ZEROS CORRESPONDING)

Op code: 2400

Format:



Definition: When the bits in the destination operand workspace register that correspond to the one bits in the source operand are all equal to a logic zero, set the equal status bit. The source and destination operands are unchanged.

Addressing modes: Format III instructions

Status affected: Equal

Execution results: Compare for logic zero and set/reset equal status bit.

Application notes: Use the CZC instruction to test single/multiple bits within a word in a workspace register. For example, if the memory location labeled TESTBI contains the value $C102_{16}$, and workspace register 8 contains 2301_{16}, then the instruction

```
CZC    @TESTBIT,8
```

results in the equal status bit reset. If workspace register 8 contained the value 2201_{16}, then the equal status bit would set.

Assembly language format: [<label>] ⊬ CZC ⊬ <ga_s>,<wa_d>[⊬<comment>]

4.5 CONTROL AND CRU INSTRUCTIONS

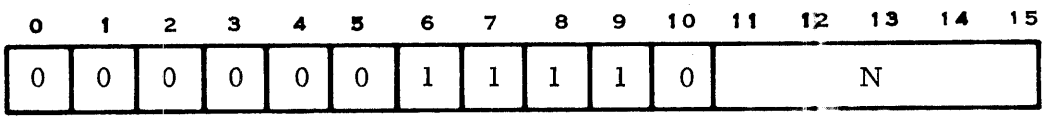
Control instructions affect the operation of the AU and associated portions of the CPU. CRU instructions affect the modules connected to the communication register unit. The instructions included in this paragraph are:

Instruction	Mnemonic	Paragraph
Clock Off	CKOF	4.5.1
Clock On	CKON	4.5.2
Load CRU	LDCR	4.5.3
Idle	IDLE	4.5.4
Reset	RSET	4.5.5
Set Bit to Logic One	SBO	4.5.6
Set Bit to Logic Zero	SBZ	4.5.7
Store CRU	STCR	4.5.8
Test Bit	TB	4.5.9

4.5.1 CKOF (CLOCK OFF)

Op code: 03C0

Format:



Definition: Stop the line frequency clock (120 Hz). No status bits are changed and the clock interrupt will not occur as long as the clock is off.

Addressing mode: Format VII instructions

Status affected: None

Execution results: The line frequency clock is turned off

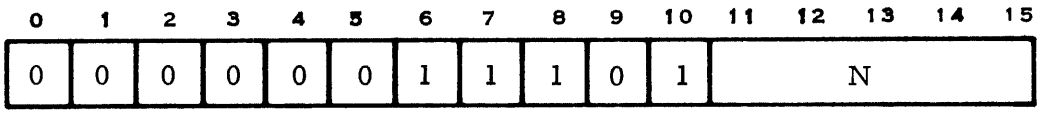
Application notes: Refer to Section V

Assembly language format: [< label >] ⊬ CKOF [⊬ < comment >]

4.5.2 CKON (CLOCK ON)

Op code: 03A0

Format:



Definition: Enable the line frequency clock. If interrupt level five is enabled, an interrupt will occur every 8.33 ms after the initial interrupt, which may occur from 1 μs to 8.33 ms after the clock is turned on. Interrupt five may be enabled/disabled by the interrupt mask as necessary.

Addressing mode: Format VII instructions

Status affected: None

Execution results: The line frequency clock is turned on

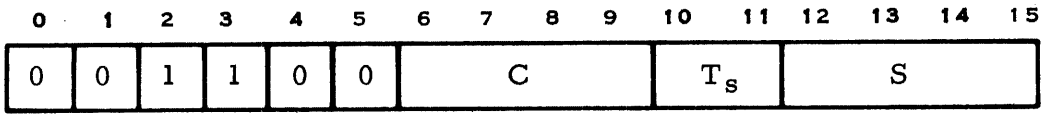
Application notes: Refer to Section V

Assembly language format: [< label >] ⊬ CKON [⊬ < comment >]

4.5.3 LDCR (LOAD COMMUNICATIONS REGISTER UNIT)

Op code: 3000

Format:



Definition: Transfer the number of bits specified in the C field from the source operand to the CRU. The transfer begins with the least significant bit of the source operand. The CRU address is contained in bits 3 through 14 of workspace register 12. When the C field contains zero, the number of bits transferred is 16. If the number of bits to be transferred is from one to eight, the source operand address is a byte address. If the number of bits to be transferred is from 9 to 16, the source operand address is a

word address. If the source operand address is odd, the word used for output is byte-reversed after memory read and before transfer to the CRU. When the number of bits transferred is a byte or less, the source operand is compared to zero and the status bits are set/reset, according to the results of the comparison. The odd parity status bit sets when the bits in a byte (or less) to be transferred establish odd parity.

Addressing modes: Format IV instructions

Status affected: Logical greater than, arithmetic greater than, equal, and odd parity (only when the number of bits to transfer is less than nine).

Execution results: The number of bits specified by the C field are transferred to the CRU.

Application notes: Use the LDCR instruction to transfer a specific number of bits from memory to the CRU at the address contained in bits 3 through 14 of workspace register 12. Refer to Section V for a detailed example and explanation of the LDCR instruction.

Assembly language format:

[<label>] ⍽ LDCR ⍽<ga_s>,<cnt> [⍽<comment>]

4.5.4 IDLE (IDLE)

Op code: 0340

Format:

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

Definition: Place the computer in the idle state. Note that the PC is incremented prior to the execution of this instruction and the contents of the PC point to the instruction word in memory immediately following the IDLE instruction. The computer will remain in the IDLE state until an interrupt or start signal occurs. An idle with an interrupt level of 4 (or level 5 with the line frequency clock off) is a lock-up. Only power failure or the load switch pressed can interrupt to recover because level 3 and 4 (level 5 with clock off) cannot occur. These interrupts cannot occur when the AU is not executiong instructions.

Addressing mode: Format VII instructions

Status affected: None

Execution results: The computer is idle.

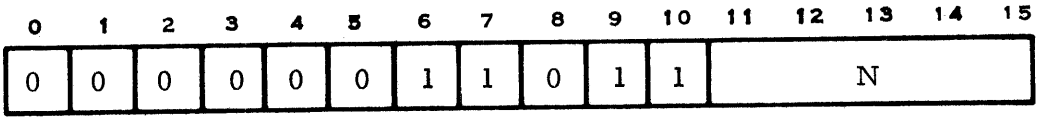
Application notes: Use the IDLE instruction to place the computer in the idle state. This instruction is useful in timing delays using the clock or in waiting for interrupt signals.

Assembly language format: [< label >] ⍽ IDLE [⍽ < comment >]

4.5.5 RSET (RESET)

Op code: 0360

Format:



Definition: The RSET instruction clears the interrupt mask, which disables all except level 0 interrupts. It also resets all directly connected input/output devices and those CRU devices that provide for reset in the interface with the CRU. RSET also resets all pending interrupts and turns the clock off.

Addressing modes: Format VII instructions

Status affected: Resets all bits of the interrupt mask.

Execution results: See definition above.

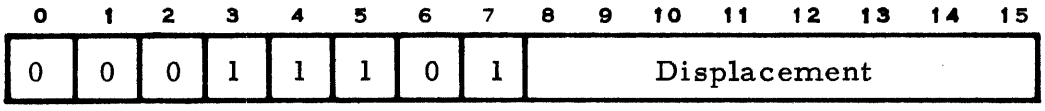
Application notes: Use the reset instruction to reset the interrupt mask to zero, turn off the clock, and (depending on the device and interface) clear any pending interrupt and reset interface electronics.

Assembly language format: [< label >] ⍽ RSET [⍽ < comment >]

4.5.6 SBO (SET BIT TO LOGIC ONE)

Op code: 1D00

Format:



Definition: Set the digital output bit to a logic one on the CRU at the address derived from this instruction. The derived address is the sum of the user supplied signed displacement and the contents of workspace register 12, bits 3 through 14. The execution of this instruction does not affect the status register or the contents of workspace register 12.

Addressing mode: Format II instructions

Workspace register accessed: WR12

Status affected: None

Execution results: 1 ⟶ CRUbit$_{WR12+displacement}$

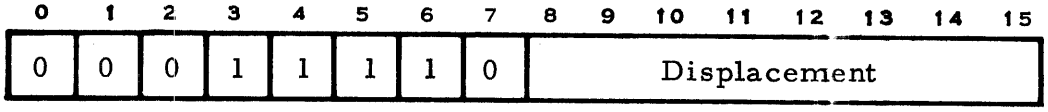
Application notes: Use the SBO instruction to set a CRU bit to a logic one. Refer to Section V for additional application notes.

Assembly language format: [< label >] ♭ SBO ♭ < disp > [♭ < comment >]

4.5.7 SBZ (SET BIT TO LOGIC ZERO)

Op code: 1E00

Format:



Definition: Set the digital output bit to a logic zero on the CRU at the address derived from this instruction. The derived address is the sum of the user supplied signed displacement and the contents of workspace register 12, bits 3 through 14. The execution of this instruction does not affect the status register or the contents of workspace register 12.

Addressing mode: Format II instructions

Status affected: None

Execution results: 0 ⟶ CRUbit$_{WR12+displacement}$

Workspace register accessed: WR12

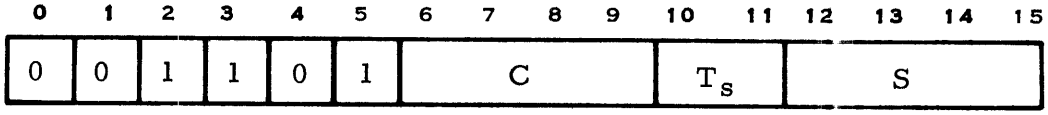
Application notes: Use the SBZ instruction to set a CRU bit to a logic zero. Refer to Section V for additional application notes.

Assembly language format: [< label >] ♭ SBZ ♭ < disp > [♭ < comment >]

4.5.8 STCR (STORE COMMUNICATIONS REGISTER UNIT)

Op code: 3400

Format:



Definition: Transfer the number of bits specified in the C field from the CRU to the source operand. The transfer begins from the CRU address specified in bits 3 through 14 of workspace register 12 to the least significant bit of the source operand and fills the source operand toward the most significant bit. When the C field contains a zero, the number of bits to transfer is 16. If the number of bits to transfer is from one to eight, the source operand address is a byte address. Any bit in the memory byte not filled by the transfer is reset to a zero. When the number of bits to transfer is from 9 to 16, the source operand address is a word address. If the source operand address is odd, the word used for input is byte-reversed after transfer from the CRU and before memory write. If the transfer does not fill the entire memory word, unfilled bits are reset to zero. When the number of bits to transfer is a byte or less, the bits transferred are compared to zero and the status bits set/reset to indicate the results of the comparison. Also, when the bits to be transferred are a byte or less, the odd parity bit sets when the bits establish odd parity.

Addressing modes: Format IV instructions

Status affected: Logical greater than, arithmetic greater than, equal, and odd parity (only when the bits to be transferred total less than nine).

Execution results: The number of bits specified by the C field are transferred to the source operand from the CRU.

Application notes: Use the STCR instruction to transfer a specified number of CRU bits from the CRU to the memory location supplied by the user as the source operand. Note that the CRU base address must be in workspace register 12 prior to the execution of this instruction. Refer to Section V for a detailed explanation and examples of the use of the STCR instruction.

Assembly language format:

[<label>] ⍽ STCR ⍽ <ga_s>,<disp> [⍽<comment>]

4.5.9 TB (TEST BIT)

Op code: 1F00

Format:

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

Definition: Read the digital input bit on the CRU at the address specified by the sum of the user supplied signed displacement and the contents of workspace register 12, bits 3 through 14 and set the equal status bit to the logic value read. The digital input bit and the contents of workspace register 12 are unchanged.

Addressing modes: Format II instructions

Status affected: Equal

Execution results: Equal status bit set to CRU bit.

Application notes: Refer to Section V for a detailed explanation and example of the use of the TB instruction.

Assembly language format: [< label >] ⊬ TB ⊬ < disp > [⊬ < comment >]

4.6 LOAD AND MOVE INSTRUCTIONS

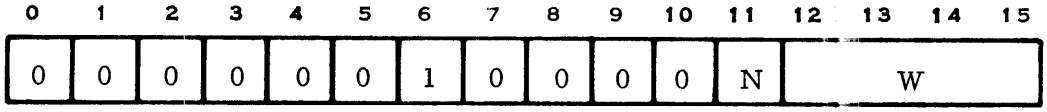
Load and move instructions permit the user to establish the execution environment and the execution results. The instructions included in this paragraph are:

Instruction	Mnemonic	Paragraph
Load Immediate	LI	4.6.1
Load Interrupt Mask Immediate	LIMI	4.6.2
Load ROM and Execute	LREX	4.6.3
Load Workspace Pointer Immediate	LWPI	4.6.4
Move Words	MOV	4.6.5
Move Bytes	MOVB	4.6.6
Store Status	STST	4.6.7
Store Workspace Pointer Immediate	STWP	4.6.8
Swap Bytes	SWPB	4.6.9

4.6.1 LI (LOAD IMMEDIATE)

Op code: 0200

Format:



Definition: Place the immediate operand (the word of memory immediately following the instruction) in the user specified workspace register (W field). The immediate operand and the status register are not affected by the execution of this instruction.

Addressing mode: Format VIII instructions

Status affected: None

Execution results: iop⟶(wa)

Application notes: Use the LI instruction to place an immediate operand in a specified workspace register. This is useful for initializing a workspace register as a loop counter. For example, the instruction

```
LI      7, 5
```

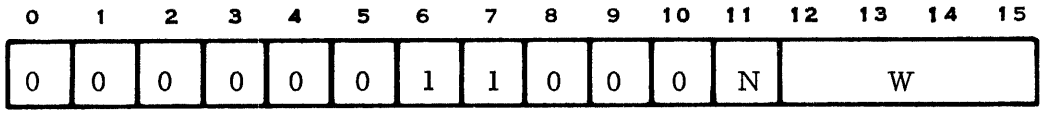
initializes workspace register 7 with the value 0005_{16}. No status bits are affected.

Assembly language format: [< label >] ⍽ LI ⍽ < wa >,< iop > [⍽ < comment >]

4.6.2 LIMI (LOAD INTERRUPT MASK IMMEDIATE)

Op code: 0300

Format:



Definition: Place the low order four bits of the contents of the immediate operand (bits 12 through 15) in the interrupt mask of the status register. The remaining bits of the status register (0 through 11) are not affected. The W field of this instruction is not used and may contain any value.

Addressing mode: Format VIII instructions

Status affected: None

Execution results: $iop_{bits\ 12\text{-}15}$⟶(interrupt mask)

Application notes: Use the LIMI instruction to initialize the interrupt mask for a particular level of interrupt to be accepted. For example, if

```
LIMI    3
```

is input, the interrupt mask is set at level three and the following interrupts are accepted:

Power On

Power Fail

Memory Parity Error

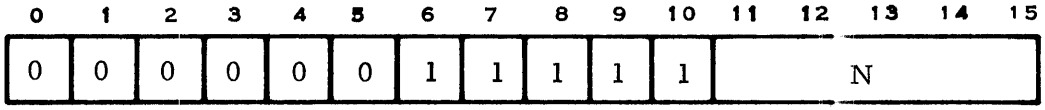
Illegal Instruction

Assembly language format: [< label >] ♭ LIMI ♭ < iop > [♭ < comment >]

4.6.3 LREX (LOAD ROM AND EXECUTE)

Op code: 03E0

Format:



Definition: Load the ROM contents into memory beginning at location zero. Clear the interrupt mask, disabling all except level zero interrupts. Place the first word in memory into the WP register and the second word in memory in the PC register. The computer begins execution at the instruction indicated by the PC. Note that the execution of an LREX instruction overlays memory addresses zero through 255.

Addressing mode: Format VII instructions

Status affected: None

Execution results: (memory word 0000_{16}) ⟶ (WP)
(memory word 0002_{16}) ⟶ (PC)
(old WP) ⟶ (new WR13)
(old PC) ⟶ (new WR14)
(old ST) ⟶ (new WR15) (Interrupt mask = 0000)

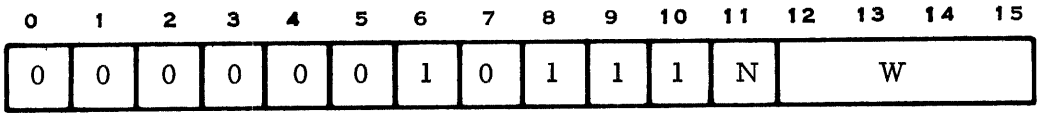
Application notes: Use the LREX instruction to load the ROM contents into memory and execute the set of instructions indicated by the first two words that are loaded into memory. Refer to Section V for additional application notes.

Assembly language format: [< label >] ♭ LREX [♭ < comment >]

4.6.4 LWPI (LOAD WORKSPACE POINTER IMMEDIATE)

Op code: 02E0

Format:



Definition: Replace the contents of the WP register with the immediate operand. The immediate operand is the word of memory immediately following the instruction LWPI. The W field is not used and may contain any value.

Addressing mode: Format VIII instructions

Status affected: None

Execution results: iop ⟶ (WP)

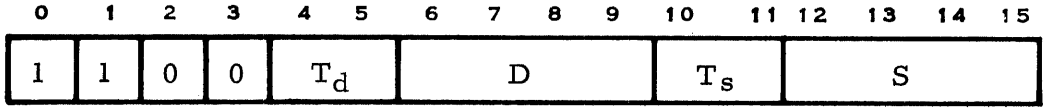
Application notes: Use the LWPI instruction to initialize or change the WP register to alter the workspace environment of the program module. The user should use either a BLWP or a LWPI instruction prior to the use of any workspace register in a program module.

Assembly language format: [< label >] ⏨ LWPI ⏨ < iop > [⏨ < comment >]

4.6.5 MOV (MOVE WORDS)

Op code: C000

Format:



Definition: Replace the destination operand with a copy of the source operand. The AU compares the resulting destination operand to zero and sets/resets the status bits according to the comparison.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: $(ga_s) \longrightarrow (ga_d)$

Application notes: MOV is used to move 16-bit words as follows:

Memory-to-memory (non register)

Load register (memory-to-register)

Register-to-register

Register-to-memory

MOV may also be used to compare a memory location to zero by the use of

```
MOV     7, 7
JNE     TEST
```

which would move register 7 to itself and compare the contents of register 7 to zero. If the contents are not equal to zero, the equal status bit is reset and control transfers to TEST. Another use of MOV, for example, is if workspace register 9 contains 3416_{16} and location ONES contains $FFFF_{16}$, then

```
MOV     @ONES, 9
```

changes the contents of workspace register 9 to $FFFF_{16}$, while the contents of location ONES is not changed. For this example, the logical greater than status bit sets and the arithmetic greater than and equal status bits reset.

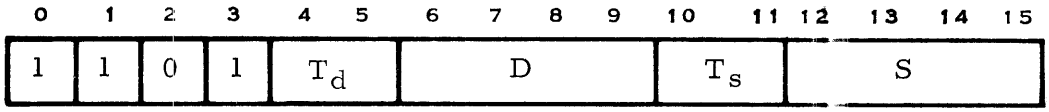
Assembly language format:

[<label>] ⊬ MOV ⊬<ga_s>,<ga_d> [⊬<comment>]

4.6.6 MOVB (MOVE BYTES)

Op code: D000

Format:



Definition: Replace the destination operand (byte) with a copy of the source operand (byte). If the destination operand is addressed in the workspace register mode, the byte addressed is the most significant byte of the word (bits 0-7) and the least significant byte (bits 8-15) is not affected by this instruction. The AU compares the destination operand to zero and sets/resets the status bits to indicate the result of the comparison. The odd parity bit sets when the bits in the destination operand establish odd parity.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, equal, and odd parity.

Execution results: $(ga_s) \longrightarrow (ga_d)$

Application notes: MOVB is used to move bytes in the same combinations as does the MOV instruction. For example, if memory location $1C14_{16}$ contains a value of 2016_{16} and TEMP is located at $1C15_{16}$, and if workspace register 3 contains $542B_{16}$, then the instruction

```
MOVB        @TEMP, 3
```

changes the contents of workspace register 3 to $162B_{16}$. The logical greater than, arithmetic greater than, and odd parity status bits set while the equal status bit resets.

Assembly language format:

[<label>] ⊬ MOVB ⊬<ga_s>,<ga_d> [⊬<comment>]

4.6.7 STST (STORE STATUS)

Op code: 02C0

Format:

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

Definition: Store the status register contents in the specified workspace register.

Addressing mode: Format VIII instructions

Status affected: None

Execution results: (ST) ⟶ (wa)

Application notes: Use the STST instruction to store the ST register contents when applicable.

Assembly language format: [< label >] ⊬ STST ⊬ < wa > [⊬ < comment >]

4.6.8 STWP (STORE WORKSPACE POINTER IMMEDIATE)

Op code: 02A0

Format:

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

Definition: Place a copy of the workspace pointer contents in the specified workspace register.

Addressing mode: Format VIII instructions

Status affected: None

Execution results: (WP) ⟶ (wa)

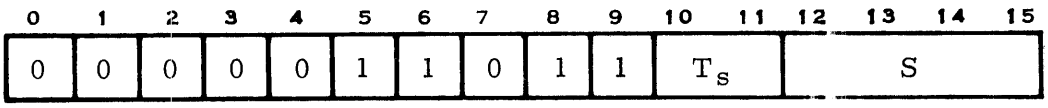
Application notes: Use the STWP instruction to store the contents of the WP register as applicable.

Assembly language format: [< label >] ♭ STWP ♭ < wa > [♭ < comment >]

4.6.9 SWPB (SWAP BYTES)

Op code: 06C0

Format:



Definition: Replace the most significant byte (bits 0-7) of the source operand with a copy of the least significant byte (bits 8-15) of the source operand and replace the least significant byte with a copy of the most significant byte.

Addressing modes: Format VI instructions

Status affected: None

Execution results: $(ga_{s\ byte\ 1}) \longrightarrow (ga_{s\ byte\ 2})$
$(ga_{s\ byte\ 2}) \longrightarrow (ga_{s\ byte\ 1})$

Application notes: Use the SWPB instruction to interchange bytes of an operand prior to executing various byte instructions. For example, if workspace register 0 contains 2144_{16} and memory location 2144_{16} contains the value $F312_{16}$, then the instruction

```
SWPB    *0+
```

changes the contents of workspace register 0 to 2146_{16} and the contents of memory location 2144_{16} to $12F3_{16}$. The status register remains unchanged.

Assembly language format: [< label >] ♭ SWPB ♭ < ga_s > [♭ < comment >]

4.7 LOGICAL INSTRUCTIONS

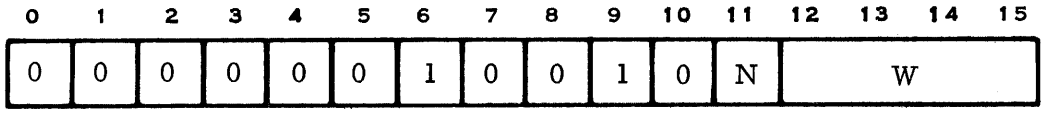
The set of available logical instructions permits the user to perform various logical operations on memory locations and/or workspace registers. The instructions included in this paragraph are:

Instruction	Mnemonic	Paragraph
AND Immediate	ANDI	4.7.1
Clear	CLR	4.7.2
Invert	INV	4.7.3
OR Immediate	ORI	4.7.4
Set to One	SETO	4.7.5
Set Ones Corresponding	SOC	4.7.6
Set Ones Corresponding, Byte	SOCB	4.7.7
Set Zeros Corresponding	SZC	4.7.8
Set Zeros Corresponding, Byte	SZCB	4.7.9
Exclusive OR	XOR	4.7.10

4.7.1 ANDI (AND IMMEDIATE)

Op code: 0240

Format:



Definition: Perform a bit-by-bit AND operation of the 16 bits in the immediate operand and the corresponding bits of the source operand. The immediate operand is the word in memory immediately following the instruction word. Place the result in the workspace register specified in the W field. The AU compares the resulting AND to zero and sets/resets the status bits according to the results of the comparison.

Addressing mode: Format VIII instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: (wa)AND iop ⟶ (wa)

Application notes: Use the ANDI instruction to perform a logical AND with an immediate operand and a workspace register. Each bit of the 16-bit word of both operands follows the truth table

Immediate Operand Bit	Workspace Register Bit	AND Result
0	0	0
0	1	0
1	0	0
1	1	1

For example, if workspace register 0 contains $D2AB_{16}$, the instruction

```
ANDI        0,>6D03
```

results in workspace register 0 changing to 4003_{16}. This AND operation on a bit-by-bit basis is

```
0110110100000011   (Immediate operand)
1101001010101011   (Workspace register 0)
0100000000000011   (Workspace register 0 result)
```

For this example, the logical greater than and arithmetic greater than status bits set while the equal status bit resets. ANDI is also useful for masking out bits of a workspace register.

Assembly language format: [< label >] ⍽ ANDI ⍽ < wa > , < iop > [⍽ < comment >]

4.7.2 CLR (CLEAR)

Op code: 04C0

Format:

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

Definition: Replace the source operand with a full, 16-bit word of zeros.

Addressing modes: Format VI instructions

Status affected: None

Execution results: 0 ⟶ (ga_s)

Application notes: Use the CLR instruction to set a full, 16-bit, memory addressable word to zero. For example, if workspace register 11 contains the value 2001_{16}, then the instruction

```
CLR         *>B
```

results in the contents of memory location 2000_{16} set to 0. Workspace register 11 and the status register are unchanged.

Assembly language format: [< label >] ⊬ CLR ⊬ < ga_s > [⊬ < comment >]

4.7.3 INV (INVERT)

Op code: 0540

Format:

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

Definition: Replace the source operand with the one's complement of the source operand. The one's complement is equivalent to changing each zero in the source operand to a logic one and each logic one in the source operand to a logic zero. The AU compares the result to zero and sets/resets the status bits to indicate the result of the comparison.

Addressing modes: Format VI instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: Refer to the definition above.

Application notes: INV changes each logic zero in the source operand to a logic one and each logic one to a logic zero. For example, if workspace register 11 contains $A54B_{16}$, then the instruction

```
INV         11
```

changes the contents of workspace register 11 to $5AB4_{16}$. The logical greater than and arithmetic greater than status bits set and the equal status bit resets.

Assembly language format: [< label >] ⊬ INV ⊬ < ga_s > [⊬ < comment >]

4.7.4 ORI (OR IMMEDIATE)

Op code: 0260

Format:

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

Definition: Perform an OR operation of the 16-bit immediate operand and the corresponding bits of the workspace register specified in the W field. The immediate operand is the memory word immediately following the ORI instruction. Place the result in the workspace register. The AU compares the result to zero and sets/resets the status bits to indicate the result of the comparison.

Addressing modes: Format VIII instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: iop OR(wa) ⟶ (wa)

Application notes: Use the ORI instruction to perform a logical OR with the immediate operand and a specified workspace register. Each bit of the 16-bit word of both operands is OR'd using the truth table

Immediate operand	Workspace register	OR Result
0	0	0
1	0	1
0	1	1
1	1	1

For example, if workspace register 5 contains $D2AB_{16}$, then the instruction

```
ORI     5,>6D03
```

results in workspace register 5 changing to $FFAB_{16}$. This OR operation on a bit-by-bit basis is

```
0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1   (Immediate operand)
1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1   (Workspace register 5)
-------------------------------
1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1   (Workspace register 5 result)
```

For this example, the logical greater than status bit sets, and the arithmetic greater than and equal status bits reset.

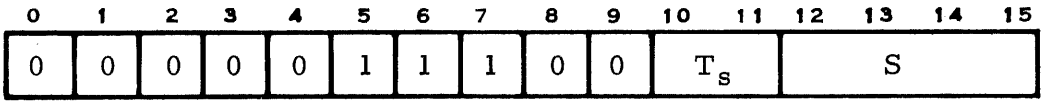
Assembly language format:

```
[<label>] ⊬ ORI ⊬<wa>,<iop> [⊬<comment>]
```

4.7.5 SETO (SET TO ONE)

Op code: 0700

Format:



Definition: Replace the source operand with a 16-bit word of logic one values.

Addressing modes: Format VI instructions

Status affected: None

Execution results: $FFFF_{16} \longrightarrow (ga_s)$

Application notes: Use the SETO instruction to initialize an addressable memory to a -1 value. For example, the instruction

```
SETO    3
```

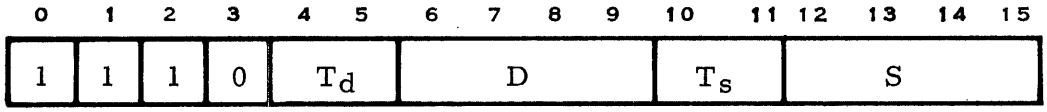
initializes workspace register 3 to a value of $FFFF_{16}$. The contents of the status register is unchanged. This is a useful means of setting flag words.

Assembly language format: [< label >] ⊬ SETO ⊬ < ga_s > [⊬ < comment >]

4.7.6 SOC (SET ONES CORRESPONDING)

Op code: E000

Format:



Definition: Set to a logic one the bits in the destination operand that correspond to any logic one bit in the source operand. Leave unchanged the bits in the destination operand that are in the same bit positions as the logic zero bits in the source operand. The changed destination operand replaces the original destination operand. This operation is effectively an OR of the two operands. The AU compares the result to zero and sets/resets the status bits to indicate the result of the comparison.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: $(ga_s)OR(ga_d) \longrightarrow (ga_d)$

Application notes: Use the SOC instruction to OR the 16-bit contents of two operands. For example, if workspace register 3 contains $FF00_{16}$ and location NEW contains $AAAA_{16}$, then the instruction

```
SOC     3, @NEW
```

changes the contents of location NEW to $FFAA_{16}$ while the contents of work space register 3 is unchanged. This is shown as

```
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0   (Source operand)
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0   (Destination operand)
-------------------------------
1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0   (Destination operand result)
```

For this example, the logical greater than status bit sets and the arithmetic greater than and equal status bits reset.

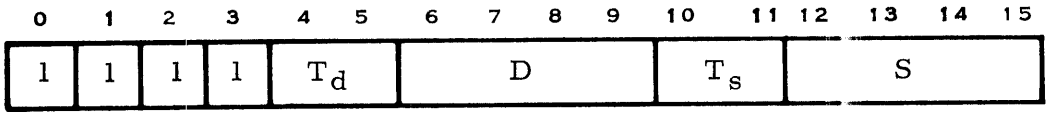
Assembly language format:

[<label>] ⊬ SOC ⊬<ga_s>,<ga_d> [⊬<comment>]

4.7.7 SOCB (SET ONES CORRESPONDING, BYTE)

Op code: F000

Format:



Definition: Set to a logic one the bits in the destination operand byte that correspond to any logic one in the source operand byte. Leave unchanged the bits in the destination operand that are in the same bit positions as the logic zero bits in the source operand byte. The changed destination operand byte replaces the original destination operand byte. This operation is also effectively an OR of the two operand bytes. The AU compares the resulting destination operand byte to zero and sets/resets the status bits to indicate the results of the comparison. The odd parity status bit sets when the bits in the resulting byte establish odd parity.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, equal and odd parity.

Execution results: $(ga_s)OR(ga_d) \longrightarrow (ga_d)$

Application notes: Use the SOCB instruction to OR two byte operands. For example, if workspace register 5 contains the value $F013_{16}$ and workspace register 8 contains the value $AA24_{16}$, then the instruction

```
SOCB        5, 8
```

changes the contents of workspace register 8 to $FA24_{16}$, while the contents of workspace register 5 is unchanged. This is shown as

```
1 1 1 1 0 0 0 0 0 0 0 1 0 0 1 1   (Source operand)
1 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0   (Destination operand)
-------------------------------
1 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0   (Destination operand result)
                   ^^^^^^^^^^^^
                   (Unchanged)
```

For this example, the logical greater than status bit sets while the arithmetic greater than, equal, and odd parity status bits reset.

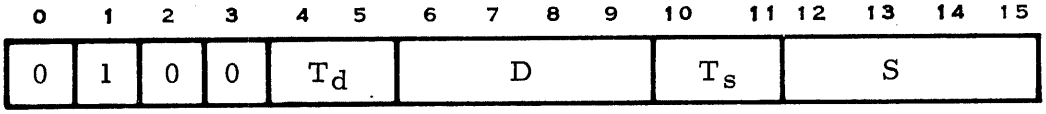
Assembly language format:

[<label>] ⱡ SOCB ⱡ<ga_s>,< ga_d>[ⱡ<comment>]

4.7.8 SZC (SET ZEROS CORRESPONDING)

Op code: 4000

Format:



Definition: Set to a logic zero the bits in the destination operand that correspond to the bit positions equal to a logic one in the source operand. This operation is effectively an AND operation of the one's complement of the source operand and the destination operand. The AU compares the resulting destination operand to zero and sets/resets the status bits to indicate the results of the comparison.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: (INV (ga_s))AND(ga_d)⟶(ga_d)

Application notes: Use the SZC instruction to turn off flag bits or AND the contents of the one's complement of the source operand and the destination operand. For example, if workspace register 5 contains $6D03_{16}$ and workspace register 3 contains $D2AA_{16}$, then the instruction

```
SZC         5, 3
```

changes the contents of workspace register 3 to $92A8_{16}$ while the contents of workspace register 5 remain unchanged. This is shown as

```
0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1   (Source operand)
1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0   (Destination operand)
-------------------------------
1 0 0 1 0 0 1 0 1 0 1 0 1 0 0 0   (Destination operand result)
```

For this example, the logical greater than status bit sets while the arithmetic greater than and equal status bits reset.

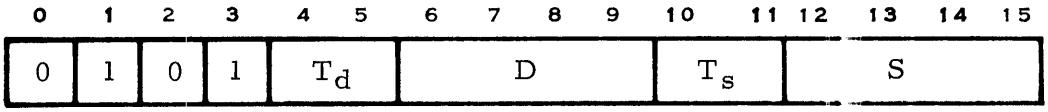
Assembly language format:

[<label>] ⊬ SZC ⊬<ga_s>,<ga_d>[⊬<comment>]

4.7.9 SZCB (SET ZEROS CORRESPONDING, BYTE)

Op code: 5000

Format:



Definition: Set to a logic zero the bits in the destination operand byte that correspond to the bit positions equal to a logic one in the source operand byte. This operation is effectively an AND operation of the one's complement of the source operand byte and the destination operand byte. The AU compares the resulting destination operand byte to zero and sets/resets the status bits to indicate the result of the comparison. The odd parity status bit sets when the bits in the resulting destination operand byte establish odd parity. When the destination operand is addressed in the workspace register mode, the least significant byte (bits 8-15) is unchanged.

Addressing modes: Format I instructions

Status affected: Logical greater than, arithmetic greater than, equal, and odd parity.

Execution results: $(INV(ga_s))AND(ga_d) \longrightarrow (ga_d)$ (Byte, bits 0-7 or 8-15)

Application notes: The SZCB instruction is used for the same applications, except for bytes instead of words. For example, if location BITS contains the value $F018_{16}$, and location TESTVA contains the value $AA24_{16}$, then

```
SZCB    @BITS, @TESTVAL
```

changes the contents of TESTVA to $0A24_{16}$ while BITS remains unchanged. This is shown as

1 1 1 1 0 0 0 0 0 0 1 1 0 0 0 (Source operand)

1 0 1 0 1 0 1 0 0 0 1 0 0 1 0 0 (Destination operand)

0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 (Destination operand result)

(Unchanged)

For this example, the logical greater than and arithmetic greater than status bits set while the equal and odd parity status bits reset.

