Burroughs

B 6900 SYSTEM

REFERENCE MANUAL

PRICED ITEM

Printed in U.S.America

July 1981

Form 5010986

Burroughs

B 6900 SYSTEM

REFERENCE MANUAL

PRICED ITEM

Printed in U.S.America

July 1981

Form 5010986

Burroughs believes that the information described in this publication is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, is accepted for any consequences arising out of use of this information.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Warning: This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. As temporarily permitted by regulation, it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Correspondence regarding this publication should be forwarded using the Remarks form at the back of the manual, or may be addressed directly to TIO West Documentation, Burroughs Corporation, P.O. Box 4040, El Monte, California 91734, U.S.A.

LIST OF EFFECTIVE PAGES

Page	Issue	Page	Issue
Title	Original	5-45 thru 5-53	Original
ii	Original	5-54	Blank
iii	Original	5-55	Original
iv	Blank	5-56	Blank
v thru xxi	Original	5-57 thru 5-71	Original
xxii	Blank	5-72	Blank
xxiii thru xxiv	Original	5-73 thru 5-88	Original
1-1 thru 1-5	Original	6-1 thru 6-7	Original
1-6	Blank	6-8	Blank
1-7 thru 1-26	Original	7-1 thru 7-33	Original
2-1 thru 2-34	Original	7-34	Blank
3-1 thru 3-18	Original	8-1 thru 8-12	Original
4-1	Original	9-1 thru 9-4	Original
4-2	Blank	10-1 thru 10-8	Original
4-3	Original	11-1 thru 11-42	Original
4-4	Blank	A-1 thru A-5	Original
4-5 thru 4-94	Original	A-6	Blank
5-1 thru 5-3	Original	B-1 thru B-6	Original
5-4	Blank	C-1 thru C-2	Original
5-5 thru 5-43	Original	D-1 thru D-2	Original
5-44	Blank	Index-1 thru Index-7	Original
		Index-8	Blank

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Cont)

LIST OF ILLUSTRATIONS (Cont)

LIST OF ILLUSTRATIONS (Cont)

LIST OF ILLUSTRATIONS (Cont)

LIST OF TABLES

INTRODUCTION

The B 6900 is a large scale, modular, high-speed data processing system. The B 6900 system consists of four or more cabinets, which are joined together to form a single mainframe organization. The leading features of the B 6900 system are:

a. Monolithic circuits.

b. System memory expandable in increments of 131,072 words, to a maximum of 1,048,576 words.

c. Either local or GLOBAL(TM) memory to the B 6900 system.

d. Automatic memory error detection and correction.

e. Peripheral units expandable to 512 DLP control units.

f. Multiple MLI paths for I/O operation.

g. Data communications processing through the use of optional standard equipment.

h. Reader/sorter subsystem capability through the use of optional standard equipment.

i. Centralized power supplies, with solid metallic bus-bar organization.

A unique design concept, developed from years of experience with the B 6700 and B 6800 Information Processing Systems has resulted in the B 6900 hardware and software design. The hardware and the software were simultaneously designed in a parallel and coordinated process, such that these two parts of the system act to augment and to complement each other. This method assures that the hardware will contain the logic circuits necessary to implement the concepts of the software, and also that the software constructs will utilize the hardware circuits in an efficient manner.

The B 6900 system is designed to use the hardware stack concept which was successful in former systems. However, the hardware used in the B 6900 system also represents recent state-of-the-arts improvements in data processing circuit components. This blending of proven design with modern material results in a more efficient, and powerful data processing system.

The B 6900 system utilizes the same dynamic storage allocation concept that was utilized in former Information Processing Systems. This concept utilizes a descriptor method of segmentation which allows variable length segments of data to be used. This method is more efficient than "fixed-size" paging concepts.

A "look-ahead" logical circuit is used in the B 6900 system data processor to fetch program code words from memory. This circuit virtually eliminates the need to halt the flow of a user program to obtain the next word of program code. Use of this circuit represents an improvement in the way user programs are executed, and results in more efficient operation of the hardware system resources.

The use of new, and more compact logical circuit components has allowed the B 6900 system to have a greater degree of packaging density than was available in system design. The central processing unit, which is a single system cabinet, takes the place of 4 cabinets that were required in the B 6700 system. This improvement in packaging saves space and reduces operating costs in the B 6900 system, without requiring a loss in data processing capability.

The B 6900 system utilizes a centralized power supply cabinet. This centralized power supply eliminates the need to mount an inverter module in each mainframe cabinet. It collects most power supplies for the B 6900 system within a single cabinet and, thus, makes the power supply subsystem easier to maintain.

The B 6900 system cabinets have a fixed relative location within the mainframe cabinet layout. This fixed location scheme reduces the complexity of the system installation process, reduces interface cabling requirements, and allows more efficiency in site planning.

The B 6900 system contains the capability to be interfaced with, and to operate from GLOBAL(TM) memory applications.

GLOBAL(TM) is a trademark of Burroughs Corporation.

SECTION 1

SYSTEM DESCRIPTION

GENERAL

This manual explains how the B 6900 Information Processing System achieves flexibility and efficiency through a comprehensive system approach to problem solving without considering the areas of computer logic or circuit design. The program-independent modular system design efficiently uses available units to process programs and also permits system configuration changes without the need to reprogram or recompile. This approach also offers the user the advantages of simplified programming, ease of operation, and a complete freedom of system expansion. The B 6900 is a compiler oriented system, designed to accept the high level problem-solving language compilers such as ALGOL, COBOL, FORTRAN, and PL/I.

The B 6900 system software operates under the control of a Master Control Program (MCP), which automatically handles memory assignments, program segmentation, and subroutine linkages. The use of the MCP eliminates many arduous programming tasks which are likely to produce errors. The compilers are operated under the control of the MCP, as are the object-programs that result from the use of the compilers. The programs are debugged and corrected in the source language.

SCOPE OF THIS MANUAL

This manual will describe the major hardware characteristics of the B 6900 system. Because of the strong interdependence of the system software and system hardware this manual will discuss both parts of the system design at times. Wherever a choice is available, to discuss a part of the system in terms of either the hardware or the software, the hardware discussion will be used. Both discussions will be used where insight can be developed by the use of this method.

B 6900 HARDWARE SYSTEM ORGANIZATION

The B 6900 system consists of a series of cabinet types arranged in a specific order. The ordering of the cabinets within the system is classed as a minimum configuration B 6900 system, or as an expanded configuration B 6900 system. The arrangement of the cabinets within a B 6900 system is such that a minimum configuration B 6900 may be upgraded to an expanded configuration by adding additional cabinets. However, no reorganization of the cabinets within a B 6900 system is required to upgrade an existing system to the expanded configuration class.

There are four standard-size cabinets used in the organization of a B 6900 system. Figure 1-1 shows these four cabinet sizes, and indicates the various dimensions of the cabinets. The cabinets in a B 6900 system are joined together to form a continuous mainframe appearance. This appearance is enhanced by the use of outer panels that give the illusion of a single mainframe structure.

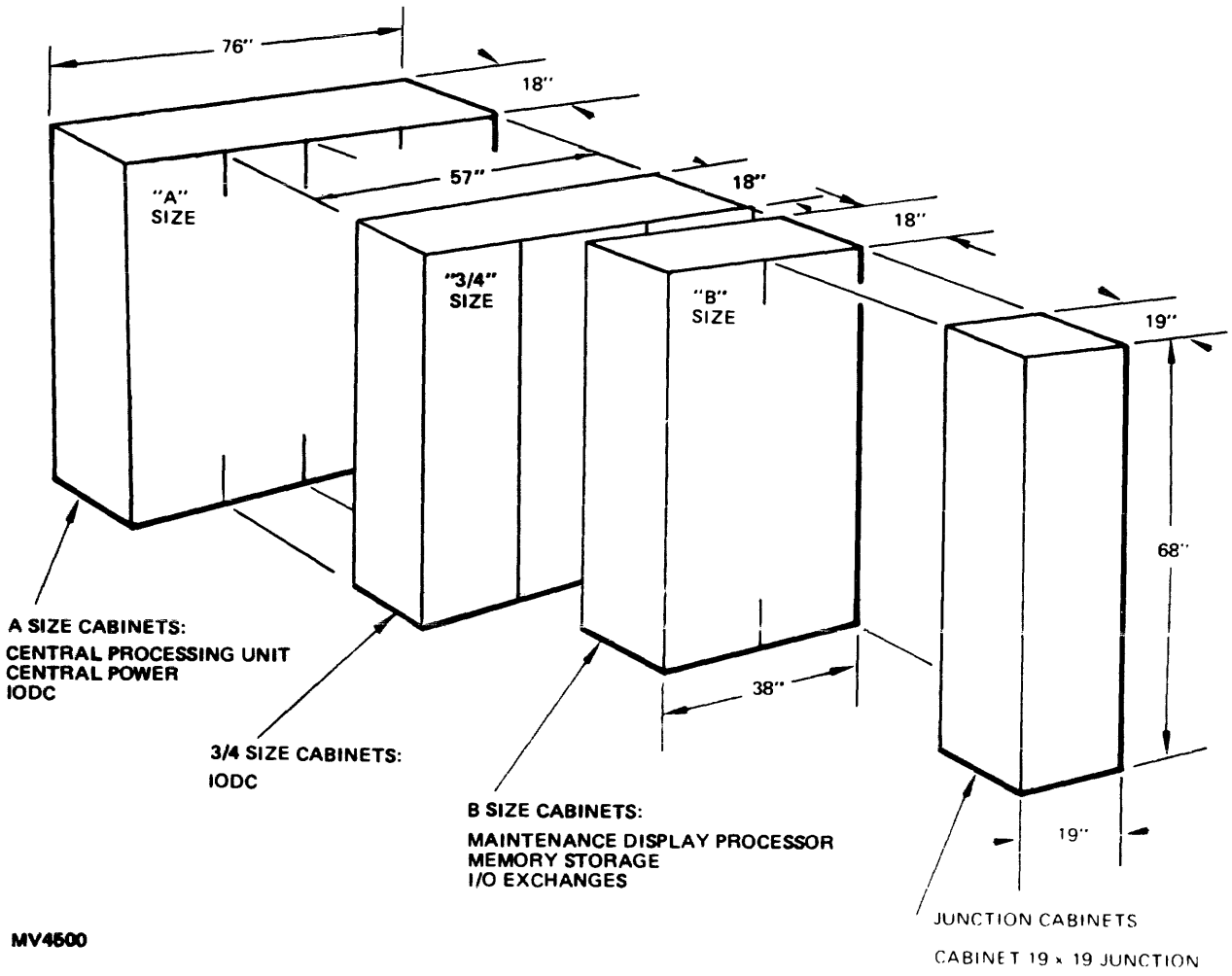


Figure 1-1. B 6900 Cabinets Sizes

Figure 1-2 shows the cabinets in a minimum configuration B 6900 system. The layout of the various cabinets within the B 6900 system mainframe structure is invariable; therefore, the minimum area for the mainframe of a B 6900 system also is invariable. The minimum area required for a B 6900 system mainframe is 21 feet, three inches wide, by 25 feet in length. This area allows for the expansion of a minimum configuration B 6900 system into a fully expanded configuration B 6900 system. But the area given in this paragraph does not include the area required to contain the peripheral devices that are connected to the B 6900 system.

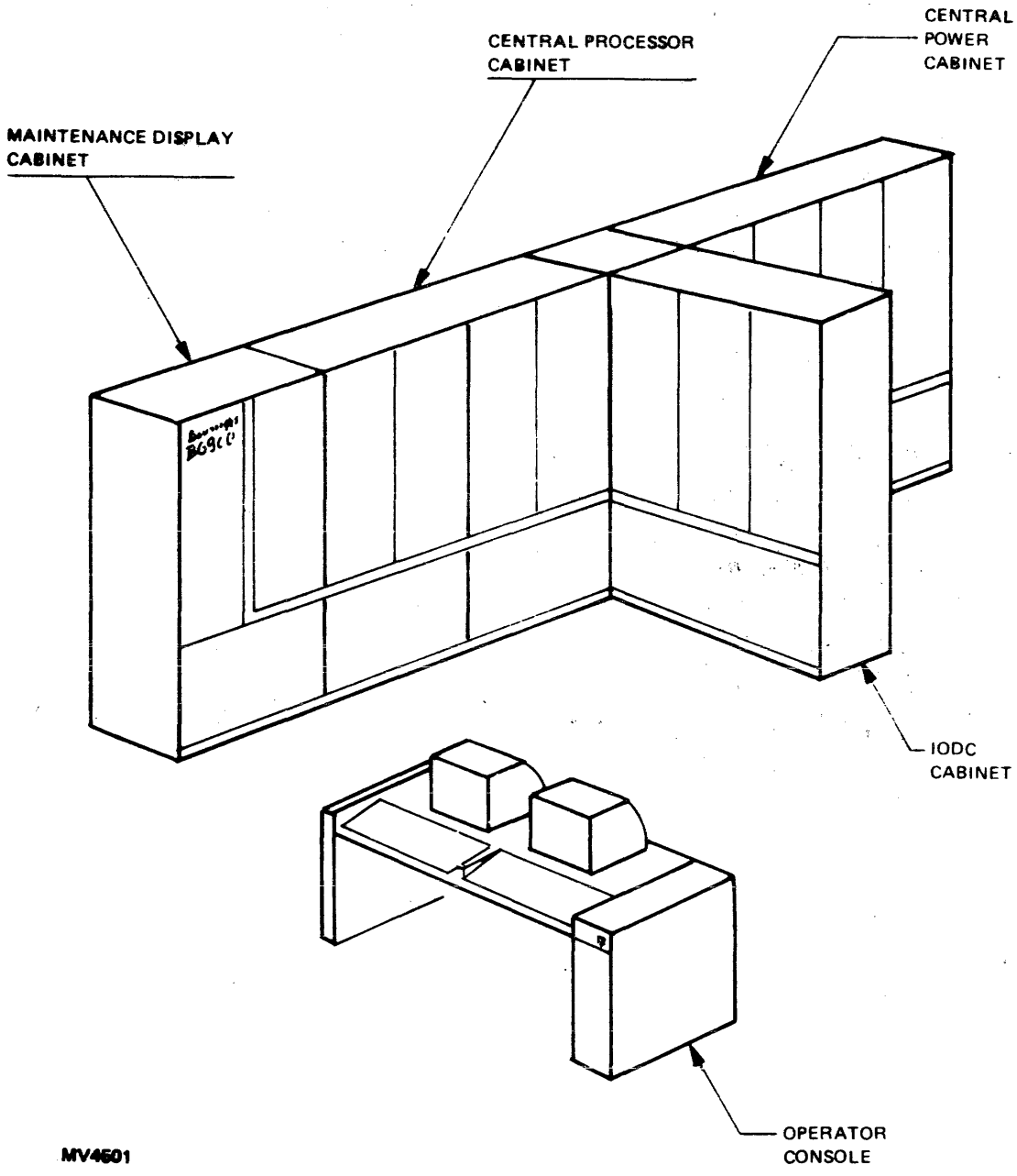


Figure 1-2. B 6900 System (Minimum Cabinets) Layout

B 6900 SYSTEM HARDWARE MODULE ORGANIZATION

The following paragraphs discuss the B 6900 system modules that are located within the system cabinets. A module in the B 6900 system is defined as a unit of hardware equipment that performs a specific function, or a set of specific functions. A module of hardware equipment in the B 6900 system is limited to a single system cabinet. Modules in separate cabinets that perform similar functions are separate modules.

A B 6900 system cabinet is not limited to a single module. The use of new types of logic circuit devices in the B 6900 have made it possible to mount within a cabinet more modules than was possible previously. Figure 1-3 is a block-diagram of the B 6900 system that shows the relationship of the modules in the B 6900 system.

B 6900 MODULE INTERFACES

Cabinets within the B 6900 system are connected together through a series of interface buses (see Figure 1-3). These buses provide a method for the transfer of information and control data between system modules.

B 6900 CENTRAL PROCESSING UNIT CABINET

The Central Processing Unit (CPU) is the heart of the B 6900 system. The CPU (see Figure 1-3) contains the data processor module, the Message Level Interface Port (MLIP) module, the memory exchange, internal local memory modules and the memory tester. The CPU generates system clock pulses that are distributed to other modules in the system. The CPU contains logic circuits that operate with the maintenance processor to perform memory testing.

The master clock frequency of the B 6900 system is 6.67 megahertz. This clock frequency produces clock pulses that occur every 150 nanoseconds. These clock pulses, distributed throughout the logic circuits of the system, are used to synchronize circuits contained in various modules of the system. In this manner, each circuit operates in concert with other circuits in the system, in an efficient and harmonious manner.

DATA PROCESSOR MODULE

The Data Processor (DP) is the key module through which the B 6900 software operating system directs and controls the resources of the B 6900 system. The DP (see Figure 1-3) initiates all operations performed by the other system modules, including the operation of all peripheral devices. The DP also performs data arithmetic operations, and manipulates data within the system. The DP contains logic circuits to sense interrupts from other modules, and also within itself. Whenever the DP senses an interrupt, the software operating system also becomes aware of the interrupt and handles the cause of it. The DP performs comparisons and other logical operations that allow the software operating system to evaluate conditions represented as data, and to make decisions based on the results of the evaluation. Because the software makes decisions, it provides the capability for altering the future course of programmed operations not only within the operating user programs, but also within the MCP itself.

The B 6900 system uses look-ahead logic in the DP. This feature fetches words of program code before the DP is ready to execute the code. As a result, it virtually eliminates the need for halting a program to fetch words of program code. The memory accesses that are performed by the look-ahead logic are independent of other memory cycles performed for the DP, and do not cause delays in obtaining data for normal DP functions. When a new word of program code is required, the first resource is the buffer circuit of the look-ahead logic. A memory cycle will be performed only if the look-ahead logic has not already fetched the word of code that is needed, or if a branch operator causes a change in the sequential program code addressing. If the next word of program code is the proper program word, and is present in the look-ahead logic buffer circuit, then circuit becomes the source from which the next word will be taken.

The DP of the B 6900 system contains an adder circuit for performing arithmetic functions. The mantissa adder circuit is a double-precision, high-speed adder. The adder circuits use micro logic algorithms for double-precision arithmetic operations.

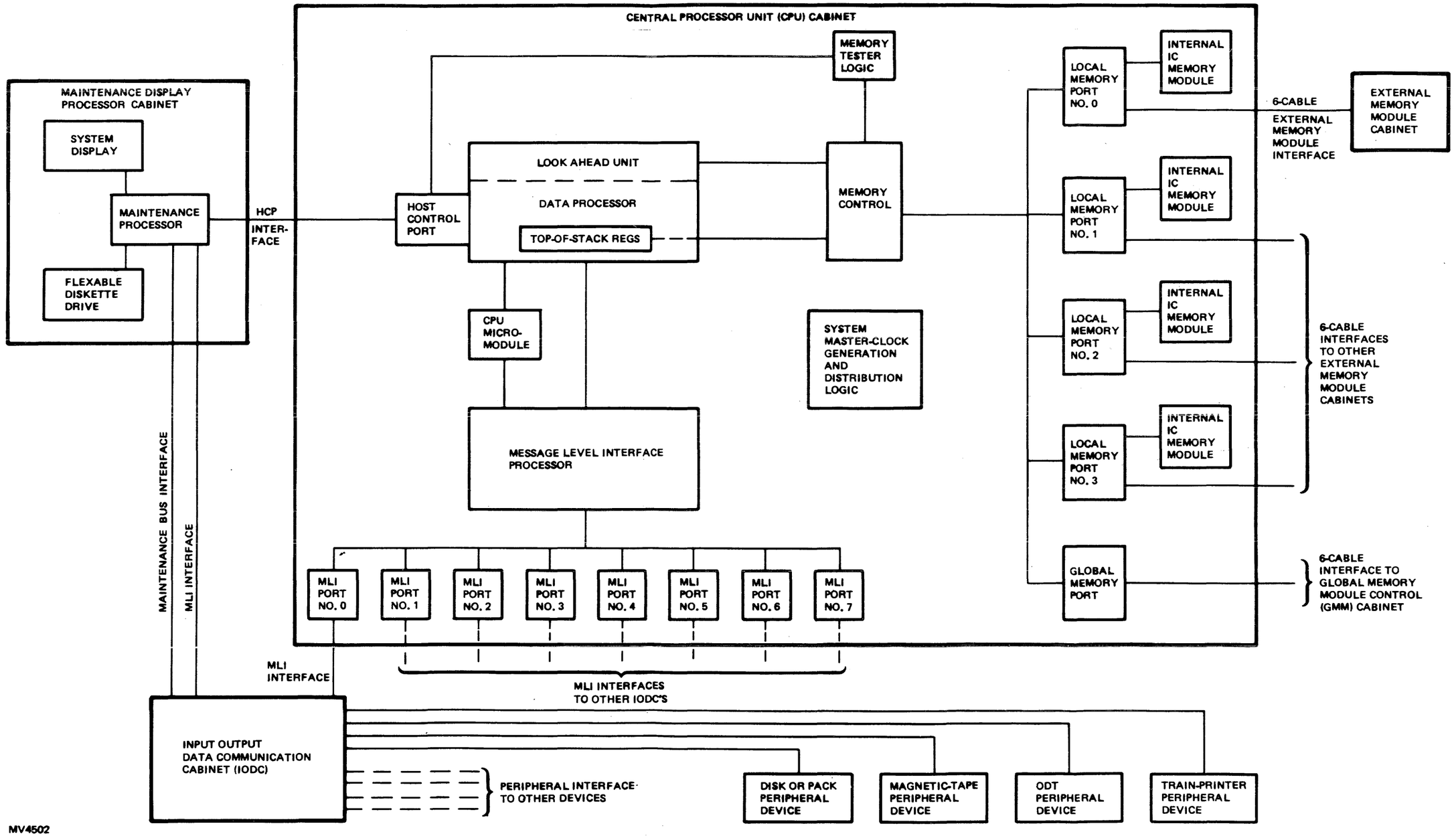


Figure 1-3. B 6900 System Module Block Diagram

The B 6900 DP contains logic circuits that provide for a retry of a DP operator that fails during its execution. This retry of failed operators is applicable only up to a predetermined point in the flow of an operator. If an operator fails and a retry is possible, then a flag is set to indicate that the retry can occur. If an operator fails and a retry is not possible, then the failure will result in the execution of the interrupt procedure for failed DP operators.

A failed operator retry operation is controlled by the system software. The system hardware indicates whether or not a retry may be attempted, but the decision to retry a particular operation is made by the software.

The B 6900 DP makes extensive use of RAM, and PROM memory integrated circuit components. Parity testing is performed on these component parts in the DP. When a failure of one of these component parts is detected, an entry is made in the Error register. The error register is decoded and written in the system log. The log entry will provide such pertinent data as

a. The location of the card package that failed.

b. The J-count sequence number and the OP code of the DP operator that was being executed when the card package failed.

The B 6900 system DP performs recursive confidence testing when the DP is in an IDLE condition. The confidence tests check circuits such as

a. The top of stack registers.

b. The shift paths for data that is placed in the top of stack registers.

c. The barrel shifter logic.

d. The mantissa adder logic.

e. The exponent adder logic.

f. The address adder logic.

g. The arithmetic operation algorithms.

h. The DP control buses.

If an interrupt occurs while the DP is performing a confidence test, the DP immediately exits from the IDLE state. If the exit is caused by an error exit, the error will be reported in the SYSTEM SUM LOG disk file.

The DP performs residue testing of the contents of the integrated circuit memory address registers. Residue testing is also performed on literal values that are used as indices to the addresses that are contained in the integrated circuit address registers. The purpose of residue testing is to increase the integrity of the address adder circuits. Residue testing is an automatic function that detects addressing errors, and cause the software operating system to make log entries that identify the nature of the error.

MESSAGE LEVEL INTERFACE PROCESSOR (MLIP)

The Message Level Interface Processor (MLIP) of the CPU cabinet (see Figure 1-3) functions to control all peripheral devices that are connected to the B 6900 system. The types of peripheral devices that may be connected to the MLIP are Universal Input/Output Data Link Processor (UIO-DLP) controlled devices. The DLPs that control B 6900 system peripheral devices are located in Input Output Data Communications (IODC) cabinets, and are connected to the MLIP module of the CPU cabinet by means of a Message Level Interface (MLI). The MLIP logic also performs various timing functions, such as Time-Of-Day and Processor Timer operations for the B 6900 system.

The IODC Module cabinet is defined in the IODC Base Module FETM, Number 1115565. Each DLP located in an IODC is a separate module. Each DLP device is documented in a separate technical manual, according to the type of peripheral device controlled by the DLP logic. The various DLP technical manuals are referenced in the IODC Base Module FETM.

The MLIP module contains provisions for as many as eight MLI interface connections to IODC modules (see Figure 1-3). Each MLI interface can connect to eight IODC modules through the use of Line Expansion Modules or LEMS. Each IODC module can contain up to eight UIO-DLP peripheral device controls; consequently, an MLIP module in a B 6900 system CPU cabinet can communicate with and control up to 512 UIO-DLP devices. The MLIP can communicate over only one of its MLI interfaces at a time. Simultaneous DLP operations by more than one of 64 possible UIO-DLPs connected to an MLI cause communication interlacing on the MLI. In the same way, communications to and from the MLIP are interlaced for simultaneous DLP operations over two different MLI interfaces.

The following list names the types of peripheral devices that may be connected to a B 6900 CPU MLIP module, by means of DLPs located in a B 6900 system IODC cabinet.

a. 300/600/800 CPM Card Reader

b. 1100/1500 LPM Train Printer

c. 300 CPM Card Punch

d. Operator Display Terminal (ODT)

e. 235 Disk Pack

f. 5N Disk File

g. 4A/5A PE Magnetic Tape

h. 206/207 Disk (Interlaced Mode)

i. 5E NRZ Magnetic Tape

j. 206/207 Disk (Sequential Mode)

k. 5G GCR/PE Magnetic Tape

l. OEM GCR/PE Magnetic Tape

m. 2000 LPM Printer (CDC Drum)

n. 225 Disk Pack

o. ICMD mini-disk

p. 750 LPM Train Printer

q. Data Communications Processor

MEMORY CONTROL MODULE

The memory control module (see Figure 1-3) operates a memory interface exchange that allows two system requestors to access one of five memory modules. The two requestors are as follows:

a. The look-ahead logic.

b. The data processor module or the MLIP module. These modules share a common requestor path to the memory control exchange, as was defined in the subsection on the MLIP.

The five memory storage module ports that may operate as respondents to the two memory control requestors are defined as follows:

a. Each of the first four modules is either a 128K or a 256K local memory module (1K = 1024 words).

b. The fifth module is an interface to the Global Memory of the B 6900 system.

In addition to controlling the interface paths through the memory exchange, the memory control module also performs memory retries, and memory read data error corrections. A read memory retry consists of detecting an error in the data fetched from memory, and causing a second memory read strobe pulse to be generated. A read memory retry is not a second memory cycle.

A memory retry is also performed when a memory module detects a parity error on the address data lines. The memory address error retry will repeat the complete memory cycle operation.

An error correction memory cycle will be performed for a read memory cycle that detects a single-bit error in the data that was stored. If a memory cycle still produces an error in the data after a read memory cycle retry has been performed then the memory control module will perform an error correction cycle. An error correction cycle can only correct single bit errors.

B 6900 MAINTENANCE PROCESSOR AND SYSTEM DISPLAY

All B 6900 systems contain provisions for control of system operations and for display of system status. B 6900 systems with low serial numbers include the Maintenance Display Processor cabinet (MDP) that provides for system control and displays system status. B 6900 systems with high serial numbers do not contain an MDP cabinet, and the functions of the MDP cabinet logic circuits are distributed to other cabinets and modules.

Figures 1-2 and 1-3 show a B 6900 system organization having an MDP cabinet present in the mainframe cabinet organization. Figure 1-4 shows how system control and status display functions are distributed in a B 6900 system mainframe organization that does not contain an MDP cabinet.