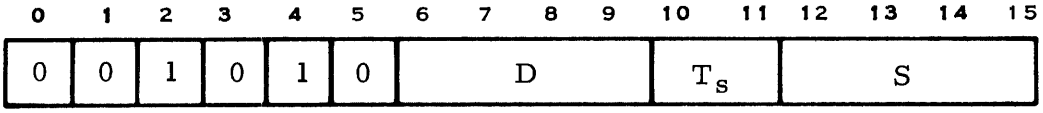
Assembly language format:

[< label >] ⊬ SZCB ⊬<ga_s>,<ga_d>[⊬< comment >]

4.7.10 XOR (EXCLUSIVE OR)

Op code: 2800

Format:



Definition: Perform a bit-by-bit exclusive OR of the source and destination operands, and replace the destination operand with the result. This exclusive OR is accomplished by setting the bits in the resultant destination operand to a logic one when the corresponding bits of the two operands are not equal. The bits in the resultant destination operand are reset to zero when the corresponding bits of the two operands are equal. The AU compares the resultant destination operand to zero and sets/resets the status bits to indicate the result of the comparison.

Addressing modes: Format III instructions

Status affected: Logical greater than, arithmetic greater than, and equal.

Execution results: $(ga_s)XOR(wa_d) \longrightarrow (wa_d)$

Application notes: Use the XOR instruction to perform an exclusive OR on two word operands. For example, if workspace register 2 contains $D2AA_{16}$ and location CHANGE contains the value $6D03_{16}$, then the instruction

```
XOR     @CHANGE,2
```

results in the contents of workspace register 2 changing to $BFA9_{16}$. Location CHANGE remains $6D03_{16}$. This is shown as

```
0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1   (Source operand)
1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0   (Destination operand)
-------------------------------
1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1   (Destination operand result)
```

For this example, the logical greater than status bit sets while the arithmetic greater than and equal status bits reset.

Assembly language format:

[<label>] ⱡ XOR ⱡ< ga_s >,< wa_d > [ⱡ<comment>]

4.8 WORKSPACE REGISTER SHIFT INSTRUCTIONS

Workspace register shift instructions permit the shifting of the contents of a specified workspace register from one to sixteen bits. The shifting instructions included in this paragraph are:

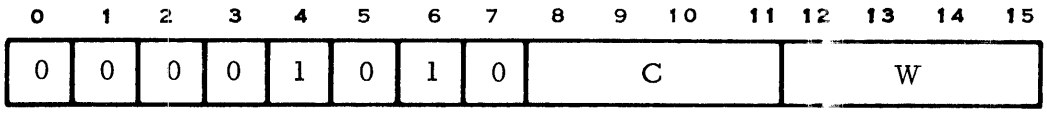
Instruction	Mnemonic	Paragraph
Shift Right Arithmetic	SRA	4.8.1
Shift Right Logical	SRL	4.8.2
Shift Left Arithmetic	SLA	4.8.3
Shift Right Circular	SRC	4.8.4

For each of these instructions, if the shift count in the instruction is zero, the shift count is taken from workspace register 0, bits 12 through 15. If the four bits of workspace register 0 are then equal to zero, the shift count is 16 bits. The logic value of the last bit shifted out of the workspace register is placed in the carry status bit position, bit 3 of the status register. The shifted value is always compared to zero and the results of this comparison are shown in status bits logical greater than, arithmetic greater than, and equal (bits 0 through 2 of the status register).

4.8.1 SRA (SHIFT RIGHT ARITHMETIC)

Op code: 0800

Format:



Definition: Shift the contents of the specified workspace register to the right for the specified number of bit positions, filling vacated bit positions with the sign bit. (Refer to paragraph 4.8 for restrictions and results.)

Addressing mode: Format V instructions

Status affected: Logical greater than, arithmetic greater than, equal, and carry.

Application notes: An example of an arithmetic right shift is: If workspace register 5 contains the value 8224_{16}, and workspace register 0 contains the value $F326_{16}$, then the instruction

```
SRA     5, 0
```

changes the contents of workspace register 5 to $FE08_{16}$. The logical greater than and carry status bits set while the arithmetic greater than and equal status bits reset. Additional examples are shown in Section V.

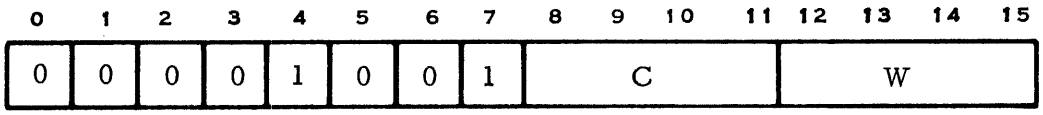
Assembly language format:

```
[<label>] b SRA b<wa>, <scnt>[b<comment>]
```

4.8.2 SRL (SHIFT RIGHT LOGICAL)

Op code: 0900

Format:



Definition: Shift the contents of the specified workspace register to the right for the specified number of bits while filling the vacated bit positions with logic zero values. (Refer to paragraph 4.8 for restrictions and results.)

Addressing modes: Format V instructions

Status affected: Logical greater than, arithmetic greater than, equal, and carry.

Application notes: An example of a logical right shift is: If workspace register zero contains the value $FFEF_{16}$, then the instruction

```
SRL     0, 3
```

changes the contents of workspace register 0 to $1FFD_{16}$. The logical greater than, arithmetic greater than and carry status bits set while the equal status bit resets. Additional examples are shown in Section V.

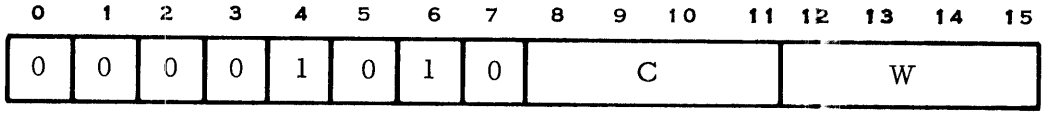
Assembly language format:

```
[<label>] b SRL b<wa>, <scnt>[b<comment>]
```

4.8.3 SLA (SHIFT LEFT ARITHMETIC)

Op code: 0A00

Format:



Definition: Shift the contents of the specified workspace register to the left for the specified number of bit positions while filling the vacated bit positions with logic zero values. (Refer to paragraph 4.8 for execution restrictions and results.) Note that the overflow status bit sets when the sign of the word changes during the shifting operation.

Addressing mode: Format V instructions

Status affected: Logical greater than, arithmetic greater than, equal, and carry and overflow.

Application notes: An example of an arithmetic left shift is: If workspace register 10 contains the value 1357_{16}, then the instruction

```
SLA     10, 5
```

changes the contents of workspace register 10 to $6AE0_{16}$. The logical greater than, arithmetic greater than, and overflow status bits set while the equal and carry status bits reset. Refer to Section V for additional examples.

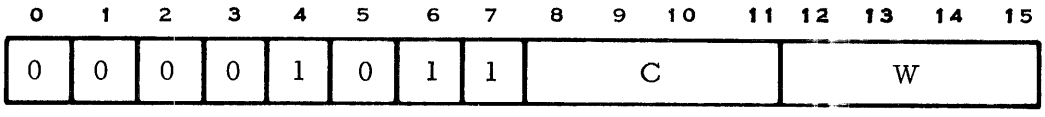
Assembly language format:

```
[<label>] ♭ SLA ♭<wa>, <scnt>[♭<comment>]
```

4.8.4 SRC (SHIFT RIGHT CIRCULAR)

Op code: 0B00

Format:



Definition: Shift the specified workspace register to the right for the specified number of bit positions while filling vacated bit positions with the bit shifted out of position 15. (Refer to paragraph 4.8 for execution restrictions and results.)

Addressing mode: Format V instruction

Status affected: Logical greater than, arithmetic greater than, equal, and carry.

Application notes: An example of a circular right shift is: If workspace reigster 2 contains the value $FFEF_{16}$, then the instruction

```
SRC     2,7
```

changes the contents of workspace register 2 to $DFFF_{16}$. The logical greater than and carry status bits set while the arithmetic greater than and equal status bits reset. Refer to Section V for additional application notes.

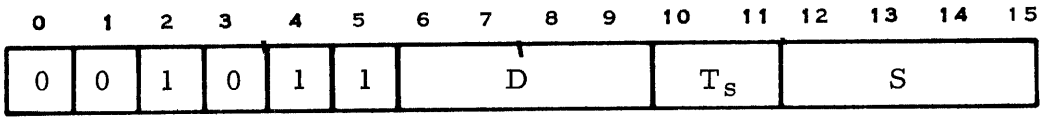
Assembly language format:

[<label>] ♭ SRC ♭<wa>,<scnt>[♭<comment>]

4.9 XOP (EXTENDED OPERATION INSTRUCTION)

Op code: 2C00

Format:



Definition: Refer to Section II for a definition of a hardware implemented XOP instruction.

When the XOP is software implemented, the D field specifies the extended operation trap location in memory. The two memory words at that location contain the WP contents and PC contents for the software implemented XOP instruction subroutine. The memory location for these two words is derived by multiplying the D field contents by four and adding the product to 0040_{16}. Note that the two memory words at this location must contain the necessary WP and PC values prior to the XOP instruction execution for software implemented instructions.

The effective address of the source operand is placed in workspace register 11 of the XOP workspace. The WP contents are placed in workspace register 13 of the XOP workspace. The PC contents are placed in workspace register 14 of the XOP workspace. The ST contents are placed in workspace register 15 of the XOP workspace. Control is transferred to the new PC address and the software implemented XOP is executed. (XOP execution of software implemented XOP instruction is similar to an interrupt trap execution.)

Addressing modes: Format IX instructions

Status affected: XOP (bit 6). (The remaining status bits may or may not be affected by the execution of the sequence of executable code called the software implemented XOP.)

Execution results: $(ga_s) \longrightarrow (WR11)$ (Of new XOP workspace)

$(WP) \longrightarrow (WR13)$ (Of new XOP workspace)

$(PC) \longrightarrow (WR14)$ (Of new XOP workspace)

$(ST) \longrightarrow (WR15)$ (Of new XOP workspace)

$(0040_{16} + (wa_d)*4) \longrightarrow (WP)$

$(0040_{16} + (wa_d)*4 + 2) \longrightarrow (PC)$

Application notes: Refer to Section V for a detailed example of the execution of a software implemented XOP instruction.

Assembly Language format:

[<label>] ⊬ XOP ⊬<ga_s>,<wa_d> [⊬<comment>]

SECTION V

PROGRAMMING CONVENTIONS

5.1 GENERAL

The contents of this section include various descriptions and examples of the programming conventions of the Model 990 Computer. The descriptions contain pertinent information about programming in 990 assembly language. Where a description may be confusing, specific coding examples and memory diagrams are included for additional information. Where memory or word diagrams are shown, any remaining memory not specifically included in the diagram is assumed to be contiguous to that shown. Also, the diagrams that include specific memory addresses do not limit the use of memory to that shown in the diagram. Any available memory may be used at the discretion of the programmer.

5.2 SAMPLE PROGRAM FORMS

There are three types of elements in a program for the Model 990 Computer. These are the procedure, the workspace, and the data. The procedure contains the computer instructions. The workspace contains program linkage, high activity data, and addresses. As many workspaces as convenient may be allocated for a program. Data areas may be allocated as required.

The three previously described hardware registers - WP, PC, and ST - control program execution. The workspace pointer contains the address of the first word of a 16-word area of memory called the workspace. Note that the program workspace may be changed by changing the contents of the WP register. The PC contains the address of the next instruction to execute. The status register contains condition bits set by instructions already performed and the interrupt level mask. (Refer to Section IV for another description of these registers.) These three registers then, completely control and define the context of a program.

The general environment of the 990 Computer is shown in figure 5-1. This arrangement of workspace, procedure, and data is the simplest approach to 990 programming. However, though many application programs may be written in this manner, a more segregated approach, with possibly several workspaces, data areas, and connected simple procedures, would provide increased flexibility and applicability.

5.2.1 PROCEDURE

A procedure is the main body of a program and contains computer instructions. It is the action part of a program. A procedure may also be used as the body of a subroutine. Procedures could be coded to solve an equation, run a motor, determine status of a process, or condition a set of data that

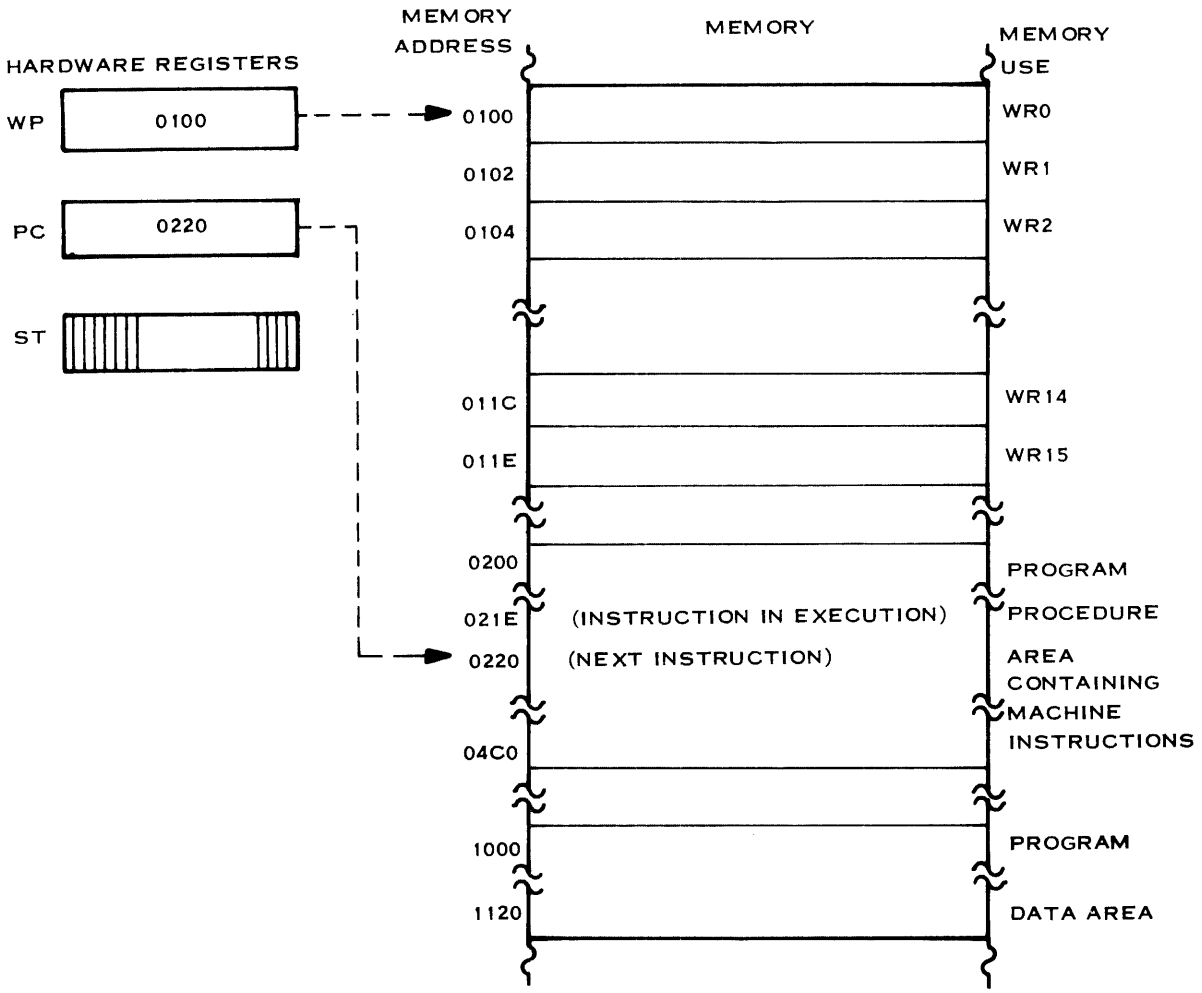


Figure 5-1. 990 Programming Environment

is to be processed by another procedure. Procedures in the 990 Computer may have workspaces and data as an integral part of the coding or may use workspaces and data passed from another procedure.

5.2.2 WORKSPACE

The Model 990 Computer uses workspaces that may be anywhere in memory and that consist of sixteen consecutive memory words. Refer to Section II and Section IV for additional information about the workspaces.

5.2.3 DATA

Data for a procedure may appear in many forms. In assembly language, there are three instructions available to the programmer to initialize data within a program module. These instructions are:

- DATA - Initializes one or more consecutive words of memory to specific values that are input on this statement.
- BYTE - Initializes one or more consecutive bytes of memory as does the DATA statement, except that bytes are initialized.
- TEXT - Initializes a textual string of characters in consecutive bytes of memory. The characters are represented in USASCII code.

Also, data input from the data terminal or device attached to the CRU or TILINE is available to procedures in the 990 Computer.

5.3 PROGRAMMING IN ASSEMBLY LANGUAGE

5.3.1 ASSEMBLY LANGUAGE

Use of an assembly language permits the programmer to express a task or problem in a symbolic notation (mnemonic) rather than a more cumbersome binary machine code. This symbolic notation permits the assembler to create machine binary coded instructions, which relieves the programmer of most of the clerical work required to determine the binary machine code and binary addressing information. Additional symbolic notation permits the programmer to include modules that are not part of the main program module. The assembler then determines the necessary addressing and control information. An example of this symbolic notation follows:

Operation	Mnemonic	Op-Code
Addition	A	A000
Move	MOV	C000
Increment By Two	INCT	05C0

As can be seen from these examples, symbolic notation more closely resembles the normal means of communication than does the Op-Code notation. Also, the use of symbolic notation permits references between parts of a program module and references to other program modules to be expressed symbolicly rather than by binary machine code.

5.3.2 LANGUAGE REQUIREMENTS

To properly code a source program to be processed by the assembler, several requirements must be adhered to. These requirements are listed in the Appendix H.

NOTE

Texas Instruments suggests that source programs be created in a sequence that is logical. This sequence would include flowcharting the problem, coding the source statements on a coding form for convenience, inputting the source statements (from cassette, or punched paper tape), and then assembling the program using the MIRA assembler.

5.4 ASSEMBLER DIRECTIVES AND PSUEDO-OPERATIONS

Appendix I defines the assembler directives that are available with the 990 Computer. Table 5-1 summarizes these directives. Also described are the two pseudo-instructions available with the 990 Computer. Assembler directives are grouped in the following categories:

- Directives that affect the location counter
- Directives that affect assembler output
- Directives that initialize constants
- Directives that link programs
- Miscellaneous directives
- Psuedo-operation directives

Each assembler directive contains the following fields that must be separated by one or more blanks (b).

<label>b<operator>b<operand>b<comment>

5.5 ADDRESSING MODES

There are eight addressing modes available with the 990 machine instructions and the exact mode used depends upon the format of the particular machine instruction. Five addressing modes are described in Section IV. The three additional addressing modes are defined as the immediate mode, the displacement mode, and the shift count mode. The immediate mode is simply placing the immediate operand in the memory word immediately following the instruction. The displacement mode is a displacement value supplied in the instruction. The shift count is supplied in the instruction for shifts.

5.6 TESTING AND JUMPING

The set of testing and jumping instructions permits the programmer to vary the execution path of the program module according to a particular status condition or to a particular CRU line logic level. (Refer to Section IV for a

Table 5-1. Assembler Directives

Title	Label	Operator	Field Contents Operand	Comment
Absolute origin	Optional	AORG	Well defined expression	Optional
Relocatable origin	Optional	RORG	Optional, relocatable expression with defined symbols	Optional[3]
Block starting with symbol	Required[1]	BSS	Well defined expression	Optional
Block ending with symbol	Required[1]	BES	Well defined expression	Optional
Word boundary	Optional	EVEN	Not used	Optional
Program identifier	Optional	IDT	Program name, 8-character string	Optional
Page title	Optional[2]	TITL[2]	50-character string (max)	Optional[2]
List source	Optional[2]	LIST[2]	Not used	Optional[2]
No source list	Optional[2]	UNL[2]	Not used	Optional[2]
Page eject	Optional[2]	PAGE[2]	Not used	Optional[2]
Initialize byte	Optional	BYTE	One or more expressions, separated by commas, with previously defined symbols	Optional
Initialize word	Optional	DATA	One or more expressions, separated by commas, with previously defined symbols	Optional

NOTES: 1. If required, must be symbol.
2. Not printed.
3. Used only if operand field used.

See Appendix I for complete definitions of these directives.

Table 5-1. Assembler Directives (Continued)

Title	Label	Operator	Field Contents Operand	Comment
Initialize text	Optional	TEXT	Character string of not more than 52 characters	Optional
Define assembly-time constant	Required[1]	EQU	Expression that contains no symbol that is not previously defined	Optional
External definition	Optional	DEF	One or more symbols that are separated by commas	Optional
External reference	Optional	REF	One or more symbols that are separated by commas	Optional
Define extended operation	Optional	DXOP	Symbol and a term separated by commas	Optional
Program end	Optional	END	Optional, contains symbol that specifies program entry point	Optional[3]

NOTES: 1. If required, must be symbol.
2. Not printed.
3. Used only if operand field used.

See Appendix I for complete definitions of these directives.

discussion of the machine instructions involved.) Several of these instructions may be used for other than the obvious purposes which are described in the following paragraphs.

TB (Test Bit) CRU line logic level test transfers the logic level from the indicated CRU line to the equal status bit without modification. If the CRU line tested is set to a logic one, the equal status bit sets to a logic one and if the line is zero, sets to a zero. JEQ will then transfer control when the CRU line is a logic one and will not transfer control when the line is a logic zero. In addition, JNE will transfer control under the exact opposite conditions.

JOP (Jump Odd Parity) transfers control if the byte tested contains an odd number (sum) of logic one bits. This factor may be used in data transmissions where the parity of the transmitted byte is used to ensure the validity of the received character at the point of reception.

JNO (Jump No Overflow) normally will transfer control during arithmetic sequences where addition, subtraction, incrementing, and decrementing may cause an overflow condition. JNO may also be used during an SLA (Shift Left Arithmetic) operation. If, during the SLA execution, the sign of the workspace register being shifted changes (+ to -, - to +), the overflow status bit sets. This feature permits transfer, after a sign change, to error correction routines or to another functional code sequence.

A feature of the ABS (Absolute) machine instruction that may be used as a switch testing function is that the status bits are set/reset according to the value of the operand prior to the execution of ABS when that operand is compared to zero. The following example program sequence illustrates this capability implemented as an entrance condition switch for a subroutine.

PROBLEM: There exists a subroutine that can only be used by one program module at a time. A flag is maintained to indicate when the subroutine is in use. A method is required to test the flag and reset it if required. The test must be insensitive to program interruption because the interrupt may alter the sequence in which the program modules are executed. The following sequence of generalized code illustrates the problem solution.

```
        SETO    @SWITCH     INITIALIZE SWITCH NEGATIVE[1]
          .
          .
          .
TEST    ABS     @SWITCH     TEST SWITCH[2]
        JLT     CALL        IF NEGATIVE, TRANSFER[3]
        IDLE                IF NOT, WAIT[4]
        JMP     TEST        TEST AGAIN
          .
          .
          .
```

```
CALL    BL      @SUBR       USE SUBROUTINE
        SETO    @SWITCH     RESET SWITCH[5]
                 :
                 :
SUBR    ....                SUBROUTINE ENTRY
                 :
                 :
        B       *11         SUBROUTINE RETURN
SWITCH  DATA    0           STORAGE AREA FOR SWITCH
```

NOTE

1. Set SWITCH to all ones, making it negative.
2. If SWITCH negative, set to positive value to prevent subsequent entry.
3. If value in SWITCH was negative, the JLT instruction transfers control.
4. Used to wait for the next interrupt point. (Refer to Section IV.)
5. Upon return, reset SWITCH to negative value to permit future use.

5.7 SHIFTING INSTRUCTIONS

There are 4 shifting instructions available with the 990 Computer that permit the user to shift the contents of a specified workspace register from one to sixteen consecutive bit positions.

The four shifting instructions are:

- Shift Left Arithmetic (SLA)
- Shift Right Arithmetic (SRA)
- Shift Right Circular (SRC)
- Shift Right Logical (SRL)

5.7.1 SHIFT LEFT ARITHMETIC (SLA)

This shifting instruction shifts the indicated workspace register a specified number of bits to the left. For example, the instruction

```
SLA     5,1
```

would shift the contents of register five one bit to the left. The carry status bit contains the value shifted out of bit position zero and the jump instructions JOC and JNC permit the user to test the shifted bit. The overflow status bit

sets when the sign of the contents of the register being shifted changes during the shift operation. If register five contained

0100111100000111

before the above instruction, the results of the instruction execution would be

1001111000001110

and the carry status bit would contain a zero and the overflow status bit would set because the contents changed from positive to negative (bit zero equal to zero changed to equal to one). If this shift sign change is important, the user could insert a JNO instruction to test the overflow condition. If there is no overflow, control transfers to the normal program sequence. Otherwise, the next instruction is then executed.

It is possible to construct double-length shifts with the SLA instruction, which could shift two or more words in a workspace. The following code will shift two consecutive workspace registers.

- Assumptions:
  1. Workspace register one and two are shifted.
  2. If overflow, transfer to ERR (error routine).
  3. There must be some additional method of determining when the shift is complete, so additional control code should be included.
- Code:

```
SLA     1,1     SHIFT W1 ONE BIT
SLA     2,1     SHIFT W2 ONE BIT
JNC     $+4     TRANSFER IF NO CARRY
INC     1       TRANSFER BIT FROM W2 TO W1
  :
  :
```

5.7.2 SHIFT RIGHT ARITHMETIC (SRA)

This shifting instruction shifts the contents of a workspace register a specified number of bits and extends the sign bit (bit zero) at the logic level that existed prior to the shift. The carry status bit contains the last bit shifted out of bit 15 of the workspace register. For example, the instruction

```
SRA     5,3
```

would shift the contents of workspace register five three bits to the right. If workspace register five contained

1100000011110000

prior to the shift, the results of this instruction would be

1111100000011110

and the carry status bit would contain a logic zero for the last shifted bit.

5.7.3 SHIFT RIGHT CIRCULAR (SRC)

The SRC instruction shifts the contents of a workspace register a specified number of bits to the right and transfers the bits shifted off the right end of the workspace to the left end of the workspace. The carry status bit contains the last bit shifted out of bit 15 of the workspace register. For example, the instruction

```
SRC     5,5
```

would shift the contents of register five five bits to the right and transfer the five bits shifted off the right end to the first five bits of workspace register five. For this example, if workspace register five contained

1100110011110101

before this instruction was executed, workspace register five would contain

1010111001100111

and the carry status bit would contain a logic one from the last bit shifted in workspace register five.

5.7.4 SHIFT RIGHT LOGICAL (SRL)

The SRL instruction shifts the contents of a specified workspace register to the right for a specified number of bits and fills the vacated bit positions on the left end of the workspace with zeros. The carry status bit contains the last bit shifted out of bit 15 of the workspace register. For example, the instruction

```
SRL     5,8
```

would shift the contents of workspace register five eight bits to the right and would fill the first eight bits of the word with zeros. If the workspace register contained

1000100011111000

prior to the SRL instruction, the contents of workspace register five would be

0000000010001000

and the carry status bit would contain a logic one for the last bit shifted off the right end of workspace register five.

5.8 INCREMENTING AND DECREMENTING

There are two decrement and two increment instructions that may be used for various types of control when passing through a loop, indexing through an array, or operating within a group of instructions.

The four incrementing and decrementing instructions available for use with the 990 Computer are:

- Decrement (DEC)
- Decrement By Two (DECT)
- Increment (INC)
- Increment By Two (INCT)

The increment and decrement instructions are useful for indexing byte arrays and for counting byte operations. The increment and decrement by two instructions are useful for indexing word arrays and for counting word operations. The following paragraphs provide some examples of these operations.

5.8.1 INCREMENT INSTRUCTION EXAMPLE

Since the INC instruction is useful in byte operations, an example problem would be to search a character array for a character with odd parity. Begin the search at the lowest address of the array and maintain an index in a workspace register. The character array for this example is called A1 (also the relocatable address of the array). The last character in the array will always be a binary zero. The sample code for this problem is:

```
            LI      1, FFFF   CLEAR COUNTER INDEX
SEARCH      INC     1         INCREMENT INDEX
            MOVB    @A1(1), 2 GET CHARACTER
            JOP     ODDP      JUMP IF FOUND
            JNE     SEARCH    CONTINUE SEARCH IF NOT ZERO
             .
             .
             .
ODDP        ...     ....
```

5.8.2 DECREMENT INSTRUCTION EXAMPLE

Since the DEC instruction is useful in byte operations, this example problem inverts a byte array and places the results in another array of the same size.

This example inverts a 26-character array called A1 (also the relocatable address of the array) and places the results in an A2 array (A2 also the relocatable address). The contents of A1 are defined with a data TEXT statement to be as follows:

```
A1      TEXT    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

Array A2 is defined with the BSS statement as follows:

```
A2      BSS     26
```

The sample code for the solution is:

```
        LI      5, 25           COUNTER AND INDEX FOR A1
        LI      4, A2           ADDRESS OF A2
INVRT   MOVB    @A1(5), *4+     INVERT ARRAY[1]
        DEC     5               REDUCE COUNTER
        JOC     INVRT           CONTINUE IF NOT COMPLETE[2]
         .
         .
         .
```

NOTE

1. @A1(5) addresses elements of array A1 in descending order as workspace register five is decremented. *4+ addresses array A2 in ascending order as workspace register four is incremented.
2. A carry occurs except when the count in workspace register five is decremented from zero to a minus one, which occurs on the 26th decrement.

Array A2 would contain the following as a result of executing this sequence of code:

```
A2      ZYXWVUTSRQPONMLKJIHGFEDCBA
```

Even though the result of this sequence of code is trivial, the example use of the MOVB instruction, with indexing by workspace register five, and the result incrementally placed into A2 with the auto-increment function can be useful in other applications.

5.8.3 DECREMENT BY TWO EXAMPLE INSTRUCTION

Since the DECT instruction is useful in word operations, the example problem chosen adds the contents of a word array to another word array and places the results in the second array. The contents of the two arrays are initialized as follows:

```
A1      DATA    500,300,800,1000,1200,498,650,3,27,0
A2      DATA    36,192,517,29,315,807,290,40,130,1320
```

The sample code that adds the two arrays is as follows:

```
        LI      4,18            INITIALIZE COUNTER[1]
SUMS    A       @A1(4),@A2(4)   ADD ARRAYS[2]
        DECT    4               DECREMENT COUNTER BY TWO
        JOC     SUMS            REPEAT ADDITION[3]
```

NOTE

1. Counter is initialized to 18 so that when the addition process is complete, there will be a negative value of two in the counter and the carry status bit will not be set.
2. Addressing of the two arrays through the use of the @ sign is indexed by the counter, which is decremented after each addition.
3. There is a carry status bit set when the counter changes from a zero to a negative two and the JNC instruction will not be taken when the addition process is complete.

The contents of the A2 array after the addition process is as follows:

```
A2      536,492,1317,1029,1515,1305,940,43,157,1320
```

There is another method by which this addition process may be accomplished. This method is shown in the following code:

```
        LI      4,10            INITIALIZE COUNTER[1]
        LI      5,A1            LOAD ADDRESS OF A1[2]
        LI      6,A2            LOAD ADDRESS OF A2[2]
SUMS    A       *5+,*6+         ADD ARRAYS[3]
        DEC     4               DECREMENT COUNTER
        JGT     SUMS            REPEAT ADDITION[4]
```

NOTE

1. Counter preset to 10 (the number of elements in the array).
2. This address will be incremented each time an addition takes place. The increment is via the auto-increment function (+).
3. The * indicates that the contents of the register is to be used as an address and the + indicates that it will be automatically incremented by two each time the instruction is executed.
4. Workspace register four will only be greater than zero for ten executions of the DEC instruction and control will be transferred to SUMS nine times after the initial execution.

The second method is not as efficient as the first, but it is available to the user. It is not as efficient because it takes more instructions to accomplish the same result and it requires two more workspace registers than the first. The contents of array A2 are the same for this method as for the first.

5.9 SUBROUTINES

There are two types of subroutine linkage available with the 990 Computer. The primary difference is in workspace allocation. The BL instruction simply saves the present contents of the PC in the current workspace register 11 and transfers control to the address specified by the instruction. This is illustrated in figure 5-2. The BLWP instruction is used to store linkage and switch context in one operation. This instruction saves the WP in workspace register 13 of the subroutine, the PC in workspace register 14 of the subroutine, and the ST in workspace register 15 of the subroutine. This is shown in figure 5-4.

To return from a subroutine entered by a BL instruction, the user may use either the RT psuedo-instruction or a B *11 (branch to the address in workspace register 11). To return from a subroutine entered by a BLWP instruction and restore the original environment, use the RTWP instruction.

5.9.1 BL SUBROUTINE CALL EXAMPLE

Figure 5-2 shows an example of memory contents prior to a BL call to a subroutine. Note that the content of workspace register 11 is not important to the main routine. When the BL instruction is executed, the CPU stores the contents of the PC in workspace register 11 of the main routine and transfers control to the instruction located at the address indicated by the operand of the BL instruction. This type of subroutine uses the main program workspace. Figure 5-3 shows the memory contents after the call to the subroutine with the BL instruction.

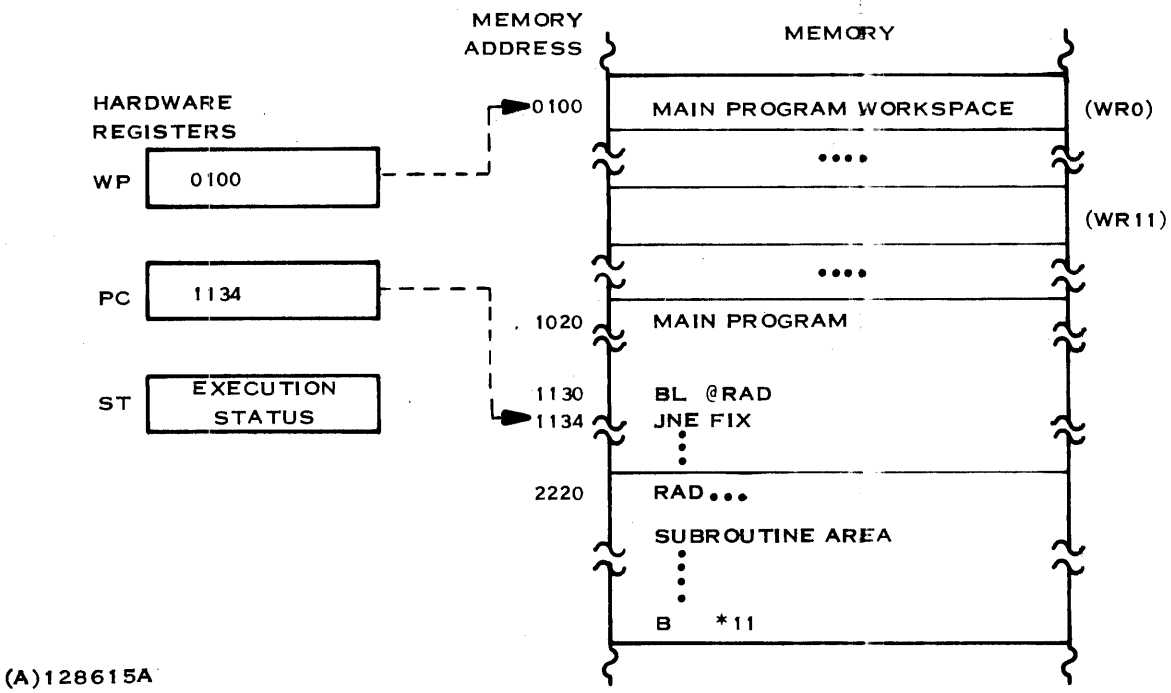


Figure 5-2. Example Subroutine Call (BL)

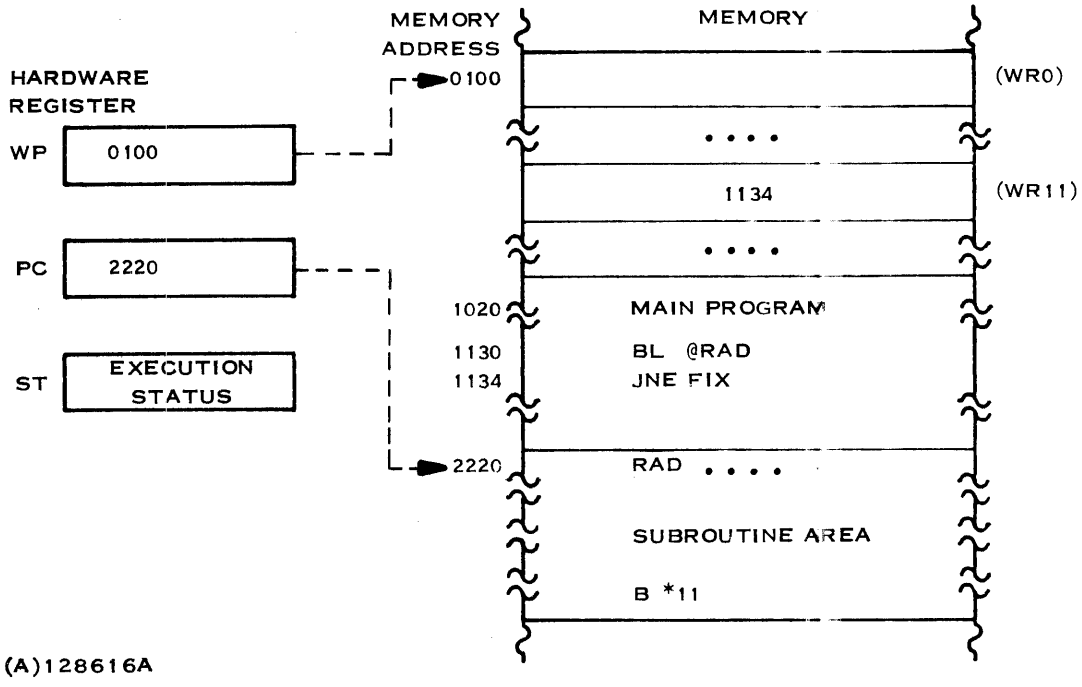


Figure 5-3. Status After BL Execution

When the instruction at location 1130_{16} is executed (BL @RAD), the present contents of the PC, which point to the next instruction, are saved in workspace register 11. WR11 would then contain an address of 1134_{16}. The PC is then loaded with the address where the label RAD appears, which is address 2220_{16}. This example subroutine returns to the main program with a branch to the address in WR11 (B *11).

5.9.2 BLWP SUBROUTINE CALL EXAMPLE

Figure 5-4 shows the example memory contents prior to the call to the subroutine. Note that the contents of workspace registers 13, 14, and 15 are not

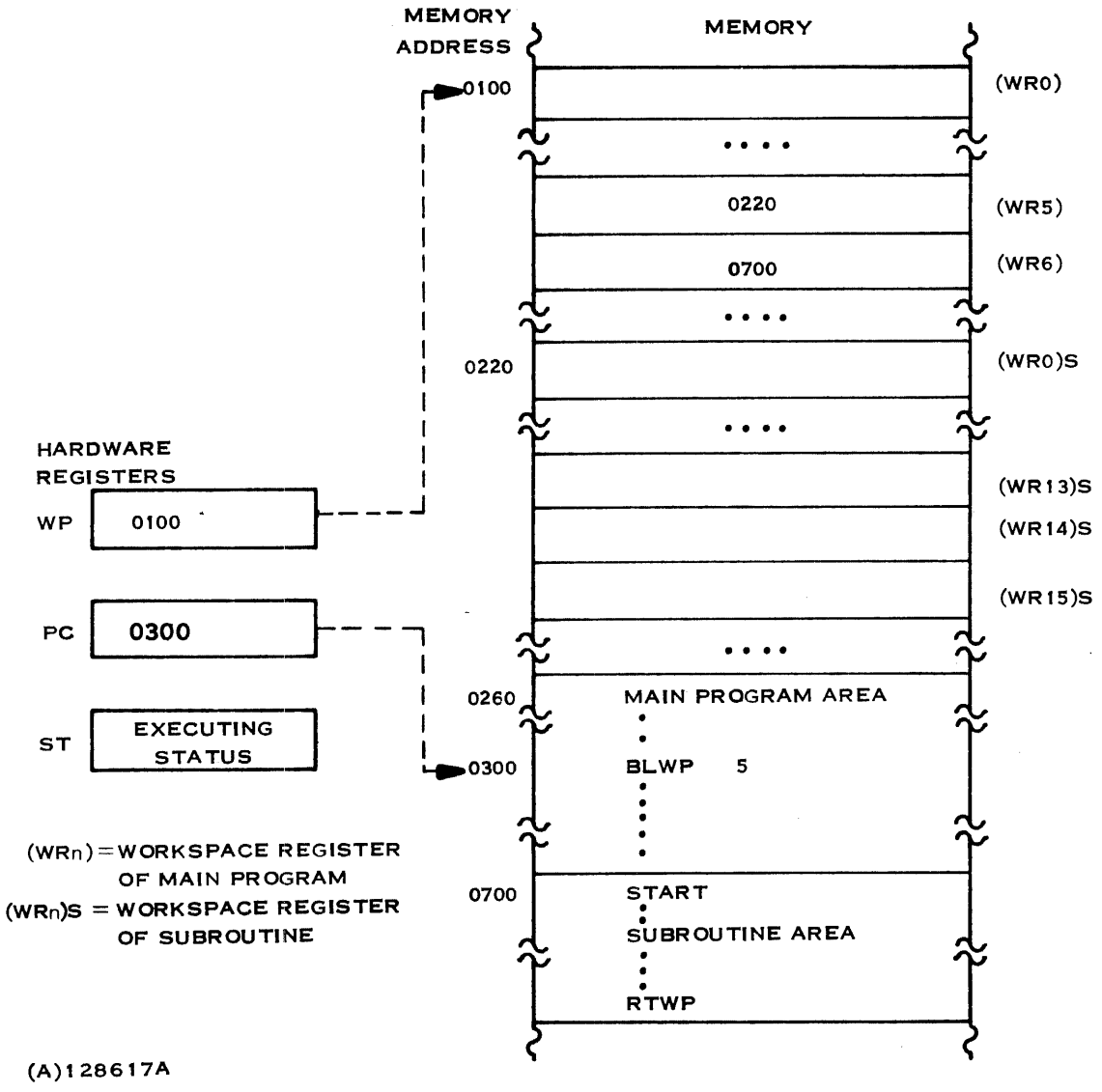


Figure 5-4. BLWP Subroutine Call Before Execution

presently used. When the BLWP instruction is executed at location 0300, there is a context switch from the main program to the subroutine. The context switch then places the main program PC, WP, and ST register contents in workspace registers 13, 14, and 15 of the subroutine. This saves the environment of the main program for return. This type of subroutine maintains a workspace that is possibly not the same as the workspace of the main program.

After the instruction at location 0300 is executed, the memory contents are shown in figure 5-5. This illustration shows the subroutine in control, with the WP pointing to the subroutine workspace and the PC pointing to the first instruction of the subroutine. The contents of the status register are not

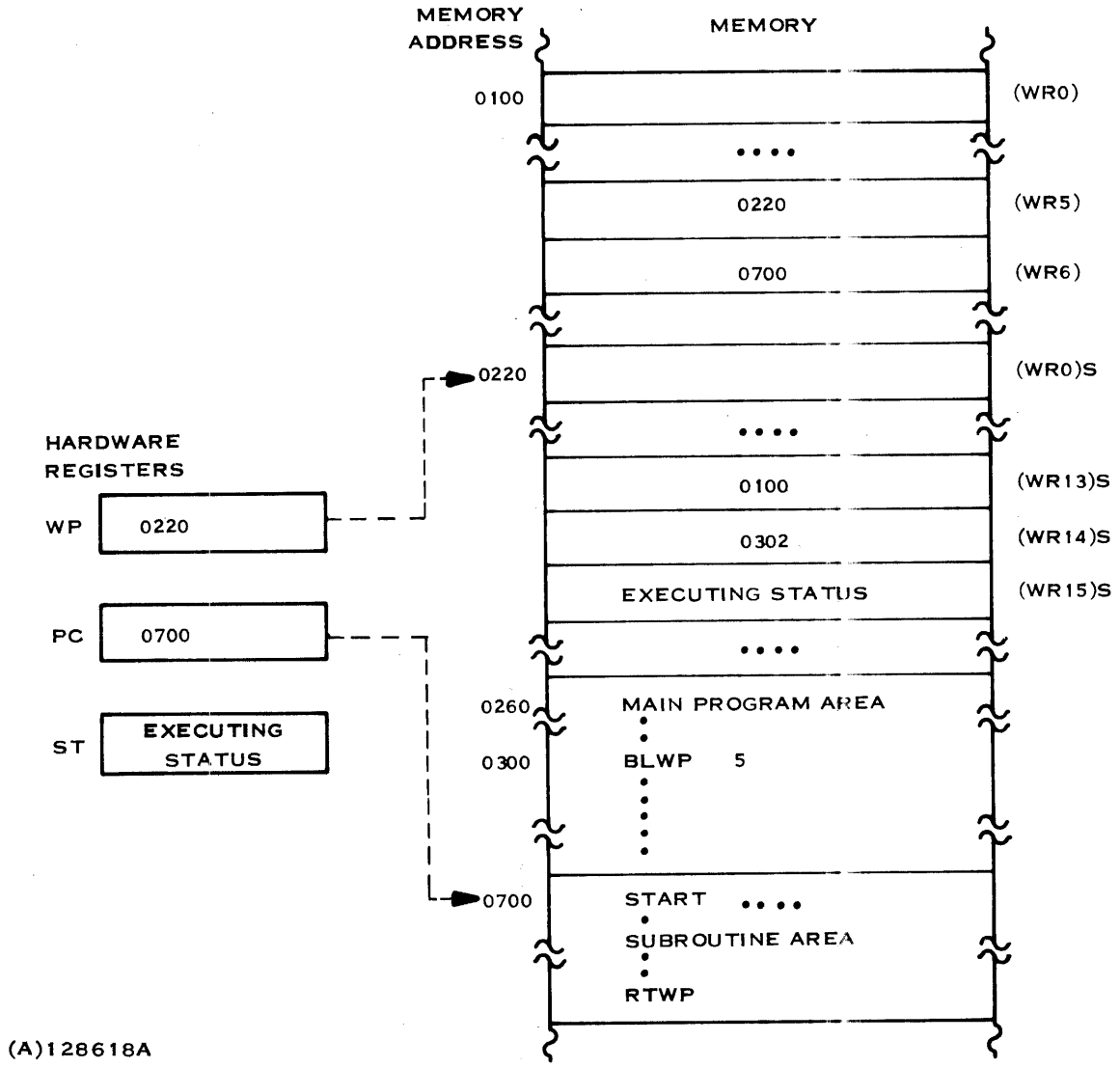


Figure 5-5. BLWP Subroutine Call After BLWP 5 Execution

reset prior to the execution of the first instruction of the subroutine, so the status indicated will actually be the status of the main program execution. A subroutine may then execute in accordance with the status of the main program.

This example subroutine contains a RTWP return from the subroutine. The results of executing the RTWP instruction are shown in figure 5-6. Control is transferred to the main program at the instruction following the BLWP to the subroutine. The status register is restored from workspace register 15 and the workspace pointer points to the workspace of the main program.

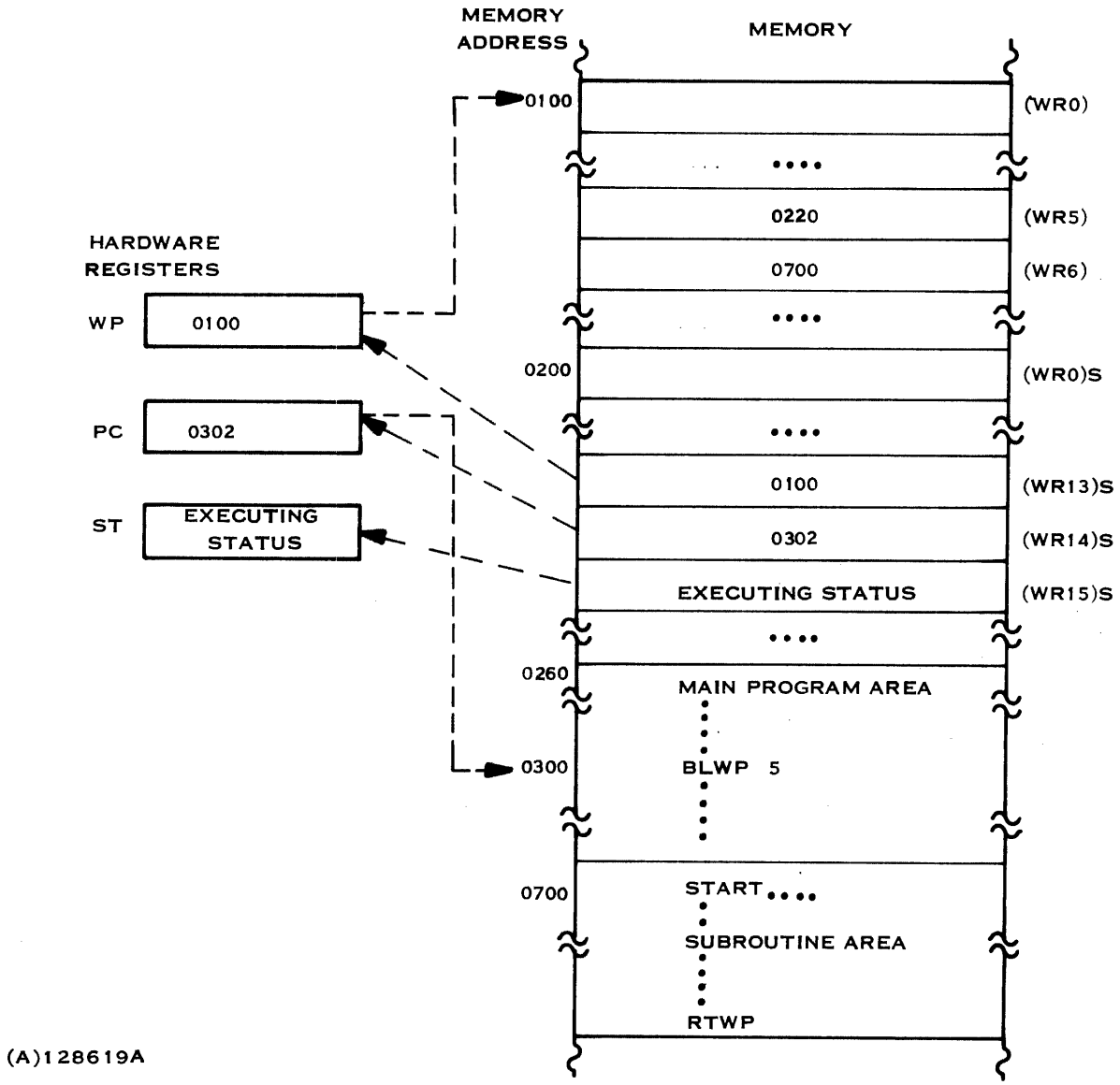


Figure 5-6. BLWP Subroutine Call After RTWP Execution

5.9.3 BLWP PROGRAMMING NOTES

Workspace register 13 of the subroutine contains the memory address of the calling program's workspace. WR14 of the subroutine contains the memory address of the next memory cell following the BLWP instruction. This particular memory cell may contain instructions or data at the discretion of the programmer. For example:

PROBLEM: Call a subroutine that uses three local variables of the main program module. The following sample code could be used to accomplish this objective.

```
            BLWP   @SUB              SUBROUTINE CALL
            DATA   V1
            DATA   V2
            DATA   V3
            JEQ    ERROR             TEST FOR ERROR (Subrou-
             .                       tine sets the EQUAL status
             .                       bit to one for error.)
             .
SUB         DATA   SUBWS, SUBPRG     ENTRY POINT FOR SUB
             .                       AND SUB WRKSPCE
             .
             .
SUBWS       BSS    32
SUBPRG      MOV    *14+, 1           FETCH V1 PLACED IN WR1
            MOV    *14+, 2           FETCH V2 PLACED IN WR2
            MOV    *14+, 3           FETCH V3 PLACED IN WR3
             .
             .
            RTWP                     RETURN FROM SUBROUTINE
```

The three MOV instructions retrieve the variables from the main program module and place them in workspace registers one, two, and three of the subroutine.

When the BLWP instruction is executed, the main program module status is stored in workspace register 15 of the subroutine. If the subroutine returns with a RTWP instruction, this status is placed in the status register after the RTWP instruction is executed. This feature is useful in returning single bits of information to the calling program. The calling program can then test the appropriate bit of the status word with jump instructions.

Variables may also be passed to a subroutine in the workspace of the calling program module. In this case, workspace register 13 of the subroutine contains the memory address of the calling program workspace. Variable pick-up may be accomplished by using the indexed addressing mode with WR13 as the index. An example of this code is

```
MOV     @10(13), 10
```

which will move the contents of workspace register 5 (10 bytes) of the calling program to workspace register 10 of the subroutine, indexed by workspace register 13 of the subroutine. Re-entrant procedures will typically use this method of indexed workspace register addressing for data access.

5.10 INTERRUPTS

Sixteen priority vectored interrupt levels are implemented in the Model 990 Computer; six are used for internal interrupts and ten are used for external interrupts. The contents of the interrupt mask in the status register define the interrupt level. Low-order memory, address 0 through 3F, is reserved for addresses used by the interrupts (table 5-2). When an interrupt request at an enable level occurs, the contents of the reserved memory words corresponding to the level are used to enter a subroutine to serve the interrupt.

The reserved memory locations are shown on the memory map (refer to Section IV). Two memory words are reserved for each interrupt level. The first of the two words for a given level contains an address that is placed in the WP register when the interrupt is requested and enabled. The second contains the entry point of the interrupt subroutine for that level; its contents are placed in the PC. The user must load the reserved memory locations corresponding to the interrupts defined for the program. The user must also load an interrupt subroutine for these interrupts.

5.10.1 GENERAL INTERRUPT STRUCTURE

The interrupt levels, numbered 0 through 15, determine the interrupt priority. Level 0 has the highest priority and level 15 the lowest. The contents of the interrupt mask, bits 12 through 15 of the ST register, determine the enabled interrupt levels. Table 5-3 shows the interrupt levels enabled by the contents of the interrupt mask. Note that level 0 cannot be disabled since the level contained in the mask is always enabled.

5.10.2 INTERRUPT SEQUENCE

The level of the highest priority pending interrupt request is continually compared with the interrupt mask contents. When the level of the pending request is equal to or less than the mask contents (equal or higher priority) the interrupt is taken after the currently executing instruction has completed.

Table 5-2. Interrupt Vector Addresses

Memory Address	Interrupt Vector	Vector Contents
0000	0	WP address for interrupt 0
0002	0	PC address for interrupt 0
0004	1	WP address for interrupt 1
0006	1	PC address for interrupt 1
0008	2	WP address for interrupt 2
000A	2	PC address for interrupt 2
000C	3	WP address for interrupt 3
000E	3	PC address for interrupt 3
0010	4	WP address for interrupt 4
0012	4	PC address for interrupt 4
0014	5	WP address for interrupt 5
0016	5	PC address for interrupt 5
0018	6	WP address for interrupt 6
001A	6	PC address for interrupt 6
001C	7	WP address for interrupt 7
001E	7	PC address for interrupt 7
0020	8	WP address for interrupt 8
0022	8	PC address for interrupt 8
0024	9	WP address for interrupt 9
0026	9	PC address for interrupt 9
0028	10	WP address for interrupt 10
002A	10	PC address for interrupt 10
002C	11	WP address for interrupt 11
002E	11	PC address for interrupt 11
0030	12	WP address for interrupt 12
0032	12	PC address for interrupt 12
0034	13	WP address for interrupt 13
0036	13	PC address for interrupt 13

Table 5-2. Interrupt Vector Addresses (Continued)

Memory Address	Interrupt Vector	Vector Contents
0038	14	WP address for interrupt 14
003A	14	PC address for interrupt 14
003C	15	WP address for interrupt 15
003E	15	PC address for interrupt 15

Table 5-3. Interrupt Mask

STATUS REGISTER BIT 12	BIT 13	BIT 14	BIT 15	INTERRUPT LEVELS ENABLED	MASK SET BY INTERRUPT LEVEL
0	0	0	0	0	0,1
0	0	0	1	0,1	2
0	0	1	0	0,1,2	3
0	0	1	1	0,1,2,3	4
0	1	0	0	0,1,2,3,4	5
0	1	0	1	0,1,2,3,4,5	6
0	1	1	0	0,1,2,3,4,5,6	7
0	1	1	1	0,1,2,3,4,5,6,7	8
1	0	0	0	0,1,2,3,4,5,6,7,8	9
1	0	0	1	0,1,2,3,4,5,6,7,8,9	10
1	0	1	0	0,1,2,3,4,5,6,7,8,9,10	11
1	0	1	1	0,1,2,3,4,5,6,7,8, 9,10,11	12
1	1	0	0	0,1,2,3,4,5,6,7,8, 9,10,11,12	13
1	1	0	1	0,1,2,3,4,5,6,7,8, 9,10 11,12,13	14
1	1	1	0	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14	15
1	1	1	1	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14,15	-

(A)128630

The workspace defined for the interrupt subroutine becomes active and the entry point is placed in the program counter. The CPU also stores the previous contents of the WP register in the new workspace register 13, the previous contents of the program counter in the new workspace register 14, and the contents of the ST register in the new workspace register 15. This preserves the program environment existing when the interrupt is taken. No additional interrupt is taken until the first instruction of the interrupt subroutine is completed. Thereafter, interrupts of higher priority can interrupt processing of the current interrupt.

After storing the ST register contents, the CPU subtracts one from the level of the interrupt taken and places the result in the interrupt mask, disabling the current interrupt level, and leaving only higher priority levels enabled. Should a higher priority level interrupt be taken, and the original interrupt request remain active when the return from the higher priority level interrupt subroutine occurs, the original interrupt remains disabled and is not taken again. Control returns to the interrupt subroutine at the point at which the higher priority interrupt occurred.

5.10.3 INTERNAL INTERRUPTS

Levels 0 through 5 are assigned to internal interrupts, as follows:

- Level 0 - Power restored. Whenever ac power is restored to the computer, a level 0 interrupt occurs. The interrupt mask is set to 0.
- Level 1 - Power failure imminent. When the power supply senses that ac power is failing, a level 1 interrupt request occurs. Following the interrupt request, 1.5 ms of program time is available before a power supply reset occurs. When the interrupt is taken, the interrupt mask is set to 0.
- Level 2 - Memory error. When the data read from memory does not match the error correction data, a level 2 interrupt request occurs. The instruction being executed when the error is detected is the instruction preceding the instruction at the address stored in workspace register 14. When the interrupt is taken, the interrupt mask is set to 1.
- Level 3 - Illegal operation code. When the CPU acquires an instruction from memory that cannot be executed, a level 3 interrupt request occurs. If level 3 is disabled, the Program Counter is incremented by two and execution of the instruction at that address is attempted. The interrupt request remains active until the interrupt is taken. When the interrupt is taken, the interrupt mask is set to 2.

- Level 4 - TILINE timeout. When the CPU attempts a memory access or communication with a device on the TILINE, and receives no response within 10 μsec, a level 4 interrupt request occurs. When the interrupt is taken, the interrupt mask is set to 3.
- Level 5 - Real time clock. When the real time clock supplies a signal, a level 5 interrupt request occurs. The interrupt request remains active until the interrupt is taken. When the interrupt is taken, the interrupt mask is set to 4.

5.10.4 EXTERNAL INTERRUPTS

Levels 6 through 15 are available for assignment to devices on the CRU or the TILINE. Several interrupt lines may be combined at one level. Assignment and grouping are determined by system considerations. Any interrupt request must remain active until the interrupt is taken, and must be reset before the interrupt subroutine is completed. The interrupt sequence, priority, and modification of the interrupt mask are the same as for internal interrupts.

5.10.5 INTERRUPT PROCESSING EXAMPLE

Refer to figure 5-7 for the following discussion. Prior to the example interrupt (eight for this example), the PC contains 1022 for the executing program, the WP contains 780 for the executing program workspace, and the ST register contains the executing program status. At this point, the example external interrupt, number eight, occurs and there is a context switch from the executing program to the interrupt subroutine. As shown in the memory map in Section IV, the two words of memory required for external interrupt eight are found in memory locations 0020 and 0022. Figure 5-7 shows that these two words of memory contain 0270 and 0290, respectively, for the WP and PC that are to be used by the interrupt subroutine.

At the point of interrupt, the CPU transfers the present WP, PC, and ST register contents to the interrupt routine workspace in workspace registers 13, 14, and 15, respectively. Once these are stored, the CPU transfers the interrupt subroutine WP and PC into the WP and PC registers. When these actions are completed, the contents of memory and the registers are as shown in the figure 5-8.

After the completion of the interrupt subroutine, the CPU restores the executing program WP, PC, and ST registers. Completion of the interrupt subroutine occurs when the RTWP instruction in the interrupt subroutine is executed.

5.11 EXTENDED OPERATIONS

Extended operation instructions permit the extension of the existing instruction set to include additional instructions that may be either hardware or software implemented.

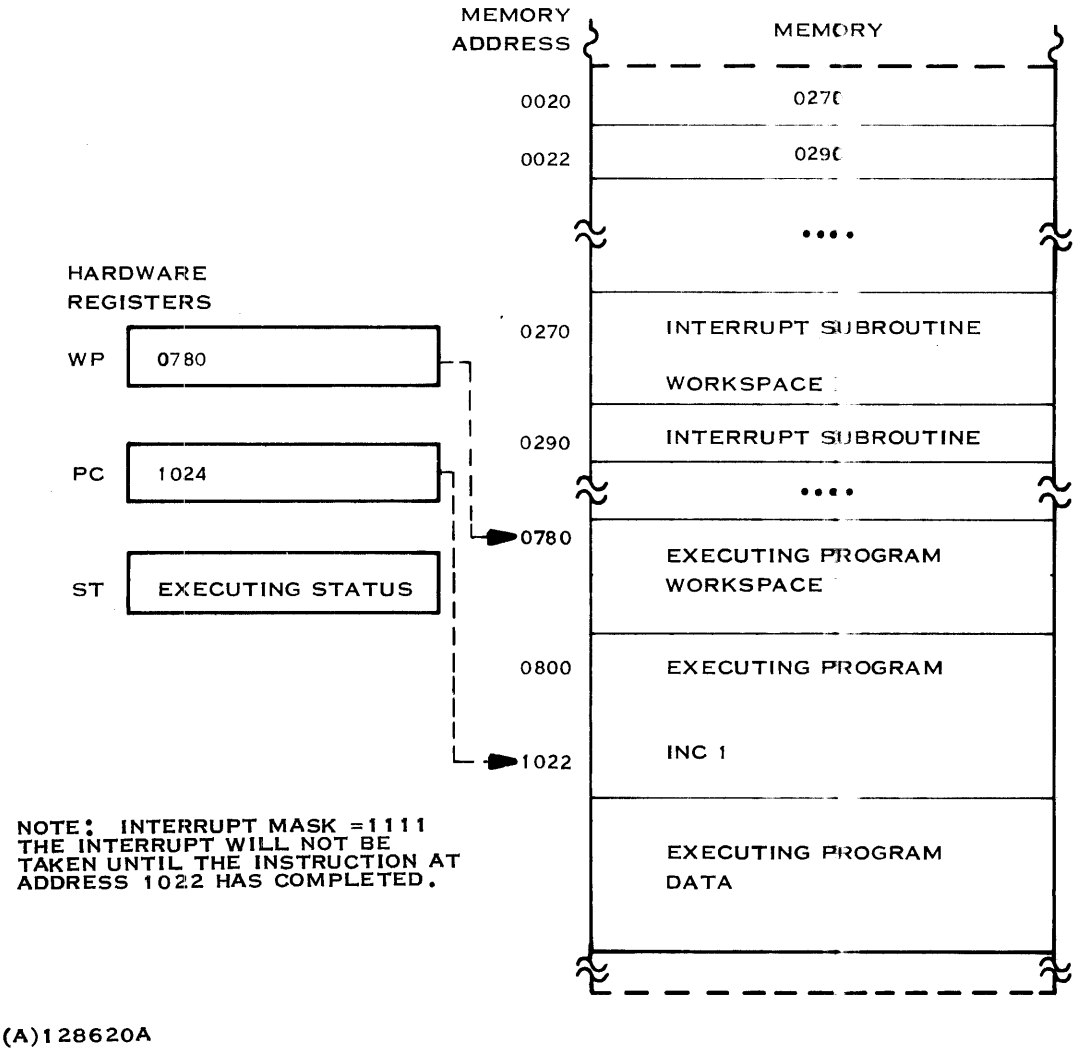


Figure 5-7. Example Memory Prior to Interrupt

Memory locations 0040_{16} through $007E_{16}$ are used for XOP vectors for software implemented XOPs. Vector contents are user supplied WP and PC addresses for the XOP routine workspace and starting address. Table 5-4 contains the addresses and contents of the 16 XOP vectors. Note that these vectors must be supplied and loaded prior to the XOP instruction execution.

When the program module contains an XOP instruction that is software implemented, the AU locates the XOP workspace pointer and PC words in the XOP reserved memory locations and loads the WP and PC. When the WP and PC are loaded, the AU transfers control to the XOP instruction set through a context switch. When the context switch is complete, the XOP workspace contains the calling routine return data in WRs 13, 14, and 15.

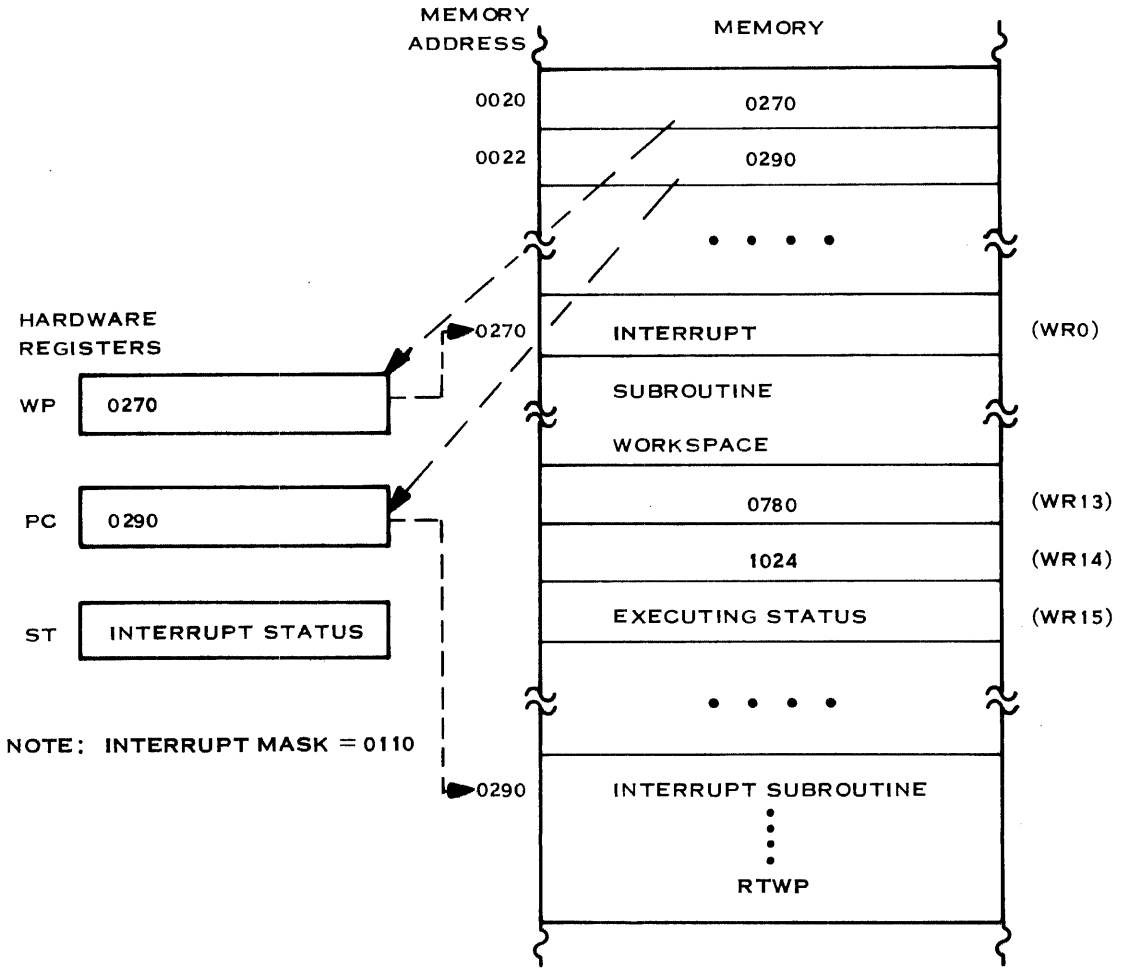


Figure 5-8. Memory Contents After Interrupt Occurs

Table 5-4. XOP Vectors

Memory Address	XOP Number	Vector Contents
0040	0	WP address for XOP workspace
0042	0	PC address for XOP routine
0044	1	WP address for XOP workspace
0046	1	PC address for XOP routine

Table 5-4. XOP Vectors (Continued)

Memory Address	XOP Number	Vector Contents
0048	2	WP address for XOP workspace
004A	2	PC address for XOP routine
004C	3	WP address for XOP workspace
004E	3	PC address for XOP routine
0050	4	WP address for XOP workspace
0052	4	PC address for XOP routine
0054	5	WP address for XOP workspace
0056	5	PC address for XOP routine
0058	6	WP address for XOP workspace
005A	6	PC address for XOP routine
005C	7	WP address for XOP workspace
005E	7	PC address for XOP routine
0060	8	WP address for XOP workspace
0062	8	PC address for XOP routine
0064	9	WP address for XOP workspace
0066	9	PC address for XOP routine
0068	10	WP address for XOP workspace
006A	10	PC address for XOP routine
006C	11	WP address for XOP workspace
006E	11	PC address for XOP routine
0070	12	WP address for XOP workspace
0072	12	PC address for XOP routine
0074	13	WP address for XOP workspace
0076	13	PC address for XOP routine
0078	14	WP address for XOP workspace
007A	14	PC address for XOP routine
007C	15	WP address for XOP workspace
007E	15	PC address for XOP routine

Also, the XOP instruction passes one operand to the XOP (input to the XOP routine in WR11 of the XOP workspace). At the completion of the software XOP, the XOP routine should return to the calling routine with an RTWP instruction that will restore the execution environment of the calling routine to that in existence at the call to the XOP.

An example of a software implemented XOP, shown in figure 5-9, causes XOP number two to be executed on the data stored at the address contained in WR1 of the calling program module. Prior to the execution of the XOP, the PC contains the address of the XOP *1, 2 instruction and the WP contains the address of the calling program workspace. At this point, the PC increments by two, to 922, and the XOP is executed. This execution is a context switch in which the XOP routine gains control of the execution sequence.

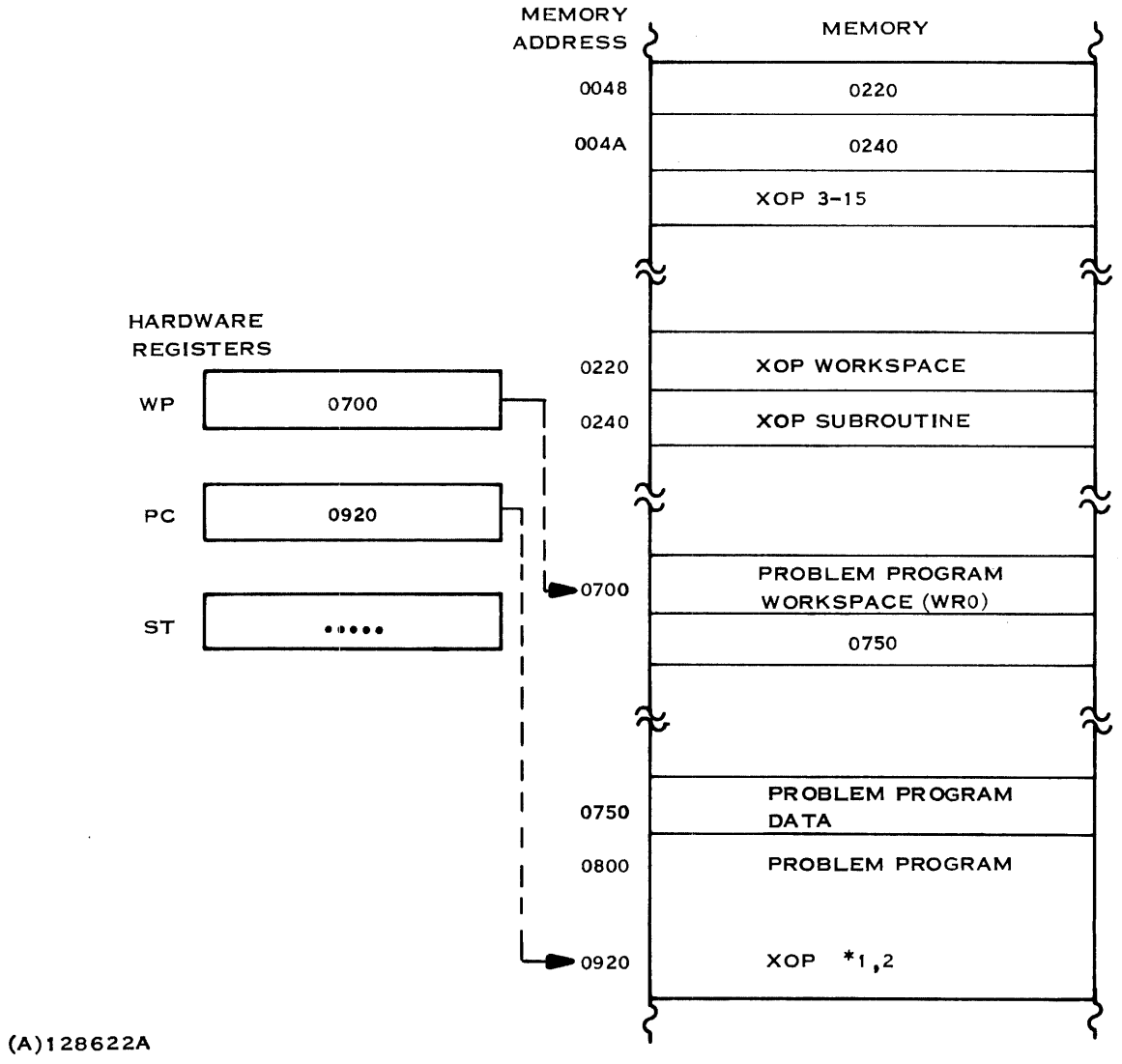


Figure 5-9. Memory Prior to XOP Instruction Execution

Note that WR1 of the calling program module contains the data address for the operand that is passed to the XOP routine.

After the context switch was complete and the XOP subroutine is in control (figure 5-10), the PC contains the starting address of the XOP subroutine and the WP contains the address of the XOP subroutine workspace. WR11 of the XOP subroutine contains the effective address of the data to be used as an operand. Workspace registers 13, 14, and 15 contain the return control information, which is used to return control to the main program module when the XOP subroutine completes execution.

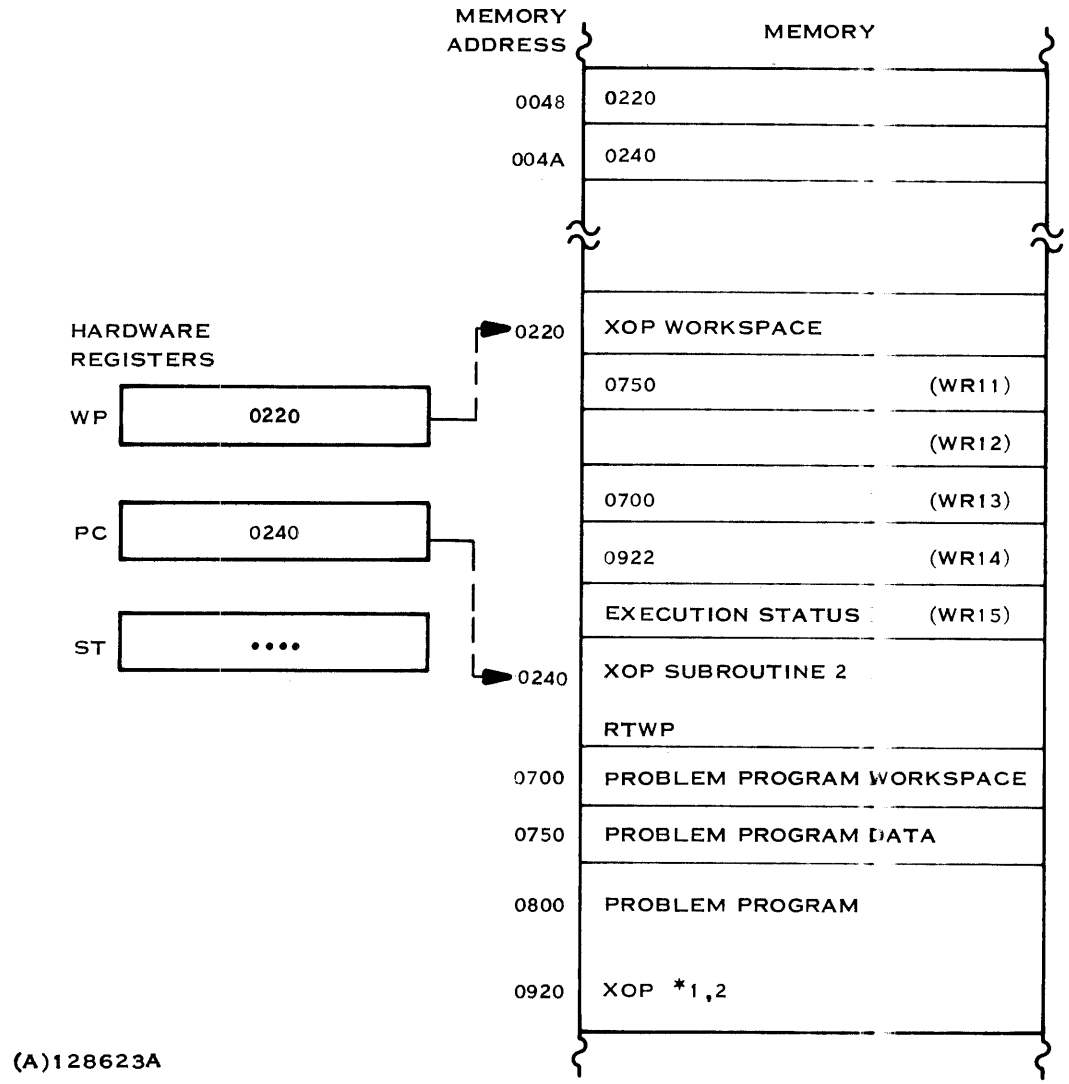


Figure 5-10. Memory After XOP Instruction Prior to XOP Routine Execution

5.12 SPECIAL CONTROL INSTRUCTIONS

There are five special control instructions that permit the programmer to control the state of the execution process of the 990 Computer. These instructions are:

Instruction	Mnemonic	Paragraph
Load ROM And Execute	LREX	5.12.1
Clock On and Clock Off	CKON/CKOF	5.12.2
Reset	RSET	5.12.3
Execute	X	5.12.4

5.12.1 LREX APPLICATIONS

The use of the LREX instruction is dependent upon the contents of the ROM that is installed in the computer. Most systems contain (at least) a warm start load capability so that LREX can be used under all systems as a restart function. This capability may then be used to halt runaway programs or to end a program. Other ROMs may provide capabilities for debug operations or overlay applications.

5.12.2 CKON/CKOF APPLICATIONS

These two instructions are used to turn on and turn off the clock, respectively. Through the use of these two instructions, the programmer may use the clock for timing operations. As an example, the clock may be used to time-out I/O procedures by turning the clock on, counting the clock interrupts until the desired time is passed, and turning the clock off. This is possible only if the interrupt level 5 for the real time clock has previously been enabled.

5.12.3 RSET APPLICATIONS

RSET is primarily used to initialize the state of the computer before performing an interrupt mask change to a lower interrupt level, i.e. change the interrupt level from 3 to 5. This instruction has the effect of clearing any pending interrupts that should not be taken when the interrupt level is changed. This instruction is also useful at the start of a program to clear the state in existence so that the new application will not be adversely affected by the previous state of the computer.

5.12.4 X APPLICATIONS

The execute instruction may be used to execute one of a table of instructions dependent upon the computed value, which represents an index into the table of instructions.

5.13 CRU PROGRAMMED INPUT/OUTPUT

The communications register unit (CRU) performs single and multiple bit programmed input/output in the Model 990 Computer. All input consists of reading CRU line logic levels into memory and output consists of setting CRU output lines to bit values from a word or byte of memory. The CRU provides a maximum of 4096 input and output lines that may be individually selected by a 12-bit address. The 12-bit address is located in bits 3 through 14 of workspace register 12 and is the base address for all CRU communications. (Each workspace may have a different base address.)

5.13.1 CRU I/O INSTRUCTIONS

There are five instructions for communications with CRU lines. They are:

- SBO - Set Bit To One. This instruction sets a CRU output line to a logic one. If the device on the CRU line is a data module, SBO results in zero volts at the data module terminal corresponding to the addressed bit.
- SBZ - Set Bit To Zero. This instruction sets a CRU output line to a logic zero. If the device on the CRU line is a data module, SBZ results in a float (no signal applied) at the data module terminal corresponding to the addressed bit.
- TB - Test Bit. This instruction reads the digital input bit and sets the equal status bit (bit 2) to the value of the digital input bit.

NOTE

The CRU address of the SBO, SBZ, and TB instructions is determined as follows:

Bits 3-14 of workspace register 12
equal the CRU base address

+

The user supplied displacement in
the instruction with sign bit extended

=

Effective CRU address

- LDCR - Load Communications Register. This instruction transfers the number of bits specified by the C field of the instruction onto the CRU from the source operand. When less than nine bits are specified, the source operand address is effectively a byte address.

When more than eight bits are specified, the source operand is effectively a word address. The CRU address is the address of the first CRU digital output affected. The CRU address is determined by the contents of workspace register 12, bits 3 through 14.

- STCR - Store Communications Register. This instruction transfers the number of bits specified by the C field of the instruction from the CRU to the memory address specified by the S and T fields of the instruction. When less than nine bits are specified, the source operand address is effectively a byte address. When there are nine or more bits specified, the source operand address is effectively a word address. The CRU address is determined by workspace register 12, bits 3 through 14.

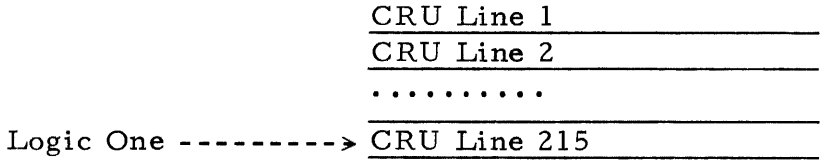
5.13.2 CRU I/O EXAMPLES

The following examples further define the input/output characteristics.

5.13.2.1 SBO. Problem - Set communications register line 215 to one (turn on a motor, for example). Workspace register 12 contains the base address of 0200_{16}. One instruction that would accomplish this objective is

```
SBO     15
```

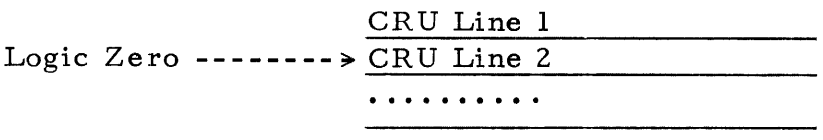
where CRU line 215 is set to one by the SBO instruction. Note that no other output line if affected. This operation is shown as follows:



5.13.2.2 SBZ. Problem - Set communications register line 2 to a logic zero to turn off a sensor. Workspace register 12 contains the base address of 0000_{16}. Note that no other output line is affected. One instruction that would accomplish this objective is

```
SBZ     2
```

where CRU line 2 is set to zero by the SBZ instruction. This operation is shown as follows:



5.13.2.3 TB. Problem - Determine if the CRU line is set to a logic one or to a logic zero for CRU line 704. Workspace register 12 contains the base address of 0700_{16}. The following instructions test the CRU line indicated and go to a particular instruction if the bit is set to a logic one or continue with another instruction if the bit is set to a logic zero.

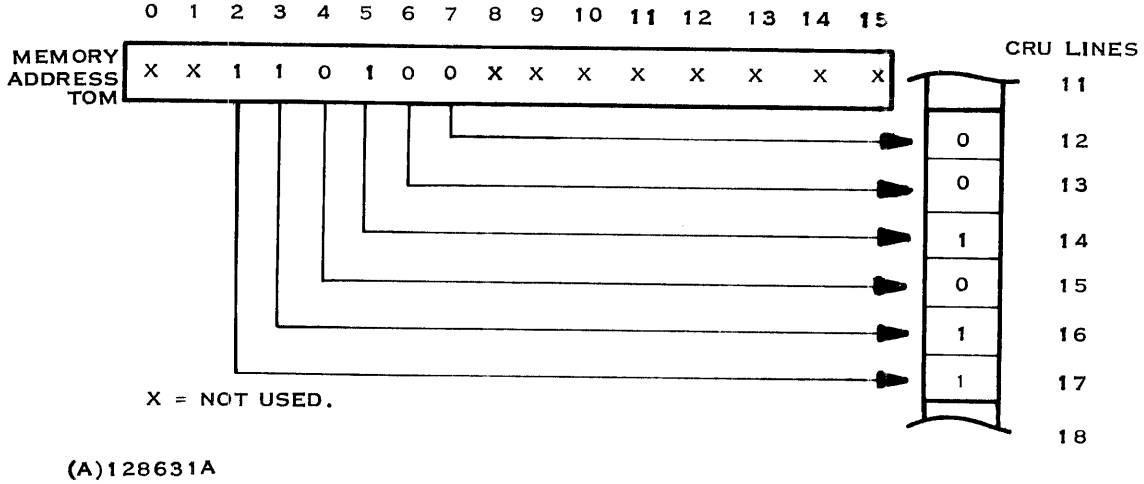
```
        TB      4       CRU LINE 704
        JEQ     RUN     IF ON, GO TO RUN
        ...             ELSE, CONTINUE
        :
        :
RUN     ...
```

If the CRU input line is set to a logic one (zero volts for a data module), the JEQ instruction transfers control to the instruction labeled RUN. If the CRU line is set to a logic zero (+5 volts for a data module), the equal status bit is reset and control does not transfer to the instruction labeled RUN but to the instruction in sequence.

5.13.2.4 LDCR. Problem - Transfer six bits from the location labeled TOM (even address) to the CRU lines that are twelve lines below the address in workspace register 12. One set of instructions that would accomplish this objective is

```
AI      12, 24      Increment the Base Address by 12
                    (Least significant bit not used)
LDCR    @TOM, 6     Transfer six bit from TOM
AI      -24         Restore Base Address In WR12
```

The following diagram illustrates the bit transfer from TOM to the CRU lines.

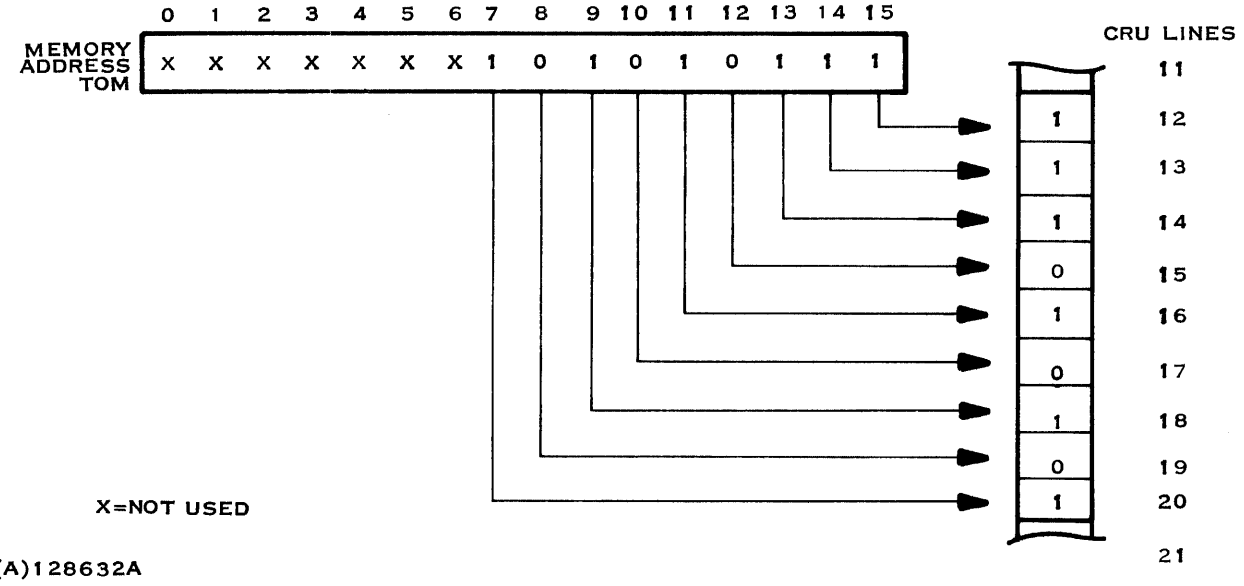


(A)128631A

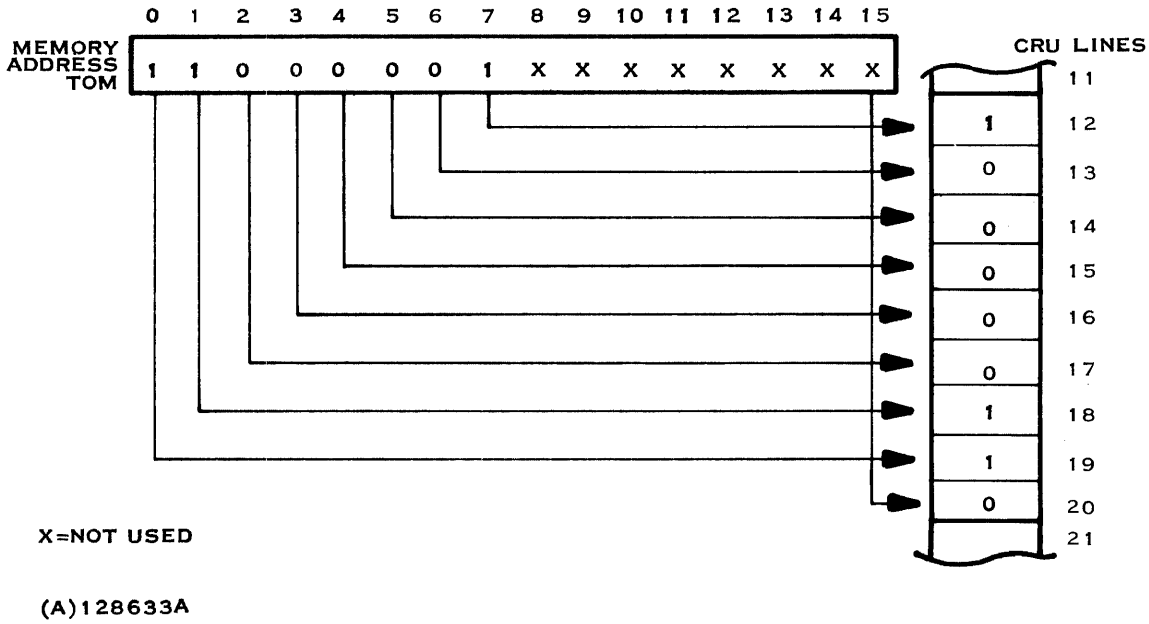
If the LDCR instruction were changed as follows

```
LDCR    @TOM, 9
```

there would be a transfer of nine bits from TOM to the same address. This instruction would result in the following diagram.



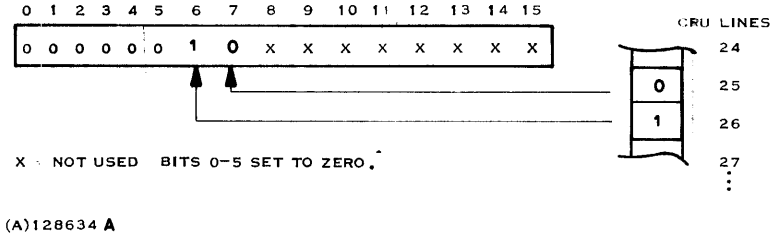
If the memory address of location TOM is an odd address, the following transfer takes place between the memory and the CRU:



5.13.2.5 STCR. Problem - Transfer two bits from the CRU lines that are 25 lines below the base address in workspace register 12 to the address specified by the contents of workspace register 2. One set of instructions that would accomplish this objective is

```
AI      12,50       SET CR BASE ADDRESS
STCR    *2,2        TWO BITS INTO ADDRESS IN WR2
AI      12,-50      RESTORE WR12
```

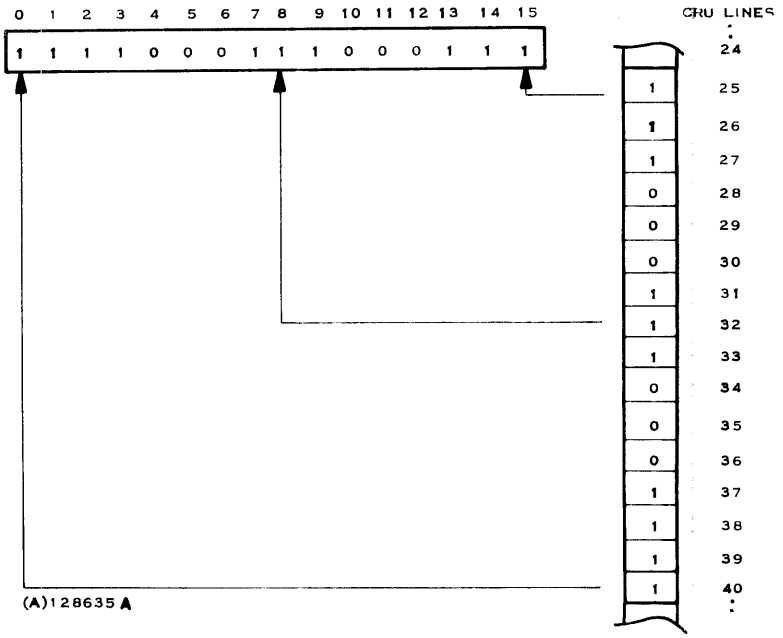
The following diagram illustrates the transfer of two bits from the CRU lines to the address in workspace register 2.



If the STCR instruction is changed to

```
STCR    *2,0
```

sixteen bits would be transferred from the CRU lines specified by workspace register 12 to the address that is specified by the contents of workspace register 2 and would result in the following diagram:

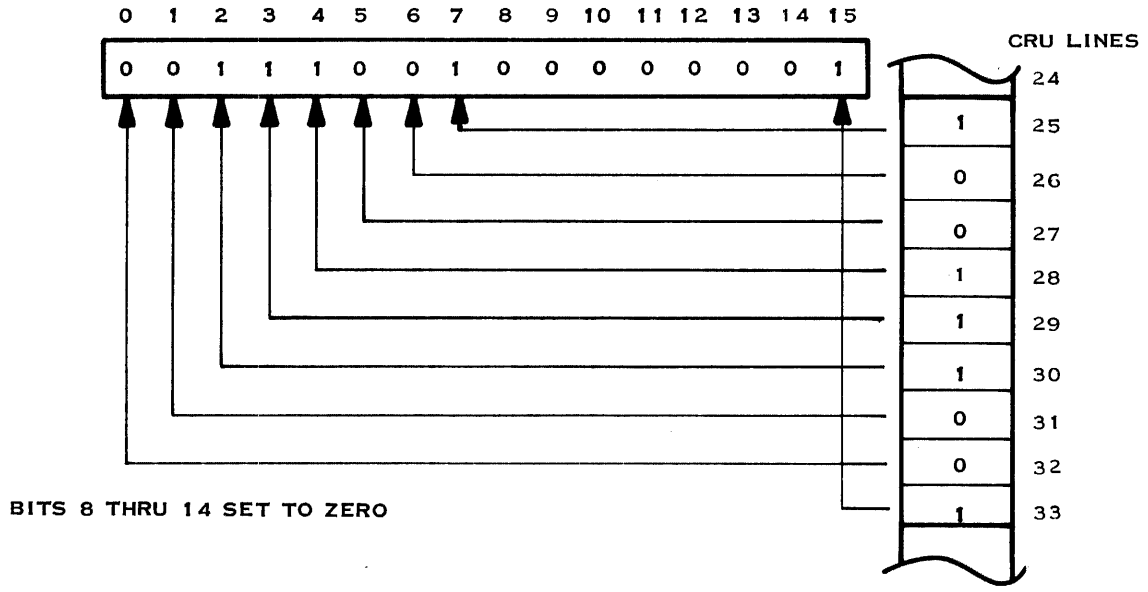


When a zero is specified for the number of bits to transfer, there is a complete word transfer of sixteen bits from the memory word to the CRU or from the CRU to the memory word.

If the STCR instruction is changed to

```
STCR    *2, 9
```

then nine bits would be stored from the address in workspace register 2. If the address in workspace register 2 is an odd address, the following transfer takes place:



(A)128636A

5.14 TILINE INPUT/OUTPUT

The set of assembly language machine instructions (Section IV) that communicate with memory may also be used to communicate with devices connected to the TILINE. To communicate with the TILINE, these instructions must be coded with appropriate addresses for the TILINE. Refer to Section II for TILINE address definitions.

In addition to appropriate addresses, the instructions must comply with the requirements of the device actually connected to the TILINE. These devices are fixed-head disc controllers, moving-head disc controllers, line printer controllers, or other high-speed peripheral controllers.

The possible instructions are:

- Format I instructions
- Format III instructions

- Format VI instructions
- Format IX instructions

Actual applicability of any of these instructions depends upon the peripheral connected to the TILINE.

5.15 RE-ENTRANT PROGRAMMING

A re-entrant procedure will typically have one procedure and multiple workspace and data areas to perform the same operation on multiple sets of data. The design of re-entrant procedure requires the following considerations:

- The procedure should not modify any of its instructions.
- The procedure should not directly address any data that is not general in application to the various tasks.
- User data that applies to a particular task and is passed to a re-entrant procedure should be symbolicly addressed by a workspace register. Data addresses in instructions should not be modified by the procedure so that the data may be subsequently addressed when that task is re-executed.
- If there is data that is general in application to all tasks for the procedure, that data may be directly addressed.

The general environment of a re-entrant procedure is shown in figure 5-11. This environment illustrates a single procedure and sixteen tasks that use the procedure. Note that the workspace for each task and the data for each task are separately located in memory. If this re-entrant procedure operates equipment connected to sixteen CRU interfaces, then WR12 for each task would contain the base address for that task's CRU interface.

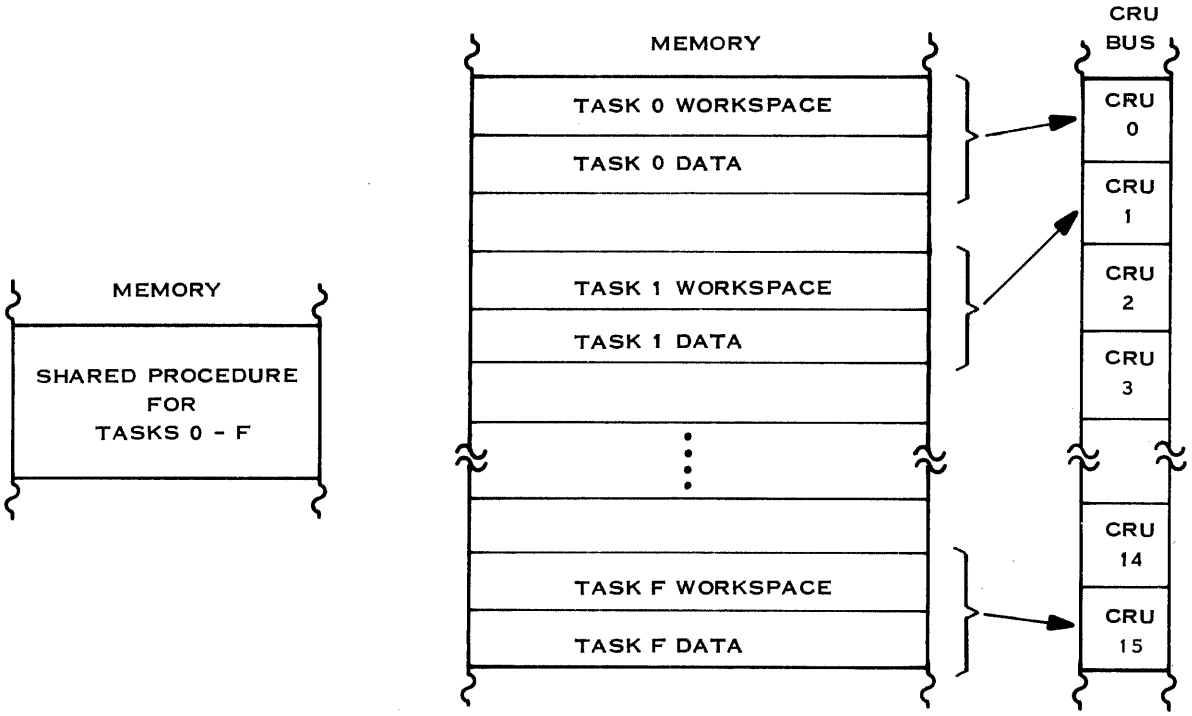
An example of re-entrant programming is as follows:

The task to be performed is to search a byte character array for a carriage return and if the carriage return is not found before the end of the array is located (the last word in the array is filled with zeros), proceed to another section of code that performs another set of instructions. Workspace register nine contains the address of the data array.

The following code is considered re-entrant:

```
ENTER   MOV   *14+,3      GET BUFFER SIZE FROM USER
        MOV   9,8         GET START ADDRESS
        A     3,8         POINT TO END OF BUFFER
```

```
LOOK     C      9,8              CHECK FOR AN END ADDRESS
         JH     NOTFND           BRANCH WHEN END
         CB     *9+,@CARRET      CHARACTER CHECK
         JNE    LOOK             BRANCH WHEN NOT FOUND
FOUND    .                       CHARACTER FOUND
         .
         .
CARRET   BYTE > D                SEARCH CHARACTER
                                 INITIALIZATION
```



(A)128624A

Figure 5-11. 990 Re-entrant Procedure Environment

The following code is considered non-re-entrant because of suggested restriction violation:

```
ENTER MOV   MOV     *14+,@ADDLOC
            MOV     9,8
            AI      8,$-$
ADDLOC      EQU     $-2
LOOK        C       9,8
            JH      NOTFND
            CB      *9+,@CARRET
            JNE     LOOK
FOUND         .
              .
              .
CARRET      BYTE  >  D
```

5.16 CREATING A SOURCE PROGRAM USING TSE990

Prior to an operation involving the use of TSE990, ensure that all operation characteristics and procedures for TSE990 are understood. Once the programmer is familiar with TSE990, the following sequence of steps permits creation of a source program on paper tape or on a cassette.

1. Load TSE990 according to the loading procedures in the System Operation Guide.
2. Cassette: Place the scratch cassette in the object output position and place the 733 ASR in the Ready and Record mode.
   Paper Tape: Place the paper tape in the punch station and place the 33 ASR in the On-Line mode.
3. Type the instruction I0 (Insert After Zero).
4. Type each line of symbolic instructions ended with a carriage return. Use the sample program in figure 5-12 for an example.

NOTE

If the available memory fills up while typing in the program, use the K (Keep) command to scroll out lines on the output media. Scrolling out lines frees memory for more input symbolic lines.

5. When the entire program has been entered, enter the Q0 (Quit-Zero) command, which scrolls out the remaining symbolic lines in memory and then terminates TSE990.

NOTE

To maintain horizontal spacing in a particular column format, and to conserve space in the source symbolic lines, use the tab feature, which will automatically space the four fields of the source program.

5.17 ASSEMBLING SOURCE PROGRAMS USING MIRA 990

To assemble source programs using MIRA 990, use the following sequence of steps:

1. Load the MIRA assembler according to the loading procedures in the System Operation Guide.
2. When MIRA is properly loaded and ready to accept a source program, a question mark (?) is printed on the teleprinter.
3. When the input file is ready (cassette loaded or paper tape placed in the read station), press any character on the keyboard except the carriage return and MIRA will assemble the program. A carriage return will terminate MIRA.
4. When the assembly process is complete, the MIRA assembler prints another ? on the teleprinter, which indicates that the MIRA assembler will accept another program if necessary. To terminate MIRA at this point in the sequence, press the carriage return.

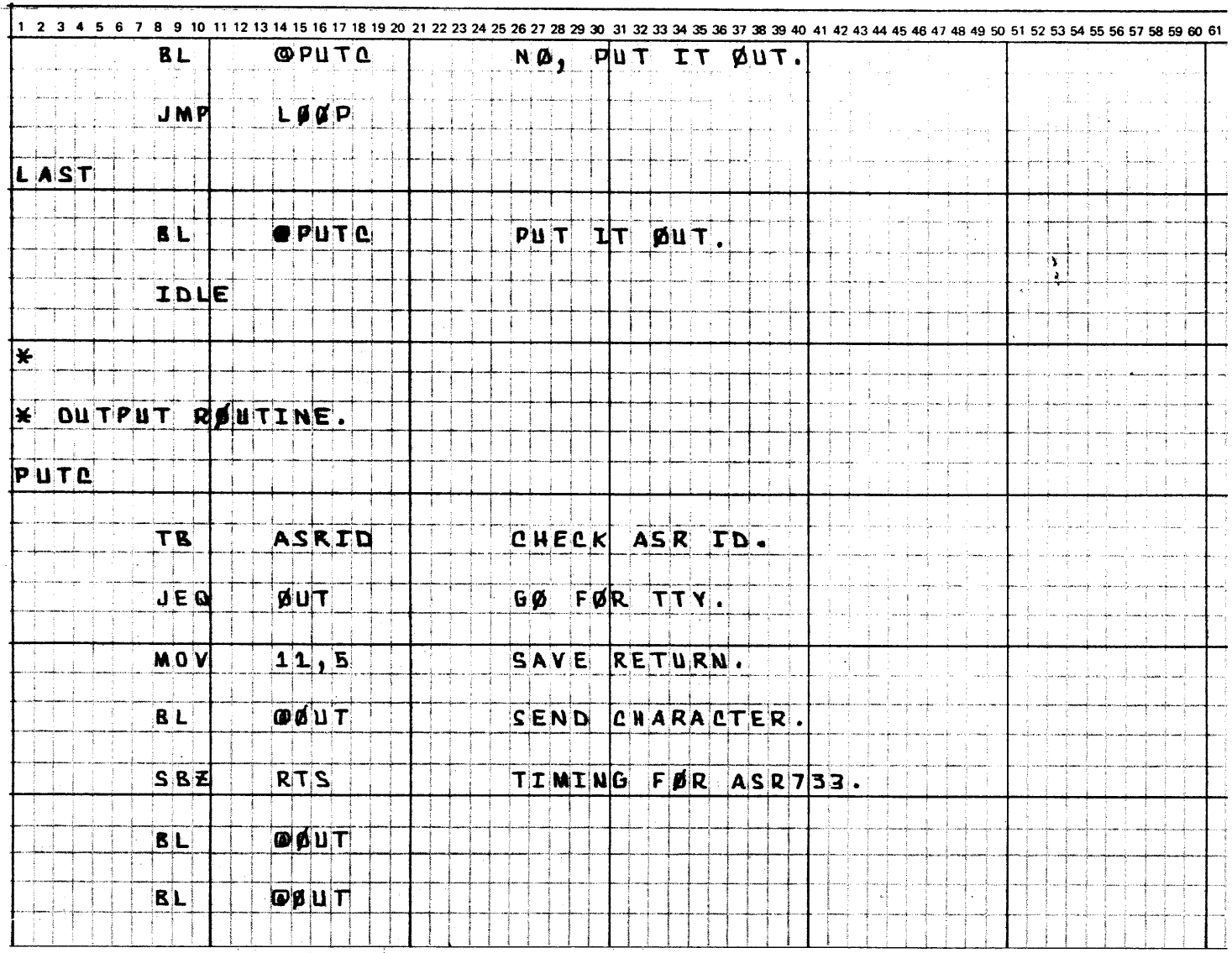
5.18 EXAMPLE PROGRAM

The example program in figure 5-13 was assembled using MIRA, and executed on the 990 Computer. Figure 5-14 shows the results of the execution of this sample program.