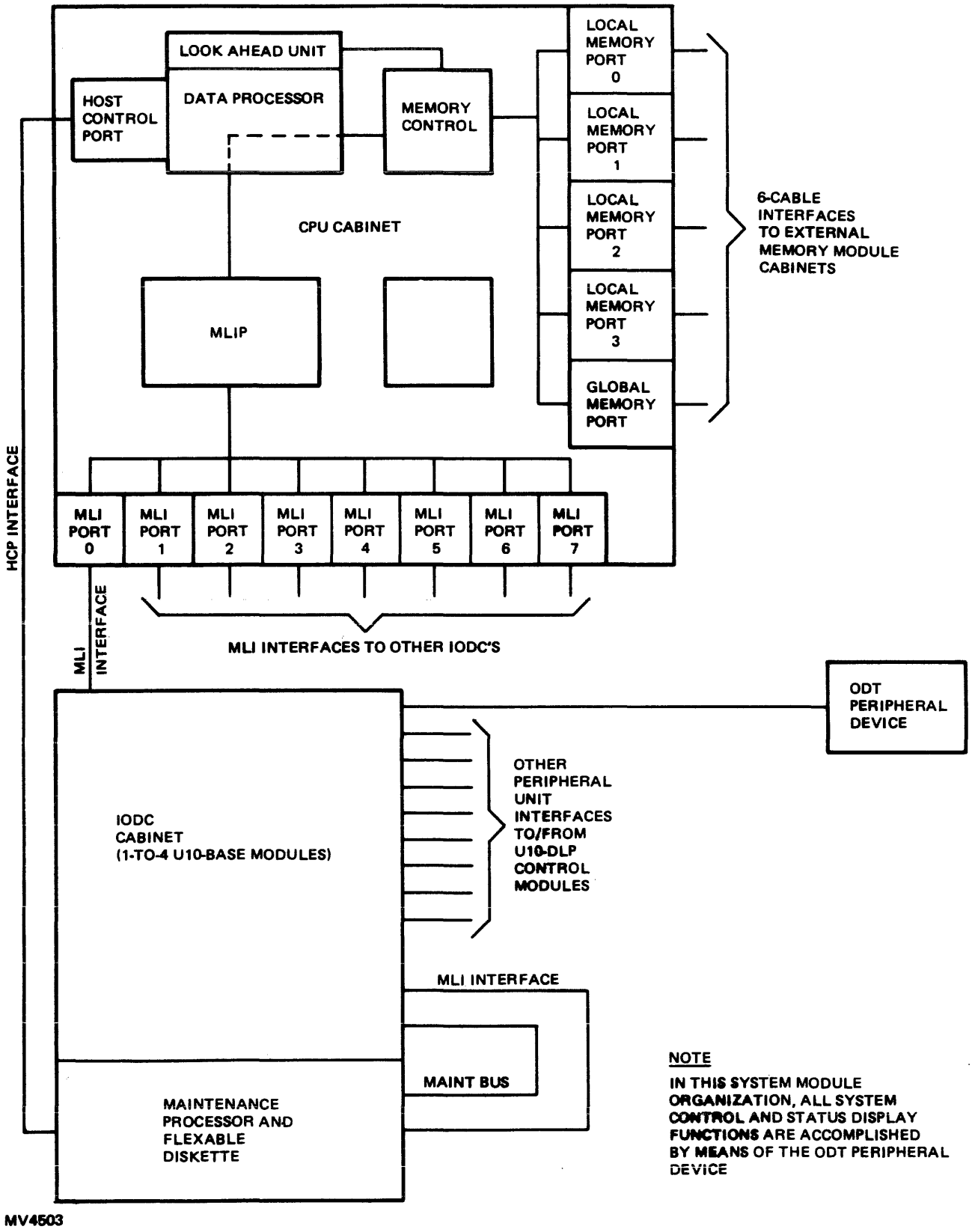


Figure 1-4. B 6900 System Module Block Diagram Without MDP Cabinet

The leading features and functions of the B 6900 Maintenance Display Processor are described below:

a. The MDP displays the states of as many as 4096 logic devices.

b. The MDP can write into and verify the code of a PROM device.

c. The MDP contains logic card package testing capability which exercise test cases for all non-discrete logic cards.

d. The MDP can operate DLP controlled I/O devices.

e. The MDP can be programmed to beam test (at single clock level) and to compare all flip-flops within the system, as well as any flip-flop that is under test.

f. The MDP can be programmed to allow a system operator to test the logic circuits of the system at the single clock level.

g. The MDP can be programmed to dynamically isolate most failures that occur in the hardware elements of the system.

The MDP cabinet can be divided into two parts:

a. The upper half of the cabinet, which contains the displays.

b. The lower half of the cabinet, which contains the maintenance processor, the display control logic, the flex-disk device, and a power supply for the cabinet.

The MDP contains a flex-diskette drive device, which is used to load initial system operating FIRMWARE into the maintenance processor RAM memory.

The upper half of the cabinet consists of a display panel and several control panels. The display is on the left-hand side of the MDP cabinet. The display panel is not always visible. To view the panels, swing-out covers must be extended. Four display registers, and various control panels are exposed to view when the swing-out covers are extended. Two switch panels are located at the bottom of the display panel. These switch panels are used to control operation of the MDP maintenance processor and CPU cabinet logic circuits.

The maintenance processor is the principal operating unit in the MDP cabinet. The maintenance processor operates in either of two modes, which are Maintenance Test Routine mode (MTR) or normal mode. These two modes are discussed in the following paragraphs.

The PROC ENABLE switch (on the MDP switch panel) is used to place the maintenance processor in the MTR mode. The MTR mode provides a way of testing the maintenance processor through the use of test-routines that are stored in PROM memory. The PROM memory is an integral part of the maintenance processor. This PROM memory contains firmware that is used

a. To test the maintenance processor circuits.

b. Test the memory interface logic between the maintenance processor and the RAM memory, which is an integral part of the maintenance processor.

c. Test the RAM memory up to a checkerboard test.

d. Test the micro-logic controllers of the maintenance processor.

e. Perform an extensive (Galpat) test on the RAM memory.

f. Load an MTR test-routine program from the flexible-diskette unit to the RAM memory of the MDP.

g. Perform a program branch to the start of the MTR test-routine that was loaded into the RAM memory.

h. Handle Interrupts that occur during the operation of the maintenance processor in MTR mode.

The same switch that was used to place the maintenance processor in MTR mode (the PROC ENABLE switch) is also used to select normal mode. The normal mode of operation provides a way to test the B 6900 system through the use of the MTR test routines that are loaded to the RAM memory. The maintenance processor uses the PROM memory to initiate the loading of MTR test routines into the RAM memory as follows:

a. Uses switches on the MDP System Control Panel to select a DLP controlled peripheral unit.

b. Provides a quick confidence check for the peripheral unit to be used.

c. Initializes the RAM memory to receive the data from the I/O device.

d. Purges the RAM memory of all parity errors.

e. Communicates with the system operator to determine which part or parts of the system MTR test program are to be loaded into maintenance processor RAM memory.

f. Loads the selected system MTR program parts into the RAM memory.

g. Performs a program branch to the start of the system MTR test routine residing in the RAM memory.

h. Handles interrupt procedures during system operation.

The maintenance processor logic contains the Keyboard/Switch/Indicator (KSI) controller, the purpose of which is to interface the maintenance processor to the control panels of the MDP. The control panels are used manually as source input devices, to direct that various functions of the maintenance processor be performed. The KSI controller coordinates and synchronizes these manual control demands with the normal logical operations of the maintenance processor. The orderly responses of the maintenance processor to a control panel demand are returned to the maintenance processor control panel for display by the KSI controller.

The PROM Write I/O controller provides a method of creating a selected bit pattern in a PROM device. In addition, a PROM device can be verified to have the correct pattern inserted.

The MDP contains three other controllers, as follows:

a. The Mainframe Input Output (MFIO) controller.

b. The Message Level Interface Input Output (MLIO) controller.

c. The UIO Maintenance Input Output (UMIO) controller.

The purpose and use of each of these three controllers is defined in the following paragraphs.

The purpose of the Mainframe I/O (MFIO) controller is to allow either the maintenance processor or the display logic to set and to sample the state of mainframe flip-flops. In addition, the maintenance processor can monitor various conditions within the controller through use of status and data transfers. The MFIO controller interfaces the logic of the MDP with one of two connectors that are identified as normal, and as alternate interfaces. The PROC ENABLE switch selects either the maintenance processor or the display logic to control the data lines between the MDP and the CPU cabinet.

The maintenance processor uses a set of command words and fixed format status reports to control the operation of the MFIO controller. These controller directing commands and status reports are passed between the maintenance processor and the MFIO logic over the DIN and DOUT lines of the MFIO interface bus.

When the PROC ENABLE switch is in the ENABLED position (UP), the maintenance processor is permitted to control data that is sent to the CPU cabinet and, therefore, to control the setting of mainframe flip-flops. When the PROC ENABLE switch is in the DOWN position, the display logic controls the data sent to the CPU cabinet and, consequently, the setting of mainframe flip-flops.

The purpose of the Message Level Interface Controller (MLIO) is to provide the maintenance processor with a way to communicate with the peripheral units that are attached to the system. The MLIO controls an MLI interface bus between the IODC cabinet and the MDP. The MLIO controller contains a 1024 byte IC memory buffer that is used to hold data received from an I/O device. The MLIO controller can initiate different I/O devices, but only one I/O operation can be in process at any one time. MDP cabinet system control panel switches are used to select those I/O devices the MLIO controller can initiate. The maintenance processor uses a set of command words and status reports to control the MLIO controller, and the MLIO interface bus to the IODC cabinet.

The purpose of the Universal Maintenance Input Output (UMIO) controller is to connect the maintenance processor to system IODC Base-module maintenance card-packages. This interface provides for FIRMWARE programs executed by the maintenance processor to initiate and control maintenance tests on system peripheral devices.

DISPLAY CONTROL LOGIC

The display control logic (see Figures 1-5 and 1-6) is controlled by the maintenance processor.

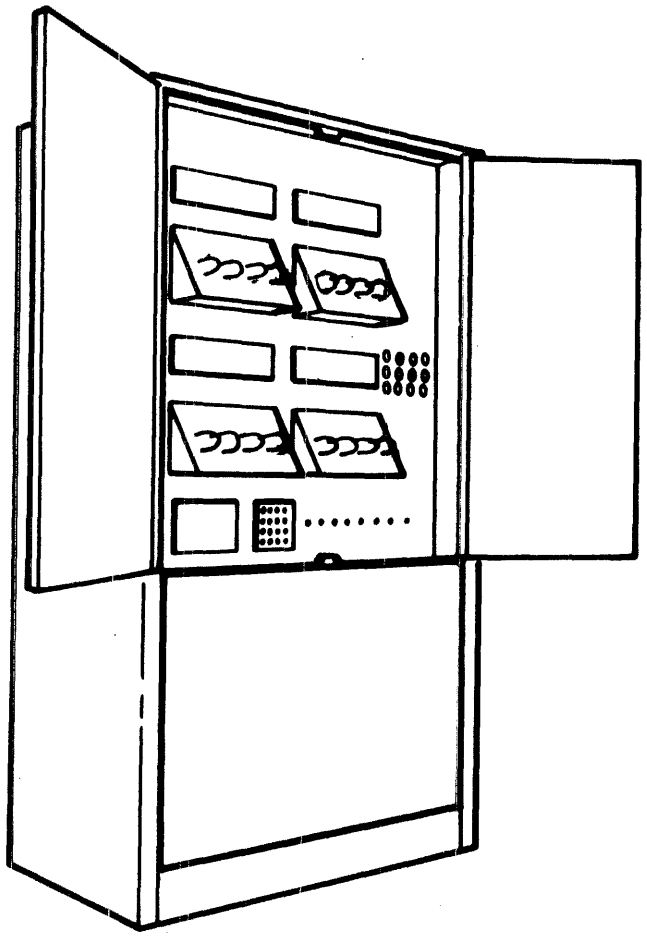
If an MDP cabinet is installed in a B 6900 system the maintenance display indicates circuit device status, and causes the circuit devices to SET or to RESET.

If an MDP cabinet is not installed in a B 6900 system the status of system circuit devices is displayed on an Operator Display Terminal (ODT) peripheral device screen. The maintenance processor executes a FIRMWARE "SOFT-DISPLAY" program which controls the display of system status on the ODT screen. The SOFT-DISPLAY program is executed in response to an input message which is written on the ODT screen. Other input messages on the ODT screen are used to cause circuit devices in the system to be SET or RESET.

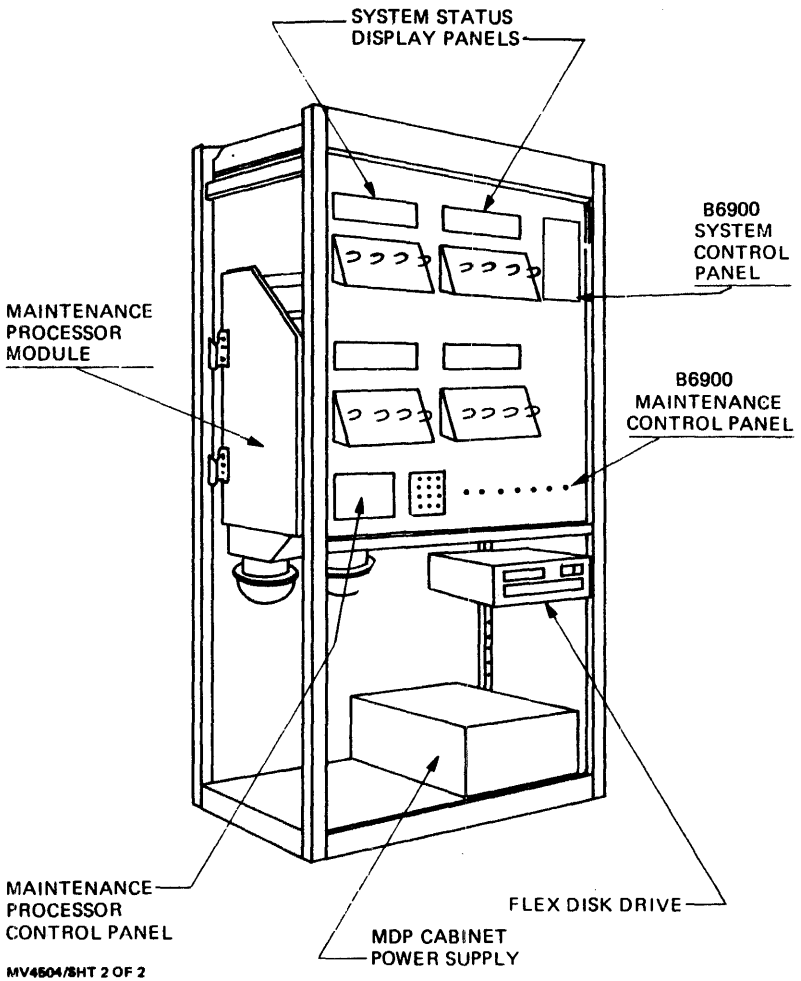
B 6900 CENTRAL POWER SUPPLY CABINET

The Central Power Supply Cabinet (PSC) of the B 6900 system is an A size cabinet (see Figure 1-6), which provides centralized power to all cabinets within the B 6900 system except for independently powered cabinets. Power buses route the power generated in the PSC to other cabinets in the B 6900 system. The source power to the B 6900 system PSC is discussed in the B 6900 System Installation Planning Manual, number 5011364.

The power supplies in the B 6900 system PSC are capable of supplying electrical power to the mainframe cabinets of the system. The power supplies in the PSC use constant voltage transformers, that provide sufficient pre-regulation conditions to ensure constant voltage outputs with a loss of input power of up to 30 percent of normal line supply. These design characteristics in the PSC provide for continuous system operation during "brown-out" operations. A "brown-out" is defined as a reduction by as much as 15 percent of normal operating line voltage, and for an unspecified period of time.



MV4504/SHT 1 OF 2



MV4504/SHT 2 OF 2

Figure 1-5. Maintenance Display Processor Cabinet

Figure 1-6 shows the major parts of the PSC, and the relative location of these parts within the cabinet. Figure 1-7 shows the power bus distribution between the PSC, and other cabinets within the B 6900 system mainframe.

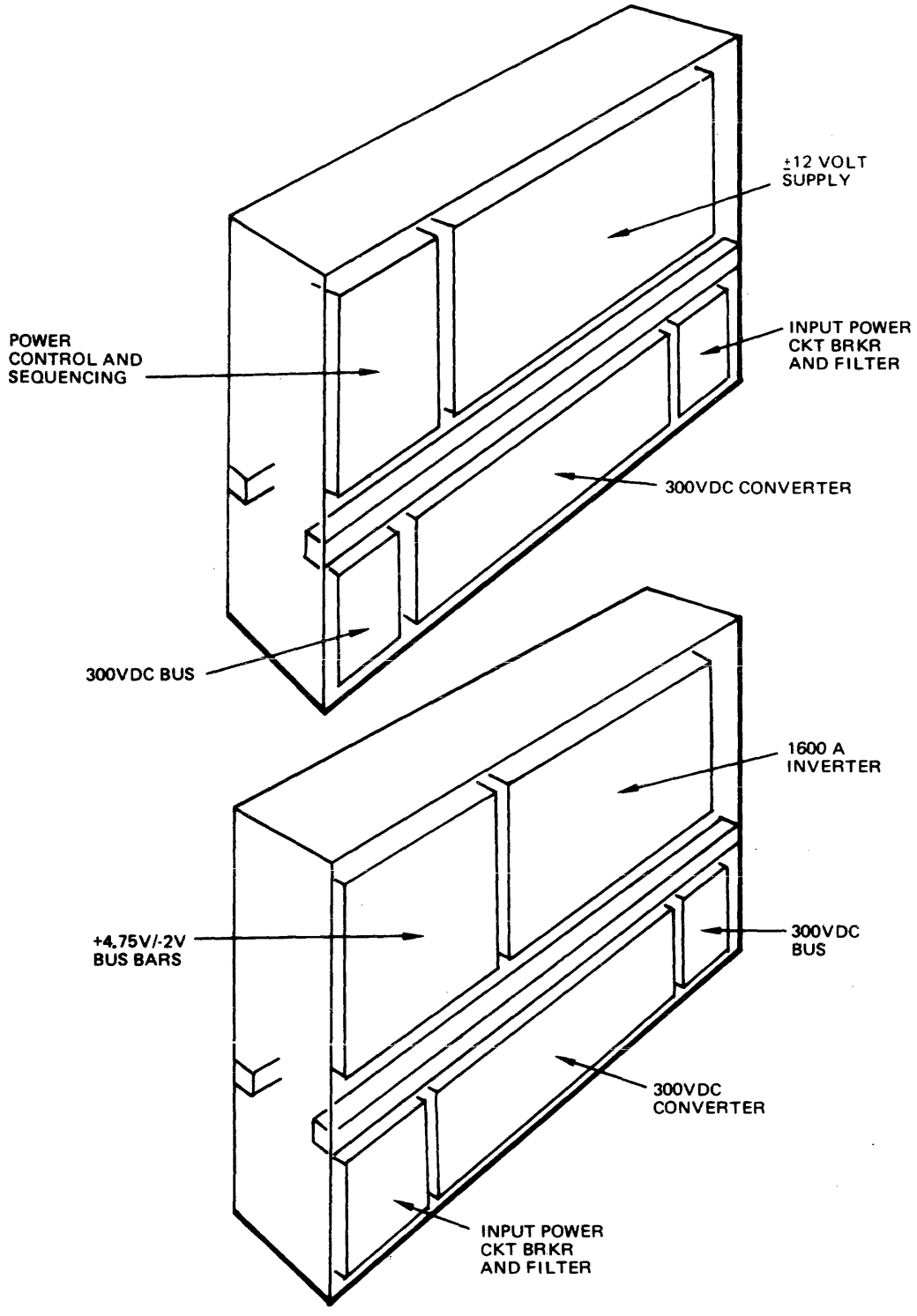


Figure 1-6. Central Power Cabinet

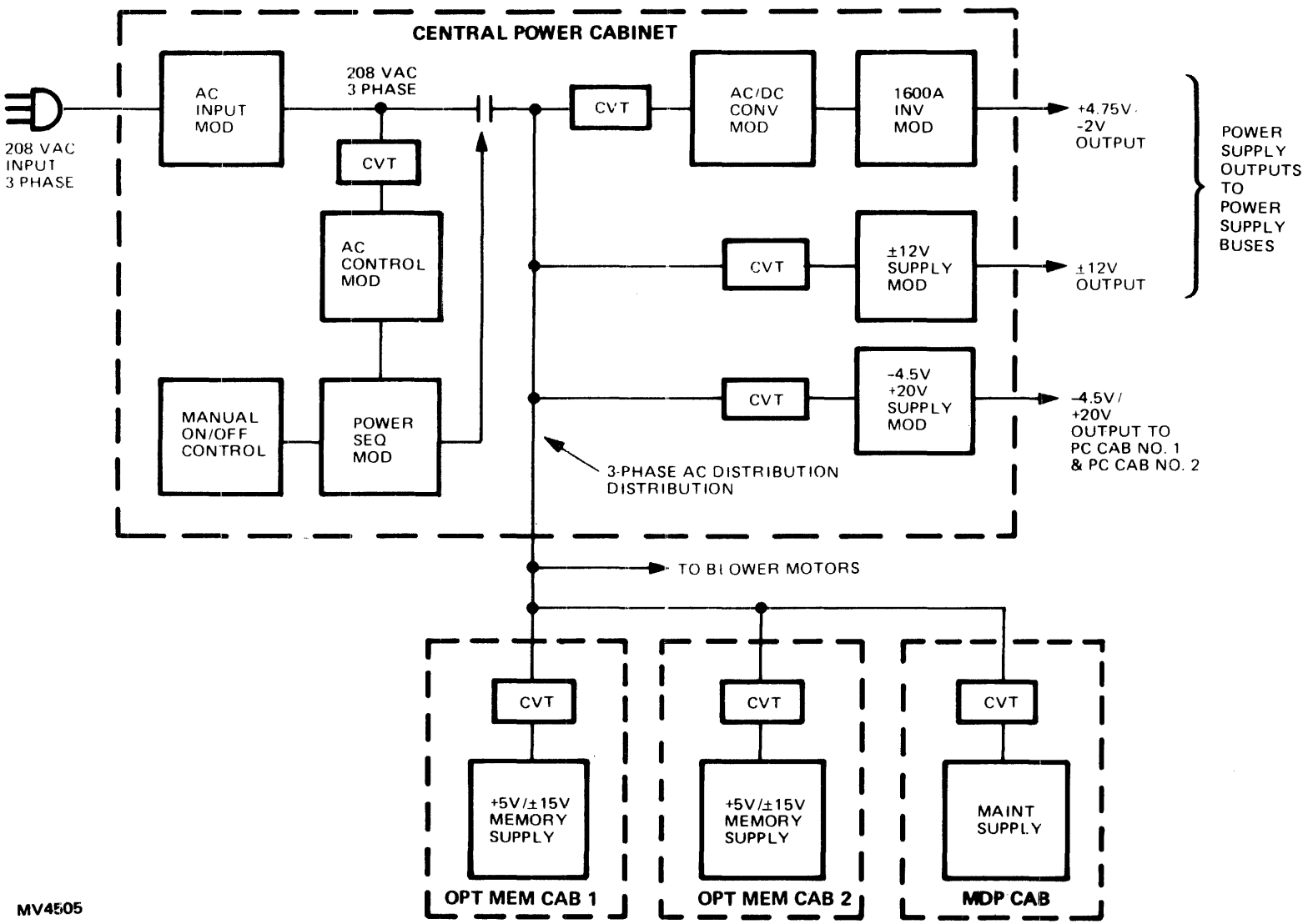


Figure 1-7. B 6900 Power Subsystem Distribution Diagram

INPUT OUTPUT DATA COMMUNICATION (IODC) CABINET

In low serial-numbered B 6900 systems an IODC cabinet is a 3-quarter size cabinet (see Figure 1-8) that contains from 1-to-4 Universal Input Output Base (UIO-Base) modules. In high serial-numbered B 6900 systems an IODC cabinet may be an A-size cabinet (see Figure 1-9) which contains a maintenance processor module as well as 1-to-4 UIO Base modules.

There are two kinds of IODC cabinets, system-powered cabinets, and independently-powered cabinets. A B 6900 system must contain one system-powered IODC cabinet. If a B 6900 system contains multiple IODC cabinets, the first IODC cabinet is system-powered, and all other IODC cabinets are independently-powered. If a B 6900 system does not contain an MDP cabinet, the first IODC cabinet is A sized, system-powered, and contains a maintenance processor module.

A B 6900 CPU can interface to 8 UIO-Base modules by means of the MLI ports in the MLIP module. Each UIO-Base module can be inter-connected to 7 other UIO-Base modules by means of UIO Line Expansion Modules (LEMS)s. Therefore, a B 6900 CPU can interface to a maximum of 64 UIO-Base modules, located in up to 64 IODC cabinets.

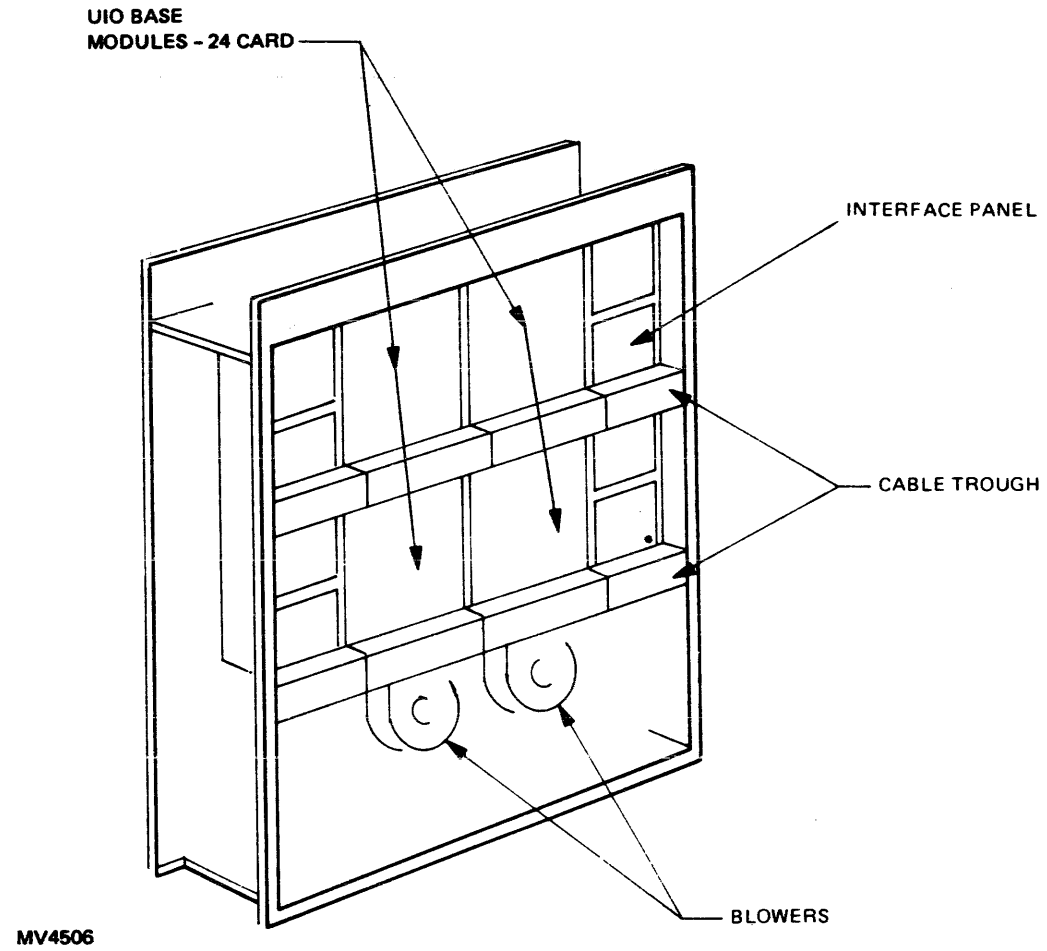


Figure 1-8. IODC Cabinet (3/4 Size)

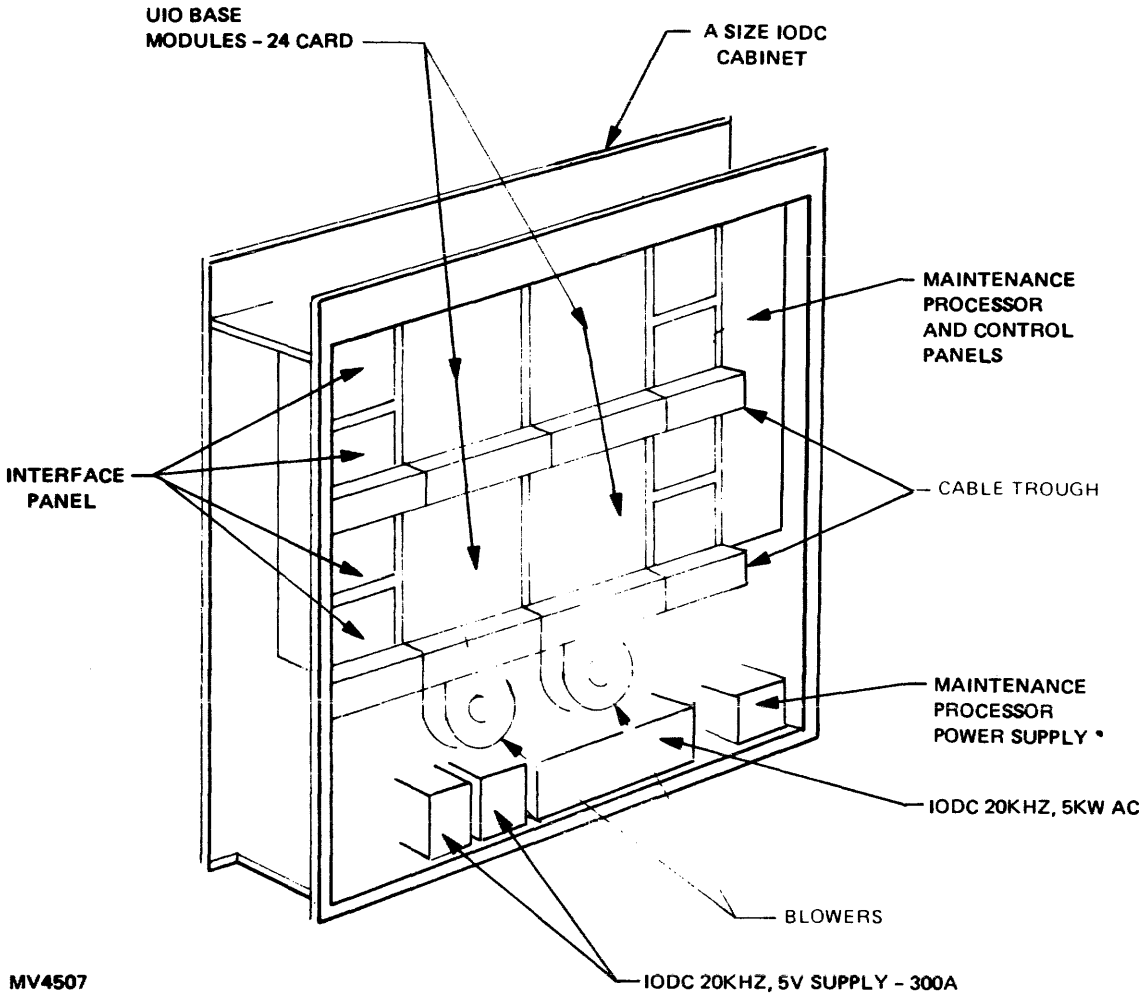


Figure 1-9. IODC Cabinet (A Size)

B 6900 Memory Cabinets

The B 6900 system CPU cabinet contains provisions for installing 1-to-4 Integrated Circuit (IC) local memory modules. Memory modules mounted in a CPU cabinet contain 128K words of storage capacity. The use of internal IC memory modules is optional.