```
1 2 3 4 5 6 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
        TITL 'HELLØ! PRØGRAM'
* THIS PRØGRAM PRINTS 'HELLØ!' ØN THE TELEPRINTER.
DTR     EQU  13          DATA TERMINAL READY.
WRQ     EQU  11          WRITE REQUEST.
RTS     EQU  10          REQUEST TØ SEND.
ASRID   EQU  10          ASR733/33 ID.
*
HELLØ
        LWPI REGS        INITIALIZE WØRKSPACE PØINTER.
        LIMI 0           DISABLE INTERRUPTS BELØW ZERØ.
        LI   12,>100     INITIALIZE CRU BASE.
        LI   2,TABLE     LØAD TABLE ADDRESS.
LØØP
        MOVB *2+,8       GET A CHARACTER.
        JLT  LAST        LAST CHARACTER?
```

(A)128625 (1/4)

Figure 5-12. Sample Program (Sheet 1 of 4)



```
                BL      @PUTC           NØ, PUT IT ØUT.
                JMP     LØØP
LAST
                BL      @PUTC           PUT IT ØUT.
                IDLE
*
* OUTPUT RØUTINE.
PUTC
                TB      ASRID           CHECK ASR ID.
                JEQ     ØUT             GØ FØR TTY.
                MOV     11,5            SAVE RETURN.
                BL      @ØUT            SEND CHARACTER.
                SBZ     RTS             TIMING FØR ASR733.
                BL      @ØUT
                BL      @ØUT
```

(A)128625 (2/4)

Figure 5-12. Sample Program (Sheet 2 of 4)

```
1 2 3 4 5 6 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
          BL    @ØUT
          SBØ   RTS
          B     *5          RETURN TØ CALLER.
*
ØUT
          TB    DTR         ASR ØN LINE?
          JNE   $-2         WAIT UNTIL IT IS.
          LDCR  8,8         ØUTPUT CHARACTER.
          TB    WRQ         WAIT ØN IT.
          JNE   $-2
          SBØ   WRQ
          B     11          RETURN TØ CALLER.
*
* MESSAGE TABLE.
TABLE
```

(A)128625 (3/4)

Figure 5-12. Sample Program (Sheet 3 of 4)

```
1 2 3 4 5 6 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
            TEXT    'HELLØ'
            BYTE    '!'+>80         SET PARITY BIT.
            EVEN
REGS        BSS     32              WØRKSPACE AREA.
            END     HELLØ
```

(A)128625 (4/4)

Figure 5-12. Sample Program (Sheet 4 of 4)

```
HELLO! PROGRAM                                              PAGE 0001

0002                    * THIS PROGRAM PRINTS 'HELLO!' ON THE TELEPRINTER..
0003          000D      DTR     EQU     13        DATA TERMINAL READY.
0004          000B      WRQ     EQU     11        WRITE REQUEST.
0005          000A      RTS     EQU     10        REQUEST TO SEND.
0006          000A      ASRID   EQU     10        ASR733/33 ID.
0007                    *
0008                    HELLO
0009  0000    02E0              LWPI    REGS      INITIALIZE WORKSPACE POINTER.
      0002    ----
0010  0004    0300              LIMI    0         DISABLE INTERUPTS.
      0006    0000
0011  0008    020C              LI      12,>100   INITIALIZE CRU BASE.
      000A    0100
0012  000C    0202              LI      2,TABLE   LOAD TABLE ADDRESS.
      000E    ----
0013                    LOOP
0014  0010    D232              MOVB    *2+,8     GET A CHARACTER.
0015  0012    11--              JLT     LAST      LAST CHARACTER?
0016  0014    06A0              BL      @PUTC     NO, PUT IT OUT.
      0016    ----
0017  0018    10FB              JMP     LOOP
0018                    LAST
      0012**1103
0019  001A    06A0              BL      @PUTC     PUT IT OUT.
      001C    ----
0020  001E    0340              IDLE                STOP.
0021                    *
0022                    * OUTPUT ROUTINE.
0023                    PUTC
      0016**0020'
      001C**0020'
0024  0020    1F0A              TB      ASRID     CHECK ASR ID.
0025  0022    13--              JEQ     OUT       GO FOR TTY.
0026  0024    C14B              MOV     11,5      SAVE RETURN.
0027  0026    06A0              BL      @OUT      SEND CHARACTER.
      0028    ----
0028  002A    1E0A              SBZ     RTS       TIMING FOR ASR733.
0029  002C    06A0              BL      @OUT
      002E    ----
0030  0030    06A0              BL      @OUT
      0032    ----
0031  0034    06A0              BL      @OUT
      0036    ----
0032  0038    1D0A              SBO     RTS
0033  003A    0455              B       *5        RETURN TO CALLER.
0034                    *
0035                    OUT
      0022**130C
      0028**003C'
      002E**003C'
      0032**003C'
      0036**003C'
0036  003C    1F0D              TB      DTR       ASR ONLINE?
0037  003E    16FE              JNE     $-2       WAIT 'TILL IT IS.
0038  0040    3208              LDCR    8,8       OUTPUT CHARACTER.
```

Figure 5-13. Sample Program Assembly (Sheet 1 of 2)

```
HELLO! PROGRAM                                          PAGE 0002

0039  0042  1F0B          TB      WRQ     WAIT ON IT.
0040  0044  16FE          JNE     $-2
0041  0046  1D0B          SBO     WRQ
0042  0048  045B          B       *11     RETURN TO CALLER.
0043                *
0044                * MESSAGE TABLE.
0045                TABLE
      000E**004A'
0046  004A    48          TEXT    'HELLO'
      004B    45
      004C    4C
      004D    4C
      004E    4F
0047  004F    A1          BYTE    '!'+>80 SET PARITY BIT.
0048                      EVEN
0049  0050        REGS    BSS     32      WORKSPACE AREA.
      0002**0050'
0050                      END     HELLO
0000  ERS
```

Figure 5-13. Sample Program Assembly (Sheet 2 of 2)