From 1-to-4 external memory modules (in independently powered cabinets) can be connected to a B 6900 CPU cabinet by external cable interface connections. Externally connected memory modules may be IC memory of PLANAR CORE memory modules which contain 128K words of storage capacity. The use of external memory modules is optional.

Global Memory (TM) is optional in a B 6900 system. If Global memory is installed, it is interfaced to the CPU cabinet by an external cable interface; therefore, it must be independently powered.

A B 6900 system must have a minimum of 128K words of memory available to the CPU, and may access a maximum of 1000K words of memory. The memory resources of a B 6900 system may be any combined mixture of local/global memory, from the minimum to the maximum number of words.

The B 6900 Planar Memory Cabinet (refer to Figure 1-10) is an optional independently powered B size cabinet that can contain a maximum of 256K words of local memory. With a maximum of two Planar memory cabinets in a B 6900 system, a maximum of 512K words of local Planar memory is available to the system. Local Planar memory is expandable from 128K words to 512K words, in increments of 128K words. In the common context, one K of memory is actually 1024 words in length.

A B 6900 I/C Memory Cabinet (see Figure 1-11) is an optional B size independently powered cabinet that can contain up to 512K words of memory. The I/C memory is installed in modules of 128K words, up to a maximum of four such modules. If only I/C memory is installed in a B 6900 system, then the system contains a single optional memory cabinet.

Each word of memory consists of 60 bits. These 60 bits are divided to provide 51 bits of data, one parity bit, and eight bits which are utilized for error detection and correction.

A B 6900 memory interface consists of six cables. Figure 1-12 shows these six cables, and how they operate to provide the interface between the memory control module of the CPU cabinet, and B 6900 memory modules.

The B 6900 memory modules are capable of performing one of three types of operations as follows:

a. Read/Restore operation

b. Clear/Write operation

c. Read/Modify/Write operation

A memory Read cycle is the minimum time that must occur between two consecutive Initiate Memory Cycle (IMC) pulses. A Read/Restore memory operation, or a Clear/Write memory operation may be performed in the time given for a memory Read cycle. A Read/Modify/Write memory cycle requires a longer memory cycle time because this operation requires that both a memory Read, and a memory Write function must be performed (two IMC pulses are required) to complete a memory cycle.

A Read/Modify/Write memory cycle accepts input data, and a memory address from the memory requestor. A memory cycle is performed on the address specified, and the data present at the address is made available to the memory requestor.

A Read/Modify/Write operation in the memory control may be changed into a Read/Restore operation under either of the following conditions:

a. A protected memory operation is in progress, and the data in the word addressed by the Read part of the Read/Modify/Write operation determines that the memory protect bit (bit 48) is true. If this condition exists, the data Readout of the memory address is rewritten into the same address, and the Memory Protect Interrupt is detected by the memory control.

b. A parity error occurs during the read part of the Read/Modify/Write operation. If this condition exists after a Memory Retry has been attempted, then the data with the parity error is rewritten into the same address, and the Memory Parity Error Interrupt is detected by the requesting function.

If the memory control does not detect a memory protect interrupt, or a parity error interrupt during the read part of a Read/Modify/Write operation, then the operation continues as follows.

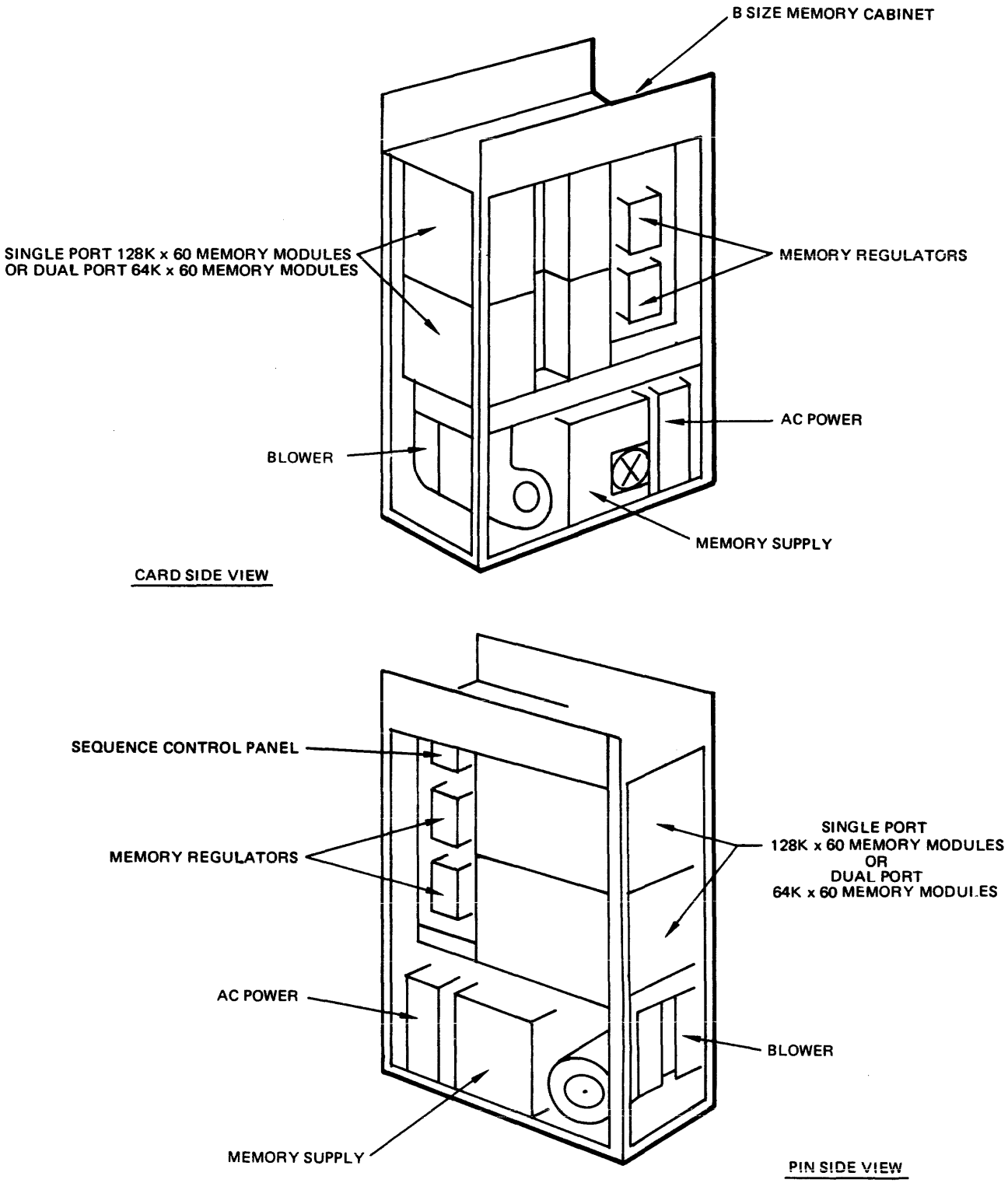


Figure 1-10. B 6900 Planar Core (Optional) Memory Cabinet

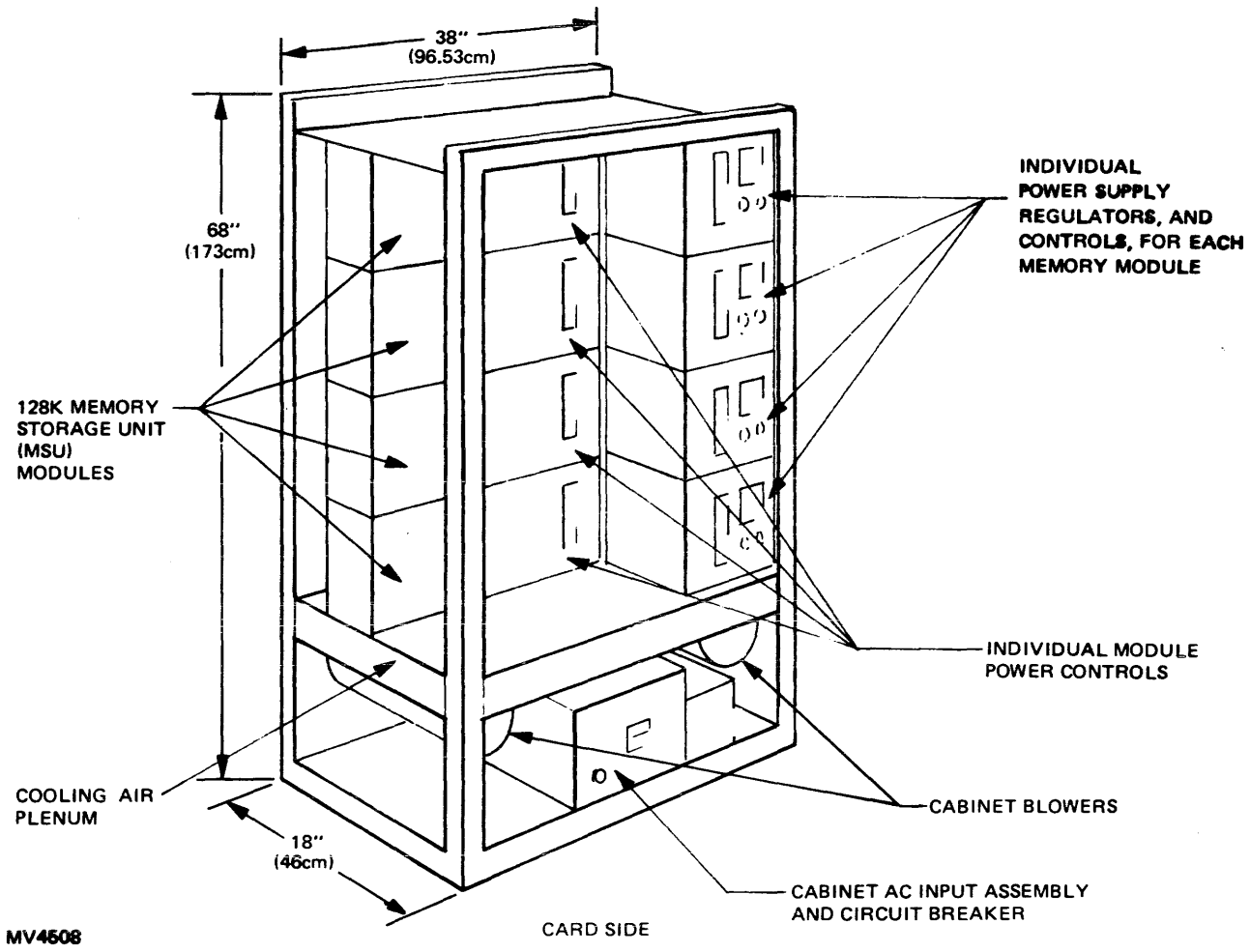


Figure 1-11. B 6900 IC Memory (Optional) Cabinet

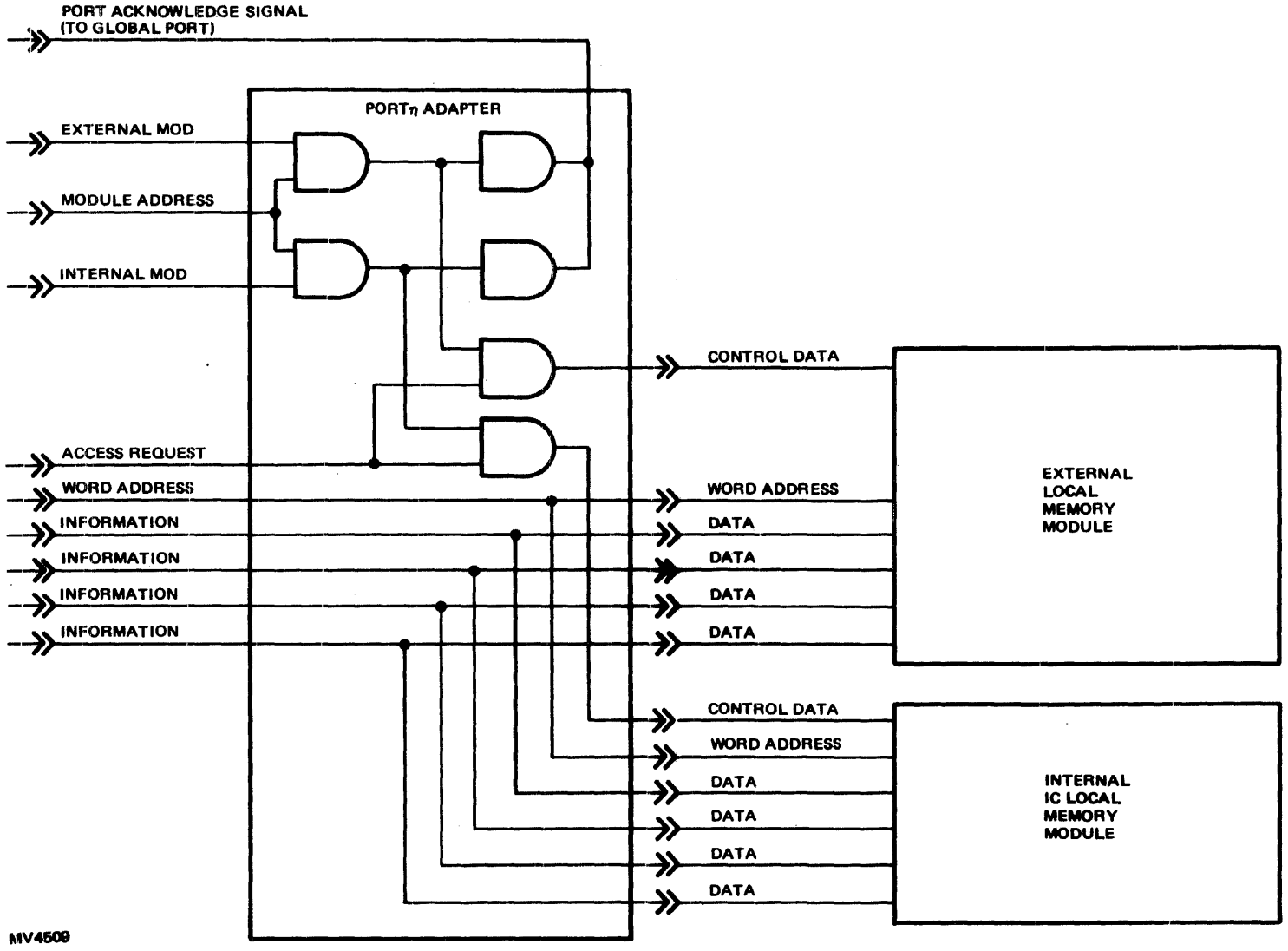


Figure 1-12. Memory Port η Module Interfaces

The data that was accepted by the memory module is written into the same address from which the memory read operation was performed and thus, the original data is destroyed. The B 6900 system uses the Read/Modify/Write mode of operation to perform normal memory write functions.

A Read/Restore memory cycle accepts an address from the memory requestor, a read memory cycle is performed on the address specified, and the data that is present at the address is made available to the memory requestor. The same data that was present in the specified address is written back into the specified address. The B 6900 system uses the Read/Restore mode of operation to perform normal memory read functions.

A Clear/Write memory cycle accepts an address from the memory requestor, and writes a requestor supplied data word into the address. Changing the Clear/Write operation into a Read/Restore operation (for a parity error), is analogous to that change previously defined for the Read/Modify/Write operation.

B 6900 OPERATORS DISPLAY CONSOLE

The purpose of this console (see Figure 1-2) is to provide a position where all necessary system operating controls are collected in one physical place. Collecting the normal operating controls into a single central location is efficient, and provides a logical place for the system operational staff to function.

There are two parts to the operators display console (see Figure 1-13), in addition to the tabletop work area. The two parts of the console are the video display and the keyboard for the video display. The video display terminal sets on the tabletop. The system control panel is part of the keyboard for the terminal, and is mounted in front of the display screen.

The operators display console contains two separate operator stations. Full control of the system is possible from only the left-hand station of the console because the right-hand station does not include a system control panel. A locking device is installed for each operators station. The locking device is a security feature used for system integrity. When the device is locked, the keyboard is disconnected, and the operators station cannot communicate with the software operating system. The locking device is activated by the use of a hand key that must be turned to open or lock the operator's console station keyboard. The locking device has no effect on the system control panel, and the controls on the panel may be operated regardless of whether the keyboard is locked.

The controls for the video display (see Figure 1-14) consist of a thumbwheel type adjustment, and an ON-OFF switch for the video display. The purpose and use of the video display controls are as follows:

a. The ON-OFF switch. This switch controls the power utilized by the video display.

b. The BRIGHTNESS thumbwheel controls the lighting intensity of the video display.

The controls for the B 6900 system control panel consist of eight indicator/switch pushbutton controls shown in Figure 1-13. The purpose and use of the B 6900 system controls are as follows:

a. The ENABLE pushbutton switch allows the use of the HALT, POWER ON, and POWER OFF pushbutton switches. If the ENABLE pushbutton is not depressed then the three other pushbuttons listed are inoperative, and have no effect on System operation. If the ENABLE pushbutton is depressed then the other three pushbuttons listed are enabled, and depressing any one of the pushbuttons will cause the circuit corresponding to the switch to be activated. The purpose of the ENABLE pushbutton is to prevent accidental system operation caused by inadvertently depressing one of the pushbutton controls listed.

b. The POWER OFF pushbutton is used to remove source power from the circuits of the system that are supplied power from the central power supply cabinet. The POWER OFF pushbutton does not remove power from circuits that receive their source power from some other source.

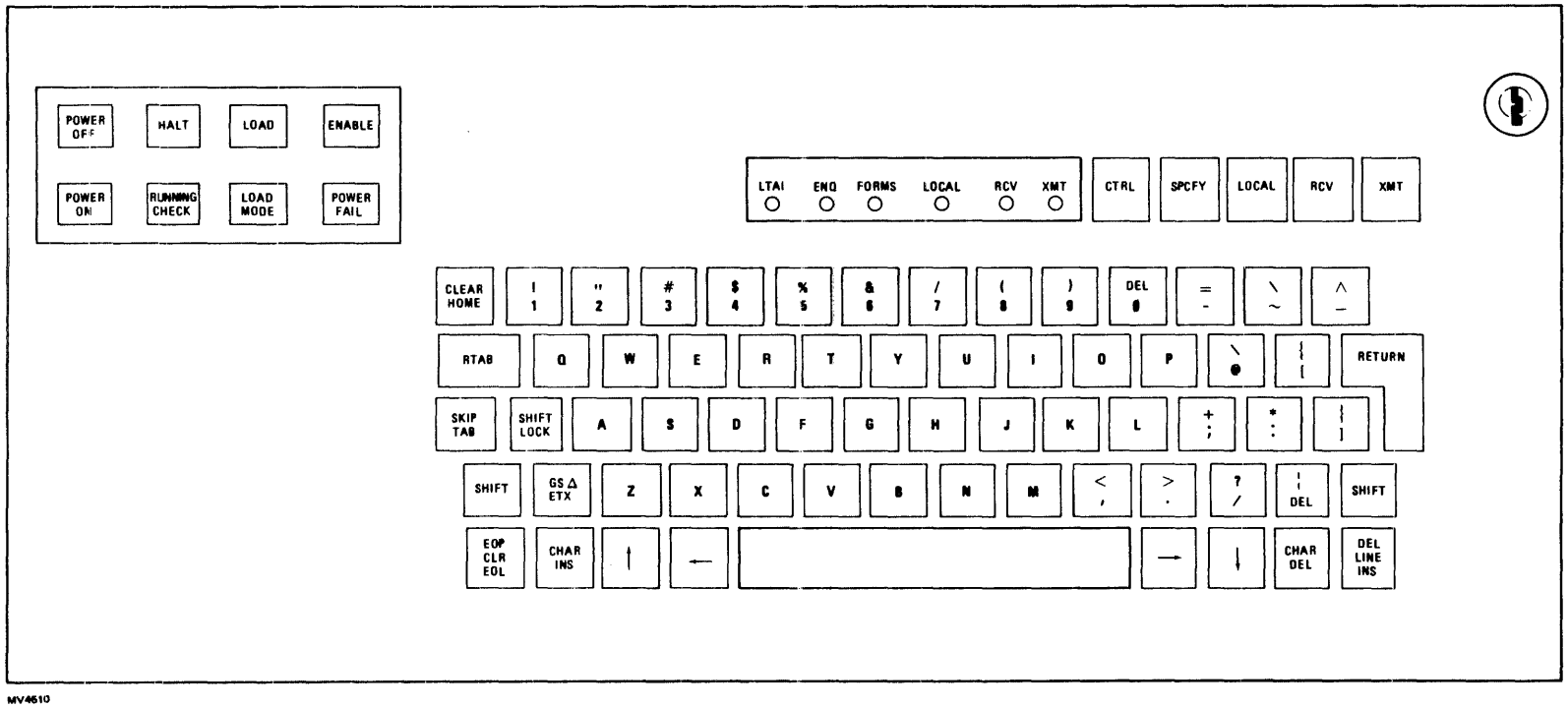


Figure 1-13. Left-Hand System Operators Keyboard

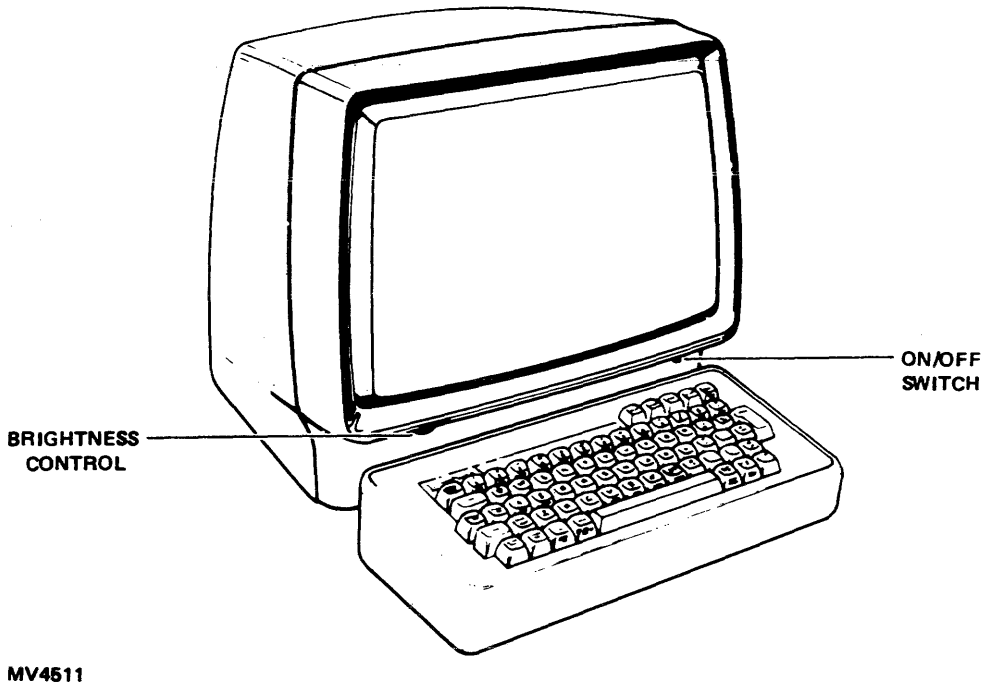


Figure 1-14. B 6900 Operators Console Video Screen

c. The POWER ON pushbutton is used to apply source power to the B 6900 system cabinets that derive their power input from the central power supply cabinet. The POWER ON pushbutton does not provide a method for applying source power to cabinets and peripheral units that do not derive their source power from the central power supply cabinet.

d. The HALT pushbutton is used to stop the B 6900 system at the end of the current machine language operator that is in process.

e. The LOAD pushbutton is used to cause the B 6900 system to initiate a Halt/Load sequence of operations. When the LOAD pushbutton is depressed the B 6900 system logic is general cleared (Set to the binary zero condition). When the pushbutton is released the Load operation is initiated. The Halt/Load sequence is a predetermined set of operations that results in the software operating system being placed in control of the system hardware.

f. The LOAD MODE pushbutton is used in conjunction with the LOAD pushbutton, to control the Halt/Load sequence of operations. If the LOAD MODE pushbutton is illuminated, and a system Halt/Load sequence is initiated (by depressing the LOAD pushbutton), then a Load operation proceeds from a predetermined peripheral device. If the LOAD MODE pushbutton is not illuminated when the LOAD pushbutton is depressed, then the Load sequence proceeds to perform a load operation from an alternate peripheral device. The selection of either device from which to perform a system Load operation depends on whether the pushbutton is illuminated or extinguished.

g. The RUNNING/CHECK indicator lamp illuminates dimly when the system is operating. The purpose of the RUNNING indicator is to provide an indication of whether or not the system is capable of responding to certain stimuli during system operations. A RUNNING indication is necessary because under certain conditions there is no other visible way to determine whether the system is trapped in a perpetual operating loop.

   The RUNNING/CHECK lamp illuminates brightly if a CHECK condition (FAULT) is detected by the maintenance processor.

   If the RUNNING/CHECK lamp is extinguished, the system is not RUNNING and a CHECK condition has not been detected by the maintenance processor logic.

The operators keyboard (Figure 1-13) is used by a system operator to input commands and data to the operating system. The operators display console and keyboard are commonly referred to as an Operators Display Terminal (ODT), or alternately as a Supervisory Printer Output (SPO).

When the security lock mechanism for system integrity is engaged, the keyboard is disabled, and has no effect on system operations. However, if the keyboard is disabled, but the video display switch (Figure 1-14) is in the ON position, the video screen will display status messages and other pertinent data about current system oeprations.

The operators display video screen (Figure 1-14) is used to pass communications between the B 6900 operating software system and one who operates the system manually. The display screen is similar to a home television receiver, except that the display screen can display only characters and numbers, not pictures. The only sound that the display is capable of making is the bleep tone used to gain the operators attention when the software operating system needs a response from the operator.

When the operator needs to communicate with the operating system, the keyboard is used to write data which is displayed on the screen. The screen is capable of displaying 1920 characters, arranged in a matrix that consists of 24 rows of characters. Each row contains 80 character positions. A cursor blinks at the position that the next character will occupy. If the next character position contains a valid character then the valid character blinks, but if the next character position is not occupied then the cursor illuminates the character position, and causes the illuminated position to blink. The cursor moves from left to right, and from top to bottom on the screen. The display screen has automatic line-feed, and carriage-return features so that the operator is not required to control these functions. When the operator writes data on the screen, the last character written is the End-Of-Text (▮) special character. This special character is used to indicate where an input message terminates.

SECTION 2

DATA REPRESENTATION

GENERAL

All data in the B 6900 system is in binary form. The basic unit of data is the memory word (see Figure 2-1), which consists of 60 consecutive binary bits. All words of data in the B 6900 system have four distinct parts: the check-bit field, a parity bit, a tag field, and the information field. The 60 bits in a word are numbered for identification.

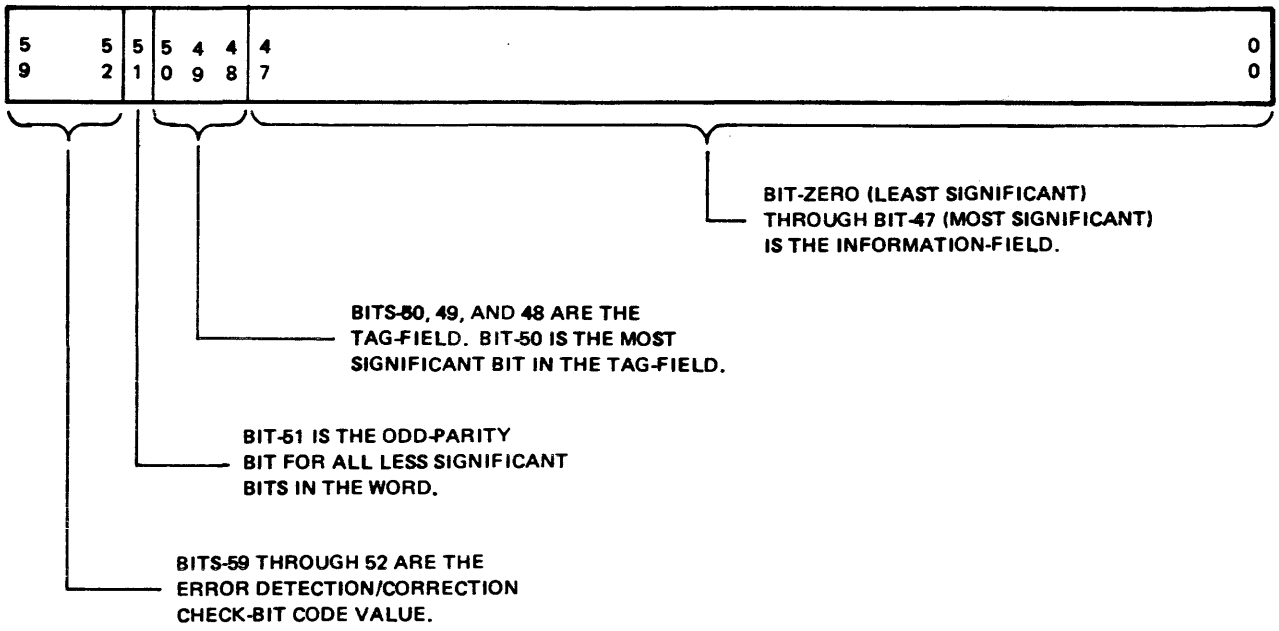


Figure 2-1. B 6900 Word Structure

Bits 52 through 59 are the Error Detection/Error Correction field. These bits are not available to a system user; they are intended for internal system use only. The purpose of these check-bits is to provide a method for detecting single-bit errors in a memory word, and for correcting single-bit errors. Multiple-bit errors may be detected by the system Memory Controller, but cannot be corrected.

The B 6900 Memory Controller inserts check-bits into a word as it is written into memory. When a memory word is read, its check-bits are used to detect bit-errors and to correct any single-bit error that is detected. A check-bit code is part of a data word only while the word is present in system memory and in the Memory Controller logic. Data words not present in system memory or Memory Controller logic circuits are tested for errors by means other than check-bit codes.

Bit number 51 (the most significant bit in a word) is the parity bit. The parity bit is used to represent the odd parity of the word. If the number of binary ONES present in the tag field and in the information field is an even number then the parity bit is a binary one value. If the number of binary ONES present in the tag field, and the information field is an odd number, then the parity bit is a binary ZERO value. The B 6900 system uses the parity bit to monitor the quality of data in a word. Logic circuits in the B 6900 system count the number of bits in a word, and compare the count against the parity bit state. If the result of the comparison is not equal, then the B 6900 system recognizes

that a parity error has occurred. The process of parity checking is an automatic feature of the B 6900 system. The parity bit for a word is not directly available to the user of the system because it is only used when words are transferred from one module to another. Data that is internal to a module has already been tested for parity.

Bits 50, 49, and 48 are the tag field. The tag field is used to identify the type of interpretation that is to be applied to the data that is present in the information field of the word. There are eight different values that may be present in the tag field, and each value specifies a different interpretation to be used. The meaning of the tag field values are as follows:

TAG (50)	FIELD (49)	BITS (48)		MEANING
0	0	0	–	A tag field of ZERO indicates that single-precision data is present in the information field of the word.
0	0	1	–	A tag field of ONE indicates that the information field contains an indirect address, not data.
0	1	0	–	A tag field of TWO indicates that double-precision data is present in the information field of the word.
0	1	1	–	A tag field of THREE indicates that a control word is present in the information field of the word. There are several different types of control words used in the B 6900 system. These types of control words are discussed individually, later in this section of this manual.
1	0	0	–	A tag field of FOUR normally indicates that a step index word is present. The meaning and use of a step index word is discussed later in this section of this manual.

NOTE

A special use for a word that has a tag of FOUR may be invoked by the MCP when a fault condition is to be handled by a user program.

The compiler will place a word with a tag of FOUR in the stack as a flag word. This flag is used to indicate that the program using the stack is responsible for handling one or more of the interrupts that may occur when the program is executed.

This special use for a word with a tag field of FOUR is only invoked when the programmer of the user program specifies that the user program is responsible for interrupt handling. The compilers that utilize this special case are the ALGOL, FORTRAN, ESPOL, and the PL/I compilers.

1	0	1	–	A tag field of FIVE indicates that a descriptor word is present. The meaning and use of a descriptor word is discussed later in this section of this manual.
1	1	0	–	A tag of SIX indicates that a software control word is present. The meaning and use of a software control word is discussed later in this section of this manual.
1	1	1	–	A tag of SEVEN indicates that a program control word is present. The meaning and use of a program control word is discussed later in this section.

This manual uses a convention to refer to data bits in a word. The rules of this convention follow:

a. A data field within a word is represented by two numbers separated by a colon character and enclosed in brackets.

b. The meaning of the two numbers enclosed in the brackets is as follows:

   1. The first (left-most) number identifies the most significant bit in the field of data bits.

   2. The second (right-most) number identifies the number of bits that are contained in the field of data bits (including the most significant bit, which was identified in rule b.1 above).

c. Bits in the tag field are not included in the field unless the most significant bit (rule b.1., above) is one of the tag field bits.

d. All bits in the information field are considered to "wrap-around" the word in such a way that the next least significant bit after bit ZERO is bit 47.

Examples of this convention are as follows:

Bits [50:3] (the tag field) – Beginning with bit 50 for three bits, or bits 50, 49, and 48.

Bits [06:9] (a data field) – Beginning with bit 06 for 9 bits, or bits 06, 05, 04, 03, 02, 01, 00, 47, 46.

Bits [47:48] (a data field) – Beginning with bit 47 for 48 bits, or all of the information field.

The convention that was stated in the previous paragraph is used to further define the bits that make up the information field of the B 6900 system words. There are 48 bits in this field: bit 47 is the most significant bit, and bit ZERO is the least significant bit.

INTERNAL CHARACTER CODES

The only internal code that is used in a B 6900 system is Extended Binary Coded Decimal Interchange Code (EBCDIC). EBCDIC is an 8-bit alphanumeric code containing four zone bits, followed by four numeric bits. The character code used for Data Communications Subsystems (external character code) is the American Standard Code for Information Interchange (ASCII). ASCII may be a 6-bit, 7-bit, or 8-bit alphanumeric code. Within the B 6900 system, EBCDIC codes may be compacted by deleting the zone bits, and by retaining the numeric portion of the character. When data in the B 6900 system is compacted it is said to be packed.

Appendix C of this manual lists the character codes of the character sets that are used in the B 6900 system. Appendix D gives the card codes that are required to produce an EBCDIC, or hexadecimal coded character representation.

NUMBER BASES

Number bases used in the B 6900 system are base 10 (decimal), base 16 (hexadecimal), base 2 (binary), and base 8 (octal) (see Figure 2-2). Because the system utilizes various of these number bases in performing its functions, it is necessary that the user of the system be familiar with the number bases, and know how to convert a value from one number base to any of the other number bases. A brief discussion of the number systems used follows.

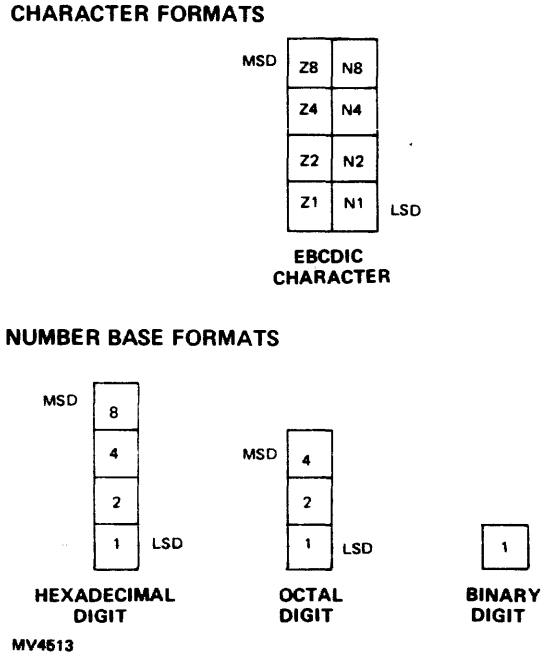


Figure 2-2. Character and Digit Formats

The decimal numbering system is based on the numeric digits zero through nine, and on the powers of ten. Similarly, the binary numbering system is based on the numeric digits zero and one, and on the powers of two. In the case of the numbering systems described above, it is apparent that a decimal digit may have any value from zero through nine, and that a binary digit may have a value of either zero, or one.

The octal numbering system is based on the numeric digits zero through seven, and on the powers of eight. An octal digit may have any value from zero through seven. Further, two raised to the third power is eight, the base of the octal numbering system. Therefore, because the octal numbering base is a multiple of the binary number base, an octal number can be conveniently converted to a binary number, and vice versa.

The hexadecimal numbering system is based on the numeric digits zero through nine, and A through F: where A equals decimal 10, B equals decimal 11, C equals decimal 12, D equals decimal 13, E equals decimal 14, and F equals decimal 15. Hexadecimal numbering is also based on the powers of sixteen. Two raised to the fourth power is sixteen, the base of the hexadecimal numbering system. Therefore, because the hexadecimal numbering base is a multiple of the binary numbering base, a hexadecimal number can be conveniently converted to a binary number, and vice versa.

A B 6900 word contains 48 bits in the value field of the word (see Figure 2-3). These 48 bits can be converted into hexadecimal, octal, BCL, or EBCDIC values by arrangement of the 48 bits in the proper order. A hexadecimal digit is equivalent to four binary digits because 1111 binary is equal to hexadecimal F. Since a hexadecimal digit contains four

binary digits, the value field of a B 6900 word contains 12 complete hexadecimal digits (48/4 =12). The same value field can also be considered to contain 16 octal digits (48/3 = 16), or 6 EBCDIC characters (48/8 = 6).

From the foregoing discussion it is clear that the choice of 48 bits for the value field of a B 6900 word was not a random choice, but rather was chosen because that number is a multiple of the common character codes and number bases used in the B 6900 System.

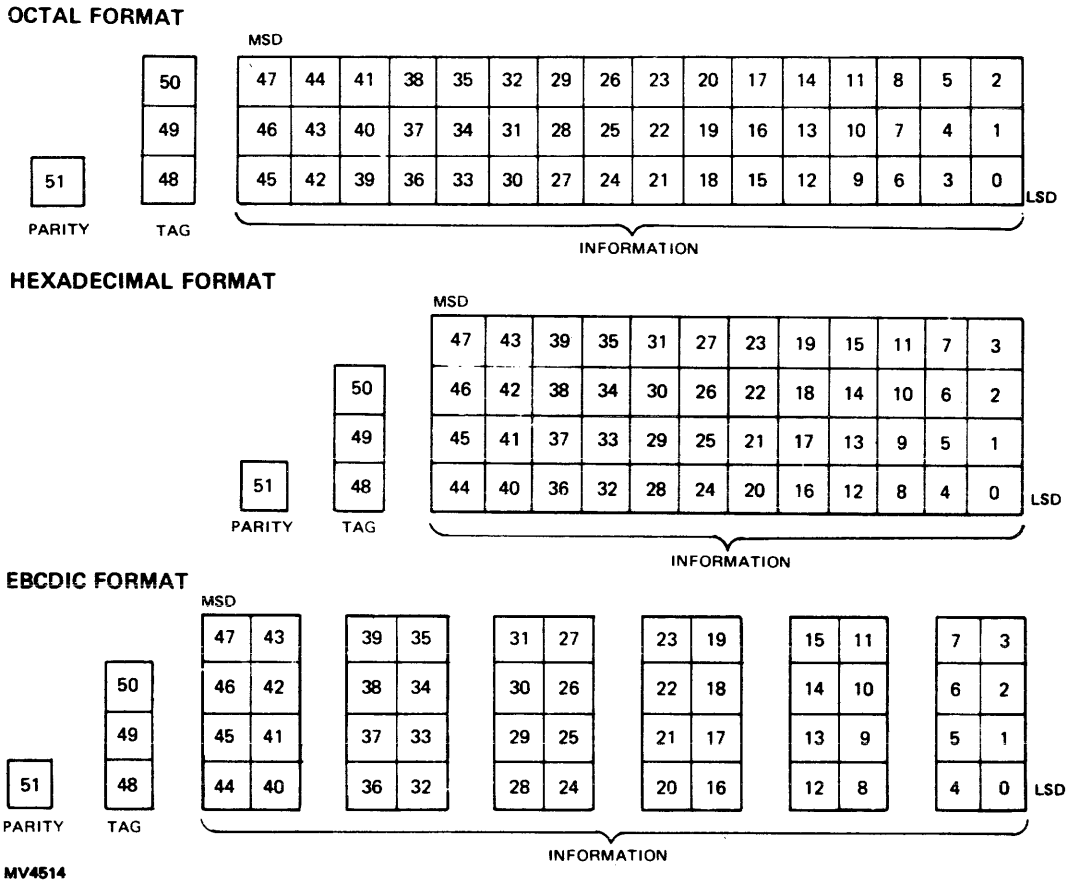


Figure 2-3. B 6900 Word Formats

NUMBER CONVERSION

The B 6900 system normally converts decimal data that is input to the system from decimal notation to EBCDIC codes. An exception to this normal mode of operation may occur in the case of the data communications subsystem, where external input data may be in ASCII code. It is also possible to find that the input data has been packed, is in hexadecimal notation in the system. The user of the system must be familiar with the forms in which the data can be stored. The user must be able to perform manual conversion of numeric data from one form to another so that the internal data conversion processes can be assessed for proper operation. The following paragraphs present methods for performing manual conversion of numeric data from one form to other forms.

DECIMAL TO NONDECIMAL

Decimal numeric data is converted from base 10 to some other number base by repeatedly dividing the decimal value by the base number for the numbering system to which it is to be converted. Each time a division is performed, the remainder becomes the next most significant digit or bit in the new number base. When no more whole numbers occur during the division, the conversion is complete.

EXAMPLES:

a. Convert the decimal number 1776 to octal (base 10 converted to base 8).

1776/8 = 222 with a remainder of 0;

222/8 = 27 with a remainder of 6;

27/8 = 3 with a remainder of 3;

3/8 = 0 with a remainder of 3.

1776 decimal = 3360 octal.

b. Convert the decimal number 1776 to hexadecimal (base 10 converted to base 16).

1776/16 = 111 with a remainder of 0;

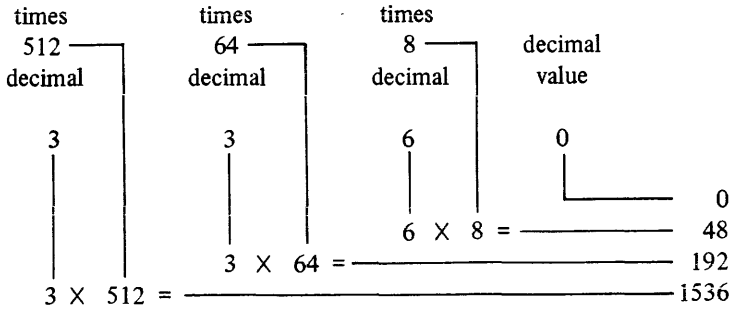
111/16 = 6 with a remainder of 15 F (15 decimal = F hex);

6/16 = 0 with a remainder of 6.

1776 decimal value = 6F0 hexadecimal.

NONDECIMAL TO DECIMAL

Nondecimal numeric data is converted to decimal data by multiplying each digit of the numeric value by the value of the digit position in decimal values. For example, in the preceding subsection of this manual the decimal number 1776 was converted to octal and hexadecimal notation. The successively more significant digits of the octal notation are as follows:



The decimal equivalent value is 1776

By the same logic, a hexadecimal number is converted to decimal as follows:

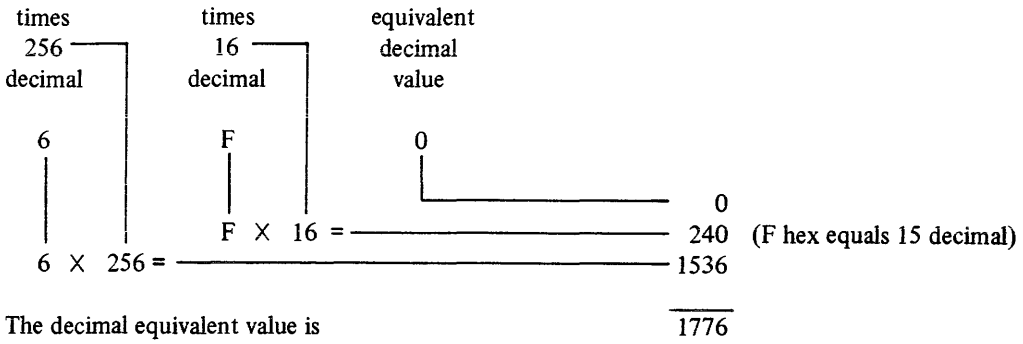


Table 2-1 gives the value of each succeeding digit in a number. These values are provided for binary, octal, and hexadecimal digit positions. The values in this table are expressed in decimal equivalents for the corresponding digit positions. There are 16 octal digits in a B 6900 word (see Figure 2-3) and, therefore, Table 2-1 gives the place values for 16 octal digits. A B 6900 word contains 12 hexadecimal digits, and, therefore, Table 2-1 gives the place values for 12 hexadecimal places.

Observing Table 2-1 while again reading the examples of converting a nondecimal value to a decimal value shows the origin of the place values used to perform the multiplication portions of the examples. The sum of the multiplications provides the decimal values of the nondecimal numbers used in the examples.

NONDECIMAL TO NONDECIMAL

It is occasionally necessary to convert a hexadecimal number to an octal number or vice versa. The easiest way to perform this conversion is to first convert this binary value to the final form.

EXAMPLE:

Convert the hexadecimal value ABCDE to octal notation.

a. Convert hexadecimal ABCDE to binary form as follows:

An A in the fifth position is 1010 in binary form
A B in the fourth position is 1011 in binary form
A C in the third position is 1100 in binary form
A D in the second position is 1101 in binary form
An E in the first position is 1110 in binary form

The binary representation for the hexadecimal value is

1010 1011 1100 1101 1110.

b. Convert the binary value from step a to octal notation as follows:

10 101 011 110 011 011 110

2 5 3 6 3 3 6

Thus, the octal equivalent for the hexadecimal value ABCDE is 2536336. Reversing the procedure of the preceding example converts the octal value to hexadecimal notation.

Table 2-1. Decimal Place Values of Digits in Various Number Bases

Digit Place	Binary Number Place Value	Octal Number Place Value	Hexadecimal Number Place Value
1	1	1	1
2	2	8	16
3	4	64	256
4	8	512	4096
5	16	4096	65536
6	32	32768	1048576
7	64	262144	16777216
8	128	2097152	268435456
9	256	16777216	4294967296
10	512	134217728	68719476736
11	1024	1073741824	1099511827776
12	2048	8589934592	17592189244416
13	4096	68719476736	
14	8192	549755813888	
15	16384	4398047311104	
16	32768	35184378488832	
17	65536		
18	131072		
19	262144		
20	524288		
21	1048576		
22	2097152		
23	4194304		
24	8388608		
25	16777216		
26	33554432		
27	67108864		
28	134217728		
29	268435456		
30	536870912		
31	1073741824		
32	2147483648		
33	4294967296		
34	8589934592		
35	17179869184		
36	34359738368		
37	68719476736		
38	137438953472		
39	274877906944		
40	549755813888		
41	1099511827776		
42	2199023655552		
43	4398047311104		
44	8796094622208		
45	17592189244416		
46	35184378488832		
47	70368756977664		
48	140737513955328		

The example shown works well when the present form of the value to be converted to another form is relatively small. However, it can be seen that a five digit hexadecimal number converts into a twenty digit binary number (as in the preceding example), and from this it is evident that larger hexadecimal numbers become long strings of binary digits. Extremely long strings of binary digits are cumbersome, and become awkward in performing the conversion. Another method that may be used to perform conversions in this case is as follows:

EXAMPLE:

Convert the hexadecimal value ABCDE to octal notation.

a. Using the values in Table 2-1, convert the hexadecimal number to its equivalent decimal value, as follows:

(1) The value of the fifth position in a hexadecimal number (from Table 2-1) is 65,536. The fifth position of the value to be converted is hexadecimal A (A hexadecimal is equal to 10 decimal). Therefore, the hexadecimal A in the fifth position is equal to 10 times 65,536, or 655,360 decimal.

(2) The fourth position of a hexadecimal number has a value of 4,096 (from Table 2-1). The fourth position of the hexadecimal number to be converted is B (hexadecimal B is equal to 11 decimal). Decimal 11 times 4,096 is equal to 45,056.

(3) Hexadecimal C times 256 decimal is equal to 3,072.

(4) Hexadecimal D times 16 decimal is equal to 208.

(5) Hexadecimal E is equal to 14 decimal.

655,360	hexadecimal Annnn
45,056	hexadecimal nBnnn
3,072	hexadecimal nnCnn
208	hexadecimal nnnDn
14	hexadecimal nnnnE
703,710	hexadecimal ABCDE equals 703,710 decimal

b. Convert the decimal number 703,710 (from step a. above) to the equivalent octal value, as follows:

703,710/8 = 87,963 with a remainder of 6;
87,963/8 = 10,995 with a remainder of 3;
10,995/8 = 1,374 with a remainder of 3;
1,374/8 = 171 with a remainder of 6;
171/8 = 21 with a remainder of 3;
21/8 = 2 with a remainder of 5;
2/8 = 0 with a remainder of 2;

Hexadecimal ABCDE equals 2 5 3 6 3 3 6 octal.

The procedure for converting nondecimal numbers to nondecimal numbers shown in the preceding example can also be used to convert an octal number to a hexadecimal equivalent. The only difference is that the place values from Table 2-1 (used in step a. of the procedure) must be taken from the octal column instead of from the hexadecimal column.

WORD TYPES AND PHYSICAL WORD LAYOUTS

As explained in the beginning paragraphs of this section, a B 6900 system word consists of a parity bit, a tag field, and an information field. The tag field defines an interpretation that is to be applied to the contents of the information field. This subsection of this manual will define the interpretations that are to be used for the data in the B 6900 system, and will present the format of the data in the information field of each type of word used in the B 6900 system.

The two types of data used in the B 6900 system are character strings and operands. The following paragraphs define character strings and operands.

CHARACTER TYPE WORDS

Character type words are used to contain character strings. A character type word has a tag field of ZERO (a single precision word) and contains EBCDIC, or hexadecimal coded data. A string may occupy more than a single word of character data. However, a string must have at least one character type word.

The most significant character in a character string occupies the left-most character position in the field character word of the string. Each word in a character string will contain 6 EBCDIC character positions or 12 hexadecimal character positions. The final word in a character string may contain less than a full word of characters if the number of characters in the string is not a multiple of the number of characters in a full word. Figures 2-4 through 2-5 show the various formats that are used for character type words.

		A	A	B	B	C	C	D	D	E	E	F	F
	0	A	A	B	B	C	C	D	D	E	E	F	F
	0	A	A	B	B	C	C	D	D	E	E	F	F
P	0	A	A	B	B	C	C	D	D	E	E	F	F

P = WORD PARITY VALUE
0 = BINARY ZERO VALUES (TAG FIELD)
A ⇒ F 6 EBCDIC CHARACTER FIELDS
A IS THE MOST SIGNIFICANT CHARACTER

MV 2573

Figure 2-4. EBCDIC Character Word Format

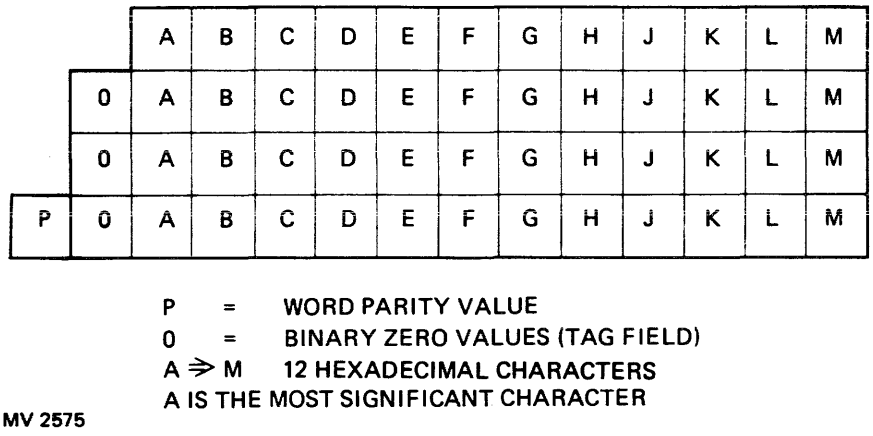


Figure 2-5. Hexadecimal Character Word Format

OPERANDS

Operands are words of data that are used to contain numeric values or logical information. An operand may be either a single precision word (tag field of ZERO), or a double precision word (tag field of TWO). Single, and double precision words are used for mathematical operations. Logical information is used for decision-making processes, and operations. The following paragraphs discuss the uses of operands in the B 6900 system.

Single-Precision Operand

A single-precision operand is a numeric value that has an exponent part and a mantissa part. Figure 2-6 shows the format for a single-precision operand. The fields in a single-precision operand are as follows:

bits [50:3] are the tag field, and are always equal to zero for a single-precision operand.

bit 47 bit 47 is not used in a single-precision operand.

bit 46 bit 46 is used as the sign of the mantissa field. If the sign bit is a binary one then the mantissa field contains a negative value, and if bit 46 is a binary zero then the mantissa contains a positive value.

bit 45 bit 45 is used as the sign of the exponent field. If the sign bit is a binary one then the exponent field contains a negative value, and if bit 45 is a binary zero then the exponent contains a positive value.

bits [44:6] are the exponent field. Bit 44 is the most significant bit in the exponent value. The value of the bits in this field are as follows:

- bit 39 value is decimal one
- bit 40 value is decimal two
- bit 41 value is decimal four
- bit 42 value is decimal eight
- bit 43 value is decimal sixteen
- bit 44 value is decimal thirty-two

The maximum value that the exponent field can contain is decimal 63. When the exponent is used in conjunction with the exponent sign bit (45), the range of the exponent value is from +63 to -63 decimal.

bits [38:39] are the mantissa field. Bit 38 is the most significant bit in the mantissa value. The mantissa is divided into thirteen octal fields, of which bits [38:3] are the most significant octal digit, and bits [2:3] are the least significant digit.

An octal point (similar to a decimal point) is always located to the right of bit zero in the mantissa field. This point is not displayed in any way and must be assumed to exist.

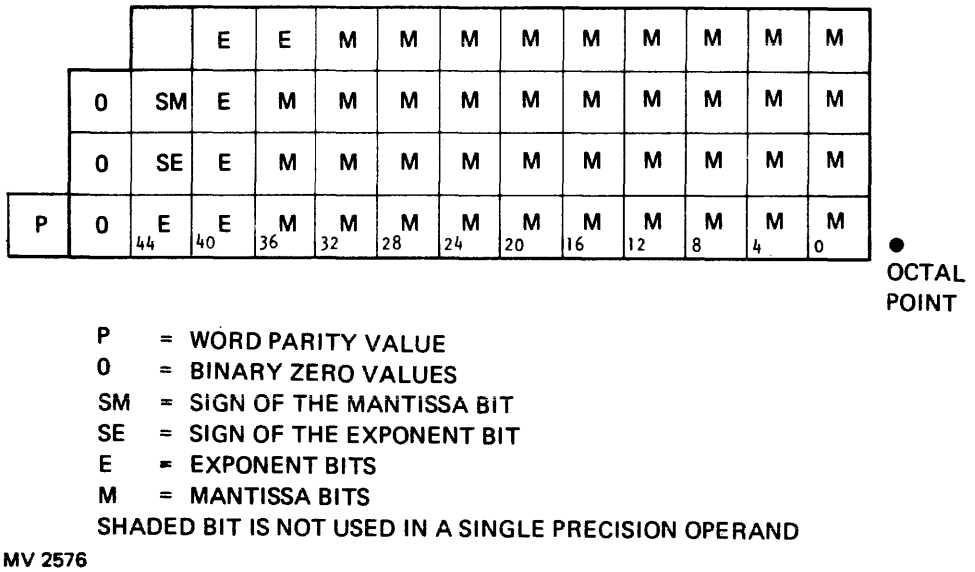


Figure 2-6. Single-Precision Operand Format

The software of the B 6900 system classes numeric data into two classes: INTEGER, and REAL. An INTEGER number is a single-precision or double-precision numeric value with an exponent value of zero. The maximum value that an INTEGER may have in the B 6900 system is +7777777777777 octal, or 549,755,813,887 decimal. The minimum integer value is -7777777777777 octal. A REAL numeric value is any value that has an exponent that is not equal to zero, or any value that contains a part value (contains a decimal, or octal point prior to the least significant digit of the value). From the format given for a single-precision operand it is evident that REAL numbers may not qualify to be expressed as single-precision values.