```
HELLO!
```

Figure 5-14. Sample Program Execution

SECTION VI

SOFTWARE PACKAGES

6.1 GENERAL

This section of the manual contains information about the various software packages available with the 990 Computer. The available packages are:

- MIRA990 - Assembler that assembles object code from assembly language statements. This assembler executes in the 990 Computer.
- MIRA990/360 - This version of the assembler executes on the IBM System/3x0 family of machines.
- LAL990 - Link And Load module that provides external reference definitions and loading instructions to the loader to load any linked object programs into memory.
- ASR I/O - Automatic Send/Receive Input/Output software package that permits the 990 Computer to communicate with peripheral devices that are described in Section III.
- TSE990 - Terminal Source Editor software package that permits the programmer to edit programs using the avilable terminal peripherals.
- XDB990 - A hexadecimal debug package that permits the programmer to debug programs in the stand-alone mode.

These descriptions are general in nature and leave specific information and descriptions to the particular User's Guide.

6.2 MIRA990 ASSEMBLER

MIRA990 is a one-pass assembler designed to assemble absolute and/or relocatable assembly language source code (in the same program module, if desired) in a minimum memory configuration of 4K (4096) words. This same assembler may be loaded at higher memory locations in larger memory configurations to utilize this "extra" memory space for more symbols without resorting to a "new" assembler version. Operational control of the assembly process is via either the TI Model 733 ASR Teleprinter or the Model 33 ASR Teletypewriter. In addition to operational control, these two terminals provide printed outputs from the assembly process in the form of assembly listings and error codes. MIRA also provides for a minimum of 125 6-character symbols in the symbol table. If less than six characters are used per symbol, the assembler will accept more than 125 symbols. Programmers that use this assembler may batch more than one module if desired. The object output from the MIRA assembler is in USASCII code.

There are several features related to the USASCII code object output that improve patching options. The RORG (relocatable origin) assembler directive permits over-patching of source code by the programmer that will overlay previously detected error code at the time of loading. Since the object output is in USASCII code, this object may be patched directly without passing the module through the assembly process again. In effect, scatter loading of object code permits this method of over-patching the source code or directly patching the USASCII object whenever the corrected code appears in the load sequence subsequent to the error code. Also, load addresses need not be in consecutive order as long as any IDT, DEF, or REF record appears in the specified positions and as long as a load address precedes new or changed records, if necessary.

6.3 MIRA990/360 CROSS ASSEMBLER

The MIRA990/360 Cross Assembler is designed to execute on the IBM System/3x0 family of machines under OS control. MIRA 990/360 is a two-pass assembler that requires a minimum of 250K bytes of memory on the System/3x0. All of the other features described for the MIRA assembler are available with this cross assembler, with some additions. Output listings are on the line printer and more information descriptive messages appear in place of the error codes. Operational control is via the punched card media and all options must be specified at the time of assembly. Options included with MIRA 990/360 are:

- Batching of assemblies
- Variable symbol table size (150 symbols default)

USASCII object code is also the output from this cross assembler, which permits the same types of patching of object output as with the 990 MIRA Assembler.

6.4 LINK AND LOAD (LAL990)

LAL990 is designed to resolve linking requirements from previously referenced symbols (REF assembler directives) and to load both the absolute and relocatable object output, generated by the assembler, from the source code inputs. During the linking and loading process, LAL990 maintains a dynamic symbol table size to react to the load module requirements. LAL990 will load both absolute and relocatable object code in one process and will overlay code previously loaded if directed by identical load addresses, thus implementing the scatter loading and object patching features. LAL990 also accepts user provided relocatable load bias for any or all load modules. If no bias is provided, LAL990 loads at a default address of 200_{16}. If required, LAL990 will accept load modules, from physically separate media, of the same primary media required for input. This feature permits loading of several modules from several cassettes or from several different pieces of paper tape object. LAL990 outputs to the terminal device any appropriate message, the entry address, a list of undefined references, and a list of all

loaded modules with appropriate relocatable load bias at which the modules were loaded. When loading of modules is complete and there are no fatal errors, execution may begin at the specified entry address.

6.5 TERMINAL SOURCE EDITOR (TSE990)

TSE990 is an interactive, terminal oriented program that runs stand-alone on the 990 Computer. TSE990 will store 2900 characters from a module to be edited in the minimum memory configuration of 4K. If additional memory is available, TSE990 will utilize all additional memory without modification. TSE990 accepts either source or object code from either the keyboard, the cassette, or the paper tape reader. Editing commands are input by the user from the teleprinter terminal. These editing commands are self-explanatory and easy to use or remember. In addition to a standard character editing feature, TSE990 permits string editing that replaces the entire string, a portion of the string, or one character of the string. During the editing process, TSE990 provides informative/warning messages about the status of the edit and error messages for syntax errors. Operational commands permit scrolling in and out of the editing source module. If a module contains more than 2900 characters and the computer has a 4K memory, portions of the source module may be scrolled in for editing, scrolled out after editing to save the edited source and scrolling in of the subsequent character block. When an editing process is complete, TSE990 is reusable without having to be re-loaded.

6.6 INPUT/OUTPUT PACKAGE FOR THE 990 COMPUTER

This input/output package permits the user to input and/or output characters and/or records on the Model 33 Teletypewriter and the TI Model 733 ASR Teleprinter. Input/output to/from the 733 ASR may be either the keyboard or the cassette. Input/output to/from the Model 33 ASR may be either the paper tape reader/punch or the keyboard. Character I/O routines are as follows:

- IN - Inputs the character in the left half of workspace register 8.
- OUT - Outputs the character in the left half of workspace register 8, at a 1200 baud rate, to the 733 ASR.
- OUTP - This subroutine calls OUT and outputs the character in the left half of workspace register 8, at a 300 baud rate, to the 733 ASR.

Record I/O routines are as follows:

- KEY - Reads a record from the keyboard.
- READ - Reads a record from the cassette (733 ASR) or from the paper tape reader (33 ASR).
- PUNCH - Punches a record to tape (paper tape 33 ASR and cassette 733 ASR).

- PRINT - Prints a record on the teleprinter print device.
- LEADER - Punches a paper tape leader (Ignored on 733 ASR).
- ENDFIL - Writes and end-of-file character at the end of the record.
- CONRET - Control Return. When the control character is encountered, control returns to the specified address.

The print routines for the printing device on the teleprinter permit format control similar to other existing format control characters. These characters control the printing device for top-of-form, single/double spacing, carriage return, and print from present position. All other printing is single-spaced automatically. When the terminal in control is a 733 ASR, cassette operations in the form of rewind, unload, block reverse, place in record mode and check status are permitted.

This I/O package is a non-interrupt driven set of subroutines that may be linked into the user's program module to perform any input/output operation required by that module. If required by application, this I/O package will write from one buffer while reading from another buffer, thereby permitting simultaneous read/write operations. This I/O package will operate in either the 33 ASR or the 733 ASR environment, without modification, whenever the cassette operation subroutines are included when using the 733 ASR.

6.7 HEXADECIMAL DEBUG PACKAGE (XDB990)

XDB990 is a hexadecimal debug that does not simulate execution of the module being debugged. Since execution is not being simulated, XDB990 does not keep control of the execution process. The module being debugged actually executes under its own control. Operational control of the debugging process is via the teleprinter and all communication between XDB990 and the user is in the form of symbolic mnemonics and hexadecimal memory addresses. XDB990 provides several options to check out particular executable portions of any module. The user may inspect, or inspect and change, single memory locations or consecutive memory locations (either forward or reverse). Memory dumps of sequential memory locations are permitted for any one or all memory locations of the module being debugged. The user may inspect the workspace pointer and the contents of the status register and, if necessary, alter the contents of either register. Load operations are permitted from a cassette or paper tape. Execution of the module is permitted from the load end vector or from a particular memory address within the extent of the module being debugged. There is also a breakpoint option that permits setting of a breakpoint to stop execution, altering of the breakpoint, execute from the breakpoint (continuation of the stopped execution), and clearing of the breakpoint previously set. For breakpoint operations, XDB990 uses an illegal operation trap.

SECTION VII
INSTALLATION

7.1 GENERAL

This section defines the steps required to prepare for installation and to install the Model 990 Computer for a shelf or cabinet mounted configuration. It includes instructions for wiring backpanel interrupts as well as cable routing examples.

7.2 SITE PREPARATION

The installation site for the Model 990 Computer should allow for the size, weight and electrical requirements of the computer chassis. To aid in site planning prior to delivery of the computer, table 7-1 lists the physical and electrical requirements of the computer.

Table 7-1. Model 990 Computer Physical and Electrical Requirements

Characteristic	Requirement
Chassis size	12-1/4 inches high 19 inches wide 21-1/4 inches deep (plus cabling room)
Mounting space	
Rack	12-1/4 inches vertical rack space
Shelf	25 inches wide by 26 inches deep; capable of supporting chassis weight
Weight	Less than 75 pounds fully implemented
Temperature	
Operating	0 to 50 degrees Centigrade (32 to 122 degrees F)
Storage	-40 to 65 degrees Centigrade (-40 to 149 degrees F)
Humidity	
Operating or Storage	0% to 95% relative humidity without condensation
Cooling	Free flow of air to computer chassis
Shock limits	
Operating	Force of 1 G to chassis
Shipping	Force of 15 G's to carton

Table 7-1. Model 990 Computer Physical and Electrical Requirements (Continued)

Characteristic	Requirement
Voltage	115 Vac ± 10%
Line frequency	50 Hz or 60 Hz
Current	5 amps (maximum)

7.3 UNPACKING

The computer is shipped in a corrugated cardboard container together with the circuit boards and interconnecting cables required to install the system. Upon receipt of the container, inspect to ensure that no signs of physical damage are present. Following preliminary inspection, perform the following steps to remove the computer from its container and ready it for operation. Figure 7-1 illustrates the required steps.

1. Position container so that the address label is right-side up.
2. Open top of container and remove polyfoam cushioning material from corners.
3. Remove plastic bag containing cables from between side of container and cardboard inner shroud.
4. Remove cardboard inner shroud and foam block that cushions circuit boards.
5. Remove computer and attached shipping pallet from container. Place computer on a convenient work surface.
6. Shipping pallet is held to computer by four screws and flat washers that are accessible from the underside of the shipping pallet. Remove these screws and washers.

7.4 CHASSIS CONFIGURATION

To adapt the hardware chassis to specific system requirements, the external interrupts from interface modules are not connected when the computer is delivered to the installation site. This allows the user to configure the interrupt assignments to best suit his needs. These connections must be made during the installation procedure. Before making these connections, however, each module in the system must be assigned a permanent chassis location.

7.4.1 MODULE LOCATIONS

Figure 7-2 illustrates the chassis locations within the computer and the types of circuit modules that may be used in each of these locations. Notice that some chassis locations are preassigned to the power supply, arithmetic

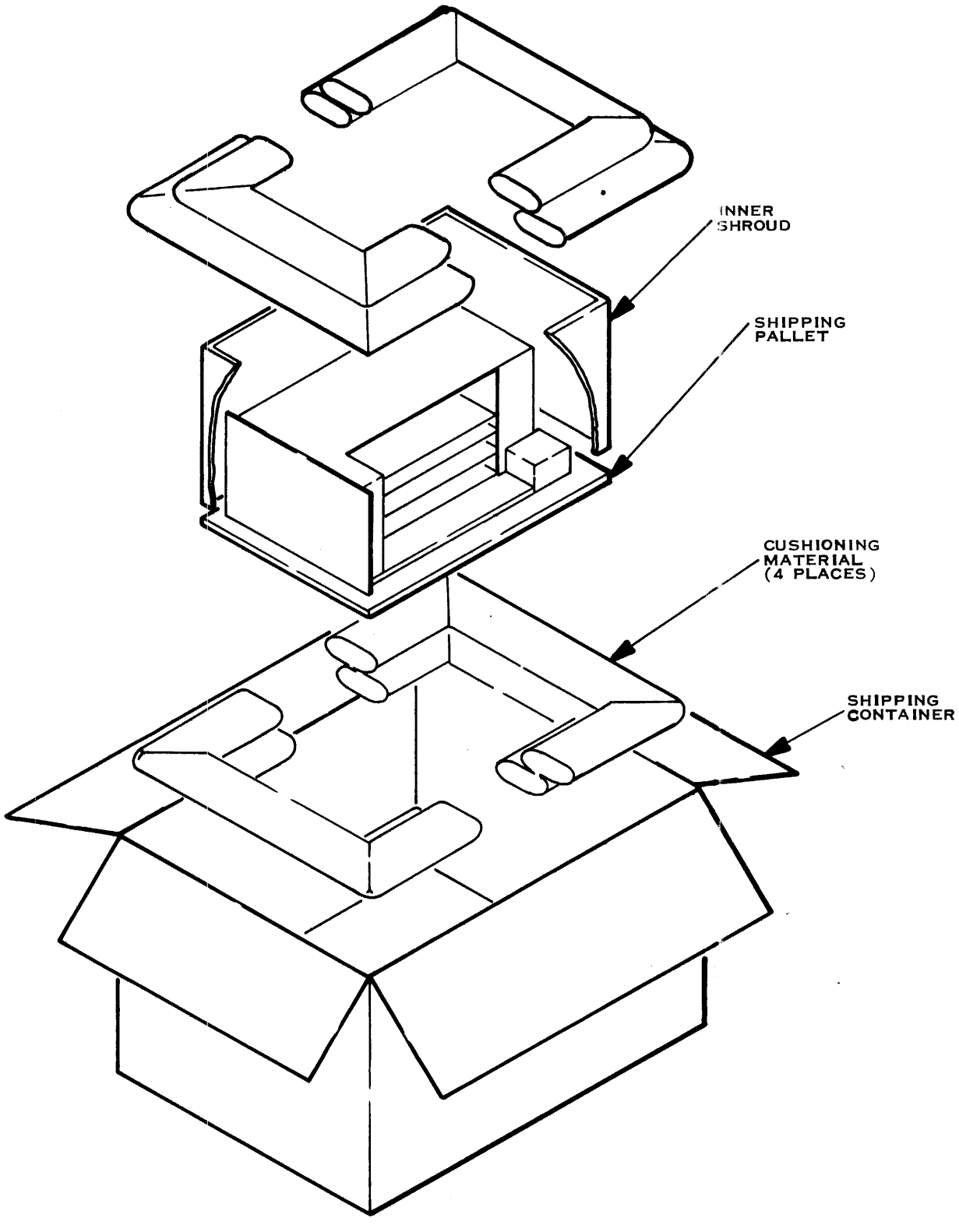


Figure 7-1. Computer Shipping Packaging

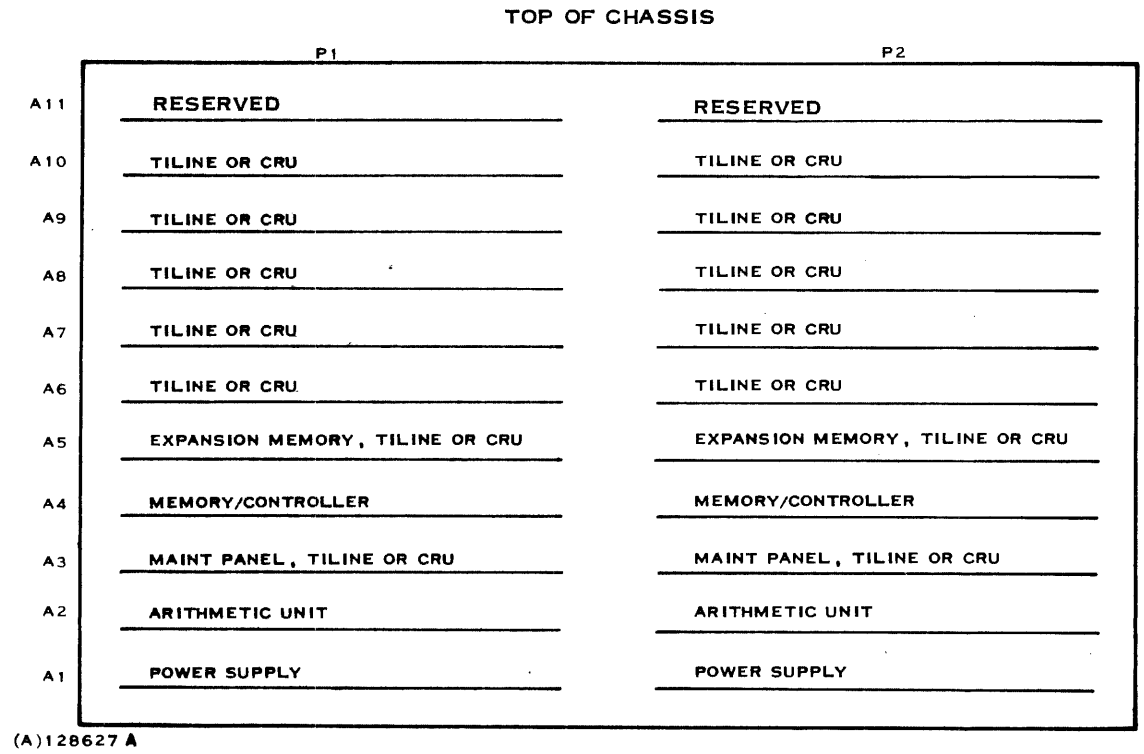


Figure 7-2. Possible Chassis Location Assignments

unit and memory controller modules. In addition, if expansion memory is included in the system it must be in location A5, next to the memory controller. Each chassis location is prewired with specific CRU addresses (module select lines). Therefore, each CRU module must be placed in the chassis location corresponding to its assigned address. Within these restrictions, assign each module in the system a specific chassis location. To aid in this process, table 7-2 tabulates the chassis locations, the corresponding CRU addresses, and the modules with preassigned locations and table 7-3 lists each standard CRU module and its module select assignments. Table 7-2 also provides space to record the location of each device within the system. Complete the P1 Device and P2 Device columns of this table with the modules for the system being installed. Remember that double-connector devices occupy both P1 and P2 connectors.

7.4.2 MODULE INTERRUPT LEVELS

The system software defines the levels of the external interrupts generated by the interface modules. If required, several modules may share the same interrupt level if the service routine for that level determines which of the modules generated an interrupt. Interrupt levels 6 through 15 are used for external interrupts. Level 6 is the highest priority and level 15 is the lowest priority. In general, mass storage devices on the TILINE interface should receive the highest priority, followed by time dependent CRU devices

Table 7-2. System Chassis Configuration (to be completed during installation)

Loc	P1 Device	CRU 1 Base Address IMODSELB/ IMODSELA	TILINE Address	2 Interrupt A/B	P2 Device	CRU 1 Base Address IMODSELB/ IMODSELA	TILINE Address	2 Interrupt A/B
A11		Not Not wired/wired		/		Not Not wired/wired		/
A10		1E0/1C0		/		1C0/1E0		/
A9		1A0/180		/		180/1A0		/
A8		160/140		/		140/160		/
A7		120/100		/		100/120		/
A6		0E0/0C0		/		0C0/0E0		/
A5		0A0/080		/		080/0A0		/
A4	Memory/ Controller	060/040	0-0BFF or 0-0FFF	(Internal 2)	Memory/ Controller	040/060	0-0BFF or 0-0FFF	(Internal 2)
A3		020/000		/		000/020		/
A2	Arithmetic Unit	- / -	-	(Internal 3, 4)	Arithmetic Unit	- / -	-	(Internal 3, 4)
A1	Power Supply	- / -	-	(Internal 0, 1, 5)	Power Supply	- / -	-	(Internal 0, 1, 5)

NOTES: 1 CRU Base Address A uses pin 48 for module selection.
CRU Base Address B uses pin 46 for module selection.

Note that P1 is different from P2.

2 Interrupt A generated on pin 66.
Interrupt B generated on pin 65.

Table 7-3. CRU Module Select Signals

Module	Module Select	Pin	Function
Modem and Controller	B	P1-46	Read status
	A	P1-48	Read data status
913 CRT and Controller	B	P2-46	Position/status input Printer data output
	A	P2-48	CRT data input/output
Full Duplex TTY/EIA Module	A	P1 or P2-48	Data input/output
Data Module (16 I/O)	A	P1 or P2-48	Data input/output

and then lower speed TILINE devices. This interrupt order may be altered to accomodate specific applications. Determine the interrupt levels assigned to each of the modules in the computer system and enter those values in the INTERRUPT column of table 7-2. To aid in this process, table 7-4 lists the interrupt pin assignments for standard modules offered with the computer system.

Table 7-4. Module Interrupt Pin Assignments

Module	Interrupt	Pin	Function
Model 913 CRT	B	P2-65	Keyboard interrupt
	A	P2-66	Printer interrupt
Modem and Controller	A	P1-66	Input interrupt
Full Duplex TTY/EIA	A	P1 or P2-66	Input interrupt
16 I/O Data Module	A	P1 or P2-66	Input interrupt

7.4.3 INTERRUPT INSTALLATION

Included in the cable package is a kit of jumper wires used to connect interrupts between module connectors and the AU circuit board connector. Perform the following procedure to install the interrupt lines required to implement the configuration for the particular system as outlined in table 7-2.

1. Position computer chassis such that the left side is easily accessible.

2. Remove four screws and metal plate from left side of chassis to expose wire-wrap connections (see figure 7-3).

3. Determine the chassis location and connector that has been assigned interrupt level 6.

4. Push one end of a jumper wire on the wire-wrap pin corresponding to P1-25 of the arithmetic unit chassis connector and connect the other end of the wire to the connector pin generating the interrupt for level 6. If more than one module is assigned to this level, connect the AU pin to each of the pins generating an interrupt for that level. Multiple connections may be daisy-chained from connector to connector to avoid overloading the AU wire-wrap pin.

5. Repeat steps 3 and 4 for the remaining interrupt levels. Refer to table 7-5 for the AU connector pin numbers of each of the interrupt level inputs.

6. Reinstall metal plate and four screws removed in step 2 of this procedure.

7.5 MOUNTING

After the computer has been removed from its shipping container and the desired interrupt signals installed, the unit can be mounted for operation. Four rubber feet, included in the parts kit with the cables, screw into the

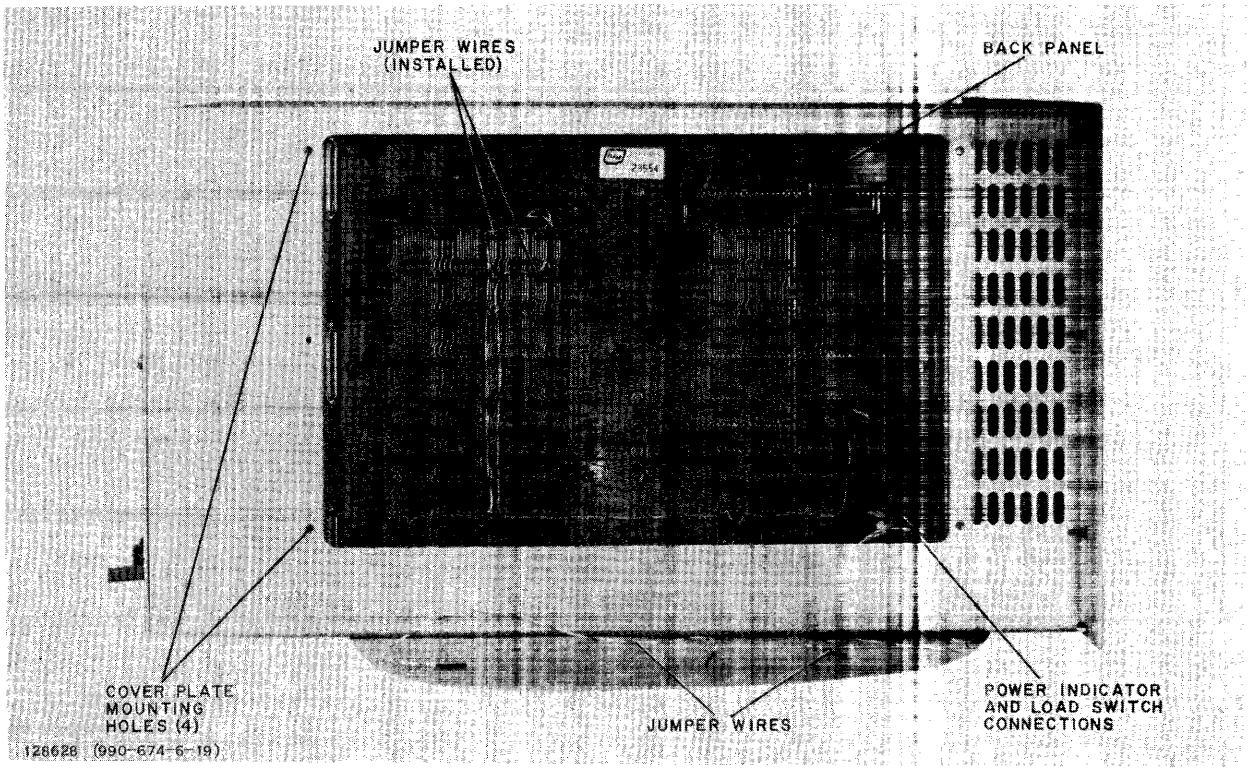


Figure 7-3. Module Interrupt Installation

Table 7-5. Interrupt Level Input Pin Assignments

Level	AU Connector Pin Number	Level	AU Connector Pin Number
6	P1-25	11	P1-16
7	P1-27	12	P1-73
8	P1-22	13	P1-15
9	P1-23	14	P1-12
10	P1-18	15	P1-11

holes used to secure the computer to the shipping pallet. These feet ensure that the shelf surface that mounts the computer will not be marred by the computer. The computer may then be placed on a shelf or table, or within a cabinet, that will support the weight of the chassis. Ample room must be provided for free air flow to cool the computer logic, and for cable connections to exterior equipment.

7.6 CABLING

Ensure that all required circuit boards are in the computer chassis in the orientation prescribed in table 7-2, and that each circuit board is firmly seated in the chassis connector. Connect the interface cables to the top edge connector of the proper interface circuit board for that device (refer to Section III of this manual for part numbers of particular cables within the peripheral device interconnection kit). Connector on the cable must be oriented so that the cable feeds out through the rear of the chassis. Secure the cable within the plastic cable clamps mounted to the computer chassis and connect the other end of the cable to the peripheral equipment or other input device. Figure 7-4 illustrates the cabling technique for each of the standard peripheral units described in Section III of this manual.

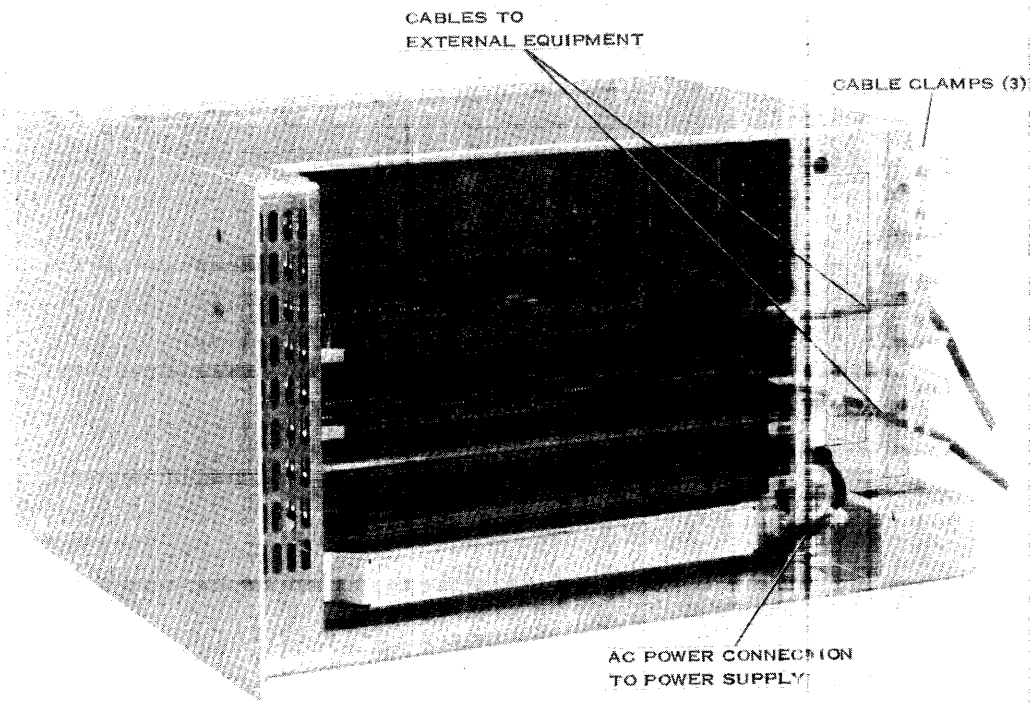
7.7 POWER APPLICATION

When all cables have been installed in the computer, ensure that the memory address switches on the memory controller board are all OFF (see memory description in Section II of this manual). When all circuit boards are in place in the chassis, the unit may be powered up. Perform the following steps to apply power to the computer:

1. Ensure that the main power connector within the computer chassis is connected to the power supply circuit board.
2. Connect power cord to a source of suitable ac power for the system being installed (115 Vac 50 Hz, or 115 Vac 60 Hz).
3. Set BATTERY toggle switch on the rear of the chassis to ON.

4. Set POWER toggle switch on the rear of the chassis to ON. Indicator on front panel of the computer should light.

5. Press the Load switch on the front panel of the computer to load the system bootstrap from the internal ROM, to enable system generation with the specified I/O device.



128629 (990-674-6-12)

Figure 7-4. Peripheral Cabling Technique

APPENDIX A
INSTRUCTION EXECUTION TIMES, MODEL 990

APPENDIX A

INSTRUCTION EXECUTION TIMES, MODEL 990

A.1 GENERAL

This appendix contains the information necessary to calculate the instruction execution times for the various instructions in the instruction set. These execution times are contained in tables in this appendix and the table number corresponds to the format number of the instruction. There are two types of memory elements available with the 990 Computer. The principal type of memory element is the MOS 4K array. An optional memory element of 1K (maximum) additional memory is 120 ns compatible. The MOS memory element contains the error correction logic while the 120 ns memory does not provide any additional bits with the 16-bit words that are used in the 990 Computer.

The following conventions are used in the tables for MOS error correcting memory:

- Clock cycle (C) = 0.250 to 0.270 μs (0.255 used as the base number for the calculation shown in the tables.)
- Read cycle (R) = 0.820 μs
- Delayed read cycle (DR) = 0.960 μs
- Write cycle (W) = 0.895 μs

The following conventions are used in the instruction execution tables for 120 ns memory. (Note that for mixed memory, both MOS and 120 ns, the 120 ns memory could be used for workspace and the MOS memory for general data storage. For 120 ns execution time to be utilized, all data must be resident in the 120 ns memory. Any data accessed from MOS memory will degrade the 120 ns execution time accordingly.)

- Clock cycle (C) = Same as MOS memory
- Read cycle (R) = 0.320 μs
- Delayed read cycle (DR) = 0.460 μs
- Write cycle (W) = 0.415 μs

Instruction execution depends upon the number of different cycles required by the addressing mode of the operand, the format of the instruction, the PC increment, and the instruction acquisition time. Each table contains the various combinations of times required by the format and addressing mode(s) of the instruction. Also included are example instruction times for the MOS and 120 ns memories.

The MOS memory requires a refresh cycle once every 31 μs. This cycle is 0.600 μs in length. Since the duration of the refresh cycle is only two percent of the active memory time (0.600/31=.02), the competition for refresh with memory access is not a significant factor. If, for example, memory refresh competed with a constant read cycle for access to memory, the read cycle time would only be degraded by the refresh cycle for an effective average of 6 nanoseconds for each read cycle.

Table A-1. Format I Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Source operand acquisition							
T_s = 00		1			0.200	0.820	0.320
= 01		1	1		0.540	1.780	0.780
= 10 (S≠0)		2	1		0.740	2.600	1.100
= 10 (S=0)		1	1	1	0.795	2.035	1.035
= 11	1	2		1	0.950	2.830	1.310
Destination operand acquisition							
T_d = 00			1		0.340	0.960	0.460
= 01			2		0.680	1.920	0.920
= 10 (D≠0)		1	2		0.880	2.740	1.240
= 10 (D=0)			2	1	0.935	2.175	1.175
= 11	1	1	1	1	1.090	2.970	1.450
Instruction execution							
SCZ, SZCB, SOC	1				0.295	0.935	0.415
SOCB, MOV, MOVB	1				0.295	0.935	0.415

Table A-1. Format I Instruction Execution Times (Continued)

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction execution (Cont.)							
A, AB, S, SB	1				0.295	0.935	0.415
C, CB				2	0.510	0.510	0.510
Program counter increment				1	0.255	0.255	0.255

Example instruction	MOS Memory MOV 5,9	120 ns Memory AB *2,@GRAD(4)
Instruction acquisition	0.960	0.460
Source operand	0.820	0.780
Destination operand	0.960	1.240
Instruction execution	0.935	0.415
Program counter increment	0.255	0.255
Total time for example	3.930	3.150

Table A-2. Format II Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Instruction execution							
JUMP (All)				2	0.510	0.510	0.510
SBO, SBZ		1		3	0.965	1.585	1.085
TB		1		4	1.220	1.840	1.340

Example instructions	MOS Memory JMP $+8	120 ns Memory SBO 3
Instruction acquisition	0.960	0.460
Instruction execution	0.510	1.085
Total time for example	1.470	1.545

Table A-3. Format III Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Source operand acquisition							
T_s = 00		1			0.200	0.820	0.320
= 01		1	1		0.540	1.780	0.780
= 10 (S≠0)		2	1		0.740	2.600	1.100
= 10 (S=0)		2			0.400	1.640	0.640
= 11	1	1		2	0.750	2.010	0.990
Instruction execution							
COC, CZC			1	1	0.595	1.215	0.715
XOR	1	1		1	0.750	2.010	0.990
Program counter increment				1	0.255	0.255	0.255

Example instructions:	MOS Memory CZC *3,7	120 ns Memory XOR @RAD,9
Instruction acquisition	0.960	0.460
Source operand acquisition	1.780	0.640
Instruction execution	1.215	0.990
Program counter increment	0.255	0.255
Total time for example	4.210	2.345

Table A-4. Format IV Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Source operand acquisition							
T_s = 00		1			0.200	0.820	0.320
= 01		1	1		0.540	1.780	0.780
= 10 (S≠0)		2	1		0.740	2.600	1.100
= 10 (S=0)		1	1	1	0.795	2.035	1.035
= 11 (Byte or word)	1	2		1	0.950	2.830	1.310
Instruction execution							
LDCR (Setup)			1	2	0.850	1.470	0.970
LDCR (Output per bit)				1	0.255	0.255	0.255
STCR (Byte)	1		1	11	3.440	4.700	3.680
STCR (Byte, input per bit)				1	0.255	0.255	0.255
STCR (Word)	1		1	18	5.225	6.485	5.465
STCR (Word, input per bit)				1	0.255	0.255	0.255
Program counter increment				1	0.255	0.255	0.255

Table A-4. Format IV Instruction Execution Times (Continued)

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Example instructions:	MOS Memory LDCR *7, 8					120 ns Memory STCR @RAD(3), 0	
Instruction acquisition					0.960		0.460
Source operand acquisition					1.780		1.100
Instruction execution					1.470		5.465
Bit transfer					2.040		
Program counter increment					0.255		0.255
Total time for example					6.505		7.280

Table A-5. Format V Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Instruction execution							
Any shift (C≠0)	1	1		1/Bit	0.495+ 0.255/Bit	1.755+ 0.255/Bit	0.735+ 0.255/Bit
Any shift (C=0)	1	1	1	1/Bit	0.835+ 0.255/Bit	2.715+ 0.255/Bit	1.195+ 0.255/Bit
Program counter increment				1	0.255	0.255	0.255

Example instructions:	MOS Memory SRA 3,9	120 ns Memory SRL 5,0
Instruction acquisition	0.960	0.460
Instruction execution	1.755	1.195
Instruction bit count	9 = 2.295	16 = 4.080
Program counter increment	0.255	0.255
Total time for example	5.265	5.990

Table A-6. Format VI Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Source operand acquisition							
T_s = 00 (Notes 1, 2)							
= 00 (Note 3)			1		0.340	0.960	0.460
= 01 (Notes 1, 2)			1		0.340	0.960	0.460
= 01 (Note 3)			2		0.680	1.920	0.920
= 10 (S≠0) (Notes 1, 2)		1	1		0.540	1.780	0.780
= 10 (S≠0) (Note 3)		1	2		0.880	2.740	1.240
= 10 (S=0) (Notes 1, 2)			1	1	0.595	1.215	0.715
= 10 (S=0) (Note 3)			2	1	0.935	1.555	1.175
= 11 (Note 1)	1	1		1	0.750	2.010	0.990
= 11 (Note 2)	1			2	0.805	1.445	0.925
= 11 (Note 3)	1	1	1	1	1.090	2.970	1.450

Notes: For source operand acquisition, the following instructions are:

1. B, BL, CLR, SETO.
2. BLWP, X, SWPB.
3. NEG, INV, ABS, INC, INCT, DEC, DECT.
4. NEG, INV, ABS, X, DEC, DECT, INC, INCT only.

Table A-6. Format VI Instruction Execution Times (Continued)

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction execution							
B				1	0.255	0.255	0.255
CLR, SETO	1			1	0.550	1.190	0.670
BL	1			2	0.805	1.445	0.925
BLWP	3	2		1	1.540	4.700	2.140
X							
SWPB	1		1	1	0.890	2.150	1.130
NEG, INV	1				0.295	0.935	0.415
ABS (+)				2	0.255	0.255	0.255
ABS (-)	1			2	0.805	1.445	0.925
DEC, DECT, INC, INCT	1				0.295	0.935	0.415
Program counter increment (Note 4)				1	0.255	0.255	0.255

Note: For source operand acquisition, the following instruction is:

4. NEG, INV, ABS, X, DEC, DECT, INC, INCT only.

Table A-6. Format VI Instruction Execution Times (Continued)

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Example instructions:	MOS Memory B	@RAD			120 ns Memory NEG	3	
Instruction acquisition		0.960				0.460	
Source operand acquisition		1.215				0.460	
Instruction execution		0.255				0.415	
Program counter increment		0.000				0.255	
Total time for example		2.430				1.590	

Table A-7. Format VII Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Instruction execution							
IDLE							
RSET				3	0.765	0.765	0.756
RTWP		2	1	1	0.995	3.475	1.355
CKON, CKOF				1	0.255	0.255	0.255
LREX (Note)							
Program counter increment				1	0.255	0.255	0.255

Note: The execution time of the LREX instruction depends upon the amount of memory written by the ROM loader installed in the CPU. If the ROM loader writes 256 words of memory, the time is 241.765 μsec. If the ROM loader writes 4096 words of memory, the time is 3832.165 μ sec.

Example instruction executions:	MOS Memory RTWP	120 ns Memory CKON
Instruction acquisition	0.960	0.460
Instruction execution	3.475	0.255
Program counter increment	0.255	0.255
Total time for example	4.690	0.970

Table A-8. Format VIII Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Instruction execution							
LI	1	1		2	1.005	2.265	1.245
AI, ANDI, ORI	1	1	1	2	1.345	3.225	1.705
CI		1	1	1	0.795	2.035	1.035
STWP, STST	1			1	0.550	1.190	0.670
LWPI, LIMI		1		2	0.710	1.330	0.830
Program counter increment				1	0.255	0.255	0.255

Example instruction executions:	MOS Memory LI 3,>ADD	120 ns Memory LIMI > 256
Instruction acquisition	0.960	0.460
Instruction execution	2.265	0.830
Program counter increment	0.255	0.255
Total time for example	3.480	1.545

Table A-9. Format IX Instruction Execution Times

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
Instruction acquisition			1		0.340	0.960	0.460
Source operand acquisition							
XOP							
T_s = 00				1	0.255	0.255	0.255
= 01			1		0.340	0.960	0.460
= 10 (S≠0)		1	1		0.540	1.780	0.780
= 10 (S=0)			1	1	0.595	1.215	0.715
= 11	1	1		1	0.750	2.010	0.990
MPY, DIV							
T_s = 00		1			0.200	0.820	0.320
= 01		1	1		0.540	1.780	0.780
= 10 (S≠0)		2	1		0.740	2.600	1.100
= 10 (S=0)		2		1	0.655	1.895	0.895
= 11	1	1		2	1.005	2.265	1.245
Instruction execution							
XOP (Software only)	4	3	1	5	3.395	8.435	4.355

Table A-9. Format IX Instruction Execution Times (Continued)

Operation	Number and Type of Cycles W	R	DR	C	CPU Time	Total Time W/MOS Memory	Total Time W/120 ns Memory
MPY	2	1		33	9.165	11.025	9.525
DIV (Normal)	2	1	1	48	13.330	15.81	13.810
(Overflow)		1	1	3	1.305	2.545	1.545
Program counter increment				1	0.255	0.255	0.255

Example instruction execution:	MOS Memory XOP *4,2	120 ns Memory XOP *4,2
Instruction acquisition	0.960	0.460
Source operand acquisition	0.960	0.460
Instruction execution	8.435	4.355
Program counter increment	0.255	0.255
	10.610	5.530

APPENDIX B
HEXADECIMAL INSTRUCTION INDEX

APPENDIX B

HEXADECIMAL INSTRUCTION INDEX

Hexadecimal Operation Code	Mnemonic Operation Code	Name	Format	Paragraph
0200	LI	Load Immediate	VIII	4.6.1
0220	AI	Add Immediate	VIII	4.2.4
0240	ANDI	AND Immediate	VIII	4.7.1
0260	ORI	OR Immediate	VIII	4.7.4
0280	CI	Compare Immediate	VIII	4.4.3
02A0	STWP	Store Workspace Pointer	VIII	4.6.8
02C0	STST	Store Status	VIII	4.6.7
02E0	LWPI	Load Workspace Pointer Immediate	VIII	4.6.4
0300	LIMI	Load Interrupt Mask Immediate	VIII	4.6.2
0340	IDLE	Computer Idle	VII	4.5.4
0360	RSET	Computer Reset	VII	4.5.5
0380	RTWP	Return From Interrupt Subroutine	VII	4.3.17
03A0	CKON	Clock On	VII	4.5.2
03C0	CKOF	Clock Off	VII	4.5.1
03E0	LREX	Load ROM And Execute	VII	4.6.3
0400	BLWP	Branch And Load Workspace Pointer	VI	4.3.3
0440	B	Branch	VI	4.3.1
0480	X	Execute	VI	4.3.18
04C0	CLR	Clear Operand	VI	4.7.2
0500	NEG	Negate	VI	4.2.11
0540	INV	Invert	VI	4.7.3
0580	INC	Increment By One	VI	4.2.8
05C0	INCT	Increment By Two	VI	4.2.9

Hexadecimal Instruction Index (Continued)

Hexadecimal Operation Code	Mnemonic Operation Code	Name	Format	Paragraph
0600	DEC	Decrement By One	VI	4.2.5
0640	DECT	Decrement By Two	VI	4.2.6
0680	BL	Branch and Link	VI	4.3.2
06C0	SWPB	Swap Bytes	VI	4.6.9
0700	SETO	Set Ones	VI	4.7.5
0740	ABS	Absolute Value	VI	4.2.3
0800	SRA	Shift Right Arithmetic	V	4.8.1
0900	SRL	Shift Right Logical	V	4.8.2
0A00	SLA	Shift Left Arithmetic	V	4.8.3
0B00	SRC	Shift Right Circular	V	4.8.4
1000	JMP	Jump Unconditional	II	4.3.11
1100	JLT (A)	Jump Less Than	II	4.3.10
1200	JLE (L)	Jump Low Or Equal	II	4.3.9
1300	JEQ	Jump Equal	II	4.3.4
1400	JHE (L)	Jump High Or Equal	II	4.3.6
1500	JGT (A)	Jump Greater Than	II	4.3.5
1600	JNE	Jump Not Equal	II	4.3.13
1700	JNC	Jump No Carry	II	4.3.12
1800	JOC	Jump On Carry	II	4.3.16
1900	JNO	Jump No Overflow	II	4.3.14
1A00	JL (L)	Jump Low	II	4.3.8
1B00	JH (L)	Jump High	II	4.3.7
1C00	JOP	Jump Odd Parity	II	4.3.15
1D00	SBO	Set Bit To One	II	4.5.6
1E00	SBZ	Set Bit To Zero	II	4.5.7
1F00	TB	Test Bit	II	4.5.9
2000	COC	Compare Ones Corresponding	III	4.4.4

Hexadecimal Instruction Index (Continued)

Hexadecimal Operation Code	Mnemonic Operation Code	Name	Format	Paragraph
2400	CZC	Compare Zeros Corresponding	III	4.4.5
2800	XOR	Exclusive OR	III	4.7.10
2C00	XOP	Extended Operation	IX	4.9
3000	LDCR	Load Communication Register	IV	4.5.3
3400	STCR	Store Communication Register	IV	4.5.8
3800	MPY	Multiply	IX	4.2.10
3C00	DIV	Divide	IX	4.2.7
4000	SZC	Set Zeros Corresponding, Word	I	4.7.8
5000	SZCB	Set Zeros Corresponding, Byte	I	4.7.9
6000	S	Subtract Word	I	4.2.12
7000	SB	Subtract Byte	I	4.2.13
8000	C	Compare Words	I	4.4.1
9000	CB	Compare Bytes	I	4.4.2
A000	A	Add Words	I	4.2.1
B000	AB	Add Bytes	I	4.2.2
C000	MOV	Move Word	I	4.6.5
D000	MOVB	Move Byte	I	4.6.6
E000	SOC	Set Ones Corresponding, Word	I	4.7.6
F000	SOCB	Set Ones Corresponding, Byte	I	4.7.7

APPENDIX C
ALPHABETICAL INSTRUCTION INDEX

APPENDIX C

ALPHABETICAL INSTRUCTION INDEX

Mnemonic Operation Code	Hexadecimal Operation Code	Name	Format	Paragraph
A	A000	Add Words	I	4.2.1
AB	B000	Add Bytes	I	4.2.2
ABS	0740	Absolute Value	VI	4.2.3
AI	0220	Add Immediate	VIII	4.2.4
ANDI	0240	AND Immediate	VIII	4.7.1
B	0440	Branch	VI	4.3.1
BL	0680	Branch and Link	VI	4.3.2
BLWP	0400	Branch and Load Workspace Pointer	VI	4.3.3
C	8000	Compare Words	I	4.4.1
CB	9000	Compare Bytes	I	4.4.2
CI	0280	Compare Immediate	VIII	4.4.3
CKOF	03C0	Clock Off	VII	4.5.1
CKON	03A0	Clock On	VII	4.5.2
CLR	04C0	Clear Operand	VI	4.7.2
COC	2000	Compare Ones Corresponding	III	4.4.4
CZC	2400	Compare Zeros Corresponding	III	4.4.5
DEC	0600	Decrement By One	VI	4.2.5
DECT	0640	Decrement By Two	VI	4.2.6
DIV	3C00	Divide	IX	4.2.7
IDLE	0340	Computer Idle	VII	4.5.4
INC	0580	Increment By One	VI	4.2.8
INCT	05C0	Increment By Two	VI	4.2.9
INV	0540	Invert	VI	4.7.3

Alphabetical Instruction Index (Continued)

Mnemonic Operation Code	Hexadecimal Operation Code	Name	Format	Paragraph
JEQ	1300	Jump Equal	II	4.3.4
JGT	1500	Jump Greater Than	II	4.3.5
JH	1B00	Jump High	II	4.3.7
JHE	1400	Jump High Or Equal	II	4.3.6
JL	1A00	Jump Low	II	4.3.8
JLE	1200	Jump Low Or Equal	II	4.3.9
JLT	1100	Jump Less Than	II	4.3.10
JMP	1000	Jump Unconditional	II	4.3.11
JNC	1700	Jump No Carry	II	4.3.12
JNE	1600	Jump Not Equal	II	4.3.13
JNO	1900	Jump No Overflow	II	4.3.14
JOC	1800	Jump On Carry	II	4.3.16
JOP	1C00	Jump Odd Parity	II	4.3.15
LDCR	3000	Load Communication Register	IV	4.5.3
LI	0200	Load Immediate	VIII	4.6.1
LIMI	0300	Load Interrupt Mask Immediate	VIII	4.6.2
LREX	03E0	Load ROM And Execute	VII	4.6.3
LWPI	02E0	Load Workspace Pointer Immediate	VIII	4.6.4
MOV	C000	Move Word	I	4.6.5
MOVB	D000	Move Byte	I	4.6.6
MPY	3800	Multiply	IX	4.2.10
NEG	0500	Negate	VI	4.2.11
ORI	0260	OR Immediate	VIII	4.7.4
RSET	0360	Computer Reset	VII	4.5.5
RTWP	0380	Return From Interrupt Subroutine	VII	4.3.17

Alphabetical Instruction Index (Continued)

Mnemonic Operation Code	Hexadecimal Operation Code	Name	Format	Paragraph
S	6000	Subtract Word	I	4.2.12
SB	7000	Subtract Byte	I	4.2.13
SBO	1D00	Set Bit To One	II	4.5.6
SBZ	1E00	Set Bit To Zero	II	4.5.7
SETO	0700	Set Ones	VI	4.7.5
SLA	0A00	Shift Left Arithmetic	V	4.8.3
SOC	E000	Set Ones Corresponding, Word	I	4.7.6
SOCB	F000	Set Ones Corresponding, Byte	I	4.7.7
SRA	0800	Shift Right Arithmetic	V	4.8.1
SRC	0B00	Shift Right Circular	V	4.8.4
SRL	0900	Shift Right Logical	V	4.8.2
STCR	3400	Store Communication Register	IV	4.5.8
STST	02C0	Store Status	VIII	4.6.7
STWP	02A0	Store Workspace Pointer	VIII	4.6.8
SWPB	06C0	Swap Bytes	VI	4.6.9
SZC	4000	Set Zeros Corresponding, Word	I	4.7.8
SZCB	5000	Set Zeros Corresponding, Byte	I	4.7.9
TB	1F00	Test Bit	II	4.5.9
X	0480	Execute	VI	4.3.18
XOP	2C00	Extended Operation	IX	4.9
XOR	2800	Exclusive OR	III	4.7.10

APPENDIX D
CRU INTERFACE EXAMPLE

APPENDIX D

CRU INTERFACE EXAMPLE

D.1 GENERAL

This appendix supplies information to aid users of the Model 990 Computer to design and implement peripheral controllers that interface with the CRU. Included in this appendix is a description of the interface unit and sample logic diagrams for the CRU interface unit, followed by a sample program for a typical device. Refer to the main body of this manual for a description of the CRU and its general requirements. Specifically, this appendix contains a description of the necessary requirements to interface a medium-speed line printer to the CRU. The line printer has the characteristics listed in table D-1.

Table D-1. Medium-Speed Line Printer Characteristics

Function	Description
Print line length	80 characters maximum
Paper width	Variable, up to 9-1/2 inches, sprocket fed
Character format	5x7 dot matrix, 10 characters per inch (horiz) 6 lines per inch (vertical)
Printer speed	60 lines per minute for 80 character lines or 150 lines per minute for 20 character lines
Printer input buffer	80 characters
Buffer data rate	75,000 characters per second (8-bit characters supplied in parallel) maximum

D.2 HARDWARE INTERFACE REQUIREMENTS

To interface with the CRU and the line printer, the interface device must be able to receive serial data from the CRU and convert the data to parallel lines for interconnections to the printer. In addition to the data lines, the interface device must supply the necessary control signals to actuate the printer mechanisms when required for the printing format. The control signals are shown in table D-2.

D.2.1 INTERFACE TIMING

Refer to figure D-1 for the following discussion. The interface device must enable the data on the lines to the printer prior to any other operation. Once

Table D-2. Printer Control and Response Signals

Signal	Definition	Hexadecimal Value
Control Characters		
LF	Line Feed	$0A_{16}$
CR	Carriage Return	$0D_{16}$
TOF	Top of Form	$0E_{16}$
PS	Printer Strobe	11_{16}
PP	Printer Prime	FF_{16}
PD	Printer De-select	13_{16}
Discrete Signals		
PL	Paper Low ←	
PSD	Printer Selected ←	
PF	Printer Fault ←	
BSY	Printer Busy ←	
IM	Interrupt Mask →	
IR	Interrupt Reset →	
ACK	Acknowledge ←	

*←—Signal from printer
—→Signal to printer

the data has been applied to the printer, the interface device then applies the data strobe to the printer. This strobe may remain on the line for a period of from 0.5 μsec to 500 μsec. After the strobe is removed from the line, the printer must respond with the acknowledge signal from 2.5 μsec to 10 μsec later. This acknowledge signal must remain on the line from 2.5 μsec to 5.0 μsec. After the acknowledge signal is removed from the line by the printer, the overall process may then be repeated.

D.2.2 INTERFACE MODULE HARDWARE

The user may design and implement a special interface card that will connect the line printer to the 990 Computer CRU. However, for purposes of this example, the 16 I/O data module (Section III) is used as the interface device.

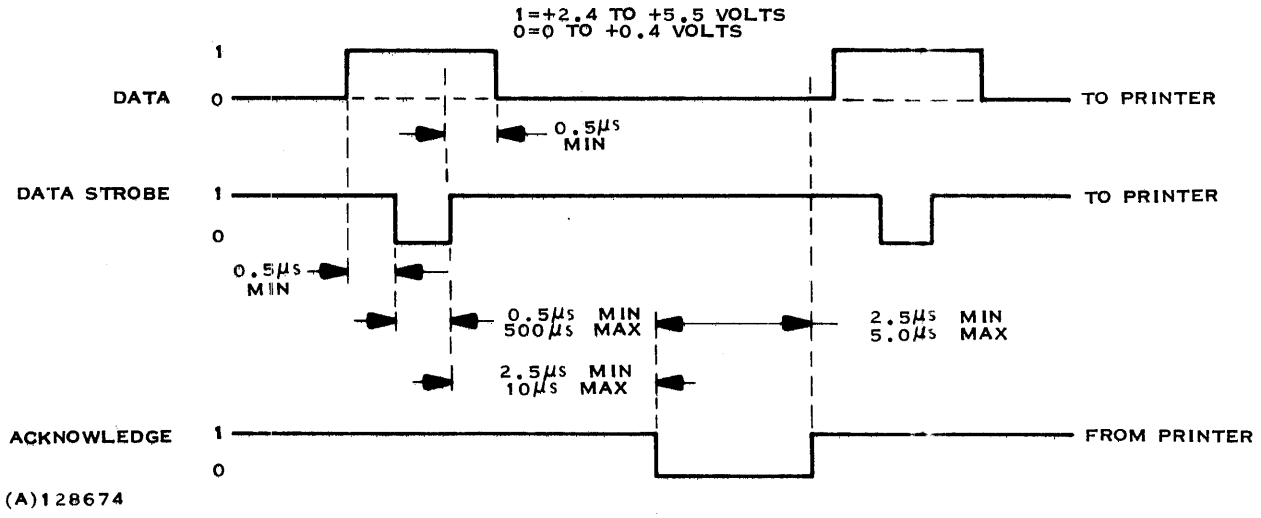


Figure D-1. Interface Timing for Data Transfer

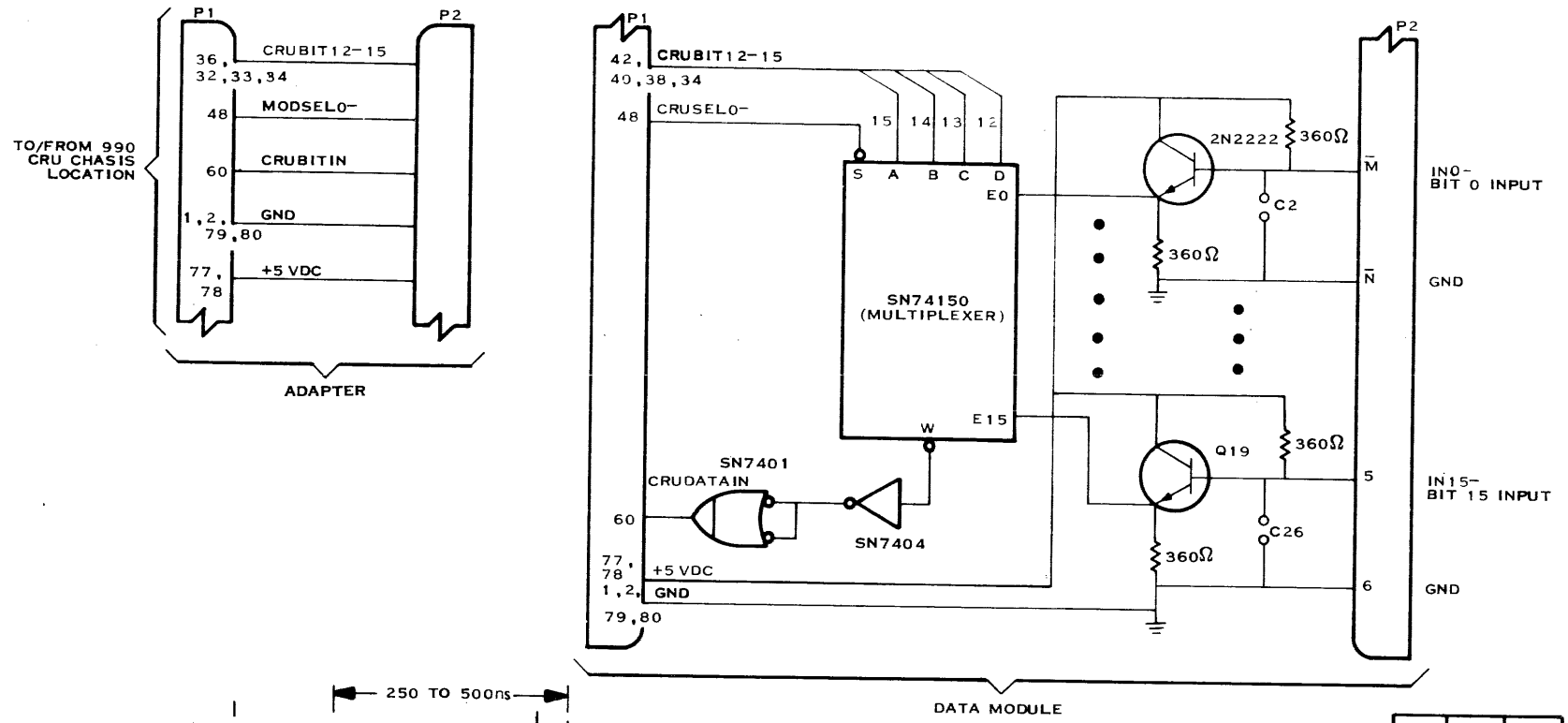
D.2.2.1 INPUT CIRCUITS. A logic diagram of the input circuits on the 16 I/O data module is shown in figure D-2. Input signals on the bit input lines are selected by the bit address applied to the Z3 multiplexer. Single or multiple input lines may be selected under program control.

D.2.2.2 OUTPUT CIRCUITS. A logic diagram of the output circuits is shown in figure D-3. Output signals on the output lines remain on these lines until changed by the CRU signals. Decode logic (Z13) permits the addressing of one or more of these output lines according to program requirements. Note that it is possible for the output circuits to be loaded into an external power source through external load resistors.

D.2.2.3 INTERRUPT CIRCUITS. A logic diagram of the interrupt circuitry is shown in figure D-4. Note that the interrupt signal may be wired to interrupt on either the positive or negative transition of the interrupt signal.

D.2.2.4 PRINTER LOGIC CIRCUITS. The printer contains input logic to decode the input control signals to operate the appropriate printer mechanism.

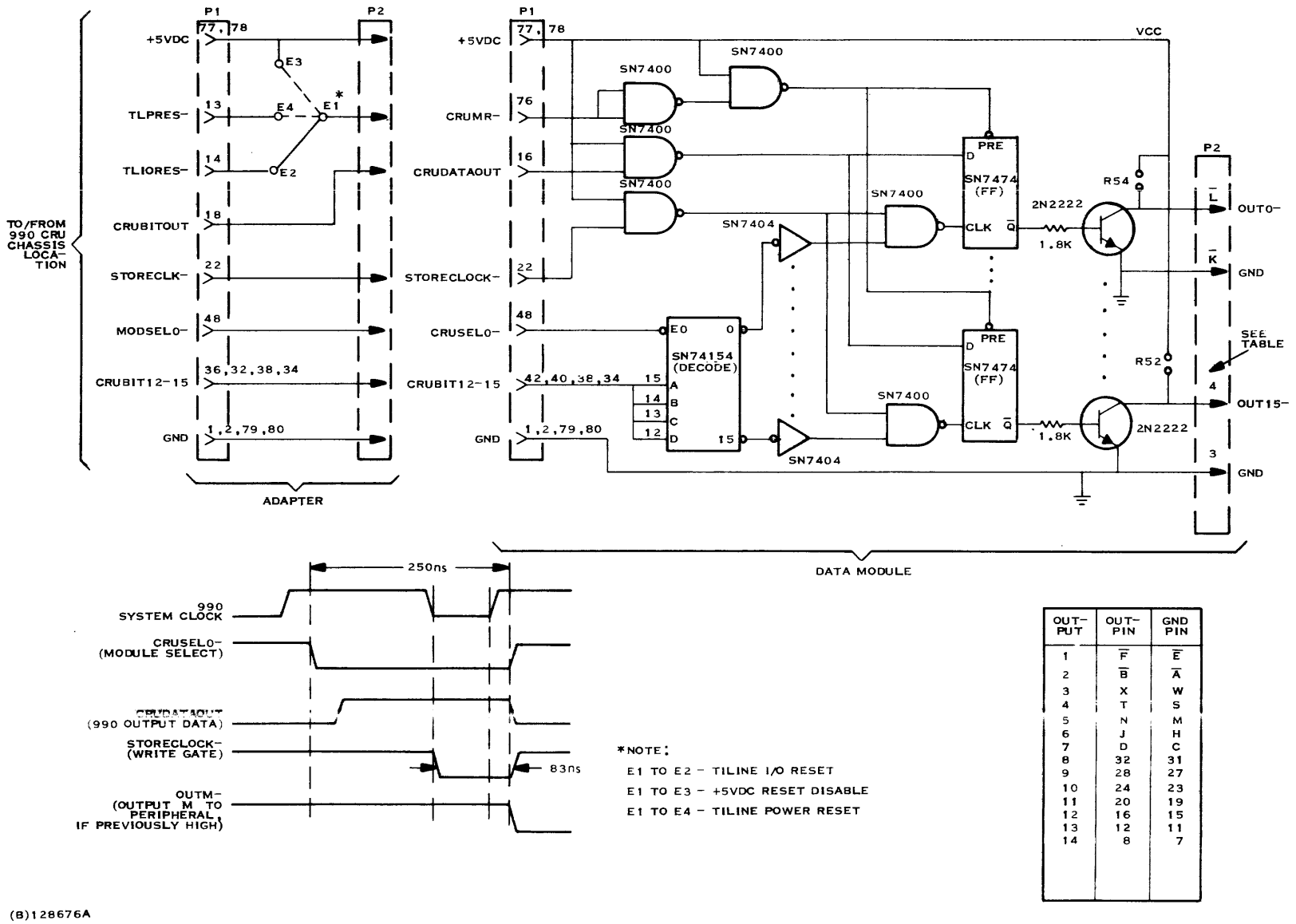
D.2.2.5 PRINTER/16 I/O DATA MODULE INTERCONNECTIONS. A cable may easily be constructed that will interconnect the 16 I/O data module and the printer. Specific printer connections may be found in printer documentation.



INPUT	IN- PIN	GND PIN
1	H	J
2	C	D
3	Y	Z
4	U	V
5	P	R
6	K	L
7	E	F
8	33	34
9	29	30
10	25	26
11	21	22
12	17	18
13	13	14
14	9	10

*CASE OF INPUT = HIGH (LOGIC 0)

Figure D-2. Data Module Input Logic and Timing



*NOTE:
E1 TO E2 - TILINE I/O RESET
E1 TO E3 - +5VDC RESET DISABLE
E1 TO E4 - TILINE POWER RESET

OUT-PUT	OUT-PIN	GND PIN
1	$\overline{F}$	$\overline{E}$
2	$\overline{B}$	$\overline{A}$
3	X	W
4	T	S
5	N	M
6	J	H
7	D	C
8	32	31
9	28	27
10	24	23
11	20	19
12	16	15
13	12	11
14	8	7

Figure D-3. Data Module Output Logic and Timing

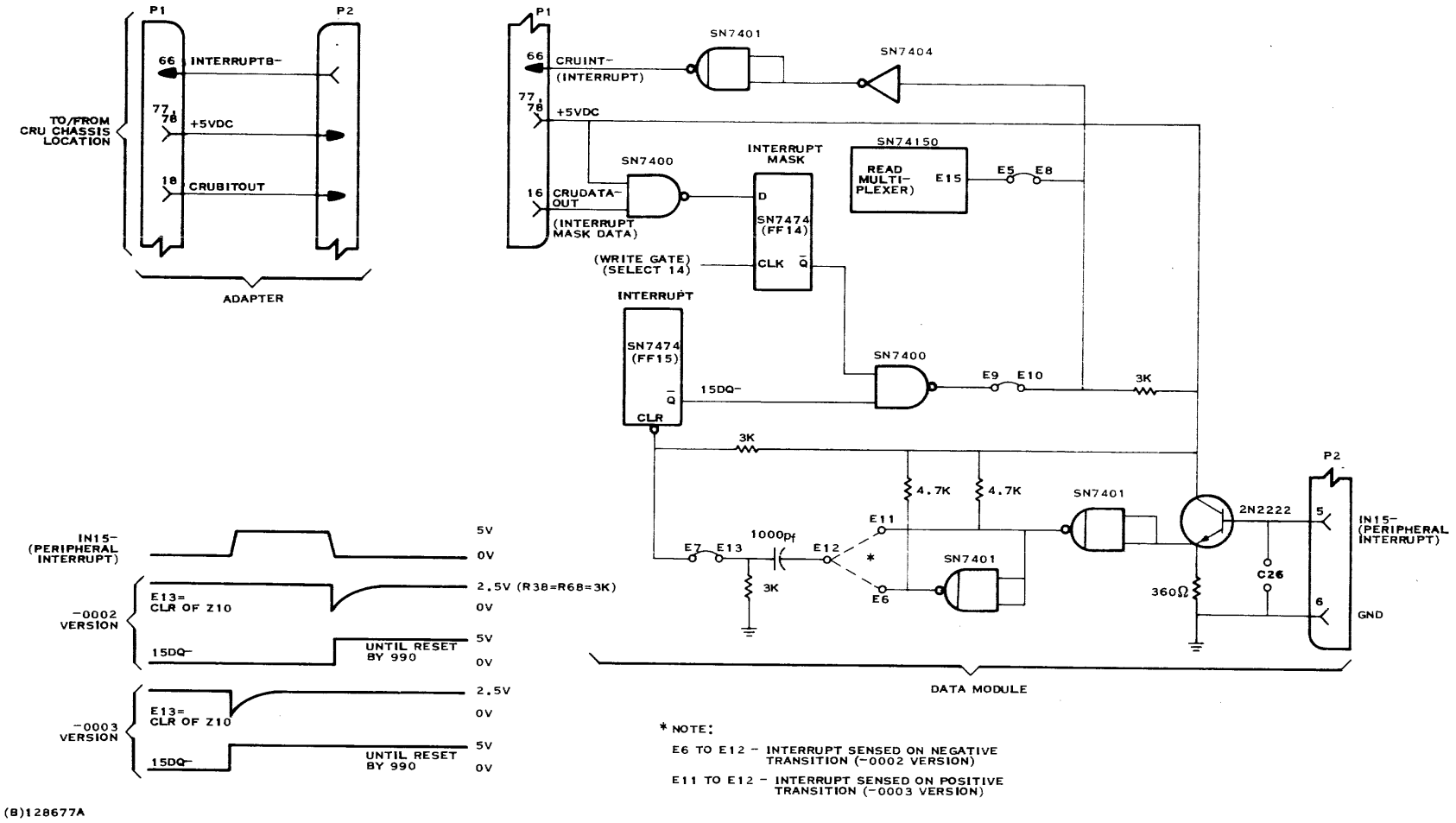


Figure D-4. Data Module Interrupt Logic and Timing

D.3 SOFTWARE INTERFACE REQUIREMENTS

An arbitrary CRU signal arrangement has been selected for this example, as shown in figure D-5.

D.3.1 ASSEMBLY LANGUAGE INSTRUCTIONS

The available assembly language instructions that may be used to cause data transfers between the CRU and the printer are:

- SBO - Set Bit to Logic One
- SBZ - Set Bit to Logic Zero
- LDCR - Load Communications Register
- TB - Test Bit

Refer to Section IV for detailed descriptions of these instructions and Section V for examples of usage of these instructions.

D.3.2 SOFTWARE ROUTINES REQUIRED

To properly operate the medium-speed line printer, software routines must provide initialization, character transfer, and end-of-data reporting. The following paragraphs define these operations and provide specific programming examples.

CRU BITS	CRU OUTPUTS	PRINTER OUTPUTS (CRU INPUTS)
0	DATA BIT 0 (LSB)	BUSY
1	DATA BIT 1	FAULT
2	DATA BIT 2	SELECTED
3	DATA BIT 3	NOT USED
4	DATA BIT 4	NOT USED
5	DATA BIT 5	NOT USED
6	DATA BIT 6	NOT USED
7	DATA BIT 7 (MSB)	NOT USED
8	STROBE	NOT USED
9	PRIME	NOT USED
10	NOT USED	NOT USED
11	NOT USED	NOT USED
12	NOT USED	NOT USED
13	NOT USED	NOT USED
14	INTERRUPT MASK	NOT USED
15	INTERRUPT RESET	ACKNOWLEDGE

(A)128706

Figure D-5. CRU Bit Assignments

D.3.2.1 INITIALIZATION. Initialization should occur when power is applied to the system. A generalized approach to initialization with specific printer initialization follows:

```
        AORG    0
        DATA    PWRONW      INITIALIZE POWER ON
        DATA    PWRONP          INTERRUPT VECTOR
          .
          .                 OTHER VECTORS
          .
PWRONP  EQU     $           POWER ON INITIALIZATION
          .
          .                 OTHER INITIALIZATIONS
          .