Double-Precision Operand

A double-precision value is two consecutive words, with a tag field of TWO (010 binary). The two words are concatenated in such a way that they form a single numeric value, with an octal point located between the two words. The most significant part of the mantissa in a double-precision operand is commonly referred to as the most significant part (MSP) and the least significant part of the mantissa is commonly referred to as the least significant part (LSP). The octal point that separates the MSP from the LSP is used to separate whole values from partial values, with whole values present in the MSP, and partial values present in the LSP. The format for the MSP of a double-precision operand is

identical with the format for a single-precision operand, except for the tag field. The LSP of a double-precision operand is an extension of the exponent field and of the mantissa field contained in the MSP of the word. Figure 2-7 shows the word format for a double-precision operand.

The largest double precision value (type REAL) that can be contained in a B 6900 is 1.94882938205028079124469, with an exponent value of +29603. The smallest double-precision value (type REAL) that can be contained in a B 6900 is 1.9385458571375858335564, with an exponent value of -29581. The value zero and the positive or negative values between the largest and smallest values given above may be represented in double-precision numbers in the B 6900 system.

When a double-precision value is used the exponent extension field (in the LSP), it is an extension to the high order end of the exponent field in the MSP. Bit 39 in the LSP word is the next bit in sequence after bit 44 of the upper-half, and has a binary value of 64. Bit 40 in the LSP word is the next bit in sequence after bit 39 of the word, and has a binary value of 128. This same order is used for all of the bits in the LSP exponent extension field, so that bit 47 of the LSP becomes the most significant bit in the exponent value. The whole exponent field in a double-precision operand is as follows:

MSP bit 39 is the least significant bit of the exponent, and has a value of 1, decimal.

LSP bit 39 is the next most significant bit in the exponent, and has a value of 64, decimal.

bit 47 is the most significant bit in the exponent, and has a value of 16384, decimal.

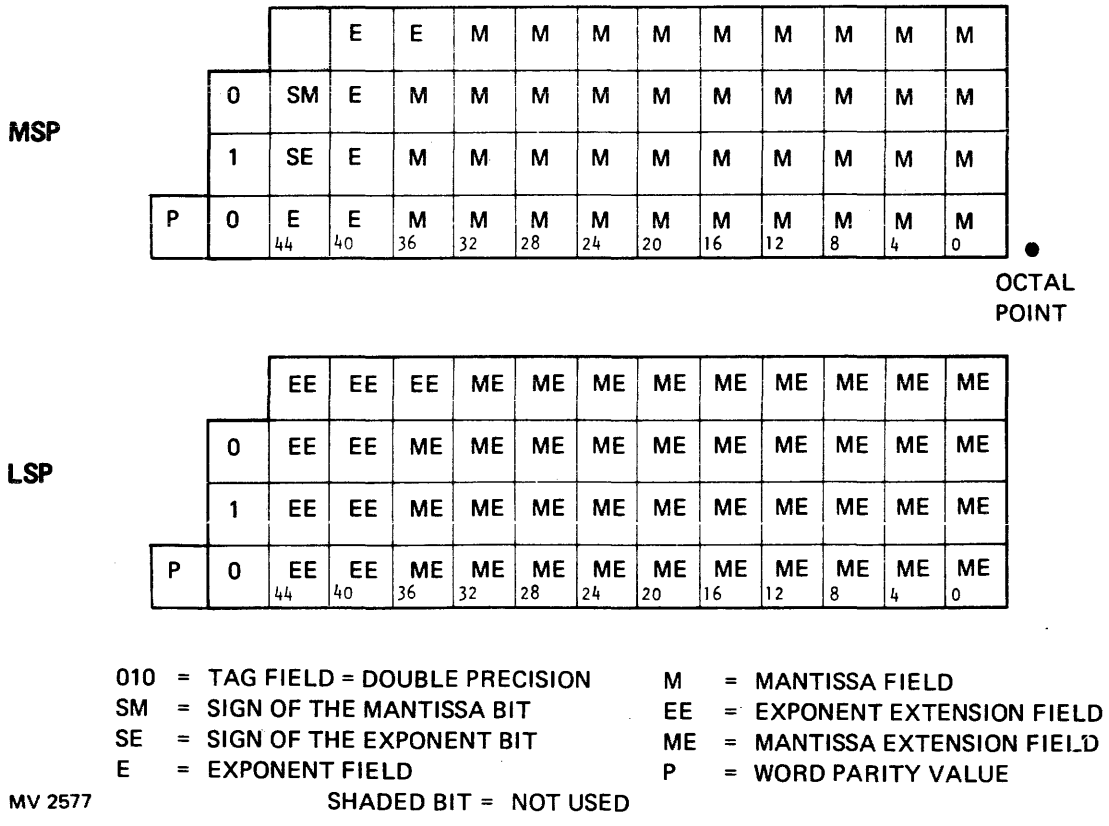


Figure 2-7. Double-Precision Operand Format

The maximum value of an exponent in the B 6900 system is 32,767 decimal, and the range of the exponent field is from +32,767, to -32,767 decimal.

The mantissa extension field (in the LSP of the double precision operand) contains that portion of the mantissa that is less than unity. The mantissa extension field is divided into 13 octades, in the same manner as the mantissa field in the MSP of the double precision operand. These octal digits are arranged in the same way as the octal digits in the MSP of the word. The least significant octade of the mantissa extension field is bits [2:3], and the most significant octade is bits [38:3].

The B 6900 system, in performing mathematical operations, utilizes two processes known as integerization and normalization. Normalization is a process that removes leading zeroes from a single-precision or double-precision word. This process is used to make the operation of the adder logic circuits more efficient. Integerization is a process that alters the value of a number such that it meets the requirements of an integer, as was defined previously in this section.

Normalization is accomplished by adjusting the value of the exponent field of a number in a positive direction until it is at the maximum value for an exponent, or until there are no leading zeroes in the mantissa of the number. Each time the exponent is incremented, the mantissa is shifted one octade to the left. There are no more leading zeroes in a mantissa when the most significant octade of the mantissa is located in bits [38:3] (of the LSP word).

The process of integerization is a two-step process. The first step is to adjust the exponent in either a positive or a negative direction until the exponent field is equal to zero. Each time the exponent is incremented or decremented, the mantissa is shifted one octade in the corresponding direction. Octades that fall out of the low order digit of the mantissa during the adjustment of the exponent are saved until the exponent is equal to zero. After the exponent has been adjusted to zero, that part of the mantissa that is less than unity (located to the right of the octal point) is either rounded upward to the next whole number, or it is truncated (deleted from the number). The process of rounding or truncating is selective in the B 6900 system, and is the second step of the integerization process.

The mathematical operations that are performed in the B 6900 system can be completed regardless of the format of the operands used. If an arithmetic operation is performed using two single precision operands, then the result of the operation will be in the single-precision format. If, however, either operand is in the double-precision format then the result of the operation will be in the double-precision format.

Logical Operands

Logical operands are words that result from the performance of either a relational operation, or a logical (Boolean) operation. A relational operation is one that determines the relative merits of two values by means of a comparison process. A logical operation is one that constructs a result based on the relative merit of each bit in a word when compared to the corresponding bits in another word.

A relational operation results in either a true or a false answer. The answer is true if the result of an algebraic comparison of two arithmetic values is valid. The answer is false if the result of the algebraic comparison of the two arithmetic values is not valid. The B 6900 constructs a single precision logical operand (tag field equal to binary zero) each time that a relational operation is performed. If the answer is valid, bit zero is a one in the logical operand; and if the answer is not valid then bit zero is a zero. All other bits in the answer word logical operand are not used, and are zeroes.

A logical (Boolean) operation results in the construction of a different type of logical operand. The constructed logical operand may contain a number of bits. The reason is that a logical operation looks at each bit in two different words, and places a corresponding bit in the result operand if the conditions of the logical operation are satisfied.

Logical operands are discussed later in this manual.

DATA DESCRIPTORS

Data descriptor words refer to data areas, including input/output buffer areas. The data descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in words is contained in the length field of the descriptor. Data descriptors may directly reference any memory word address from word number zero through word number 1, 048, 576. The structure of the data descriptor word is illustrated in Figure 2-8.

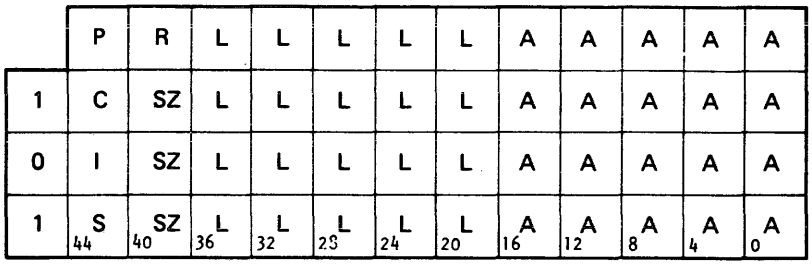


Figure 2-8. Data Descriptor Format

The fields in the data descriptor are as follows:

bits 50:3 — Bits 50, 49, and 48 are the tag field, and are always equal to a binary value of 101.

bit 47 — Bit 47 is the presence bit. The presence bit is used to indicate whether or not the information described by the data descriptor is present in main memory. If the presence bit is equal to a binary one then the data is present in main memory. If the presence bit is equal to a binary zero then the data is not in main memory. Attempting to access data with a data descriptor that has its presence bit equal to a binary zero causes a presence bit interrupt. The B 6900 system uses the occurrence of a presence bit interrupt as the preliminary step to start an MCP process which will move the data described by the data descriptor from system disk, or system pack storage into the main memory.

bit 46 — Bit 46 is the copy bit. The copy bit indicates whether the data descriptor is the original descriptor for the data, or is a copy of the original descriptor. If the copy bit is equal to a binary zero then the data descriptor is the original. If the copy bit is a binary one then the data descriptor is a copy of the original descriptor. An original data descriptor is commonly referred to as a mother (or MOM) descriptor and a copy of a mother descriptor is commonly referred to as a copy descriptor.

bit 45 — Bit 45 is the indexed bit. The indexed bit is used to indicate whether or not an indexing operation has been performed on the data descriptor. If the index bit is equal to a binary one then the descriptor has been indexed previously, and the value of the previous index is located in the length field 39:20. If the index bit is equal to binary zero, the data descriptor has neven been indexed before; and such an indexing operation must be performed before accessing the data described by the descriptor. The process that causes the indexing operation to be performed also sets the indexed bit and stores the value of the index in the field 39:20.

bit 44 — Bit 44 is the segmented bit. The segmented bit is used to identify whether or not the data described by the data descriptor is segmented. If the segmented bit is equal to a binary zero then the data is not in segments, and this descriptor describes the entire field.

bit 43 — Bit 43 is the read only bit. The read only bit is used to show whether the memory area described by the data descriptor can be written into or not. If the read only bit is equal to a binary one then the data descriptor describes a memory area that may be read, but may not be written into. If the read only bit is a binary zero then the data descriptor describes a memory area that may be written into, or read from. It is possible for a single area in memory to be described by two different data descriptors: one where the Read Only bit is a binary one, and another descriptor where the Read Only bit equals a binary zero. The memory area may be written into by use of the data descriptor that has the Read Only bit equal to a binary zero, but may not be written into by use of the data descriptor that has the Read Only bit equal to a binary one.

bits 42:3 — Bits 42, 41, and 40 are used to define the type of data contained in the memory area that is described by the data descriptor. If bits 42 and 41 are both equal to binary zeroes, then the data descriptor defines an area in memory in words. A data descriptor that describes a string of character data is commonly called a string descriptor. If either bit 42 or bit 41 is equal to a binary one then the descriptor is a string descriptor. Bits 42:3 may contain several different binary values, and the meaning of the different values that are used have the following meanings:

bit 42	bit 41	bit 40	
0	0	0	Bits 42 and 41 being equal to zero indicates that the data descriptor is a word descriptor. Bit 40 being equal to binary zero indicates that the data described by the descriptor is in single precision operands.
0	0	1	Bits 42 and 41 being equal to zero indicates that the data descriptor is a word descriptor. Bit 40 being equal to binary one indicates that the data described by the descriptor is in double precision operands.
0	1	0	Bits 42 and 41 not being equal to zero indicates that the data descriptor is a string descriptor, and bit 41 being a binary one indicates that the data described contains hexadecimal (4-bit) data.
0	1	1	Bits 42 and 41 not being equal to zero indicates that the data descriptor is a string descriptor. Bits 41 and 40 both being equal to binary ones is an illegal code in a B 6900 system.
1	0	0	Bits 42 and 41 not being equal to zero indicates that the data descriptor is a string descriptor. Bit 42 equal to binary one indicates that the data described contains EBCDIC (8-bit) data.

bits 39:20 Bits 39:20 contain either the length of the memory area (if bit 45 is a binary zero) or an index value (if bit 45 is a binary one). If bit 45 is equal to binary zero the descriptor has not been indexed. This field is used for size checking during the indexing operation. If bit 45 is equal to a binary one the descriptor has been indexed. If the data descriptor is a word descriptor, and also if bit 40 is a binary one (the word area contains double precision operands) then the index is doubled after the indexing operation and the size checking operation have been completed. The doubled index is stored in the index field.

bits 19:20 Bits 19:20 contain either a main memory or a disk file address. If the presence bit is equal to a binary one, and the copy bit is also equal to a binary one, then the address field contains the main memory address of the MOM descriptor. If the presence bit is equal to a binary one and the copy bit is equal to a binary zero then the address field contains the main memory address of the first word of data described by the descriptor. If the presence bit is equal to a binary zero, and the copy bit is also equal to a binary zero, then the address field contains a 6-bit binary coded decimal disk file address where the data described by the data descriptor is located. If the presence bit is a binary zero and the copy bit is a binary one, the address field contains the memory address of the original program segment descriptor.

STEP INDEX WORDS

Step index words are words that are used in conjunction with the step and branch operator in the B 6900 system. The purpose of the step and branch operator in the B 6900 system is to perform a series of other machine language operators in a recursive manner, but with control over the number of times the series of operators are executed. The step index word is used to provide the control part of the function of the step and branch operator.

The step index word (see Figure 2-9) contains a TAG of four (100-binary), and four other fields, as follows:

47:12 the increment value

35:16 the final value

19:04 an unused, but value-specified, field which must be equal to zero

15:16 the current value

	I	I	I	F	F	F	F	0	C	C	C	C
1	I	I	I	F	F	F	F	0	C	C	C	C
0	I	I	I	F	F	F	F	0	C	C	C	C
0	I_{44}	I_{40}	I_{36}	F_{32}	F_{28}	F_{24}	F_{20}	0_{16}	C_{12}	C_{8}	C_{4}	C_{0}

TAG = 100 – STEP INDEX WORD
I = INCREMENT FIELD [47:12]
F = FINAL VALUE FIELD [35:16]
C = CURRENT VALUE FIELD [15:16]
FIELD [19:4] MUST CONTAIN BINARY ZEROES

MV 2579

Figure 2-9. Step Index Word Format

Each time the series of machine language operators is performed the value of the increment is added to the value of the current value field. The step and branch operator then compares the current value field to the final value field. If the current value field is greater than the final value field a branch is taken out of the recursive series of operators. If the current value field is not greater than the final value field then the recursive series of operators are executed.

The increment value, the final value, and the current value are binary values. To determine the number of times a recursive series of operations will occur, binary mathematics and not decimal mathematics, must be used; and the unused but value-specified field (19:04) must be equal to zero in the step index word.

SOFTWARE WORDS

A software word is a word with a tag field of six (110 binary) that is used by the MCP of the B 6900 system for software purposes. The MCP uses the software word for several different purposes, and the format of the word is different for each purpose. The software word is utilized as a linking word for memory allocation, as a software control word, as an un-initialized pointer word, and to contain system intrinsics data. Each of these uses for software words causes a different format to be used for the fields of data that are contained in the word.

The format of the software word when it is used for uninitialized pointers or for intrinsics information are not defined in this manual. These formats are specialized applications that are properly documented in manuals that discuss the specific application subjects.

The format of the software word when it is used for a memory link word and for a software control word is given in the following paragraphs. The specific use of the software word in either of these formats is not covered in this manual. Like the uninitialized pointer word and the intrinsics information word, these specific uses are specialized applications, and are more properly documented in manuals that deal with the software system as a specific subject.

The MCP maintains linking words in main memory to show which portions of the memory are in use, and which portions are not currently in use. A software word is used as the first link word for a portion of memory that is in use. This word is defined in the memory link system as the LINKA word, and each part of the main memory that is in use begins with a LINKA word. Memory link words are a mechanism for dynamic storage allocation which will be covered in more detail later in this manual. Figure 2-10 shows the format of a LINKA word.

	CF	S	S	S	S	S		A	A	A	A	A
1	CF	S	S	S	S	S	CS	A	A	A	A	A
1		S	S	S	S	S	AS	A	A	A	A	A
0		S	S	S	S	S	1	A	A	A	A	A
	44	40	36	32	28	24	20	16	12	8	4	0

TAG = 6 (110 BINARY) = SOFTWARE CONTROL WORD.
CF [47:2] = CONTROL FIELD FOR AREA DURING THE OVERLAY AREA MCP PROCESS.
S [43:20] = SIZE OF THE IN-USE AREA IN WORDS.
CS (BIT 22) = CONTROL SAVE FIELD – IF AREA IS TEMPORARILY SAVED CS=1.
AS (BIT 21) = AREA SAVED FIELD – IF AREA IS NON-OVERLAYABLE (SAVED) AS=1.
BIT 20 = IS BINARY 1 FOR A LINKA WORD.
A [19:20] = THE CORE MEMORY ADDRESS FOR THE MOM DATA DESCRIPTOR OF THE AREA CONTENTS

MV 2580

Figure 2-10. Software Control (LINKA) Word

Software control words are used by the software operating system to indicate the existance of memory areas that are related to the operating stack, but are physically located outside of the operating stack. When the memory area of an operating stack is deallocated (the stack is cut back), related memory areas also must be deallocated. The software control word is a mask word that indicates the presence or absence of such related memory areas by the state of the bits in the mask word. At the time that the stack area is to be deallocated, a related memory area is present for each bit that is a binary one value in the mask field of the software control word. Figure 2-11 shows the format of the software control word.

	1						PLM SKF	ALM SKF	ALM SKF	ALM SKF	PC	PC
1	0						PLM SKF	ALM SKF	ALM SKF		PC	PC
1	GOTO ABO RTF						PLM SKF	ALM SKF	ALM SKF	PC	PC	PC
0						NOC PBT	PLM SKF	ALM SKF	ALM SKF	PC	PC	PC
	44	40	36	32	28	24	20	16	12	8	4	0

[50:3] = TAG FIELD = 110 = SOFTWARE CONTROL WORD
[47:2] = 2 = SOFTWARE CONTROL WORD (MASK WORD)
45 = 1 = GO TO ABORTE
24 = 1 = NOCPBIT
[23:4] = PL/I COMPILER BLOCKEXIT AND FAULT FIELD
[19:9] = MASK FIELD
19 = NOT USED
18 = FMT PSEUDO BUFFER FIB-LOCKED
17 = NON-LOCAL GOTO
16 = DIRECT ARRAY DECLARATION IN BLOCK
15 = FAULT IN BLOCK DECLARATION
14 = INTERRUPT IN BLOCK DECLARATION
13 = FILE IN BLOCK DECLARATION
12 = MULTI-DIMENSION ARRAY IN BLOCK DECLARATION
11 = SINGLE-DIMENSION ARRAY IN BLOCK DECLARATION
[9:10] = PROCESS COUNT

MV 2581

Figure 2-11. Software Control (MASK) Word

INDIRECT REFERENCE WORDS

Indirect reference words (IRW) are used in the B 6900 system to reference data that is located within the addressing environment of the current procedure. The addressing environment of the current procedure includes the current operating stack, and all stacks that are a part of the current procedure at a lower lexicographical level than the current operating stack level.

Stuffed indirect reference words (SIRW) are used in the B 6900 system to reference data that is located outside of the addressing environment of the current operating procedure.

The fields of an indirect reference word or a stuffed indirect reference word do not contain data. Instead, the fields of an indirect reference word or a stuffed indirect reference word contain addressing information that is used to point to the location of data. The fields of an IRW or a SIRW are both displayed in Figure 2-12. The fields within the IRW and the SIRW are as follows:

bits 50:3 — Bits 50:3 are the tag field. The tag field for an IRW is always 001 binary, regardless of whether the IRW is stuffed or normal.

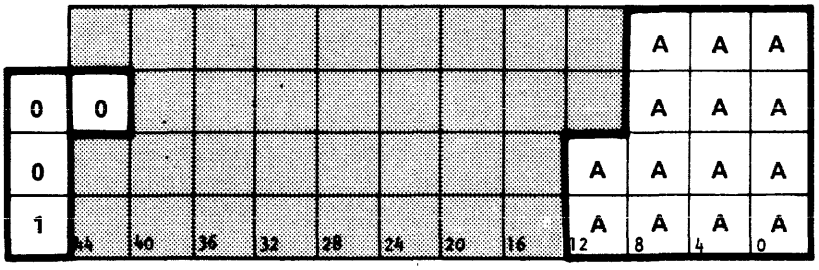
bit 46 — Bit 46 is the environment bit. If bit 46 is a binary one the IRW is stuffed. If bit 46 is a binary zero the IRW is a normal IRW.

bits 45:10 — Bits 45:10 are the stack number field. The stack number is not used in a normal IRW and is equal to binary zero. If bit 46 is a binary one then the value of the stack number field is the identification number of the stack that is to be referenced.

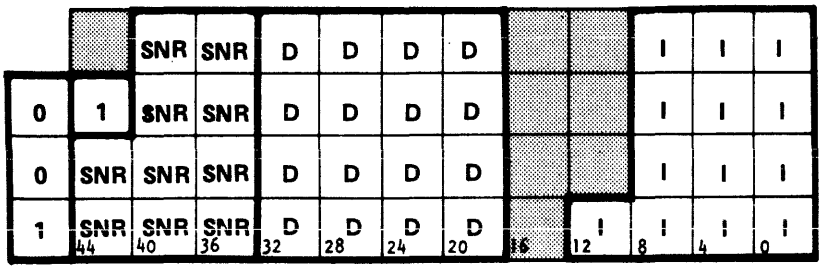
bits 35:16 — Bits 35:16 are the displacement field. The displacement field is not used for a normal IRW and is equal to binary zero. If bit 46 is a binary one then the displacement field is added to the address of the base of the stack being referenced to locate a mark stack control word within the referenced stack area.

bits 12:13 — Bits 12:13 are the index field. The index field is not used in a normal IRW; however, the same bits are used for a different purpose. If bit 46 is a binary one then the index field is added to the address of the mark stack control word in the referenced stack. The sum of these values is the address of the data that is being addressed.

bits 13:14 — Bits 13:14 are the address couple field. The address couple field is not used in the SIRW; however, the same bits are used for a different purpose. The address couple field is used in an IRW to locate data in the addressing environment of the current procedure. The address couple consists of two separate values, each of which are of variable bit length. The most significant part of the address couple contains the lexicographical level value. The least significant part of the address couple contains an index value which is added to the address of the mark stack control word that corresponds to the lexicographical control level. The sum of the address of the mark stack control word, and the index value is the address of the data referenced by the IRW.



IRW WORD FORMAT



SIRW WORD FORMAT

MV2725

Figure 2-12. IRW and SIRW Formats

The lexicographical level (program level) of a current procedure may have any value from zero, through thirty-one. The lexicographical level (LL) part of an address couple is represented by the most significant bits of the address couple. The LL requires five bits of the address couple to represent the binary value of thirty-one which is the highest LL value possible. When the LL contains a value of zero or one, only one bit is required to represent the binary LL value. The actual number of binary bits that are used to contain the LL value in an address couple is defined by the level of the current operating procedure. Thus, if the current procedure is at lexicographical level seven, the number of bits in the address couple that are used to indicate LL is three; because three binary bits are required to represent the value of seven decimal.

The index part of an address couple consists of the bits that are not required to represent the LL value. Thus, if the lexicographical level of the current procedure is seven, three binary bits (bits 13, 12, and 11) are required to represent the LL value; and the remaining bits (bits zero through ten) are used to represent the index part of the address couple.

The B 6900 system derives the absolute memory address referred to by an IRW in the following manner:

a. The LL part of the address couple defines the IC memory display register that contains the address of a mark stack control word in main memory.

b. The index part of the address couple is added to the address of the mark stack control address. This sum is the absolute address of the data referred to by the IRW.

Since the number of bits in the address couple that are required to contain the LL value is a variable number, the size of the index value is limited by the number of bits that comprise the index value. Thus, if three bits are required to contain the LL value, then the size of the index part is limited to an eleven bit binary value (or a maximum index value of 2047 decimal memory words). Table 2-2 shows the maximum number of memory words that may be contained in the index part of an address couple for any given LL value part of the address couple.

Table 2-2. Address Couple Value Fields

Lexicographical Level Value	Number of Bits Required	Bits Available for Index Value	Maximum Index Value
0	1	13	8191
1	1	13	8191
2	2	12	4095
3	2	12	4095
4	3	11	2047
5	3	11	2047
6	3	11	2047
7	3	11	2047
8	4	10	1023
9	4	10	1023
10	4	10	1023
11	4	10	1023
12	4	10	1023
13	4	10	1023
14	4	10	1023
15	4	10	1023
16 through 31	5	9	511

The B 6900 system determines the absolute address referred to by the SIRW in a way that is different from the one used for determining the absolute address referred to by an IRW. The method used to determine the absolute address referred to by a SIRW is as follows:

a. The stack number field in the SIRW is an index into the segment dictionary, which is maintained by the MCP. The segment dictionary contains a list of data descriptors that give the absolute memory addresses of all stacks in main memory. The stack number field of the SIRW identifies the descriptor containing the base address of the stack to be referenced.

b. The displacement field value of the SIRW is an index on the base address of the stack being referenced. The value of the base address of the stack, plus the value of the displacement field is the absolute memory address of a mark stack control word in the stack that is being referenced.

c. The index field value of the SIRW is an index on the address of the mark stack control word in the stack that is being referenced. The sum of the address of the mark stack control word plus the value of the index field is the address of the value that is being addressed by the SIRW.

PROGRAM CONTROL WORDS

The program control word (PCW) is used by the B 6900 system to point to the program code for a procedure or segment of a program. The PCW also contains program information about the system environment that is to be used during the execution of the segment or program.

The use of PCW's provides the flexibility that the software requires to utilize reentrant code techniques, and also dynamic storage allocation principals. The reentrant code techniques are used in the B 6900 system to provide the software capability to execute more than one job at a time while using the same machine language code.

Figure 2-13 shows the fields of data that are contained in a PCW.

		SNR	SNR	PSR	PIR	PIR	PIR	N	LL	SDI	SDI	SDI
1		SNR	SNR	PSR	PIR	PIR	PIR	LL	LL	SDI	SDI	SDI
1	SNR	SNR	SNR	PSR	PIR	PIR	PIR	LL	SDI	SDI	SDI	SDI
1	SNR 44	SNR 40	SNR 36	PIR 32	PIR 28	PIR 24	PIR 20	LL 16	SDI 12	SDI 8	SDI 4	SDI 0

50:3 = THE TAG FIELD.
7 IS A PCW TAG
45:10 = THE STACK NUMBER FIELD
35:3 = THE PROGRAM SYLLABLE REGISTER VALUE
32:13 = THE PROGRAM INDEX REGISTER VALUE
19 = THE NORMAL/CONTROL STATE BIT
18:5 = THE LEXICOGRAPHICAL LEVEL VALUE
13:14 = THE SEGMENT DESCRIPTOR INDEX VALUE

MV 1583

Figure 2-13. Program Control Word

The fields of data in a PCW are used as follows:

bits 50:3 — The tag field. The tag field for a PCW is seven decimal (111 binary).

bits 45:10 — The stack number. The stack number field is used to identify the stack that contains the PCW (not always the stack associated with the program code that is to be executed).

The MCP uses stack numbers to identify jobs that are currently being executed or that are scheduled to be executed. The MCP assigns stack numbers for program stacks on a first-come, first-served basis. Therefore the stack number for a program stack is a dynamic variable that is assigned to a program at execution time.

bits 35:3 — The program syllable register (PSR) field. The PSR field is used to indicate the first machine language operator in the first memory word of a machine language code string. A program code string is not required to begin at the first machine language operator in a memory word. There are 6 syllables in a machine language code word, and the PSR value indicates which of the 6 syllables the current string of code starts in.

bits 32:13 — The program index register (PIR) value. The PIR field is used to indicate the first word of the program machine language code string. The combination of the PIR field and the PSR field combine to identify the specific first machine language operator in the program code string. The PIR value defines the first word address of the string, and the PSR value defines the first syllable within the first word of the string.

bit 19 — The normal state/control state bit. The B 6900 system may operate in either of two states, and the proper state for the current code segment is defined by the normal state/control state bit. If the normal state/control state bit is a binary one, control state is specified and normal state is specified otherwise.

bits 18:5 — The lexicographical level (LL) field. The LL field is used to specify the lex level at which the program string is to be executed. The LL value defines one of the 32 IC memory display registers. The value in the selected IC memory display register is the base address in core memory of the program stack with which the program code segment is associated.

bit 13:1 — This bit is used to indicate that the DO stack contains the program code segment descriptor (if 0), or the D1 stack (if 1).

bits 12:13 — The segment descriptor index (SDI) field. The SDI is used to indicate the location of the segment descriptor for the program code in core memory.