PRBASE  EQU  >120           USE MODULE SELECT 9
PRIME   EQU  9              OUTPUT - RESET PRINTER
STROBE  EQU  8              OUTPUT - TAKE DATA
MASK    EQU  14             OUTPUT - INTERRUPT MASK
*
*PRINTER INITIALIZATION
*
        LI   12,PRBASE      INITIALIZE CRU BASE ADDRESS
        SBZ  PRIME          PRIME = +5V
        SBZ  STROBE         STROBE = +5V
        SBZ  MASK           INTERRUPT IS MASKED
          .
          .
          .
```

D.3.2.2 CHARACTER TRANSFER. Character transfer can be accomplished as follows by the use of a subroutine call. The assumptions for this code are:

- Workspace register 8 (WR8) contains the address of the data to be printed.
- Workspace register 9 (WR9) is used for temporary storage.

- Workspace register 10 (WR10) contains the number of characters to transfer.
- Workspace register 12 (WR12) contains the CRU base address.

The following code is one method to provide character transfers.

```
PRINTR    EQU   $        PRINT SUBROUTINE
*
* SET UP INTERRUPTS
*
          SBZ   INT      RESET INTERRUPT
          LIMI  15       ENABLE LEVEL 15
          SBO   MASK     UNMASK INTERRUPT
*
* TEST FOR PRINTER BUSY, PRINT IF
* NOT BUSY, WAIT FOR ANY INTERRUPT[1]
* IF BUSY AND RETRY TEST.
*
TSTBSY    TB    BUSY     SAMPLE BUSY BIT
          JEQ   PRINT    IF NOT BUSY
          IDLE           WAIT IF BUSY
          JMP   TSTBSY   RETRY TEST
*
* CHARACTER PRINT SUBROUTINE
*
PRINT     EQU   $        START
          MOVB  *8+,9    WR9 CONTAINS PRINT CHAR
          INV   9        INVERT BITS (FALSE DATA)
          LDCR  9,8      OUTPUT TO PRINTER
          SBO   STROBE   PULSE STROBE LINE
```

Note 1. Refer to the discussion on the interrupt routine.

```
                SBZ     STROBE   ABOUT 1.5 MICROSECONDS
                DEC     10       DECREMENT CHARACTER COUNT
                JEQ     EXIT     EXIT IF COMPLETE
                JMP     TSTBSY   GO FOR NEXT CHARACTER
*
* EXIT CODE
*
EXIT            SBZ     MASK     MASK INTERRUPT
                RTWP             RETURN TO CALLER
                .
                .
                .
INT             EQU     15       OUTPUT - INTERRUPT RESET
BUSY            EQU      0       INPUT - PRINTER BUSY
```

D.3.2.3 END-OF-DATA REPORTING. End-of-data reporting in the example code for character transfer is determined by the input character count in WR8. When this count is depleted, the required amount of character data has been sent to the printer. The end-of-data reporting is accomplished by returning to the calling program.

D.3.2.4 INTERRUPT ROUTINE. During the printer busy test in the character transfer routine, if the printer is busy, the CPU enters an idle state. The only possible exit from this state is via any enabled interrupt. For this example, the LIMI 15 instruction enables the interrupt level for the printer. When this interrupt occurs, the following interrupt routine may be used to reset the interrupt and cause the JMP BUSY instruction to be executed.

```
                AORG    >3C      INTERRUPT LEVEL 15
                DATA    PRIWP    WORKSPACE ADDRESS
                DATA    PRIPC    PROGRAM ADDRESS
                .
                .
                .
PRIWP           RORG    $-24     WORKSPACE POINTER
                DATA    PRBASE   CRU BASE ADDRESS
                RORG    $+6
```

```
PRIPC   EQU   $      INTERRUPT ROUTINE
        SBZ   INT    RESET INTERRUPT
        RTWP         RETURN
```

D.3.3 PROGRAMMING NOTES

Sophisticated techniques for operation of the printer may be designed and implemented. Examples presented here are fundamental to the operation of the printer and do not include error routines for printer faults or printer status changes. Error recovery routines may be used when desired to overcome printer error conditions.

APPENDIX E
TILINE INTERFACE EXAMPLE

APPENDIX E

TILINE INTERFACE EXAMPLE

E.1 INTRODUCTION

This appendix supplies information to aid users of the Model 990 Computer to design and implement peripheral controllers that interface with the TILINE data bus. Included in this appendix are description and sample logic diagrams for both a master and a slave interface, followed by a sample program and description of a typical service routine for a TILINE device. Refer to the main body of this manual for description of the interface and its general requirements.

E.2 TILINE SAMPLE INTERFACES

Peripheral device controllers used with the TILINE must perform both as a master and as a slave device. The slave part of the device controller receives or sends data at the instruction of the AU or another master device. The AU may use this part of the controller to set up a controller-to-controller transfer, in which one of the controllers becomes the master, to initiate a transfer to the other controller. The master part of the controller can transfer data to or from its corresponding device from or to memory (or some other slave interface). This appendix contains a sample logic diagram and description of both a master and a slave interface to be used as a guide in designing new device controllers. The master interface is standardized, and should be implemented as described in this appendix to ensure compatibility. The slave interface may be varied to match the criterion of the slave device, but may not deviate from the general requirements for a slave interface (signals required, time delays, etc.) outlined in this appendix and in the main body of the manual.

E.2.1 MASTER INTERFACE

The master interface for a peripheral device controller must be compatible with the TILINE requirements as outlined in the main portion of this manual. Use of the standard implementation for the master interface ensures compatibility and simplifies design of peripheral controllers. The logic within the interface is completely asynchronous and consists of three major functional areas: TILINE access control, data access and system clock. Figure E-1 illustrates the interrelationship of these three areas and the signals that tie them together. Table E-1 defines those signals that are not a part of the TILINE interface. TILINE signals are defined in the main body of the manual. The controller-to-interface signals require addition of a top-edge connector if the controller is implemented on a separate circuit board from the interface.

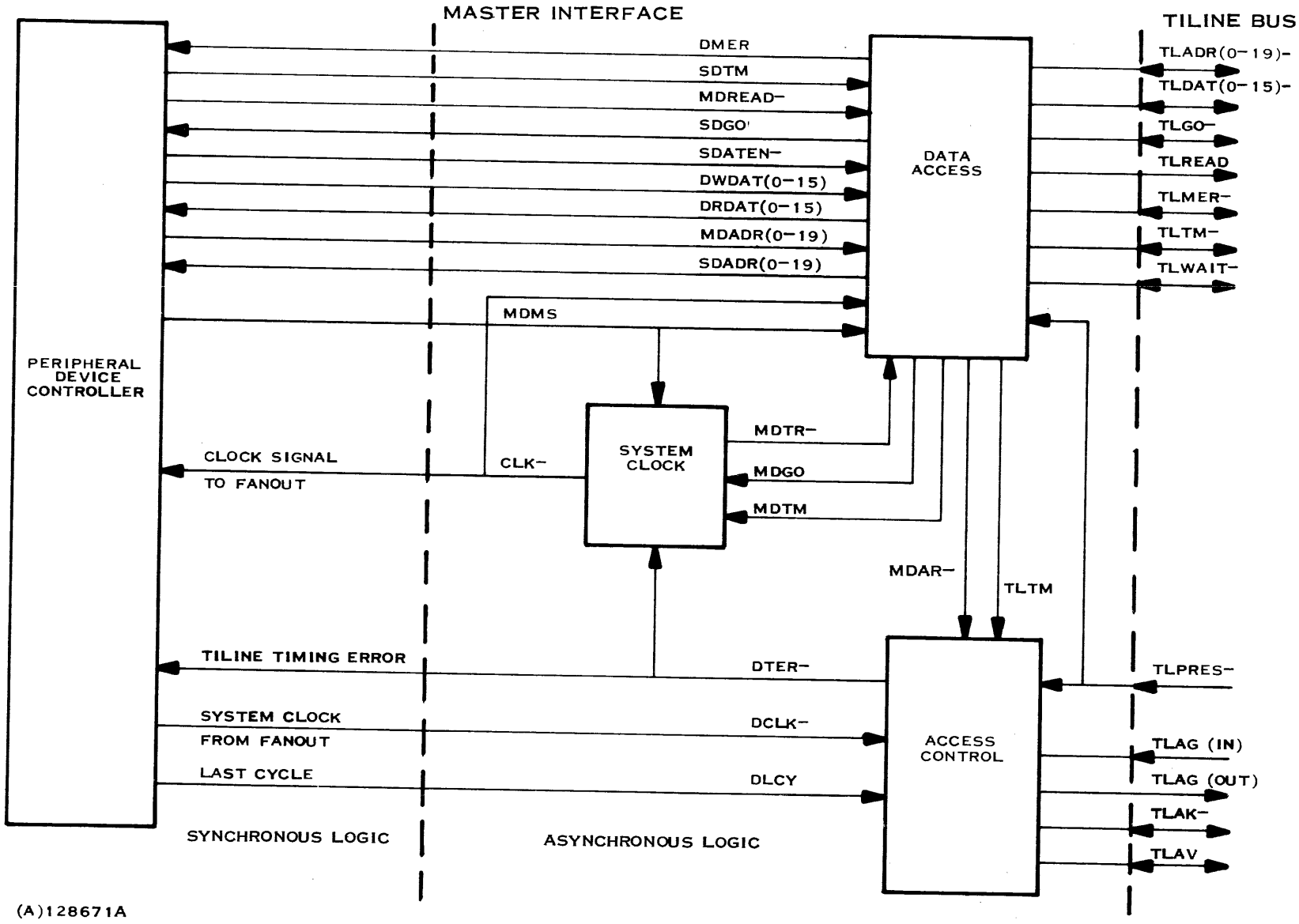


Figure E-1. Master Interface Signals

Table E-1. Peripheral Device Controller - Master Interface Signal

Signature	Definition
DMER	Memory Error: When high*, this signal indicates that a non-recoverable error occurred during a memory read cycle.
SDTM	Slave Device Terminate: When high, this signal indicates the completion of a slave data transfer.
MDREAD-	Device Read: When high, this signal indicates a write operation (data transfer from device to TILINE); when low, this signal indicates a read operation (data from TILINE to device).
SDGO	Slave Device Go: Initiates a data transfer from the slave portion of the device controller.
SDATEN-	Slave Data Enable: When low, this signal enables data transfer from the slave device to the TILINE.
DWDAT(0-15)	Write data from device to TILINE.
DRDAT(0-15)	Read data from TILINE to device.
MDADR(0-19)	Address lines from master portion of device to TILINE.
SDADR(0-19)	Address lines from TILINE to slave portion of device.
MDMS	Master Device Memory State: When high, this signal indicates to the master interface that the device controller requires a memory cycle in its current control state.
CLK-	Device System Clock: Asynchronous pulse for data and state coordination.
DTER-	Timeout Error: An asynchronous 50 nanosecond pulse that indicates that the device has failed to transfer data or has addressed a nonexistent device.
DCLK-	System clock pulse from fanout circuits.
DLCY	Device Last Cycle: When high, this signal indicates that the current memory cycle is the last cycle in a burst.
MDTR-	Device Timer Running: Indicates that the timer circuit is in operation.
MDAR-	Master Device Access Request: When low, this signal initiates an access request to the TILINE bus priority circuits.

*For all signals: High $\geq$ 2.4V
Low $\leq$ 0.4V

Figure E-2 provides a detailed logic diagram of the recommended master interface. This interface conforms to the master interface description contained in the main body of the manual. In addition, the master interface contains a clock circuit to synchronize operation of the controller with the response time of the selected slave device.

E.2.1.1 SYSTEM CLOCK. The system clock circuit provides an asynchronous clock signal for use in data transfer through the interface and within the peripheral controller. When the interface is not performing a memory transfer operation, the clock runs synchronously with a period of 260 nanoseconds. However, when the interface is performing a memory transfer, the clock becomes asynchronous and depends upon receipt of memory complete indications (MDTM). The period can never be less than 260 nanoseconds, however. Figure E-3 illustrates the timing relationship of the signals within the system clock circuit. Delay T1 in the circuit determines the pulse width of the clock pulse; delay T2 determines the timing of MDGO following the clock pulse. T1 plus T2 plus T3 determines the overall clock period.

E.2.2 SLAVE INTERFACE

Slave devices respond to assigned memory addresses that are within the highest 1K addresses of the 20-bit TILINE address bus. The address for a particular slave interface is normally set using dual inline package rocker switches, so that the particular address of a board may be changed easily. Although the actual configuration of a slave device interface will vary due to the nature of the device, the interface must conform to the timing and signal criteria prescribed in the description of TILINE interfaces in the main body of the manual. As an aid to slave interface design, figure E-4 illustrates the slave interface used by a 16-bit data register.

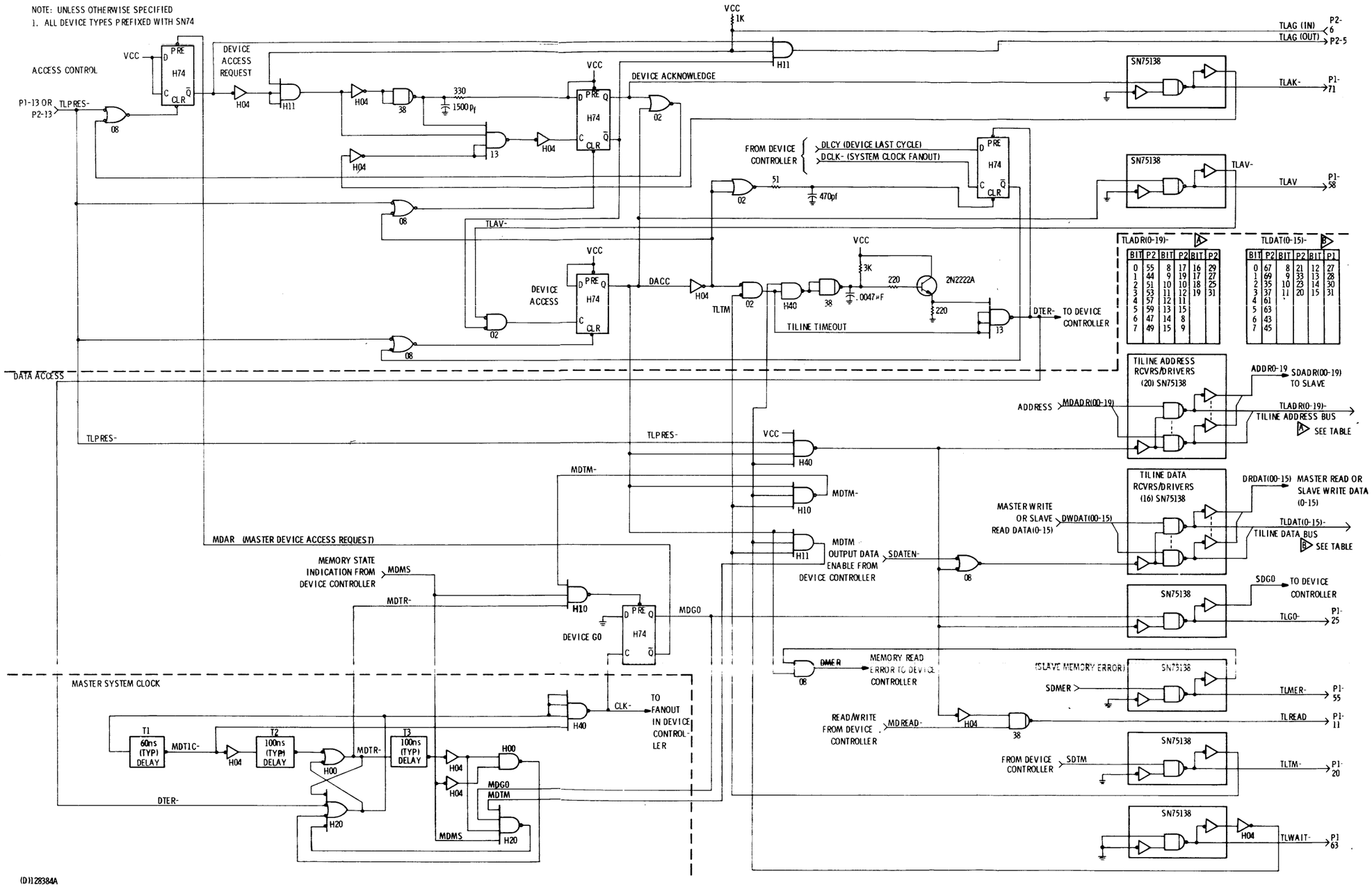


Figure E-2. TILINE Master Interface

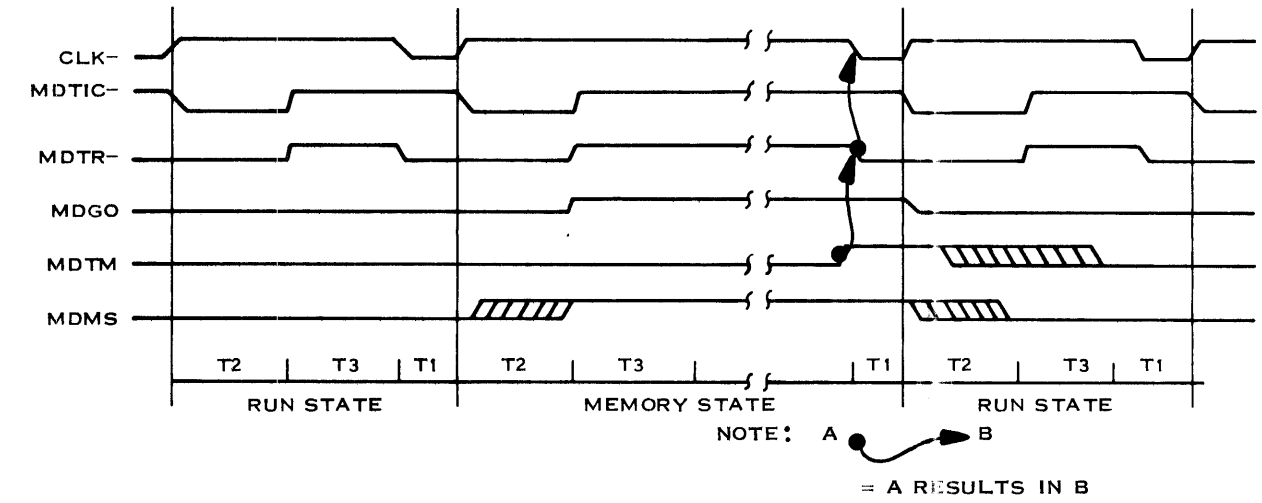


Figure E-3. System Clock Timing

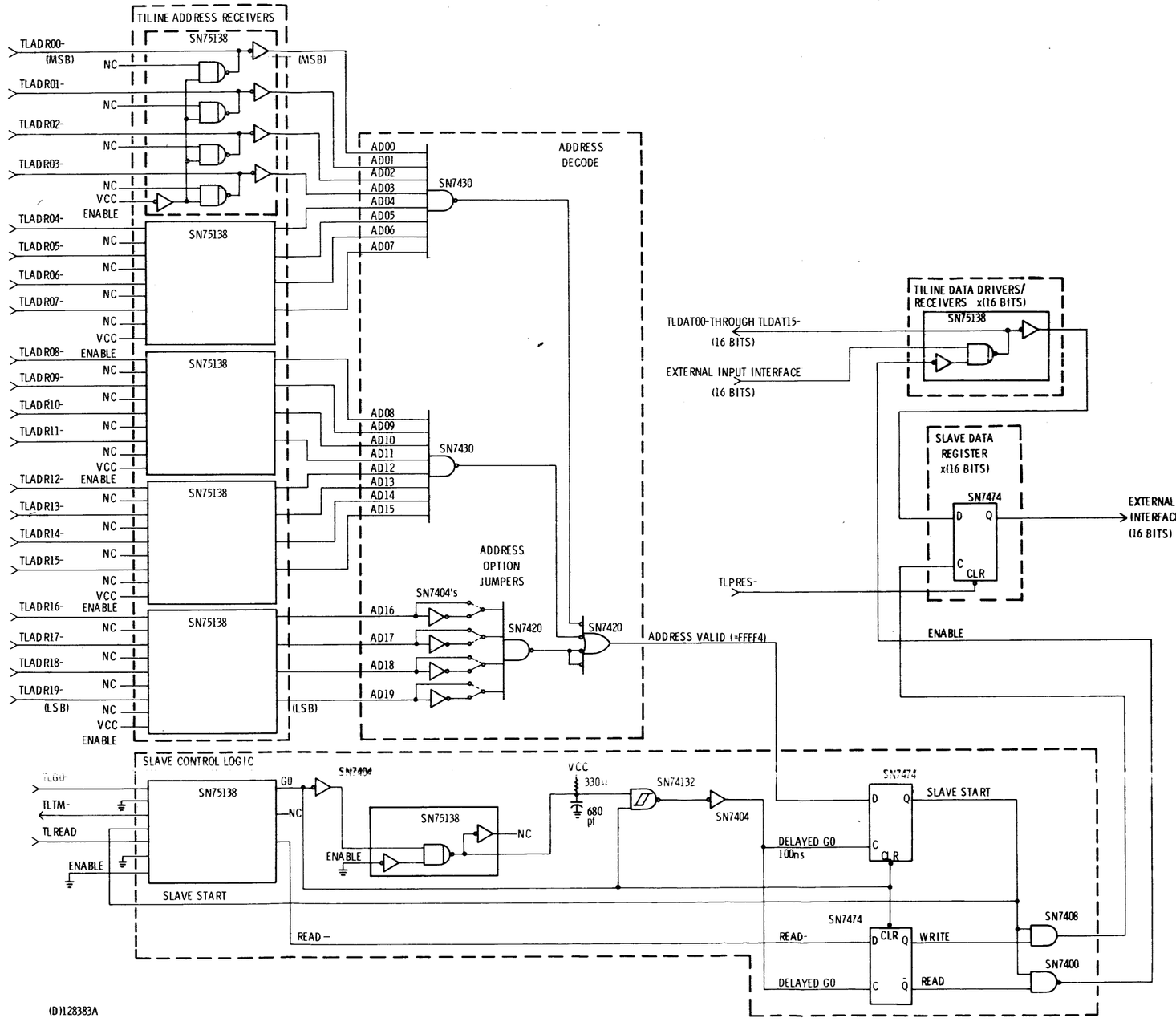


Figure E-4. TILINE Slave Interface

E.3 PERIPHERAL CONTROLLER APPLICATION

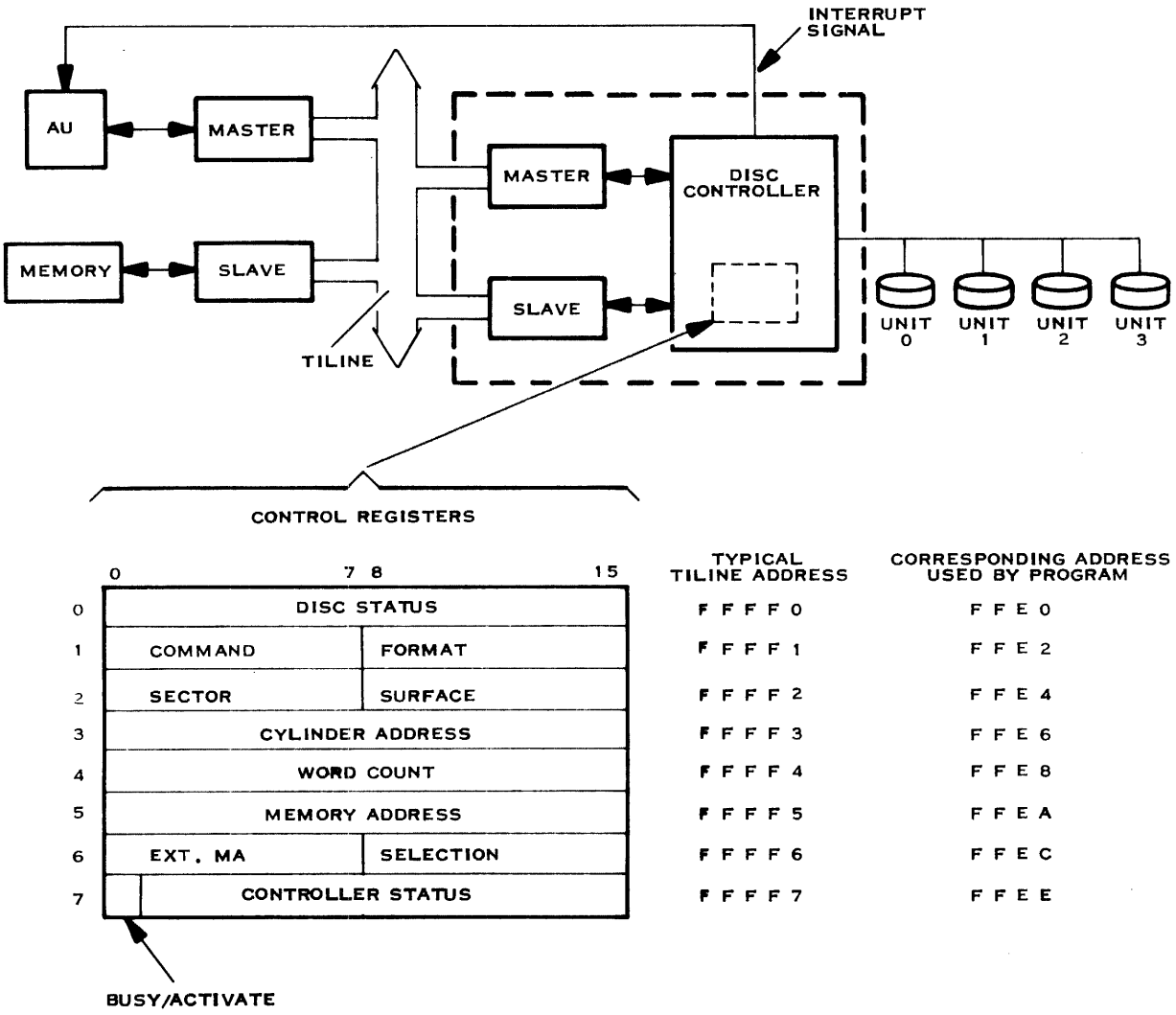
Controllers for peripheral devices connected to the TILINE use the master interface to fetch or deposit data in the system memory. The slave interface is used to receive commands and to make status data available. A simplified block diagram for a disc controller is shown in figure E-5. Typical use of control registers accessed via the slave interface is also shown in figure E-5. In a system, the programmed address $FFE0_{16}$ might be assigned to the controller. This corresponds to TILINE address $FFFF0_{16}$. A program would operate the disc controller by moving the appropriate parameters into the control registers and selecting the activation bit as follows:

```
*
* TEST FOR DISC CONTROLLER BUSY
*
        LI      7,>7FFF     WR7=BUSY TEST MASK
        COC     @>FFEE,7    TEST FOR BUSY
        JNE     BUSY        IF NOT