The 13 bits of the SDI field are a binary index value which are added to the base address from the display register (either D0 or D1) to define the absolute core memory address of the segment descriptor for the machine language code.

MARK STACK CONTROL WORDS

The mark stack control word (MSCW) is used to define an area within the stack in main memory. The MSCW and the return control word (RCW) together provide a history of the stack linkage, and a record of the stack operating environment. The historical links of a stack, and the operating environment record of the stack are key data in the reconstruction and analysis of program operations.

Figure 2-14 shows the fields of data that are contained in the MSCW.

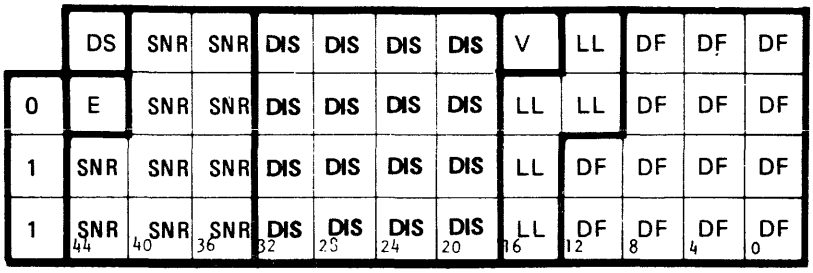


Figure 2-14. Mark Stack Control Word

The meaning of the fields of data in the MSCW are as follows:

bits 50:3 — The tag field. The tag for a MSCW is three (011 binary).

bit 47 — The different stack bit. The different stack bit indicates whether the stack number field refers to the same stack, or to a different stack. If the different stack bit is a binary zero then the stack number field refers to the same stack. If the different stack bit is a binary one then the stack number refers to a different stack.

bit 46 — The entered bit. The entered bit is used to indicate whether the stack is active or not. If the stack is currently in use (is active) then the bit will be set to a binary one. If the stack is not currently in use then the bit will be reset to a binary zero. If the entered bit is a binary one then it indicates that the MSCW is active and was entered into the stack by a procedure entry. If the entered bit is a binary zero it shows that the MSCW was entered into the stack by the mark stack machine language operator, and no procedure entry has been made in the stack. When a procedure entry is made into the stack the environment fields of the MSCW are completed from the PCW that caused entry, and the entered bit is set to a binary one.

bits 45:10 — The stack number. The stack number field is completed at procedure entry time, and contains the stack number value from the PCW that was entered. The stack number is the designation of the stack that contains the PCW, not the number of the current stack.

bits 35:16 — The displacement field. The displacement field is used to link a program together by its lexicographical levels. The value of the displacement field defines the MSCW that represents the last previous lexicographical level of the procedure. The location of the MSCW that corresponds to the preceding lexicographical level is determined by adding the value of the displacement field to the value of BOSR for the stack.

bit 19 — The value bit. The value bit is used to indicate whether or not the operator that caused entry to the current operator is to be restarted at the beginning of the operator in the procedure that caused entry. If the value bit is a binary zero then the previous operator must be restarted from the beginning. If the value bit is a binary one then the previous operator must be continued at the next operator in sequence.

bits 18:5 — The lexicographical level field. The value of the lexicographical level field defines the lexicographical level at which the program will run when the procedure is entered.

bits 13:14 — The difference field. The difference field is used to store the stack history. The value of the difference field is the number of words between the current MSCW and the previous MSCW in the stack. Subtracting the value of the difference field from the address of the current MSCW gives the address of the previous MSCW.

INTERRUPT PARAMETER WORDS

The interrupt controller of the B 6900 data processor recognizes certain types of system interrupts. The DP interrupt controller interrupts the program that is running, and causes an entry into the MCP interrupt handling procedures when a system interrupt is sensed. The interrupt handling procedures of the MCP initiate system actions that are required because of the interrupt condition that exists. At the conclusion of the interrupt handling function, the MCP returns control of the DP to the program or process that was interrupted.

The interrupt controller collects and formats data about the type of interrupt that occurred. This data is placed in a special stack (see Figure 2-15) which the interrupt controller creates for the interrupt handling procedures of the MCP. After the interrupt controller has created and filled the interrupt handling stack, a program entry is made into the interrupt handling procedures of the MCP.

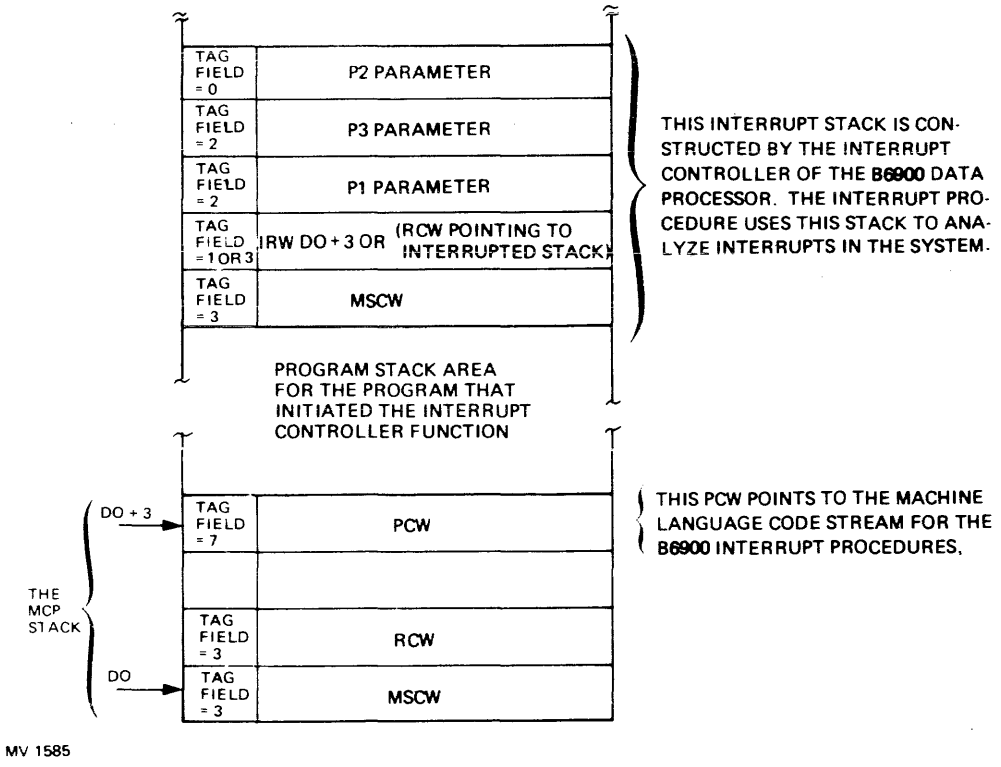


Figure 2-15. B 6900 Interrupt Stack Organization

P1 Parameter

The format and content of the data that is placed in the interrupt handling stack depends on the type of interrupt that occurred. There are five types of interrupts that are recognized by the interrupt controller of the DP, which are: Alarm type, Hardware type, General Control type, External type, and Syllable Dependent type. The first word of data in the interrupt stack is the P1 parameter. The P1 parameter defines the type of interrupt that was sensed, and indicates the cause of the interrupt. Table 2-3 shows the types of interrupts that are defined in the P1 parameter, and also shows the various causes of each type of interrupt. The P1 parameter is the first half (upper half) of a double-precision word. The last half (lower half) of the double precision word is the P3 parameter. Table 2-4 shows what information about an interrupt is to be present in the P2, and P3 parameters of the interrupt handling procedure stack.

P3 Parameter

The P3 parameter is the second half of a double precision word in the interrupt handling procedure stack.

The purpose of the P3 parameter is to provide a place to record the hardware operating environment conditions when an interrupt occurs. The B 6900 system uses the information contained in the P3 parameter to help analyze the cause of the interrupt.

The information contained in the P3 parameter is also valuable in determining the cause of a hardware failure which results in an operating system interrupt. The information that is present in the P3 parameter is recorded in the SYSTEM SUMLOG file, and thus is available to help maintenance personnel in determining the cause of hardware failures.

The P3 parameter has a variable format that depends on the type of interrupt that has occurred. There are five different formats, but only one format is used for each type of interrupt. Figure 2-16 shows the formats that are used for Alarm type, Hardware type, Syllable Dependent type, and General Control type interrupts. Table 2-4 shows what data is present in the P3 parameter for the specific cause of each of the five types of interrupts.

P2 Parameter

The P2 parameter for the B 6900 typically contains the contents of the top-of-stack register at the time the interrupt occurred. This context is true for alarm type interrupts with the single exception of the stack underflow interrupt. In the case of the stack underflow interrupt the value of the S-register will be placed in the P2 parameter word.

The B 6900 system P2 parameter for syllable dependent interrupts contains additional information. The additional information that is contained in the P2 parameter follows:

a. For a sequence error that occurs during a family C operation the P2 parameter will contain the value of the word that caused the sequence error.

b. For an invalid operation interrupt that occurs during a SPLT (9543) operator the word that caused the interrupt will be reported in the P2 parameter.

c. For an invalid operation interrupt that occurs during a JOIN (9542) operator function the word that caused the interrupt will be reported in the P2 parameter. If the information in both the A and B registers is bad then the word in the A register becomes the P2 parameter data.

The B 6900 system external type interrupts are used for I/O finished interrupts.

Table 2-3. P1 Parameter Words (Sheet 1 of 2)

Type	Cause	Parameter Bits 46 45 44 39 27 26 25 24 23 22 21 20 19 18 17 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Alarm	Loop Timer	1 1 Ø Ø 1
Alarm	Memory Addr Parity	1 1 Ø Ø Ø 1
Alarm	Scan Bus Parity	1 1 Ø Ø 1
Alarm	Inv Address-Local	1 1 Ø Ø 1
Alarm	Stack Underflow	1 1 Ø Ø 1
Alarm	Inv Program Word	1 1 Ø Ø 1
Alarm	Memory Address Residue	1 1 Ø Ø Ø 1
Alarm	Read Data Mult. Error	1 1 Ø Ø Ø 1
Alarm	Inv Address Global	1 1 Ø Ø Ø 1
Alarm	Global Memory Not Ready	1 1 Ø Ø Ø 1
Hardware	PROM Card Parity	1 1 Ø Ø 1
Hardware	RAM Card Parity	1 1 Ø Ø 1
Hardware	Bus Residue	1 1 Ø Ø 1
Hardware	Adder Residue	1 1 Ø Ø 1
Hardware	Compare Residue	1 1 Ø Ø 1
Gen. Control	Read Data Single Error	Ø Ø 1 1 X Ø 1
Gen. Control	Read Data Retry	Ø Ø 1 1 X Ø 1
Gen. Control	Read Data Check Bit	Ø Ø 1 1 X Ø 1
Gen. Control	Address Retry	Ø Ø 1 1 X Ø 1

NOTES:
1. 1 = BIT is a binary one.
   0 = BIT is a binary zero.
   Ø = BIT may be either a binary one or a binary zero.
   X = State of bit is immaterial.
2. Bit 18 indicates whether the operation is a memory operation to the Global Memory:
   If bit 18 = 0 it was a memory operation.
   = 1 it was a scan operation
3. If bit 17 is a binary one it indicates that the data in the P3 parameter is inconsistent.
4. Bit 27 is the B 6900 bit. This bit is true for B 6900 systems.

Table 2-3. P1 Parameter Words (Sheet 2 of 2)

Type	Cause	Parameter Bits 46 45 44 39 27 26 25 24 23 22 21 20 19 18 17 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
External	I/O Finished	1 X 1 1 1
SDI	Programmed Operator	ø 1 1 ø
SDI	Memory Protected	ø 1 1 ø 1
SDI	Invalid OP	ø 1 1 ø 1
SDI	Divide by Zero	ø 1 1 ø 1
SDI	Exp. Overflow	ø 1 1 ø 1
SDI	Exp. Underflow	ø 1 1 ø 1
SDI	Invalid Index	ø 1 1 ø 1
SDI	Integer Overflow	ø 1 1 ø 1
SDI	Bottom of Stack	ø 1 1 ø 1
SDI	Presence Bit	RT RT ø VS 1 1 ø 1
SDI	Seq. Error	ø 1 ø ø ø 1
SDI	Segm. Array	ø 1 1 ø 1
SDI	Interval Timer	ø 1 1 ø 1
SDI	Stack Overflow	ø 1 1 ø 1
SDI	Confidence Error	ø 1 1 X 1

NOTES: 1. 1 = BIT is a binary one.
0 = BIT is a binary zero.
ø = BIT may be either a binary one or a binary zero.
X = State of the bit is immaterial.

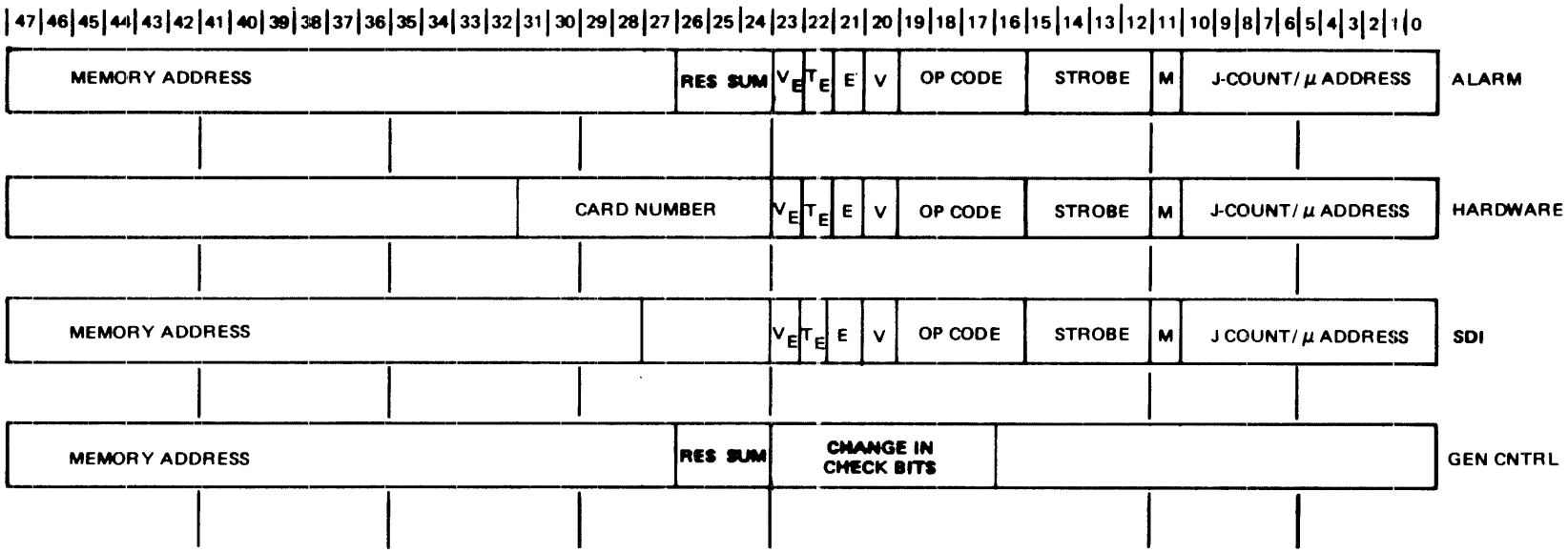
2. Bit 17 is the B 6900 bit. This bit is 1 for B 6900 systems.

Table 2-4. Interrupt Procedure Stack Parameter Contents

	Kind of Error	Interrupt Type P1 Parameter	Contents of the P2 Parameter	Contents of the P3 Parameter
1.	Loop Timer	Alarm		Strb, JC, Op
2.	Memory Address Parity	Alarm		Addr, JC, Strb, Op
3.	Inv. Address, Local	Alarm		Addr, JC, Strb, Op
4.	Stack Underflow	Alarm	S Register	Addr, JC, Strb, Op
5.	Inv. Progr. Word	Alarm	Word	JC, Strb, Op
6.	Memory Address Residue	Alarm		Addr, JC, Strb, Op
7.	Read Data Multiple Error	Alarm	Word	Addr, JC, Strb, Op
8.	Inv. Addr, Global	Alarm		Addr, Strb, JC, Op
9.	Global Memory Not Ready	Alarm		Addr, Strb, JC, Op
1.	Prom Card Parity	Hardware		JC, Strb, Op, Card #
2.	RAM Card Parity	Hardware		JC, Strb, Op, Card #
3.	Bus Residue	Hardware		JC, Strb, Op
4.	Adder Residue	Hardware		JC, Strb, Op
5.	Compare Residue	Hardware		JC, Strb, Op
1.	Read Data Single Error	Gen. Cntr.		Addr, Bit #
2.	Read Data Retry	Gen. Cntr.		Addr
3.	Read Data Check Bit	Gen. Cntr.		Addr, Bit #
4.	Address Retry	Gen. Cntr.		Addr
1.	I/O Finished	External	Empty	Empty
1.	Programmed Operator	SDI	See the text under the	JC, Str, Op
2.	Memory Protected	SDI	subheading titled P2	JC, Str, Op
3.	Invalid Op	SDI	Parameter	JC, Str, Op
4.	Divide by zero	SDI		JC, Str, Op
5.	Exponent Overflow	SDI		JC, Str, Op
6.	Exponent Underflow	SDI		JC, Str, Op
7.	Invalid Index	SDI		JC, Str, Op
8.	Integer Overflow	SDI		JC, Str, Op
9.	Bottom of Stack	SDI		JC, Str, Op
10.	Presence Bit	SDI		JC, Str, Op
11.	Seq. Error	SDI		JC, Str, Op
12.	Segm. Array	SDI		JC, Str, Op
13.	Interval Timer	SDI		JC, Str, Op
14.	Stack Overflow	SDI		JC, Str, Op
15.	Confidence Error	SDI		JC, Str, Op

Footnotes:

Addr	is the Memory or Scan address	OP	is the Op code
Strb	is the family strobe	Card #	is the number of the failing card
JC	is the family seq. counter count	Bit #	is the number of the failing bit



RES SUM = RESIDUE OF ADDRESS
V_E = VECTOR
T_E = TABLE
E = EDIT
V = VARIANT
M = MODED
M = MODE 0 MEANS J-COUNT IS ACTIVE.
1 MEANS μ ADDRESS IS ACTIVE

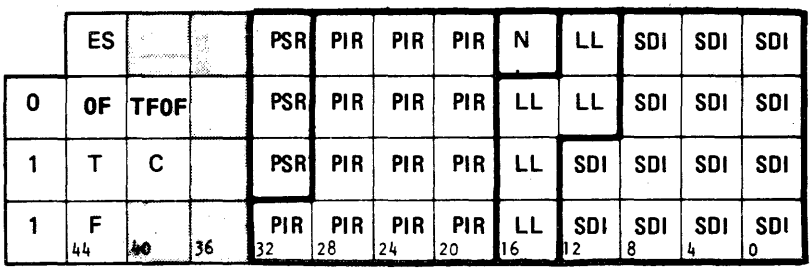
MV 1586

Figure 2-16. P3 Parameter Configurations

RETURN CONTROL WORDS

A return control word is used in the B 6900 system to provide a method for controlling a return to a previous procedure. The second entry in an active job stack is always a return control word. The hardware of the B 6900 system automatically creates the return control word (RCW) for a previous procedure or program when an entry to a new procedure is made. Prior to the hardware inserting the return control word into the stack, the second word in the stack is either a PCW or an IRW. The return control word is substituted for whichever type of word is the second word in the new procedure stack.

Figure 2-17 shows the fields of data that are present in the RCW, and defines the meaning of the data in each field. The combination of data fields that are stored in the RCW indicates what the hardware environment will be after the return to the previous procedure has been made.



50:3 = TAG FIELD.
(ALWAYS A VALUE OF 3 FOR AN RCW)
BIT 47 = EXTERNAL SIGN BIT FLIP-FLOP STATE
BIT 46 = OVERFLOW FLIP-FLOP STATE
BIT 45 = TRUE/FALSE FLIP-FLOP STATE
BIT 44 = FLOAT FLIP-FLOP STATE
BIT 42 = TRUE/FALSE FLIP-FLOP OCCUPIED FLIP-FLOP STATE
BIT 41 = COMPARE FLIP-FLOP
35:3 = VALUE OF PROGRAM SYLLABLE REGISTER FIELD
32:13 = VALUE OF PROGRAM INDEX REGISTER FIELD
BIT 19 = NORMAL/CONTROL STATE FLIP-FLOP STATE;
BINARY ZERO = NORMAL STATE
BINARY ONE = CONTROL STATE
18:5 = VALUE OF LEXICOGRAPHICAL LEVEL REGISTER
13:14 = SEGMENT DESCRIPTOR INDEX VALUE

MV 1591

Figure 2-17. Return Control Word

PROGRAM WORDS (CODE WORDS)

Program words are B 6900 words that contain the machine language instructions which the data processor executes. Program code words are grouped into units of words called segments. A segment consists of all the machine language code for a program or a segment of a program. A program segment may consist of from one program code word, to a maximum of 16,384 words. It is unusual for a program segment to exceed several hundred words. Each segment of program code in a program is referenced (and located) through the segment descriptor index field in the PCW that calls the segment to be executed by the data processor. A segment of code may call upon the system to execute another segment of code. At the conclusion of such a called segment, the system will return to the calling segment. The location of the code for the calling segment is not lost during the execution of the called segment code because the RCW of the called segment contains the SDI value for the code of the calling procedure. Thus when returning to the calling procedure the code segment location is known.

PROGRAM SEGMENTS AND THE SEGMENT DESCRIPTOR

The program code that is executed when a program job or task is performed is contained in words of machine language operator codes. All of the operator codes that comprise the task are grouped together in groups called segments. A segment may contain all of the machine language operators, or a major group of the operator codes in a program task.

When a program task is to be executed, an ENTER operator causes the PCW for the task to be brought into the stack, and distributed to the various parts of the operating system. The SDI field of the PCW word (see Figure 2-13) locates a segment descriptor (SD) for the program task. A description of the SD (Figure 2-18) is as follows:

bits 50:3 — The tag field. The tag for a SD is always three (011 binary).

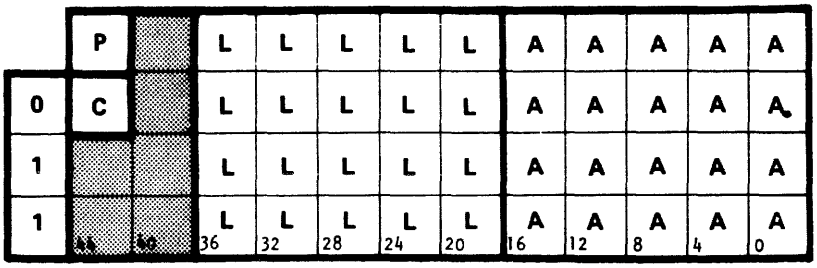
bit 47:1 — The presence bit. If this bit is binary one then the program code segment is present in local memory.

bit 46:1 — The copy bit. If this bit is a binary zero then the segment descriptor is the original segment descriptor. If this bit is a binary one then this descriptor is a copy of an original segment descriptor.

bits 45:6 — An unused field. These bits may be either binary ones or zeroes because they have no effect upon the use of the word as a segment descriptor.

bits 39:20 — The length field. This field specifies the length of the code segment, in words, in binary notation.

bits 19:20 — The address field. If the presence bit is a binary one then this field contains the absolute address of the first word in the segment. If the presence bit is a binary zero and the copy bit is also a binary zero then this field contains a five digit binary coded decimal disk address for the code segment. If the presence bit is a binary zero and the copy bit is a binary one then this field contains the absolute memory address of the original segment descriptor.



50:3 = TAG FIELD.
(ALWAYS A VALUE OF 3 FOR A SEGMENT DESCRIPTOR)
47:1 = PRESENCE BIT, 1 = PRESENT IN MEMORY
0 = PRESENT IN LIBRARY
46:1 = COPY BIT, 1 = COPY OF ORIGINAL SEGMENT DESCRIPTOR
0 = ORIGINAL SEGMENT DESCRIPTOR
39:20 = LENGTH FIELD - THE NUMBER OF WORDS IN THE SEGMENT
19:20 = ADDRESS FIELD - THE BEGINNING MEMORY ADDRESS IF [47:1] = 1.
- THE DISK OR PACK ADDRESS IF [47:1] = 0, AND [46:1] = 0.
- THE MEMORY ADDRESS OF THE ORIGINAL SEGMENT DESCRIPTOR IF [46:1] = 1, AND [47:1] = 0.

MV1689

Figure 2-18. Segment Descriptor Word

A program code segment may call another program segment to be executed. Each of these program code segments (the calling segment, and the called segment) has a separate segment descriptor. The address (SDI) for the current code segment is saved in the data processor IC memory registers. The value of the called SDI is saved when the called segment is executed. However, the SDI for the calling segment is not lost, because this address is saved in the RCW (refer to Figure 2-17). Thus. when a called segment is executed, and a return (or EXIT) to the calling segment is performed, the SDI is always available for the currently executing program segment.

The use of copy segment descriptors, and the mechanism for saving the SDI values for segments of program code are basic components used to provide for the concepts of reentrant code. Reentrant code techniques are defined in Section 3 of this manual.

A program code word is composed of six syllables, and a tag field (see Figure 2-19). The tag field for a program code word is always a value of three. The remaining 48 bits of the program word are divided into six 8-bit syllable fields. A machine language instruction consists of from one to seven syllables. An instruction is not limited to a single code word but may extend across the boundary of a code word, and into the next word of program code in sequence. For this reason the contents of a word of machine language code may be portions of two operators, plus from one to four complete operator codes.

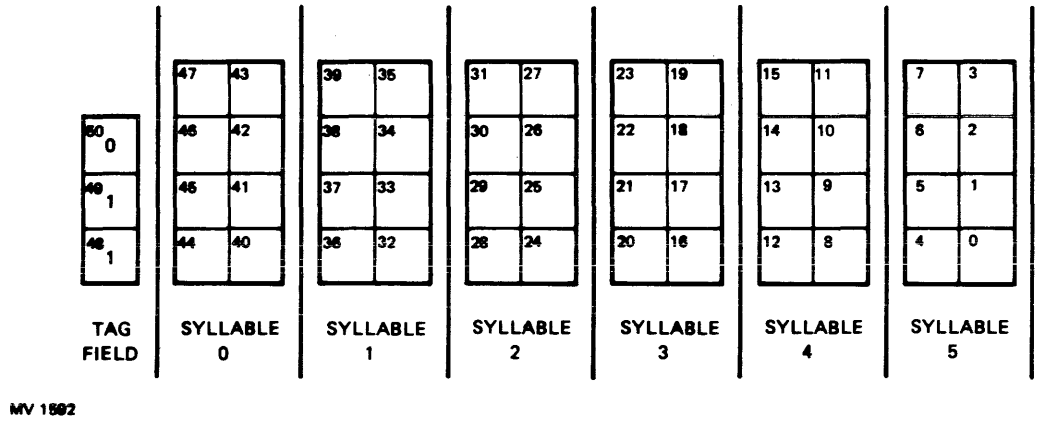


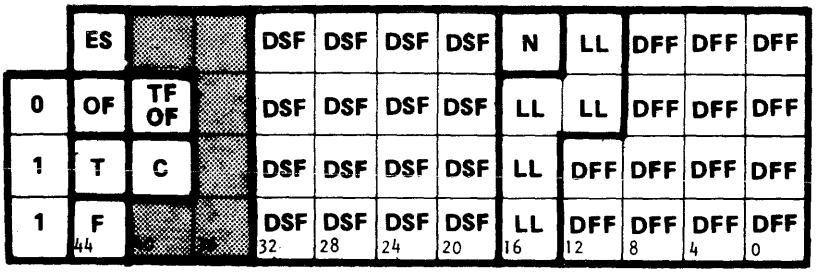
Figure 2-19. Program Word Format

TOP-OF-STACK CONTROL WORDS

A top of stack control word (see Figure 2-20) is originated when the data processor executes the move to stack operator. This word occupies the address in memory of the lower word boundary for a job or task area. A TOSCW contains the relative addressing and environment record for the program or task. The address of a TOSCW for an operating program or task is the same as the value of the BOSR address register. A TOSCW therefore also corresponds to the address of the first MSCW for a job or task.

The addressing environment for a program or task consists of the values of the BOSR, F, S, and lexicographical level registers. The values of these registers are stored in the TOSCW when another program or task is to be executed. Upon re-entry into the program or task procedures, the proper values from the TOSCW are used to restore the proper addressing environment for the program or task, in the memory address registers.

The operating environment of a job or task consists of the state of seven flip-flops. These flip-flops are the external sign, overflow, true/false, float, true/false occupied, compare, and normal/control state flip-flops. The state of these flip-flops is stored in the TOSCW when another job or task is to be executed. Upon re-entry into the original job or task, the proper values for operating environment flip-flops are restored from the TOSCW.