*
* TRANSFER DISC PARAMETER LIST FROM MEMORY TO
* DISC CONTROLLER
*

        LI      8,PARAMS    WR8=PARAMETER LIST ADDRESS
        LI      9,>FFE2     WR9=DISC CONTROLLER ADDRESS
FILL    MOV     *8+,*9+     MOVE PARAMETER LIST TO CONTROLLER
        CI      9, FFEE     STOP WHEN WR9=CONT. STATUS ADDR
        JNE     FILL        *
        INV     7           WR7=>8000
        SOC     7,*8        SET ACTIVATE BIT
```

The disc controller will perform the action requested in the "COMMAND" register. During the time the controller is active, the busy flag will be on. When the operation is complete the busy flag will be turned off and an interrupt signal will be generated. The interrupt can be connected via chassis backpanel wiring to an arithmetic unit external interrupt. The choice of interrupt used is left to the system designer.



(A)128673A

Figure E-5. Sample Disc Peripheral Controller Block Diagram

APPENDIX F
CHARACTER SET

APPENDIX F

CHARACTER SET

The Model 990 Assembly Language Uses the USASCII characters listed in table F-1. The table includes the USASCII code for each character, represented as a hexadecimal value and as a decimal value. The table also shows the corresponding Hollerith code. In addition to the characters listed in table F-1, Model 990 Assembly Language defines six characters that are undefined in USASCII. Table F-2 lists these characters, hexadecimal and decimal representations, corresponding Hollerith codes, and the corresponding character on the Model 29 keypunch.

Table F-1. Character Set

USASCII Hexadecimal Value	Decimal Value	Printable Character	Hollerith Code
20	32	Space	Blank
21	33	!	11-8-2
22	34	"	8-7
23	35	#	8-3
24	36	$	11-8-3
25	37	%	0-8-4
26	38	&	12
27	39	'	8-5
28	40	(	12-8-5
29	41	)	11-8-5
2A	42	*	11-8-4
2B	43	+	12-8-6
2C	44	,	0-8-3
2D	45	-	11
2E	46	.	12-8-3
2F	47	/	0-1
30	48	0	0
31	49	1	1
32	50	2	2
33	51	3	3
34	52	4	4
35	53	5	5
36	54	6	6
37	55	7	7
38	56	8	8
39	57	9	9
3A	58	:	8-2
3B	59	;	11-8-6

Table F-1. Character Set (Continued)

USASCII Hexadecimal Value	Decimal Value	Printable Character	Hollerith Code
3C	60	<	12-8-4
3D	61	=	8-6
3E	62	>	0-8-6
3F	63	?	0-8-7
40	64	@	8-4
41	65	A	12-1
42	66	B	12-2
43	67	C	12-3
44	68	D	12-4
45	69	E	12-5
46	70	F	12-6
47	71	G	12-7
48	72	H	12-8
49	73	I	12-9
4A	74	J	11-1
4B	75	K	11-2
4C	76	L	11-3
4D	77	M	11-4
4E	78	N	11-5
4F	79	O	11-6
50	80	P	11-7
51	81	Q	11-8
52	82	R	11-9
53	83	S	0-2
54	84	T	0-3
55	85	U	0-4
56	86	V	0-5
57	87	W	0-6
58	88	X	0-7
59	89	Y	0-8
5A	90	Z	0-9

Table F-2. Additional Characters

USASCII Hexadecimal Value	Decimal Value	Printable Character	Hollerith Code	Keypunch Character
5B	91	[	12-2-8	¢
5C	92	\	0-8-2	0-8-2
5D	93	]	12-7-8	| (vertical bar)
5E	94	^	11-7-8	¬ (logical NOT)
5F	95	_	0-5-8	_ (underscore)
00	00	Null		
09	09	Tab		

APPENDIX G
BACK PANEL CONNECTORS

ARITHMETIC UNIT CONNECTOR ASSIGNMENTS

P2

GND	1	2	GND
+5 V	3	4	+5 V
TLADR14–	5	6	TLADR13–
TLDAT00–	7	8	TLADR00–
TLDAT03–	9	10	TLADR03–
TLADR12–	11	12	TLADR15–
OPEN	13	14	OPEN
OPEN	15	16	TLADR01–
TLDAT02–	17	18	TLDAT01–
TLADR02–	19	20	TLDAT08–
OPEN	21	22	TLDAT11–
TLADR08–	23	24	TLADR11–
PINTQ–	25	26	ENXOPQ–
TLGRNTDIN	27	28	TLADR07–
XOPTHERE–	29	30	XOPSTB–
TLADR09–	31	32	TLDAT09–
TLDAT10–	33	34	TLADR10–
TLADR19–	35	36	TLADR16–
TLDAT04–	37	38	TLDAT07–
+12 V	39	40	+12 V
−12 V	41	42	−12 V
XOPIAQCK–	43	44	TLADR17–
TLADR06–	45	46	TLDAT06–
TLDAT05–	47	48	TLADR18–
TLADR05–	49	50	IMODSEL08–
IMODSEL09–	51	52	IMODSEL10–
XOPCOMP–	53	54	XOPABORT–
IMODSEL11–	55	56	IMODSEL12–
120HZ–	57	58	IMODSEL13–
IMODSEL14–	59	60	IMODSEL06–
IMODSEL07–	61	62	TLGRNTDOUT
IMODSEL15–	63	64	IMODSEL00–
TLADR04–	65	66	IMODSEL01–
IMODSEL02–	67	68	IMODSEL03–
IMODSEL04–	69	70	IMODSEL05–
−5 V*	71	72	−5 V*
+5 V*	73	74	+5 V*
+12 V*	75	76	+12 V*
+5 V	77	78	+5 V
GND	79	80	GND

P1

GND	1	2	GND
+5 V	3	4	+5 V
+12 V*	5	6	+12 V*
+5 V*	7	8	+5 V*
−5 V*	9	10	−5 V*
INTREQ15–	11	12	INTREQ14–
CRUBITIN	13	14	TLTERM–
INTREQ13–	15	16	INTREQ11–
OPEN	17	18	INTREQ10–
OPEN	19	20	FPCLKENBL–
FRCNRA–	21	22	INTREQ08–
INTREQ09–	23	24	POFF
INTREQ06–	25	26	TLGO–
INTREQ07–	27	28	CRUBIT10
TLWAIT–	29	30	CRUBIT11
TLACKD–	31	32	EXTLOADER–
TLDAT13–	33	34	TLBUSY–
CRUBIT07	35	36	TLWRITE–
TLCPUAC–	37	38	CRUBIT09
+12 V	39	40	+12 V
−12 V	41	42	−12 V
CRUBIT05	43	44	TLDAT14–
MPE–	45	46	TLDAT12–
CRUBIT08	47	48	CRUBIT06
TILCLK–	49	50	CONSNRA0
CRUBIT15	51	52	CRUBIT04
CRUBIT13	53	54	TLDAT15–
CRUBITOUT	55	56	STORECLK–
CRUBIT14	57	58	CONSNRA3
CONSNRA2	59	60	CRUBIT12
MRESET–	61	62	EXTLOADGO–
CONSNRA1	63	64	CONSNRA7
CONSNRA5	65	66	IORESET–
CONSNRA4	67	68	EOIQ–
IDLE–	69	70	RESTART–
CONSNRA6	71	72	ENTER
INTREQ12–	73	74	RESTART–
OPEN	75	76	OPEN
+5 V	77	78	+5 V
GND	79	80	GND

(B) 128626 A (1/2)

* STANDBY POWER MAINTAINED BY BATTERY DURING MAIN POWER FAILURE.

TILINE CONNECTOR ASSIGNMENTS

P2

GND	1	2	GND
+5 VOLTS	3	4	+5 VOLTS
TLAG (OUT)	5	6	TLAG (IN)
GND	7	8	TLADR14–
TLADR15–	9	10	TLADR10–
TLADR12–	11	12	TLADR11–
TLPRES–	13	14	TLIORES–
TLADR13–	15	16	TLPFWP
TLADR08–	17	18	CRUBITOUT
TLADR09–	19	20	TLDAT11–
TLDAT08–	21	22	STORECLK–
TLDAT10–	23	24	OPEN
TLADR18–	25	26	OPEN
TLADR17–	27	28	OPEN
TLADR16–	29	30	OPEN
TLADR19–	31	32	CRUBIT13
TLDAT09–	33	34	CRUBIT15
TLDAT02–	35	36	CRUBIT12
TLDAT03–	37	38	CRUBIT14
+12 VOLTS	39	40	+12 VOLTS
–12 VOLTS	41	42	–12 VOLTS
TLDAT06–	43	44	TLADR01–
TLDAT07–	45	46	IMODSELB
TLADR06–	47	48	IMODSELA
TLADR07–	49	50	XOPIAQCK–
TLADR02–	51	52	XOPTHERE–
TLADR03–	53	54	XOPSTB–
TLADR00–	55	56	ENXOPQ–
TLADR04–	57	58	OPEN
TLADR05–	59	60	CRUBITIN
TLDAT04–	61	62	PINTQ–
TLDAT05–	63	64	XOPABORT–
INTERRUPT B	65	66	INTERRUPT A
TLDAT00–	67	68	XOPCOMP–
TLDAT01–	69	70	OPEN
–5 VOLTS *	71	72	–5 VOLTS *
+5 VOLTS *	73	74	+5 VOLTS *
+12 VOLTS *	75	76	+12 VOLTS *
+5 VOLTS	77	78	+5 VOLTS
GND	79	80	GND

P1

GND	1	2	GND
+5 VOLTS	3	4	+5 VOLTS
+12 VOLTS *	5	6	+12 VOLTS *
+5 VOLTS *	7	8	+5 VOLTS *
–5 VOLTS *	9	10	–5 VOLTS *
TLREAD	11	12	GND
TLPRES–	13	14	TLIORES–
GND	15	16	TLPFWP
GND	17	18	CRUBITOUT
GND	19	20	TLTM–
GND	21	22	STORECLK–
EXTLOADER–	23	24	GND
TLGO–	25	26	GND
TLDAT12–	27	28	TLDAT13–
OPEN	29	30	TLDAT14–
TLDAT15–	31	32	CRUBIT13
OPEN	33	34	CRUBIT15
OPEN	35	36	CRUBIT12
CPUAC –	37	38	CRUBIT14
+12 VOLTS	39	40	+12 VOLTS
– 12 VOLTS	41	42	–12 VOLTS
OPEN	43	44	OPEN
OPEN	45	46	IMODSELB
OPEN	47	48	IMODSELA
CPUCLK–	49	50	CRUBIT7
OPEN	51	52	CRUBIT6
OPEN	53	54	CRUBIT5
TLMER–	55	56	CRUBIT4
GND	57	58	TLAV
GND	59	60	CRUBITIN
EXTLOADGO–	61	62	CRUBIT8
TLWAIT–	63	64	CRUBIT9
INTERRUPT B	65	66	INTERRUPT A
OPEN	67	68	CRUBIT10
OPEN	69	70	CRUBIT11
TLAK–	71	72	GND
OPEN	73	74	GND
OPEN	75	76	OPEN
+5 VOLTS	77	78	+5 VOLTS
GND	79	80	GND

(B)128626 (2/2)

*STANDBY POWER MAINTAINED BY BATTERY DURING MAIN POWER FAILURE.

APPENDIX H
LANGUAGE REQUIREMENTS AND RELOCATABILITY

APPENDIX H

LANGUAGE REQUIREMENTS AND RELOCATABILITY

H.1 SOURCE STATEMENT FORMAT

An assembly language source program consists of source statements which may contain assembler directives, machine instructions, pseudo-instructions, or comments. Each source statement is a source record as defined for the source medium. With the exception of comment statements, each statement may have as many as four fields: the label field, the operator field, the operand field, and the comment field. The fields are separated by one or more blanks; and no field, with the exception of the comment field, may contain embedded blanks. A tab character (CTRL I) may be used in place of a blank to separate fields on the ASR733 and the ASR33. Two acceptable formats of source statements are shown in figure H-1. The first four lines show the fields aligned on arbitrarily chosen character positions to produce aligned fields in the source listing. The next four lines show the fields separated by tab characters.

Comment statements consist of a single field starting with an asterisk (*) in the first character position followed by any ASCII character including a blank in each succeeding character position. Comment statements are listed in the source portion of the assembly listing and have no other effect on the assembly.

The maximum length of source records is 60 characters. However, only the first 52 characters will be printed on the ASR733 or the ASR33. The end-of-record for the source medium is placed following the last field used.

H.1.1 CHARACTER SET

The Model 990 Assembler recognizes ASCII characters as follows:

- The alphabet (capital letters only) and space character
- The numerals
- Twenty-two special characters
- Five undefined characters
- The null character
- The tab character

Appendix F contains tables that list all 66 characters and show the ASCII and Hollerith codes for each.

H.1.2 LABEL FIELD

The label field begins in character position one of the source record and extends to the first blank. The label field contains a symbol (paragraph H.4)

```
    1 2 3 4 5 6 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
01  * CONVENTIØNAL SØURCE STATEMENT FØRMAT
02
03  START   LI    3,>25       LØAD W R 3
04
05          A     5,3         ADD W R 5
06
07          RT                RETURN TO CALLING PROGRAM
08
09  * PACKED SØURCE STATEMENT FØRMAT USING TABS
10
11  START→LI→3,>25→LØAD W R 3
12
13  →A→5,3→ADD  W R 5
14
15  →RT→RETURN TØ CALLING PRØGRAM
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

(A)128440

Figure H-1. Source Statement Formats

supplied by the programmer. A label is optional for machine instructions and pseudo-instructions, and for many assembler directives. When the label is omitted, the first character position must contain a blank. A source statement consisting of a label field only is a valid statement that has the effect of an EQU directive with the same label and with a dollar sign ($) in the operand field (paragraph H. 4. 1).

CAUTION

When the location counter contains an odd location, and a source statement consisting only of a label is followed by a machine instruction or a DATA directive, the machine instruction or data word does not have the same location as the label.

H. 1. 3 OPERATOR FIELD

The operator field begins following the blank that terminates the label field, or in the first non-blank character position after the first character position when the label is omitted. The operator field is terminated by one or more blanks, and may not extend past character position 60 of the source record. The operator field contains a symbol that defines the operation, which may be a machine instruction, a pseudo-instruction, or an assembler directive. The operator field may contain a user-defined extended operation symbol.

H. 1. 4 OPERAND FIELD

The operand field begins following the blank that terminates the operator field, and may not extend past character position 60 of the source record. The operand field may contain one or more expressions, terms, or constants, according to the requirements of the operation specified in the operator field. The operand field is terminated by one or more blanks.

H. 1. 5 COMMENT FIELD

The comment field begins following the blank that terminates the operand field, and may extend to the end of the source record if required. The comment field may contain any ASCII character, including blank. The contents of the comment field are listed in the source portion of the assembly listing and have no other effect on the assembly.

H. 2 EXPRESSIONS

Expressions are used in the operand fields of assembler directives and machine instructions.

H.2.1 DEFINITION

An expression is a constant or symbol, or a series of constants, a series of symbols, or a series of constants and symbols separated by arithmetic operators. Each constant or symbol may be preceded by a minus sign (unary minus). The expression may contain no embedded blanks. The symbols may not be symbols that are defined as extended operations. Symbols that are defined as external references may not be operands of arithmetic operations. Only one symbol in an expression may be subsequently defined in the program, but that symbol must not be part of an operand in a multiplication or division operation within the expression. An expression that contains a relocatable symbol or constant immediately following a multiplication or division operator is an illegal expression. Also, when the result of evaluating an expression up to a multiplication or division operator is relocatable, the expression is illegal. An expression in which the number of relocatable symbols or constants added to the expression minus the number of relocatable symbols or constants subtracted from the expression is not equal to zero or one is an illegal expression. Refer to paragraph H.7 for definition of relocatability.

The following are examples of valid expressions:

```
BLUE+1

GREEN-4

2*16+RED

440/2-RED
```

H.2.2 WELL-DEFINED EXPRESSIONS

Some assembler directives require well-defined expressions in the operand fields. A well-defined expression must not contain any symbols or assembly-time constants that are not previously defined. No character constant may be placed in a well-defined expression. The evaluation of the entire expression must be absolute.

H.2.3 ARITHMETIC OPERATORS AND ORDER OF EVALUATION

The arithmetic operators in expressions are as follows:

- + for addition
- - for subtraction
- * for multiplication
- / for division

In evaluating an expression, the assembler first negates any constant or symbol preceded by a unary minus, then performs the arithmetic operations from left to right. The assembler does not assign precedence to any operation other than unary minus.

For example, the expression 4+5*2 would be evaluated 18, not 14. Also, the expression 7+1/2 would be evaluated 4, not 7.

H.3 CONSTANTS

Constants are used in expressions. The assembler recognizes four types of constants: decimal integer constants, hexadecimal integer constants, character constants, and assembly-time constants.

H.3.1 DECIMAL INTEGER CONSTANTS

A decimal integer constant is written as a string of numerals. When a decimal integer constant represents data, the range of values is -32,768 to +65,535. Positive decimal integer constants greater than 32,767 are considered negative when used as operands of addition and subtraction instructions.

The following are valid decimal constants:

```
 1000
-32768
  25
```

H.3.2 HEXADECIMAL INTEGER CONSTANTS

A hexadecimal integer constant is written as a string of up to four hexadecimal numerals preceded by a greater than (>) character. Hexadecimal numerals include the decimal values 0 through 9 and the letters A through F.

The following are valid hexadecimal constants:

```
>78
>F
>37AC
```

H.3.3 CHARACTER CONSTANTS

A character constant is written as a string of one or two characters enclosed in single quotes. For each single quote required within a character constant, two consecutive single quotes are required to represent the quote. The characters are represented internally as eight-bit ASCII characters, with leading bit equal to zero. A character constant consisting only of two single quotes (no character) is valid, and is assigned the value 0000_{16}.

The following are valid character constants:

Constant	Value
'AB'	4142_{16}
'C'	0043_{16}
'N'	$004E_{16}$
'''D'	2744_{16}

H.3.4 ASSEMBLY-TIME CONSTANTS

An assembly-time constant is written as an expression in the operand field of an EQU directive. Any symbol in the expression must have been previously defined. The value of the label is determined at assembly time, and is absolute or relocatable as defined in paragraph H. 7.

H.4 SYMBOLS

Symbols are used in the label field, the operator field, and the operand field. A symbol is a string of alphanumeric characters, the first of which must be an alphabetic character, and none of which may be a blank. When more than six characters are used in a symbol, the assembler prints all the characters, but accepts only the first six characters for processing. User-defined symbols are valid only during the assembly in which they are defined.

When a symbol is used in the label field, it is associated with a location in the program, and must not be used as a label in any other statement. The mnemonic operation codes and the assembler directive names are valid user-defined symbols when placed in the label field.

The DXOP directive defines a symbol to be used in the operator field. No other user-defined symbol may be used in the operator field. Any symbol that is used in the operand field must be placed in the label field of a statement, or in the operand field of a REF directive with two exceptions. One exception is the operand field of the DXOP directive. The other exception is the dollar sign character ($) used in expressions to represent the current location within the program (HERE).

The following are examples of valid symbols:

```
START
A1
OPERATION
$
```

H.5 TERMS

Terms are used in the operand fields of machine instructions and an assembler directive. A term is a decimal or hexadecimal constant, an absolute assembly-time constant, or an absolute label.

The following are examples of valid terms:

```
12
>C
WR2
```

Note that WR2 is valid as a term only if it has an absolute value. If START were a relocatable symbol and WR2 were defined as follows, WR2 would be relocatable, and not a valid term:

```
WR2  EQU  START+4
```

H.6 CHARACTER STRINGS

Several assembler directives require character strings in the operand field. A character string is written as a string of characters enclosed in single quotes. For each single quote in a character string, two consecutive single quotes are required to represent the single quote within the character string. The maximum length of the string is defined for each directive that requires a character string. The characters are represented internally as eight-bit ASCII characters.

The following are valid character strings:

```
'SAMPLE PROGRAM'

'PLAN ''C'''

'OPERATOR MESSAGE * PRESS START SWITCH'
```

H.7 RELOCATABILITY

H.7.1 RELOCATION OF CODE

The Model 990 Assembler assembles both absolute and relocatable object code. Absolute object code is code that must be placed in specified memory locations and is appropriate for programs that occupy dedicated areas of memory. Relocatable object code is code that may be placed in any available locations. All relocatable address information must be modified for the actual memory locations in which the program is placed. Relocatability allows programs to share memory in many possible combinations.

H.7.2 RELOCATABILITY OF SOURCE STATEMENT ELEMENTS

Elements of source statements are expressions, constants, symbols, and terms. Terms are absolute in all cases; the other elements may be either absolute or relocatable.

The relocatability of an expression is a function of the relocatability of the symbols and constants that make up the expression. An expression is relocatable when it contains one or more relocatable constants or symbols, and the number of relocatable symbols or constants added to the expression is one greater than the number of relocatable symbols or constants subtracted from the expression. (All other valid expressions are absolute). When the first symbol or constant is unsigned, it is considered to be added to the expression. When a unary minus follows an addition operator in an expression,

the effective operation is subtraction. When a unary minus follows a subtraction operator, the effective operation is addition. For example, when all symbols in the following expressions are relocatable, the expressions are relocatable:

```
LABEL--1

LABEL+TABLE+-INC

-LABEL+TABLE+INC
```

Decimal, hexadecimal, and character constants are absolute. Assembly-time constants defined by absolute expressions are absolute, and assembly-time constants defined by relocatable expressions are relocatable.

Any symbol that appears in the label field of a source statement other than an EQU directive is absolute when the statement is in an absolute block of the program. Any symbol that appears in the label field of a source statement other than an EQU directive is relocatable when the statement is in a relocatable block of the program.

A location may be defined as absolute or as relocatable. The location may contain either an absolute or relocatable values.

APPENDIX I
ASSEMBLER DIRECTIVES AND PSEUDO-OPS

APPENDIX I

ASSEMBLER DIRECTIVES AND PSEUDO-OPS

I.1 DIRECTIVES AFFECTING THE LOCATION COUNTER

Five assembler directives affect only the location counter of the assembler. Two of these also define the succeeding block of the program as absolute or relocatable. The location counter is a component of the assembler that contains the present location.

Until an Absolute Origin directive is processed by the assembler, the location counter contents are relocatable. Subsequent Relocatable Origin directives cause the location counter to be set to the specified relocatable value, and to continue assembling relocatable object code. This concatenates all relocatable blocks within an assembly into a single relocatable segment. The total length of this segment is the length of the relocatable code assembled.

The Block Starting with Symbol and Block Ending with Symbol directives advance the location counter, forming an area for storage of data. The Word Boundary directive aligns the location counter to a word boundary (even address).

I.1.1 ABSOLUTE ORIGIN (AORG)

AORG places a value in the location counter and defines the succeeding locations as absolute. Use of the label field is optional. When a label is used, it is assigned the value that the directive places in the location counter. The operator field contains AORG. The operand field contains a well-defined expression. The assembler places the value of the well-defined expression in the location counter. Use of the comment field is optional.

The following example shows an AORG directive:

```
AORG   >1000+X
```

Symbol X must be absolute and must have been previously defined. If X has a value of 6, the location counter is set to 1006_{16} by this directive. Had a label been included, the label would have been assigned the value 1006_{16}.

I.1.2 RELOCATABLE ORIGIN (RORG)

RORG places a value in the location counter and defines the succeeding locations as relocatable. Use of the label field is optional. When a label is used, it is assigned the value that the directive places in the location counter. The operator field contains RORG. The operand field is optional , and when the operand field is not used, zero or the value that was in the location counter following assembly of the preceding relocatable location is placed in the location counter. When the operand field is used, a relocatable expression that

contains no symbols not previously defined is placed in the operand field. The comment field may be used only when the operand field is used.

The following example shows an RORG directive:

```
RORG   $-20   OVERLAY TEN WORDS
```

The $ symbol refers to the location following the preceding relocatable location of the program. This has the effect of backing up the location counter ten words. The instructions and directives following the RORG directive replace the ten previously assembled words of relocatable code, permitting correction of the program without removing source records. Had a label been included, the label would have been assigned the value placed in the location counter. An example of a RORG directive with no operand field is as follows:

```
SEG2   RORG
```

Assume that after defining data for a program, which occupied 44_{16} bytes, an AORG directive initiated an absolute block of code. The absolute block is followed by the RORG directive in the above example, which places 0044_{16} in the location counter and defines the location counter as relocatable. Symbol SEG2 is a relocatable value, 0044_{16}. The RORG directive in the above example would have no effect except at the end of an absolute block.

I.1.3 BLOCK STARTING WITH SYMBOL (BSS)

BSS assigns the value in the location counter to the symbol in the label field and advances the location counter according to the value in the operand field. The label field contains the label of the first byte in the block. The operator field contains BSS. The operand field contains a well-defined expression that represents the number of bytes to be added to the location counter. The comment field is optional.

The following example shows a BSS directive:

```
BUFF1   BSS 80   CARD INPUT BUFFER
```

This directive reserves an 80-byte buffer at location BUFF1.

I.1.4 BLOCK ENDING WITH SYMBOL (BES)

BES advances the location counter according to the value in the operand field and assigns the new location counter value to the symbol in the label field. The label field contains the label of the location following the block. The operator field contains BES. The operand field contains a well-defined expression that represents the number of bytes to be added to the location counter. The comment field is optional.

The following example shows a BES directive:

```
BUFF2   BES   >10
```

The directive reserves a 16-byte buffer. Had the location counter contained 100_{16} when the assembler processed this directive, BUFF2 would have been assigned the value 110_{16}.

I.1.5 WORD BOUNDARY (EVEN)

EVEN places the location counter on the next word boundary (even) byte address. When the location counter is already on a word boundary, the location counter is not altered. Use of the label field is optional. When a label is used, the value in the location counter after processing the directive is assigned to the label. The operator field contains EVEN. The operand field is not used, and the comment field is optional.

The following example shows an EVEN directive:

```
WRF1    EVEN    WORKSPACE REGISTER FILE ONE
```

The directive assures that the location counter contains a word boundary address, and assigns that address to label WRF1. Use of an EVEN directive preceding or following a machine instruction or a DATA directive is redundant. The assembler advances the location counter to an even address when it processes a machine instruction or a DATA directive.

I.2 DIRECTIVES AFFECTING ASSEMBLER OUTPUT

Five assembler directives affect assembler output. One affects the object code output of the assembler and the remaining four affect the source listing output of the assembler.

The Program Identifier directive supplies a program name, which is placed in the object code for use by the linking loader.

The Page Title directive supplies a title to be printed at the top of each page of the source listing. The List Source directive restores printing of the source listing when printing has been inhibited by a No Source List directive. The Page Eject directive causes the assembler to print a heading and continue the source listing on a new page.

I.2.1 PROGRAM IDENTIFIER (IDT)

IDT assigns a name to the program. An IDT directive must precede any machine instruction or assembler directive that results in object code. Use of the label field is optional. When a label is used, the current value of the location counter is assigned to the label. The operator field contains IDT. The operand field contains the program name, a character string of up to eight characters. When a character string of more than eight characters is entered, the assembler prints a truncation error message, and retains the first eight characters as the program name. The comment field is optional.

The following example shows an IDT directive:

```
IDT   'CONVERT'
```

The directive assigns the name CONVERT to the program to be assembled. The program name is printed in the source listing as the operand of the IDT directive, but does not appear in the page heading of the source listing. The program name is placed in the object code, but serves no purpose during the assembly.

I.2.2 PAGE TITLE (TITL)

TITL supplies a title to be printed in the heading of each page of the source listing. When a title is desired in the heading of the first page of the source listing, a TITL directive must be the first source statement submitted to the assembler. This directive is not printed in the source listing. Use of the label field is optional. When a label is used, the current value of the location counter is assigned to the label. The operator field contains TITL. The operand field contains the title, a character string of up to 50 characters. When more than 50 characters are entered, the assembler retains the first 50 characters as the title, and prints a truncation error message. The comment field is optional, but the assembler does not print the comment.

The following example shows a TITL directive:

```
TITL   '** REPORT GENERATOR **'
```

The directive causes the title ** REPORT GENERATOR ** to be printed in the page headings of the source listing. When a TITL directive is the first source statement in a program, the title is printed on all pages until another TITL directive is processed. Otherwise, the title is printed on the next page after the directive is processed, and on subsequent pages until another TITL directive is processed.

I.2.3 LIST SOURCE (LIST)

LIST restores printing of the source listing. This directive is required only when a No Source List directive is in effect, to cause the assembler to resume listing. This directive is not printed in the source listing. Use of the label field is optional. When a label is used, the current value of the location counter is assigned to the label. The operator field contains LIST. The operand field is not used. Use of the comment field is optional, but the assembler does not print the comment.

The following example shows a LIST directive:

```
LIST
```

The directive causes the source listing to be resumed with the next source statement.

I.2.4 NO SOURCE LIST (UNL)

UNL inhibits printing of the source listing. The UNL directive is not printed in the source listing. Use of the label field is optional. When a label is used, the current value of the location counter is assigned to the label. The operator field contains UNL. The operand field is not used. Use of the comment field is optional, but the assembler does not print the comment.

The following example shows UNL directive:

```
UNL
```

The directive inhibits printing of the source listing. Use of the UNL directive to inhibit printing reduces assembly time and the size of the source listing.

I.2.5 PAGE EJECT (PAGE)

PAGE causes the assembler to continue the source program listing on a new page. The PAGE directive is not printed in the source listing. Use of the label field is optional. When a label is used, the current value of the location counter is assigned to the label. The operator field contains PAGE. The operand field is not used. Use of the comment field is optional, but the assembler does not print the comment.

The following example shows a PAGE directive:

```
PAGE
```

The directive causes the assembler to begin a new page of the source listing. The next source statement is the first statement listed on the new page. Use of the Page directive to begin new pages of the source listing at the logical divisions of the program improves documentation of the program.

I.3 DIRECTIVES THAT INITIALIZE CONSTANTS

Four assembler directives assign initial values to constants. The Initialize Byte directive initializes one or more bytes of memory with eight-bit two's complement numbers. The Initialize Word directive initializes one or more words of memory with 16-bit two's complement numbers. The Initialize Text directive places ASCII characters in successive bytes of memory. The Define Assembly-Time Constant directive assigns a value to a symbol.

I.3.1 INITIALIZE BYTE (BYTE)

BYTE places one or more values in one or more successive bytes of memory. Use of the label field is optional. When a label is used, the location at which the assembler places the first byte is assigned to the label. The operator field contains BYTE. The operand field contains one or more expressions separated by commas. The expressions must contain no symbols that are

not previously defined and no external references. The assembler evaluates each expression and places the value in a byte as an eight-bit two's complement number. When truncation is required, the assembler prints a truncation error message and places the rightmost portion of the value in the byte. The comment field is optional.

The following example shows a BYTE directive:

```
KONS    BYTE    >F+1,-1,'D'-'=',0,'AB'-'AA'
```

The directive initializes five bytes, starting with a byte at location KONS. The contents of the resulting bytes is 00010000, 11111111, 00000111, 00000000, and 00000001.

I.3.2 INITIALIZE WORD (DATA)

DATA places one or more values in one or more successive words of memory. The assembler advances the location counter to a word boundary (even) address. Use of the label field is optional. When a label is used, the location at which the assembler places the first word is assigned to the label. The operator field contains DATA. The operand field contains one or more expressions separated by commas. The assembler evaluates each expression and places the value in a word as a sixteen-bit two's complement number. The comment field is optional.

The following example shows a DATA directive:

```
KONS1   DATA    3200,1+'AB',-'AF',>F4A0,'A'
```

The directive initializes five words, starting with a word at location KONS1. The contents of the resulting words are $0C80_{16}$, 4143_{16}, $BEBA_{16}$, $F4A0_{16}$, and 0041_{16}. Had the location counter contents been $010F_{16}$ prior to processing this directive, the value assigned to KONS1 would be 0110_{16}.

I.3.3 INITIALIZE TEXT (TEXT)

TEXT places one or more characters in successive bytes of memory. The assembler negates the last character of the string when the string is preceded by a minus (-) sign (unary minus). Use of the label field is optional. When a label is used, the location at which the assembler places the first character is assigned to the label. The operator field contains TEXT. The operand field contains a character string of up to 52 characters, which may be preceded by a unary minus sign. The comment field is optional.

The following example shows a TEXT directive:

```
MSG1    TEXT 'EXAMPLE'    MESSAGE HEADING
```

The directive places the eight-bit ASCII representations of the characters in successive bytes. When the location counter is on an even address, the result, in hexadecimal representation, is 4558, 414D, 504C, and 45XX. XX represents the contents of the rightmost byte of the fourth word, which are

determined by the next source statement. The label MSG1 is assigned the value of the first byte address in which 45 is placed. Another example, showing the use of a unary minus, is as follows:

```
MSG2  TEXT  -"NUMBER"
```

When the location counter is on an even address, the result, in hexadecimal representation, is 4E55, 4D42, and 45AE. The label MSG2 is assigned the value of the byte address in which 4E is placed.

I.3.4 DEFINE ASSEMBLY-TIME CONSTANT (EQU)

EQU assigns a value to a symbol. The label field contains the symbol. The operator field contains EQU. The operand field contains an expression in which all symbols have been previously defined. Use of the comment field is optional.

The following example shows an EQU directive:

```
R0   EQU  0   WORKSPACE REGISTER 0
```

The directive assigns an absolute value to the symbol R0, making R0 available to use as a workspace register address. Another example of an EQU directive is:

```
TIME   EQU   HOURS
```

The directive assigns the value of previously defined symbol HOURS to symbol TIME. When HOURS appears in the label field of a machine instruction in a relocatable block of the program, the value is a relocatable value. The two symbols may be used interchangeably.

I.4 DIRECTIVES THAT LINK PROGRAMS

Two assembler directives provide links between programs that are assembled separately. The External Definition directive makes one or more symbols in a program available to other programs. The External Reference directive provides access to one or more symbols from other programs for use in a program. The programs may be linked and executed as one program.

I.4.1 EXTERNAL DEFINITION (DEF)

DEF makes one or more symbols available to other programs for reference. The use of the label field is optional. When a label is used, the current value of the location counter is assigned to the label. The operator field contains DEF. The operand field contains one or more symbols, separated by commas, to be defined in the program being assembled. The comment field is optional.

The following example shows a DEF directive:

```
DEF   ENTER,ANS
```

The directive causes the assembler to include symbols ENTER and ANS in the object code so that these symbols are available to other programs. When the DEF directive does not precede the source statements that contain the symbols, the assembler identifies the symbols as multiply defined symbols.

I.4.2 EXTERNAL REFERENCE (REF)

REF provides access to one or more symbols defined in other programs. The use of the label field is optional. When a label is used, the current value of the location counter is assigned to the label. The operator field contains REF. The operand field contains one or more symbols, separated by commas,* to be used in the operand field of a subsequent source statement. The comment field is optional.

The following example shows a REF directive:

```
REF   ARG1,ARG2
```

The directive causes the assembler to include symbols ARG1 and ARG2 in the object code so that the corresponding addresses may be obtained from other programs.

NOTE

An external reference will not be inserted by the loader at absolute location 0.

I.5 MISCELLANEOUS DIRECTIVES

Two miscellaneous directives are available. The Define Extended Operation directive assigns a symbol for an extended operation. The Program End directive terminates the source program.

I.5.1 DEFINE EXTENDED OPERATION (DXOP)

DXOP assigns a symbol to be used in the operator field to specify an extended operation. The use of the label field is optional. When a label is used, the current value in the location counter is assigned to the label. The operator field contains DXOP. The operand field contains a symbol followed by a comma and a term. The symbol assigned to an extended operation must not be used in the label or operand field of any other statement. The assembler assigns the symbol to an extended operation specified by the term, which must have a value in the range of 0 to 15. The comment field is optional.

The following example shows a DXOP directive:

```
DXOP   DADD,13
```

The directive defines DADD as extended operation 13. When the assembler recognizes the symbol DADD in the operator field, it assembles an XOP

instruction that specifies extended operation 13. The XOP instruction is described in the Model 990 Reference Manual. The following example shows the use of the symbol DADD in a source statement:

```
DADD @LABEL1(4)
```

The assembler places the operand field contents in the T_s and S fields of an XOP instruction, and places 13 in the D field.

I.5.2 PROGRAM END (END)

END terminates the assembly. The last source statement of a program is the END directive. When any source statements follow the END directive, they are ignored. Use of the label field is optional. When a label is used, the current value in the location counter is assigned to the symbol. The operator field contains END. Use of the operand field is optional. When the operand field is used, it contains a symbol that specifies the entry point of the program. When the operand field is not used, no entry point is placed in the object code. The comment field may be used only when the operand field is used.

The following example shows an END directive:

```
END  START
```

The directive causes the assembler to terminate the assembly of this program. The assembler also places the value of START in the object code as an entry point.

When a program executes in a stand-alone mode, and is loaded by the ROM loader, it must supply an entry point to the loader. When no operand is included in the END directive, and that program is loaded by the ROM loader, the loader transfers control to the entry point of the loader, and attempts to load another object program.

When a program is to be loaded by the Linking Loader (LAL990) the END directive does not require an operand unless the program is to be loaded and linked to other programs and contains the entry point for the resulting linked program. LAL990 returns control to the first relocatable location when the program or programs loaded do not specify entry points. When LAL990 loads a set of programs, and more than one of these programs specifies an entry point, LAL990 transfers control to the last entry point it receives.

I.6 PSEUDO-INSTRUCTIONS

The Model 990 Assembly Language includes two pseudo-instructions, which are predefined symbols that cause the assembler to assemble certain machine instructions with specific operands. A pseudo-instruction is a convenient way to code an operation that is actually performed by a machine instruction. The pseudo-instructions are the No Operation and the Return instructions.

I.6.1 NO OPERATION (NOP)

NOP places a machine instruction in the object code which has no effect on execution of the program. Use of the label field is optional. When the label field is used, the label is assigned the location of the instruction. The operator field contains NOP. The operand field is not used. Use of the comment field is optional.

Enter the NOP pseudo-instruction as shown in the following example:

```
MOD    NOP
```

Location MOD contains a NOP pseudo-instruction when the program is loaded. Another instruction may be placed in location MOD during execution to implement a program option. The assembler supplies the same object code as if the source statement had contained the following:

```
MOD    JMP    $+2
```

I.6.2 RETURN (RT)

RT places a machine instruction in the object code to return control to a calling routine from a subroutine. Use of the label field is optional. When the label field is used, the label is assigned the location of the instruction. The operator field contains RT. The operand field is not used.

Use of the comment field is optional. Enter the RT pseudo-instruction as shown in the following example:

```
RT
```

The assembler supplies the same object code as if the source statement had contained the following:

```
B    *11
```

When control is transferred to a subroutine by execution of a BL instruction, the link to the calling routine is stored in workspace register 11. An RT pseudo-instruction returns control to the instruction following the BL instruction in the calling routine.

USER'S RESPONSE SHEET

Manual Title: Model 990 Computer Reference Manual, Preliminary (943442-9701)

Date of Manual: 1 October 1974 Date of This Letter: _____

User: _____ Office/Dept. No.: _____

Company: _____

Street Address: _____

City/State/Zip: _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____

CUT ALONG LINE

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), STAPLE AND MAIL

First Class
PERMIT NO. 3135
Austin, Texas

BUSINESS REPLY MAIL
No Postage Necessary if Mailed in the United States

Postage Will Be Paid by

TEXAS INSTRUMENTS INCORPORATED
DIGITAL SYSTEMS DIVISION

P.O. BOX 2909 · AUSTIN, TEXAS 78767

Attn: TECHNICAL PUBLICATIONS, MS 2146

Sales and Service Offices of Texas Instruments are located throughout the United States and in major countries overseas. Contact the Digital Systems Division, Texas Instruments Incorporated, P.O. Box 1444, Houston, Texas 77001, or call (713) 494-5115, for the location of the office nearest to you.

TEXAS INSTRUMENTS
INCORPORATED