50:3 = TAG FIELD.
(ALWAYS A VALUE OF 3 FOR A TOSCW)
47:1 = EXTERNAL SIGN FLIP-FLOP
46:1 = OVERFLOW FLIP-FLOP
45:1 = TRUE FALSE FLIP-FLOP
44:1 = FLOAT FLIP-FLOP
42:1 = TRUE FALSE OCCUPIED FLIP-FLOP
41:1 = COMPARE FLIP-FLOP
55:16 = DELTA S-REGISTER FIELD (VALUE OF THE S-REGISTER DISPLACEMENT ABOVE BOSR)
19:1 = NORMAL/CONTROL STATE OF FLIP-FLOP;
0 = NORMAL STATE
1 = CONTROL STATE
18:5 = LEXICOGRAPHICAL LEVEL
13:14 = DELTA F REGISTER FIELD (VALUE OF THE F-REGISTER DISPLACEMENT, BELOW THE VALUE OF THE S-REGISTER)
2:3 = THE CPU PROCESSOR ID VALUE (001) WHEN THE TOSCW IS FOR AN ACTIVE PROCESS PROGRAM OR TAST

MV1690

Figure 2-20. TOSCW Word Layout

The TOSCW for the currently operating program or task does not contain the operating and addressing environment. Instead, the CPU data processor identity (001 for a B 6900 system) is stored in bits 2:3, and the rest of the bits (except the tag field) are zeroes. The presence of a TOSCW which only contains the data processor identity field indicates the address of the lowest word in the current job or task stack. This word is addressed by the value of the BOSR register.

SECTION 3

STACK AND REVERSE POLISH NOTATION

THE STACK

The stack is the memory storage area assigned to a job. The stack provides storage for the basic program and data references for the job. It also provides for temporary storage of data and job history. When a job is activated, four high-speed hardware registers (A, X, B, Y) are linked to the memory portion of the job's stack (see Figure 3-1). This linkage is established by the stack pointer register (the S register), which contains the memory address of the last word placed in the stack. The four hardware top-of-stack registers (A, X, B, Y) extend the stack to provide quick access for data manipulation. Another stack pointer value (the F register) always points to the most recent MSCW in the stack.

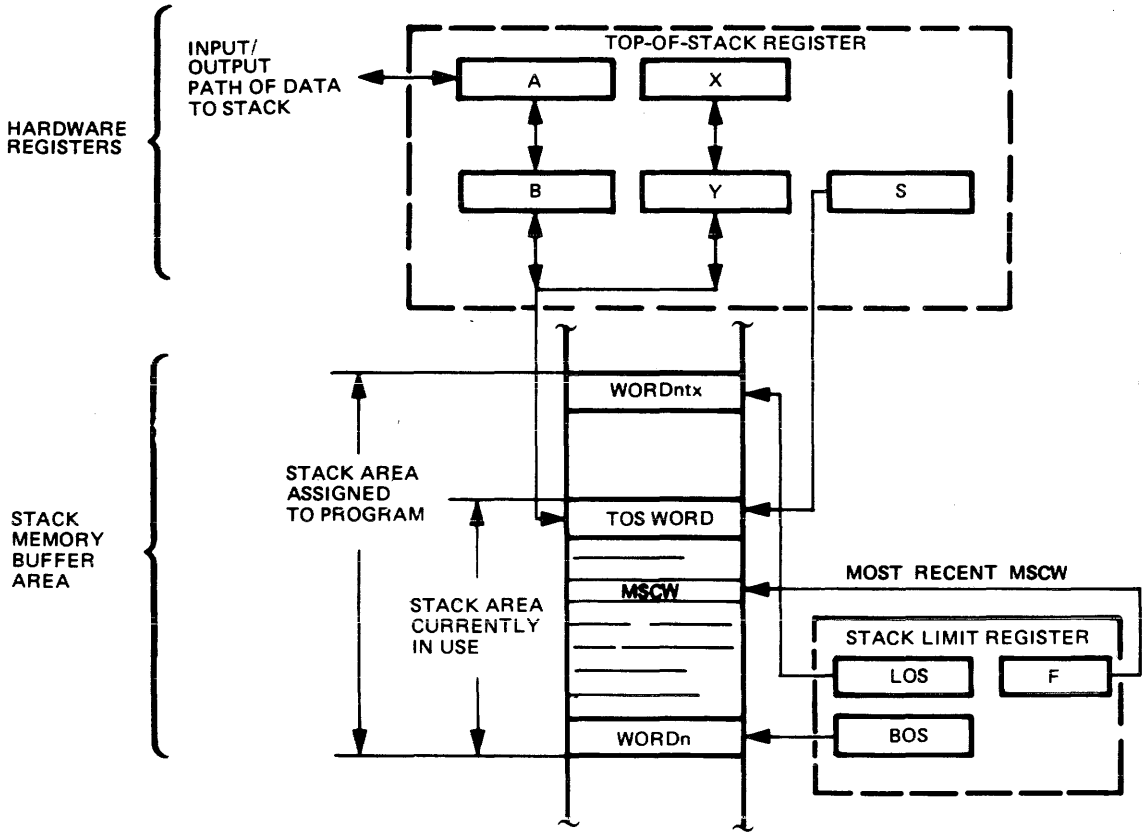


Figure 3-1. Top-of-Stack and Stack Bounds Registers

The number of words in the memory portion of the stack is equal to the difference between the values of the BOS register, and the S register (S minus BOS). Data are brought into the stack through the top-of-stack registers in a manner that the last word placed in the stack (as indicated by the value of the S register) is the first word to be extracted from the stack (last in first out method). The total capacity of the top-of-stack registers is two words or two operands. Loading a third word or operand into the top-of-stack registers causes the third word or operand to be pushed from the top-of-stack registers into the memory portion of the stack. The stack pointer value in the S register is incremented by one as a word or operand is pushed into the memory portion of the stack, and is decremented by one when a word or operand is withdrawn from the stack area and placed in the hardware top-of-stack registers. As a result, the S register continually points to the last word or operand placed into the memory portion of the job stack.

BASE AND LIMIT OF STACK

A job's stack is bounded, for memory protection, by two registers: the base-of-stack register (BOSR) and the limit-of-stack register (LOSR). The contents of BOSR define the base of the memory portion of the stack, and the contents of LOSR define the upper limit of the memory portion of the stack. The job is interrupted if the S register is set to a value that is present in either the BOSR, or the LOSR register. If the S register equals or exceeds the value of the LOSR register value a stack overflow interrupt occurs.

BI-DIRECTIONAL DATA FLOW IN THE STACK

The contents of the top-of-stack registers are maintained automatically by the data processor to meet the requirements of the current machine language operator. If the current operator requires data transfer into the memory portion of the stack, the top-of-stack registers receive the incoming data, and surplus contents in the top-of-stack registers are pushed down into the memory portion of the stack. Pushing data into the memory portion of the stack means that the bottom word or operand in the top-of-stack register is transferred to the next word or operand in sequence, in the memory portion of the stack. Pushing data down into the memory portion of the stack makes room in the top-of-stack registers to contain the incoming data that is required by the current machine language operator.

Data are also automatically brought from the memory portion of the stack and placed in the top-of-stack registers when the machine language operator requires that the top-of-stack registers be filled. This automatic function is the opposite of the push function described in the previous paragraph, and is commonly called a push up function. A push up transfers the last operand or word in the memory portion of the stack into the second word position in the top-of-stack registers. The word or operand in the memory portion of the stack is then deleted by decrementing the S register. The automatic maintenance of the top-of-stack registers takes the form of "push down", and "push up" functions which are described in the following paragraphs.

Stack Push Down

A stack push down occurs when a third word or operand is loaded into the top-of-stack registers, and both the A register and B register already contain stack words or operands. A push down consists of moving data from the top-of-stack registers to the local memory portion of the stack. Moving data to the local memory portion of the stack makes room in the top-of-stack registers so that a third operand may be loaded into the top-of-stack registers.

Stack Push Up

A stack push up occurs when an operand or word is moved from the local memory portion of the stack, to the top-of-stack register portion of the stack. A push up can only occur when a machine language operator is executed by the data processor. The data processor operator that is to be performed must require that words or operands be present in the top-of-stack registers, and such words or operands must not be present in the proper top-of-stack registers.

DOUBLE-PRECISION STACK OPERATION

The top-of-stack registers are operand oriented rather than word oriented. Calling a double-precision operand into the top-of-stack registers causes two memory words to be loaded into the top-of-stack registers. The first word is loaded into the A register, where TAG bits are checked. If the value indicates double-precision, the second word is loaded into the X register. The A and X registers are concatenated, or linked together, to form the double-precision operand. A double-precision operand located in the B and Y registers reverts to two words when pushed down into the memory portion of the stack. A double-precision operand is concatenated in the B and Y registers when pushed up from the memory portion of the stack into the hardware register portion of the stack.

TOP-OF-STACK REGISTER CONDITIONS

Two logical indicators are used to indicate the condition of the top-of-stack register portion of the stack. These two indicators are AROF (A register is occupied flip-flop), and BROF (B register is occupied flip-flop). The meaning of these two logical indicators is as follows:

AROF	BROF	MEANING
0	0	Neither the A, or the B register contains valid data. The top word in the stack is presently located in the memory address specified by the contents of the S register.
0	1	The B register contains the top word in the stack, and the contents of the A register are not valid data. The second word in the stack is presently located in the memory address specified by the contents of the S register.
1	0	The A register contains the top word in the stack, and the contents of the B register are not valid data. The second word in the stack is presently located in the memory address specified by the contents of the S register.
1	1	The A register contains the top word in the stack, and the second word in the stack is presently in the B register. The third word in the stack is in the memory address specified by the contents of the S register.

STACK ADJUSTMENTS

Each machine language operator that is executed by the data processor contains the requirement to adjust the top-of-stack registers so that their contents provide accommodation for the operation that is to be performed. A convention is used to show what stack adjustment is required, as follows:

CONVENTION NOTATION	MEANING
(ADJ 0,0)	Both the A and B registers are to be adjusted so that their contents are not valid. The top word in the stack is to be located in the memory address pointed at by the contents of the S register. The data processor will use the state of the AROF and BROF flip-flops to determine if the stack must be pushed down to achieve the required adjustment. The 0,0 portion of the convention notation shows what the logical states of AROF and BROF must be to satisfy the requirements of the adjustment. The first 0 in the expression of the notation defines what the logical state of the AROF flip-flop must be at the conclusion of the stack adjustment. The second 0 in the expression defines what the logical state of the BROF flip-flop must be at the conclusion of the adjustment. The ADJ portion of the convention notation reads "adjust the stack until AROF and BROF meet the logical states".
(ADJ 0,1)	The A register is to be adjusted so that its contents are not valid. The top word or operand in the stack is to be present in the B register, and the second word or operand in the stack is to be located in the memory address pointed at by the contents of the S register.
(ADJ 1,0)	The A register is to be adjusted so that its contents are the top word or operand in the stack. The B register must not contain valid data. The second word or operand in the stack is to be located in the memory address pointed at by the contents of the S register.
(ADJ 1,1)	The A register is to be adjusted so that it contains the top word or operand in the stack. The B register is to be adjusted so that it contains the second word or operand in the stack. The third word or operand in the stack is to be in the memory address pointed at by the contents of the S register.
(ADJ 0,2)	The A register is to be adjusted so that its contents are not valid. The B register condition is immaterial to the operation. The top word in the stack is present in the B register if BROF is set.
(ADJ 1,2)	The A register is to be adjusted so that it contains the top word in the stack. The B register condition is immaterial to the operation. The second word in the stack is located in the B register if BROF is set.
(ADJ 1,3)	The A register is adjusted so that it contains the top word in the stack if and only if the original stack condition is AROF/ and BROF/ (0,0). If any other condition than (0,0) is the original condition, then no stack adjustment occurs.

Some machine language operations require that several stack adjustments must be performed during the course of the operation. Such operations merely pause at the appropriate place until the adjustment is completed, and then continue the sequence.

Stack push down and/or stack push up (which were defined previously in this section) are intrinsic functions of the stack adjustments. That is, a push-up or a push-down may be implied because of the current state of the top of stack registers, and the required stack adjustment. Where a stack push-up or push-down is implied, such operation will be performed as an integral and automatic function of the stack adjustment procedure.

DATA ADDRESSING

The B 6900 data processor provides three methods for addressing data or program code:

a. Data descriptor (DD)/segment descriptor (SD)

b. Indirect reference word (IRW)

c. Stuffed indirect reference word (SIRW)

The data descriptor (DD) and segment descriptor (SD) provide for the add essing of data or program segments located outside of the job's stack area. Data descriptors and segment descriptors utilize absolute memory addresses. The indirect reference word (IRW) and the stuffed indirect reference word (SIRW) address data located within (IRW), or outside (SIRW) the job's stack. The IRW and SIRW address components are both relative. The IRW addresses within the immediate environment of the job relative to a display register (described later in Non-local Addressing). The SIRW addresses beyond the immediate environment of the current procedure, the addressing being relative to the base of the job's stack. Addressing across stacks is accomplished with an SIRW.

Data Descriptor

In general, the descriptor describes and locates data associated with a given job. The data descriptor (DD) is used to fetch data to the stack or to store data from the stack into an array located outside the job's stack area. The formats of the data and segment descriptors were illustrated in Section 2. The address field in each of these descriptors is 20 bits in length; this field contains the absolute address of an array in memory or in the disk file, as indicated by setting of the presence bit (P). The referenced data is in main memory when the presence bit is set.

Presence Bit

A presence bit interrupt occurs when the job references data by means of a descriptor in which the P-bit is equal to zero: that is, the data is located in a disk file, rather than in memory. The Master Control Program (MCP) recognizes the presence bit interrupt and transfers data from disk file storage to memory. After the data transfer to memory is completed, the MCP marks the descriptor present by setting the P-bit to one, and places the new memory address into the address field of the descriptor. The interrupted job is then reactivated.

Index Bit

A data descriptor describes either an entire array of data words, or a particular element within an array of data words. If the descriptor describes the entire array, the index bit (I-bit) in the descriptor is zero, indicating that the descriptor has not yet been indexed. The length field of the descriptor defines the length of the data array.

Invalid Index

A particular element of an array is described by indexing an array descriptor. Memory protection is ensured during indexing operations by performing a comparison between the length field of the descriptor and the index value. An invalid index interrupt results if the index value exceeds the length of the local memory area defined by the descriptor, or if the index is less than zero.

Valid Index

If the index value is valid, the length field of the descriptor is replaced by the index value, and the I-bit in the descriptor is set to one to indicate that indexing has taken place. The address and index fields are added together to generate the absolute machine address whenever an indexed data descriptor in which the P-bit is set is used to fetch or store data.

The double-precision bit (D) is used to identify the referenced data as single- or double-precision and directly affects the indexing operation. The D-bit equal to one signifies double-precision and causes the index value to be doubled before indexing.

Read-Only Bit

The read-only bit (R) specifies that the local memory area described by the data descriptor is read-only area. If the R-bit of a descriptor is set to one, and the area referenced by that descriptor is used for storage purposes, an interrupt results.

Copy Bit

The copy bit (C) identifies a descriptor as a copy of a master descriptor and is related to the presence-bit action. The copy bit links multiple copies of an absent descriptor (that is, the presence bit is off) to the one master descriptor. The copy bit mechanism is invoked when a copy is made in the stack. If it is a copy of the original, absent descriptor, the processor sets the copy bit to one and inserts the address of the master descriptor into the address field. Thus, multiple copies of absent data descriptors are all linked back to the master descriptor.

REVERSE POLISH NOTATION

Reverse Polish notation is an arithmetical or logical notational system using only operands and operators arranged in sequence or strings, thus eliminating the necessity for defining the boundaries of any terms. Figure 3-2 presents a flow chart for conversion to reverse Polish notation.

SIMPLIFIED RULES FOR GENERATION OF POLISH STRING

The source of expression is as follows:

Name	Action
Variable or constant	Place variable or constant in string being built and examine next symbol.
Operator-separator "(" or "["	Place in delimiter list and examine next symbol.
Arithmetic or Boolean operator and last-entered delimiter list symbol were as follows: 1. An operator of lower priority. 2. A left bracket "[" or parenthesis "(". 3. A separator. 4. Nothing (delimiter list empty).	Place operator in the delimiter list and examine next source symbol.
Arithmetic or Boolean operator and last-entered delimiter list symbol were as follows: an operator of priority equal to or greater than the symbol in the source.	Remove the operator from the delimiter list and place it in the string being built. Then compare the next symbol in the delimiter list against the source expression symbol.

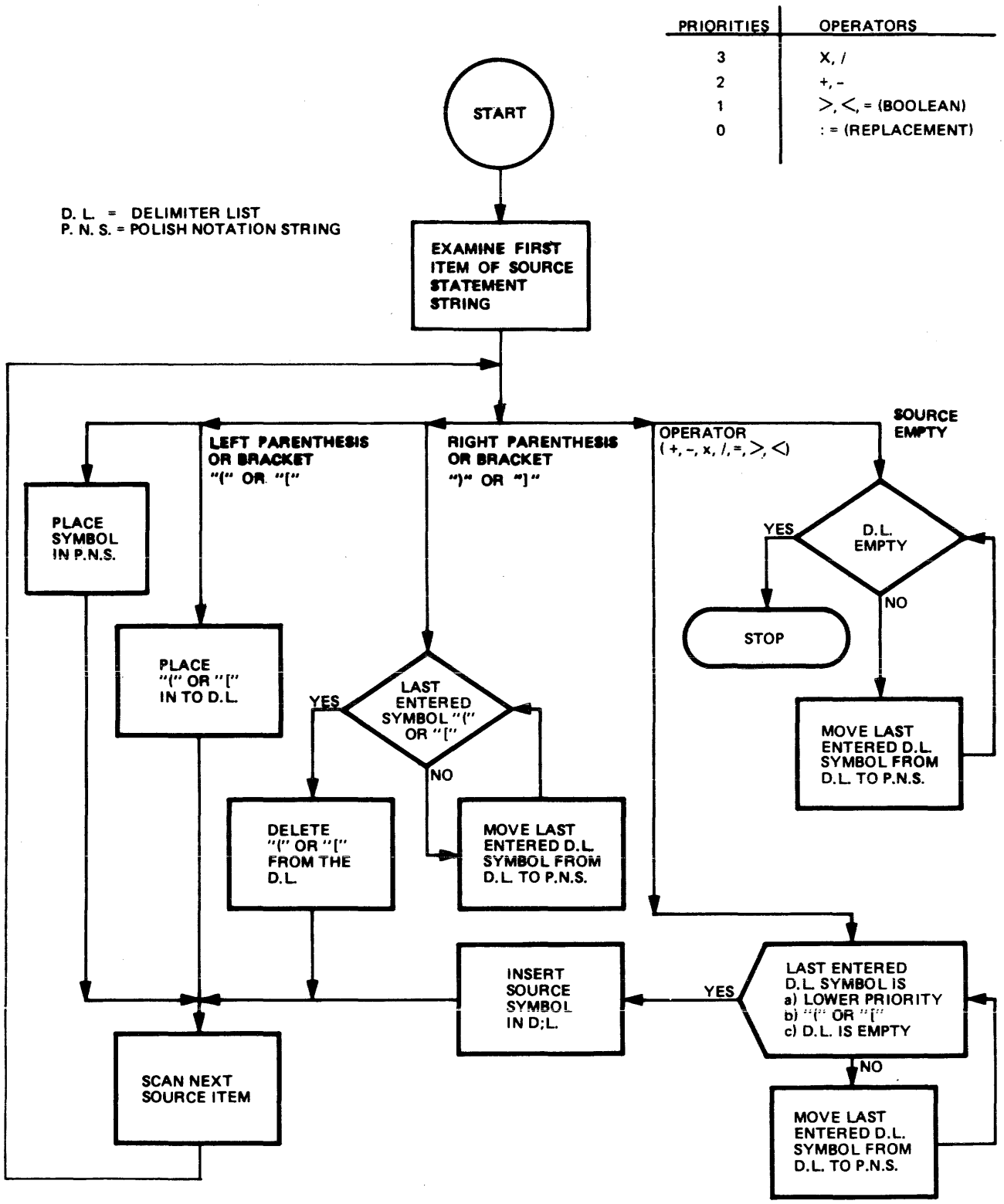


Figure 3-2. Reverse Polish Notation Flow Chart

Name	Action
A right bracket "]" or parenthesis ")".	Pull from delimiter list until corresponding left bracket or parenthesis.
End of expression.	Move last-entered delimiter list symbols to Polish notation string until empty.

POLISH STRING

The essential difference between reverse Polish and conventional notation is that operators are written to the right of the operands instead of between them. For example, the conventional B + C is written B C + in reverse Polish notation: A = 7 x (B + C) becomes A 7 B C + x :=.

Any expression written in reverse Polish notation is called a polish string. In order to fully understand this concept, the user should know the rules for evaluating a polish string.

RULES FOR EVALUATING A POLISH STRING

The following is the procedure for evaluating a polish string:

a. Scan the string from left to right.

b. Remember the operands and the order in which they occur.

c. When an operator is encountered perform the following:

   1. Record the last two operands encountered.

   2. Execute the required operation.

   3. Disregard the two operands.

   4. Consider the result of (b) as a single operand, the first of the next pair to be operated upon.

Following this rule, the reverse polish string A 7 B C + x := results in A assuming the value 7 x (B+C) (Table 3-1).

NOTE

Because replacement operators vary depending upon the language used, ←, =, and := are equivalent for this discussion.

SIMPLE STACK OPERATION

All program information must be in the system before it can be used. Input areas are allocated for information entering the system, and output areas are set aside for information exiting the system; array and table areas are also allocated to store certain types of data. Thus data is stored in several different areas: the input/output areas, data tables (arrays), and the stack. Since all work is done in the arithmetic registers, all information or data is transferred to the arithmetic registers and the stack.

Table 3-1. Evaluation of Polish String A 7 B C + x : =

Step No.	Symbol Being Examined	Symbol Type	Operands Being Remembered Order of Occurrence (1 or 2) Before Operation	Occurring Operation	Operation Results
1	B	Operand			
2	C	Operand	1 B		
3	+	Add Operator	2 C 1 B	B + C	(B + C)
4	7	Operand	1(B + C)		
5	x	Multiply Operator	2 7 1 x (B + C)	7 x (B + C)	7 x (B + C)
6	A	Name	1 7 x (B + C)		
7	:=	Replace Operator	2 A 1 7 x (B + C)	A :=7x(B + C)	A=7x(B + C)

At this point, an ALGOL assignment statement and the reverse Polish notation equivalent will be related to the stack concept of operation. The example is Z:=Y + 2x(W+V), where := means "is replaced by." In terms of a computer program, this assignment statement indicates that the value resulting from the evaluation of the arithmetic expression is to be stored in the location represented by the variable Z.

When Z:=Y + 2x(W+V) is translated to reverse Polish notation, the result is ZY2WV+ x +:=. Each element of the example expression causes a certain type of syllable to be included in the machine language program when the source problem is compiled. The following is a detailed description of each element of the example, the type of syllable compiled, and the resulting operation (see Figure 3-3 and Table 3-2).

In the example statement, Z is to be the recipient of a value, the address of Z must be placed into the stack just prior to the store command. This is accomplished by a name call syllable which places an indirect reference word (IRW) in the stack. The IRW contains the address of Z in the form of an "address couple" that references the memory location reserved in the stack for the variable Z.

Since Y is to be added to a quantity, Y is brought into the top of the stack as an operand. This is accomplished with a value call (VALC) syllable that references Y. The value 2 is then brought to the stack, with an eight-bit literal syllable (LT8). Since W and V are to be added, the respective variables are brought to the stack with value call syllables. The ADD operator adds the two top operands and places the sum in the top of stack. This example assumes, for simplicity, single-precision operands not requiring use of the X and Y registers which are used in double-precision operations.

The multiply operator is the next symbol encountered in the reverse polish string; when executed, it places the product "2x(W+V)" in the top of the stack. The next symbol, ADD, when executed, leaves the final result "Y+2x(W+V)" in the top of the stack.

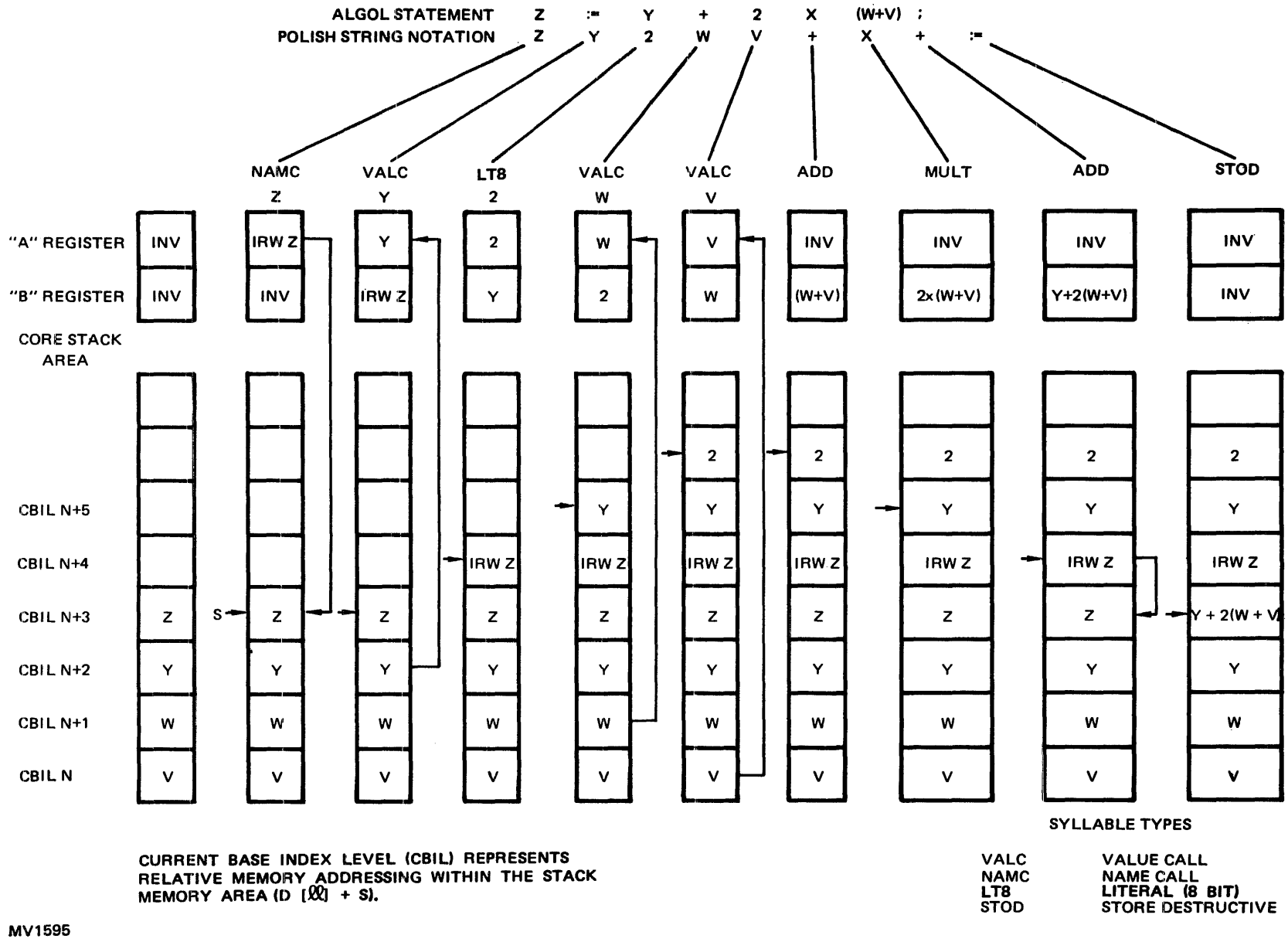


Figure 3-3. Stack Operation

Table 3-2. Description of Stack Operation

Execution Sequence	Reverse Polish Notation Element	Syllable Type Compiled	Function of Syllable During Running of the Program
0			Stack location of program variables illustrated
1	Z	Name call for Z	Build an indirect reference word that contains the address of Z and place it in the top of the stack
2	Y	Value call for Y	Place the value of Y in the top of the stack
3	2	Literal 2	Place a 2 in the top of the stack
4	W	Value call for W	Place the value of W in the top of the stack
5	V	Value call for V	Place the value of V in the top of the stack
6	+	Operator add	Add the two top words in the stack and place the result in B register as the top of the stack
7	x	Operator multiply	Multiply the two top-of-the-stack operands. The product is left in the B register as the top of the stack
8	+	Operator add	Add the two top words in the stack and leave the result in the B register as the top of the stack
9	:=	Operator store destructive	Store an item into memory. The address in which to store is indicated by an indirect reference word or a data descriptor; the address can be above or below the item stored

The store syllable completes the execution of the statement Z:=Y + 2x(W+V). The store operation examines the two top-of-stack operands looking for an IRW or data descriptor. In this example, the IRW addresses the location where the computed value of Z is to be stored. The stack is empty at the completion of this statement.

PROGRAM STRUCTURE IN MEMORY

When a problem is expressed in a source language, portions of the source language fall into one of two categories. One describes the constants and variables that will be used in the program, and the other the computations that will be executed (see Figure 3-4). When the source program is compiled, variables are assigned locations within the stack, whereas the constants are embedded within the code stream that forms the computational part. A program residing in memory occupied separately allocated areas. "Separately allocated" means that each part of the program may reside anywhere in memory, and the actual address is determined by the MCP. In particular, the various areas are not assigned to contiguous memory areas. Registers within the processor indicate the bases of the various areas during the execution of a program.

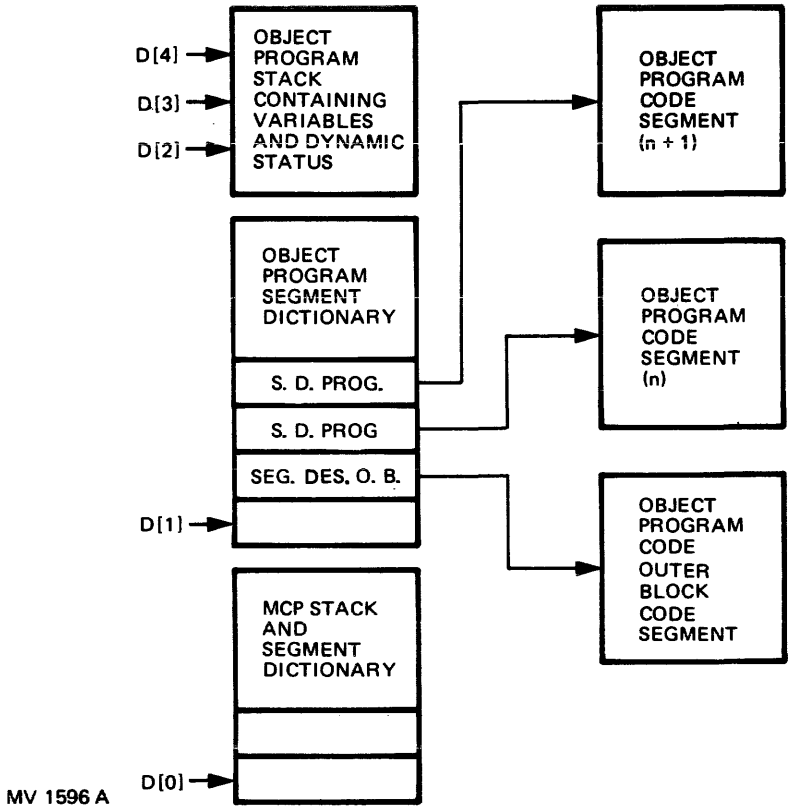


Figure 3-4. Object Program in Memory

MEMORY AREA ALLOCATION

The separately allocated areas of a program are as follows:

a. Program Segments. These are sequences of instructions (syllables) that are performed by the processor in executing the program. Note that there is a distinction between program segments and data areas. The program segments contain no data, and are not modified by the processor as it executes the program.

b. Segment Dictionary. This is a table containing one word for each program segment. This word tells whether the program segment is in memory or on the disk, and gives the corresponding memory or disk address of the program segment.

c. Stack Area. This is the pushdown stack storage, which contains all the variables and data descriptors associated with the program, including control words which indicate the dynamic status of the job as it is being executed.

STACK-HISTORY AND ADDRESSING-ENVIRONMENT LISTS

One very importamt aspect of the B 6900 is the retention of the dynamic history for the program being processed. Two lists of program history are maintained in the B 6900 stack: the stack-history list and the addressing-environment list. The stack-history list is dynamic, varying as the job proceeds along different program paths with changing sets of data. Both lists are generated and maintained by B 6900 hardware.

MARK STACK CONTROL WORD LINKAGE

The stack history is a list of Mark Stack Control Words (MSCW), linked together by their displacement fields (DF) (Figure 3-5). An MSCW is inserted into the stack as a procedure is entered and is removed as that procedure is exited. Therefore, the stack history list grows and contracts with the procedural depth of the program. Mark stack control words identify the portion of the stack related to each procedure. When the procedure is entered, its parameters and local variables are entered in the stack following the MSCW. When the procedure is executed its parameters and local variables are referenced by addressing relative to the MSCW.

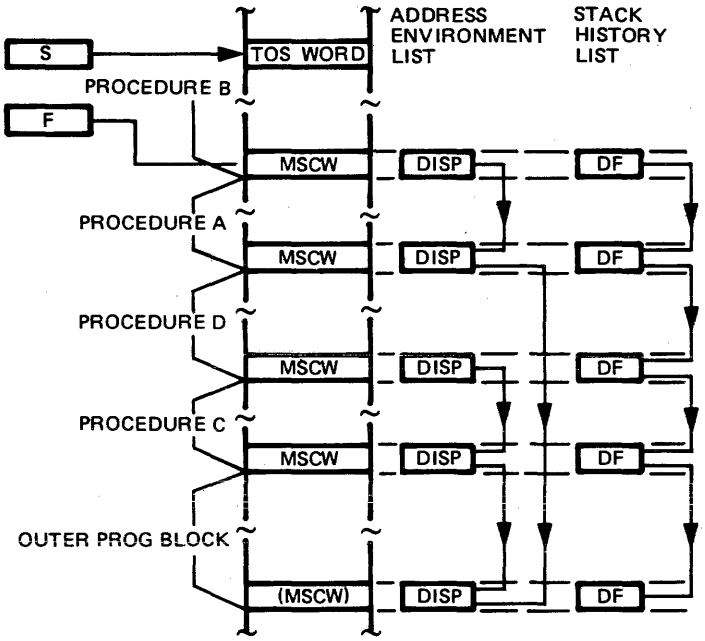


Figure 3-5. Stack History and Addressing Environment List

STACK DELETION

Each MSCW is linked to the prior MSCW through the contents of its DF field in order to identify the point in the stack where the prior procedure began. When a procedure is exited, its portion of the stack is discarded. This action is achieved by setting the stack-pointer register (S) to address the memory cell preceding the most recent MSCW (Figure 3-6). This topmost MSCW, addressed by another register (F), is deleted from the stack-history list by changing F to address the prior MSCW, placing this MSCW at the head of the stack history.

This is an efficient and convenient means of subroutine entry and exit.

RELATIVE-ADDRESSING

Analyzing the structure of an ALGOL program results in a better understanding of the relative-addressing procedures used in the B 6900 stack. The addressing environment of an ALGOL procedure is established when the program is structured

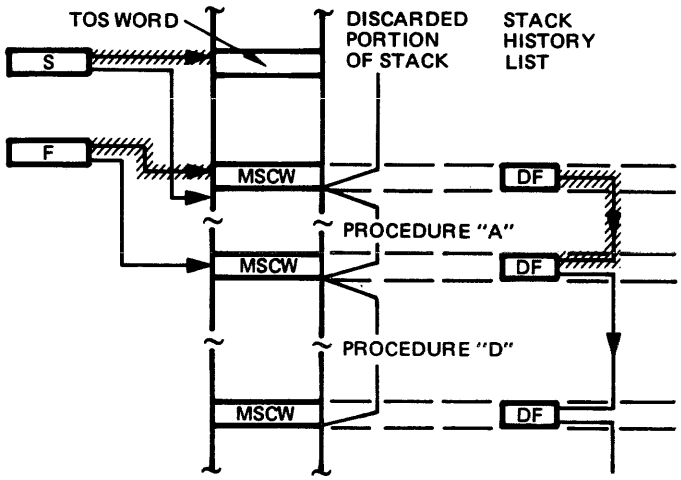


Figure 3-6. Stack Cut-Back Operation on Procedure Exit

by the programmer and is referred to as the lexicographical ordering of the procedural blocks (Figure 3-7). At compile time, the lexicographical ordering is used to form address couples. An address couple consists of the following two items:

a. The addressing level (ℓℓ) of the variable.

b. An index value (S) used to locate the specific variable within its addressing level.

The lexicographical ordering of the program remains static as the program is executed, thereby allowing variables to be referenced by means of address couples as the program is executed.

Base of Address Level Segment

The B 6900 processor contains an array of D registers (D0 through D31). These registers address the base of each addressing-level segment (Figure 3-8). The local variables of all procedures are addressed relative to the D registers.

Absolute Address Conversion

The address couple is converted into an absolute memory address when the variable is referenced. The addressing level portion of the address couple selects the D register which contains the absolute memory address of the MSCW for the environment (addressing level) in which the variable is located. The index value of the address couple is added to the contents of the D register to generate the absolute memory address.

Multiple Variables With Common Address Couples

The address couples assigned to the variables in a program are not unique. This is true because of the ALGOL scope-of-definition rules, which imply that if there is no procedure which can address both of any two quantities, then these two quantities may unambiguously have the same address couple. This addressing system works because, whereas two variables may have the same address couples, there is never any doubt as to which variable is being referenced within any particular procedure.

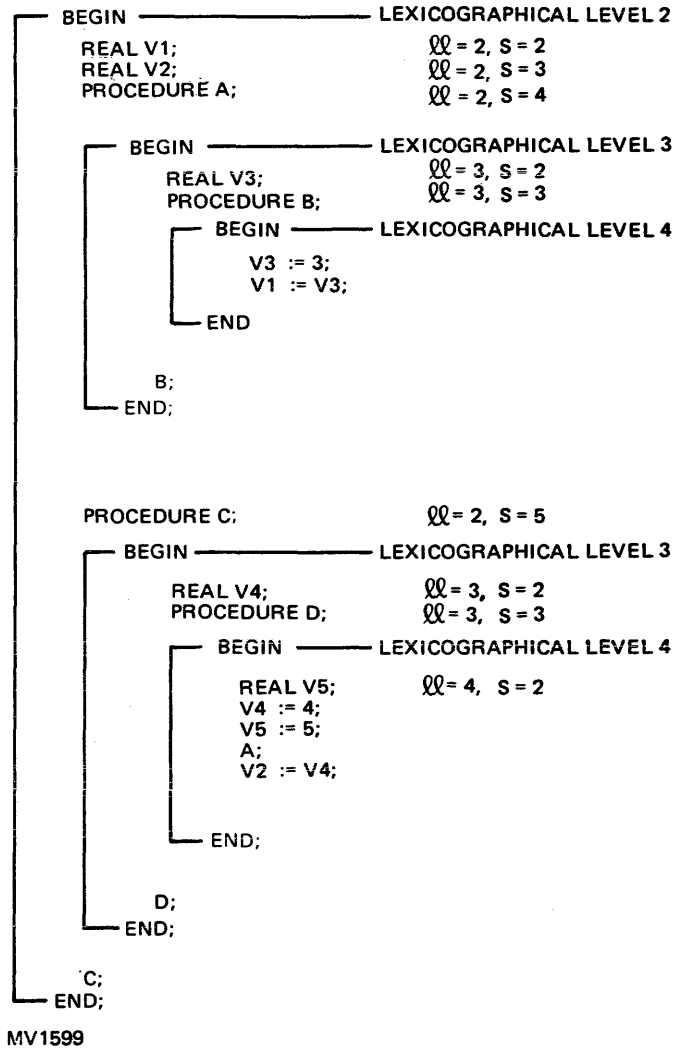


Figure 3-7. ALGOL Program With Lexicographical Structure Indicated

Address Environment Defined

There is a unique MSCW which each D register must address during the execution of any particular procedure. The D registers must be changed, upon procedure entry or exit, to address the correct MSCWs. The list of MSCWs which the D registers address is the addressing environment of the procedure.

Mark Stack Control Word Linkage

The addressing environment of the program is maintained automatically by linking the MSCWs together in accordance with the lexicographical structure of the program. This linkage is the stack number (Stack No.) and displacement (DISP) fields of the MSCW, and is inserted into the MSCW whenever the procedure is entered. The addressing environment list is formed by linking each MSCW to the MSCW immediately below the declaration for the procedure being entered. This forms a tree-structured list which indicates the addressing environment of each procedure (Figures 3-8 and 3-9). This list is used to update the D registers whenever a procedure entry or exit occurs.

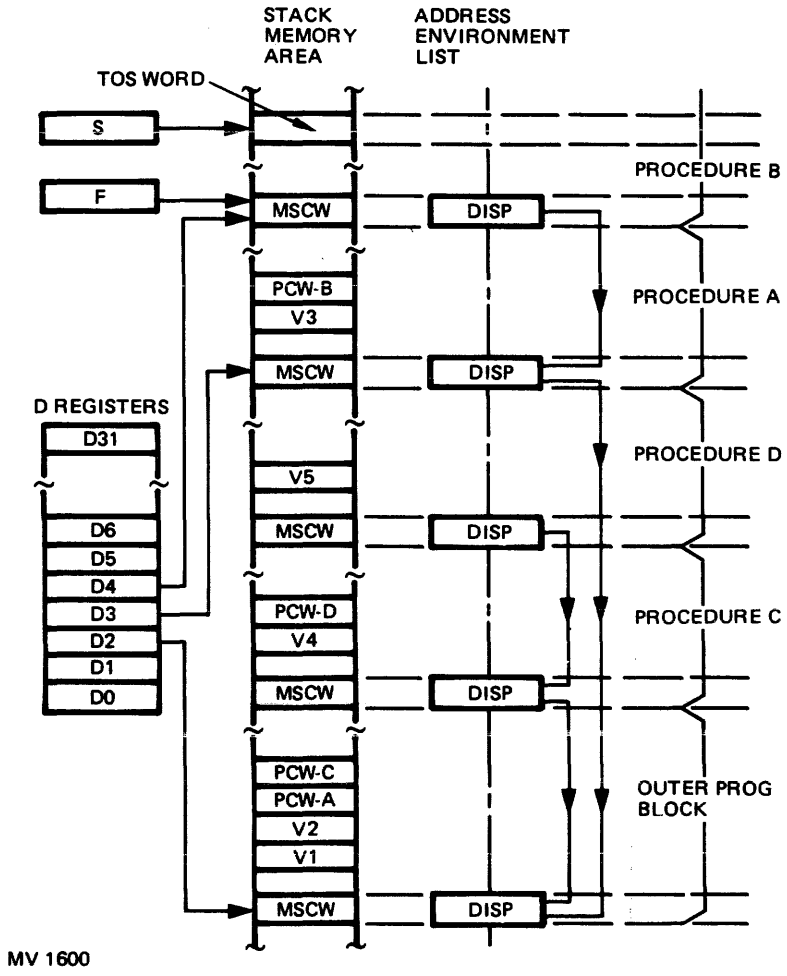


Figure 3-8. D Registers Indicating Current Addressing Environment

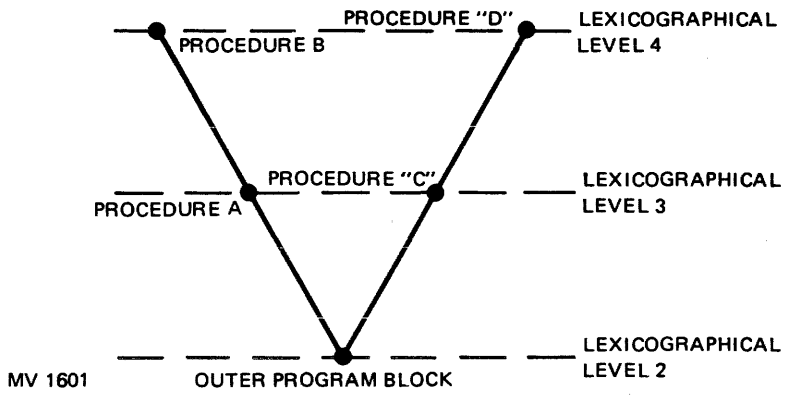


Figure 3-9. Addressing Environment Tree of ALGOL Program

STACK HISTORY SUMMARY

The entry and exit mechanism of the processor hardware automatically maintains both the stack history and address-environment lists to reflect the current status of the program. Interrupt response is a procedure entry. Therefore, the system is able to conveniently respond to, and return from, interrupts. Upon recognition of an interrupt condition, the processor creates a MSCW, inserts an indirect reference word into the stack to address the interrupt-handling procedure, inserts a literal constant to identify the interrupt condition and two other parameters, and initiates an MCP interrupt-handling procedure. The D registers are updated upon entry into the interrupt-handling procedure, to display all legitimate variables. Upon return from this procedure, the D registers are updated to display variables of the former procedure.

MULTIPLE STACKS AND REENTRANT CODE

The B 6900 stack mechanism provides a facility for handling several active stacks, which are organized in a tree structure. The trunk of this tree structure is a stack containing MCP global quantities.

LEVEL DEFINITION

A program is a set of executable instructions, and a job is a single execution of a program for a particular set of data. As the MCP is requested to run a job, a level-1 branch of the basic stack is created. This level-1 branch contains the descriptors pointing to the executable code and read-only data segments for the program. Emerging from this level-1 branch is a level-2 branch, containing the variables and data for this job. Starting from the job's stack and tracing downward through the tree structure, one finds first the stack containing the variables and data for the job (at level 2), the segment descriptor to be executed (at level 1), and the MCP's stack at the trunk (level 0).

REENTRANCE

A subsequent request to run another execution of an already-running program requires that only a level-2 branch be established. This level-2 stack branch emerges from the level-1 stack of the already-running program. Thus, two jobs which are different executions of the same program have a common node, at level-1, describing the executable code. It is in this way that program code is re-entrant and shared. This results simply from the proper tree-structured organization of the various stacks within the machine. All programs within the system are re-entrant, including all user programs as well as the compilers and the MCP.

JOB-SPLITTING

The B 6900 stack mechanism also provides the facility for a single job to split itself into two independent jobs. A common use of this facility occurs when there is a point in a job where two relatively large independent processes must be performed. This splitting can be used to make full use of a multiprocessor configuration, or to reduce elapsed time by multiprogramming the independent processes.

A split of this type establishes a new limb of the tree-structured stack, with the two independent jobs sharing that part of the stack which was created before the split was requested. The process is recursively defined and can happen repeatedly at any level.

STACK DESCRIPTOR

Stack branches are located by an array of descriptors, the stack vector array (Figure 3-10). There is a data descriptor in this array for every stack branch. This data descriptor, the stack descriptor, describes the length of the memory area assigned to a stack branch and its location in either memory or disk.

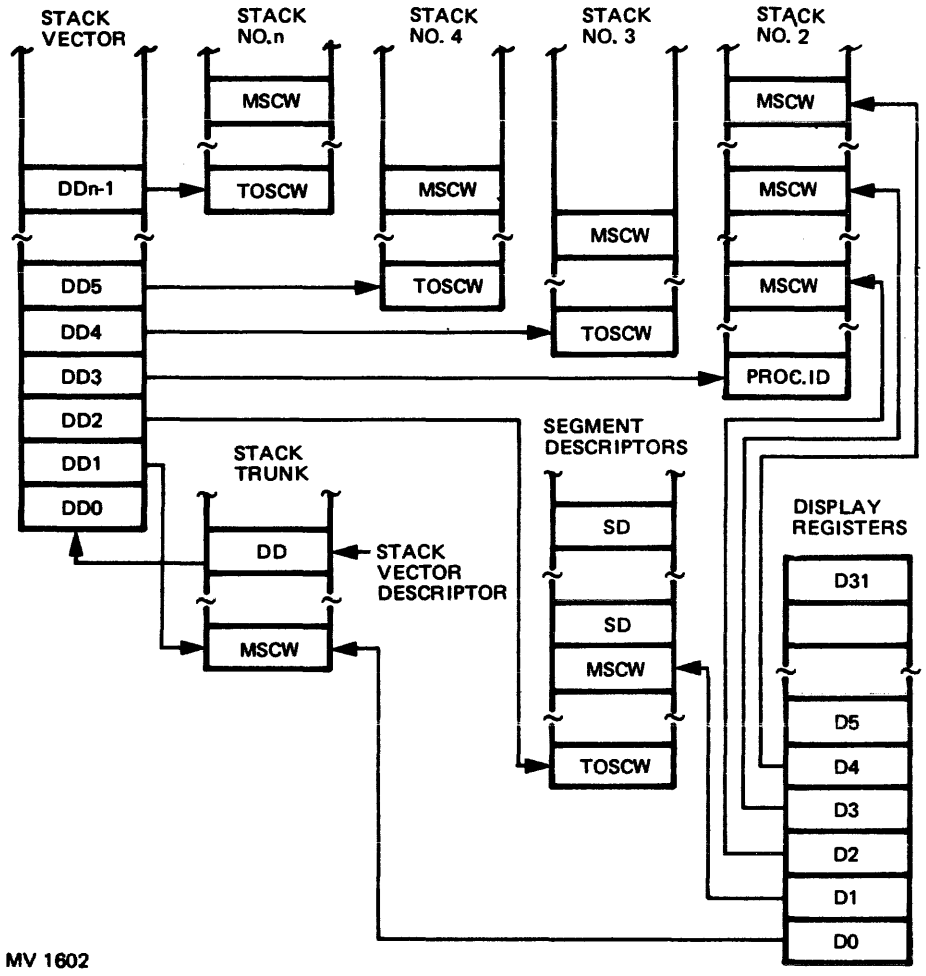


Figure 3-10. Multiple Linked Stacks

A stack number is assigned to each stack branch. The stack number is the index value of the stack descriptor in the stack vector array.

STACK VECTOR DESCRIPTOR

The array size of the stack vector and its location in memory is described by the stack vector descriptor, located in a reserved position of the trunk of the stack (DO+2, see Figure 3-10). All references to stack branches are made through the stack vector descriptor, indexed by the stack number.

PRESENCE BIT INTERRUPT

A presence bit interrupt results when an addressed stack is not present in memory. This presence bit interrupt facility permits stack overlays and recalls under dynamic conditions. Idle or inactive stacks may be moved from main memory to disk as the need arises and, when a stack is subsequently referenced, a presence bit interrupt is generated to cause the MCP to recall the non-present stack from disk.

SECTION 4

SYSTEM DISPLAY AND CONTROL

GENERAL INFORMATION

A B 6900 system provides 2 ways to control system logical circuits. If an MDP is installed in a B 6900 system, there are also 2 ways to display system status. If an MDP cabinet is not installed, there is only 1 way to display system status.

If an MDP cabinet is not installed in a B 6900 system, the Soft Display program must operate to display system status. The Soft Display program methods of operation are defined later in this section in the B 6900 Soft Display paragraphs.

DISPLAY AND CONTROL WITH MDP CABINET INSTALLED

The upper-half of the outer-panels (skins) of the B 6900 MDP cabinet are swing-out covers for the system maintenance display and control panels. The maintenance display and control panels are normally covered and, therefore, not visible. The panel covers are opened to perform maintenance operations, or to exercise control of B 6900 system FIRMWARE programs. Figure 4-1 shows the system control panels and displays when the swing-out covers are opened.

MDP Status Display Panel

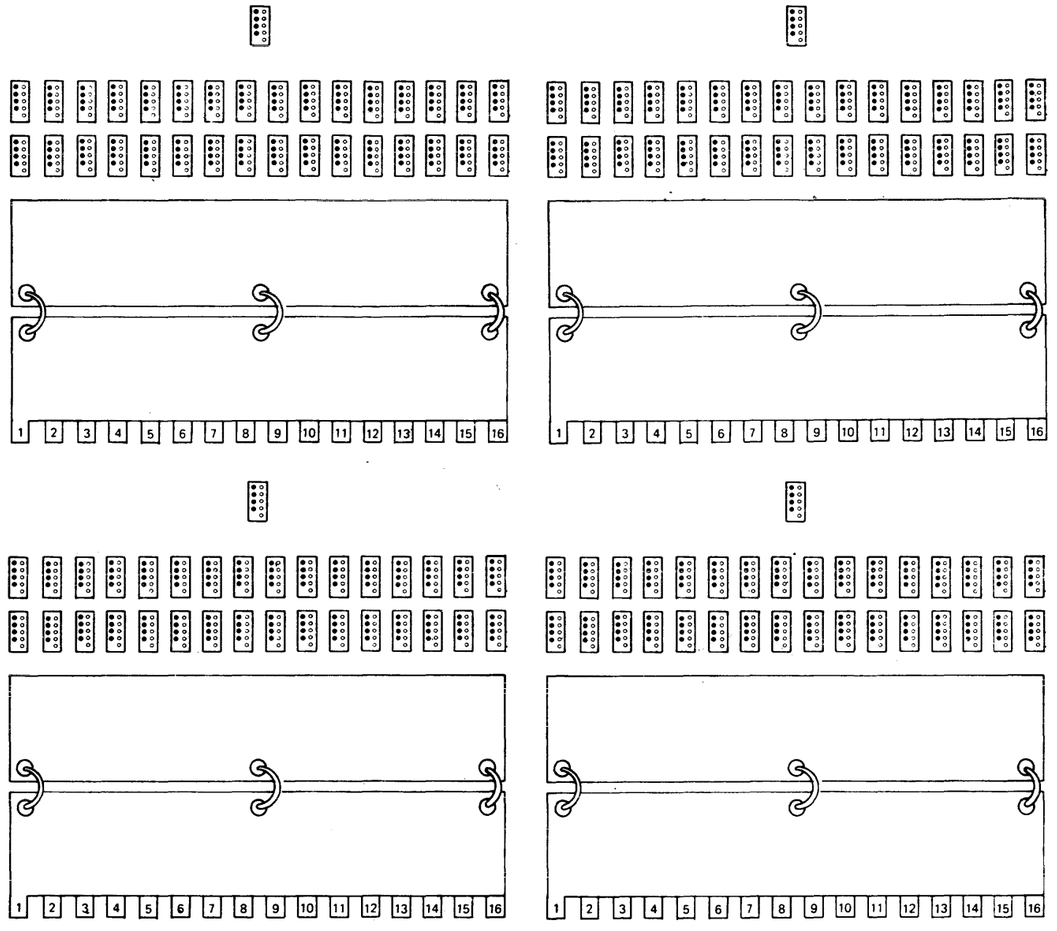
The left-hand side of Figure 4-1 shows the MDP system status display panels. There are 2 adjacent status display panels: the left-most panel is panel 1, and the right-most panel is panel 2. Each display panel contains 2 display registers. Each register has a PAGE selector device located immediately above the display register, and a flip-chart device located immediately below the display register. Figure 4-2 shows one complete display register, and there are four such display registers in the entire system status display.

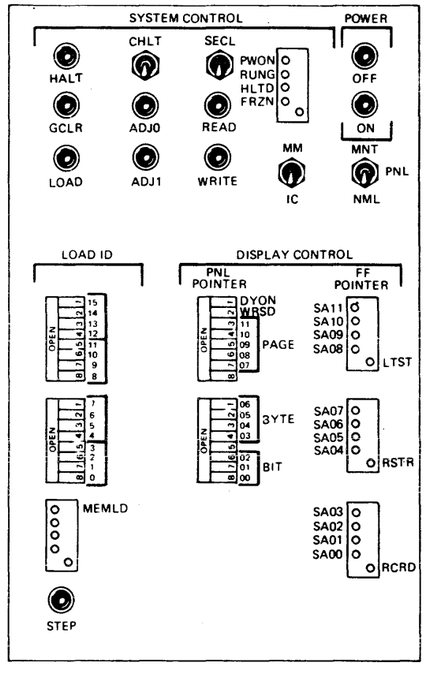
Each display register is capable of displaying the status of 128 logic signals or flip-flops. The PAGE selector is capable of selecting any of 16 different sets of 128 logic signals and/or flip-flops (PAGES) to be displayed in the register. Push-buttons provide a method for SETting or RESETing the state of the flip-flop or logic signals that are currently displayed in the register.

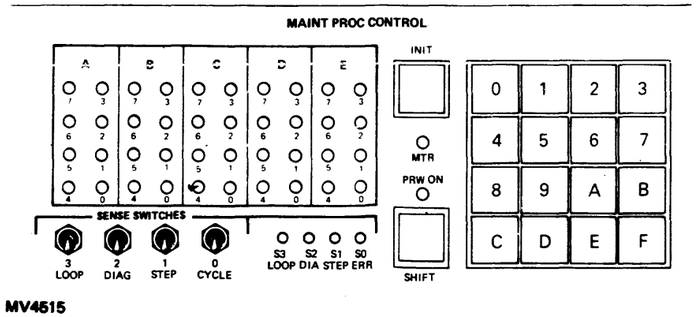
The 2 display registers on a display panel both display the same 16 PAGEs. Thus, a certain PAGE can be selected for display in the upper display register on a panel, and a different PAGE from the same set of PAGEs can be selected for display in the lower display register. It is also possible to display the same PAGE on a display panel in both display registers.

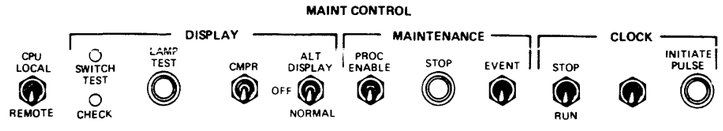
A display register consists of 33 display circuit devices, and is divided into an upper-display and a lower-display. Each display uses 16 display circuit devices. One display circuit device is used as a PAGE-selector for the display register. The upper and lower displays each indicate the status of 64 signals or flip-flops. Selecting the flip-chart that corresponds to the value of the PAGE-selector display device shows which logical status is currently displayed.

A display device (see Figure 4-3) indicates the status of 4 logic signals or flip-flops. Each display device consists of 4 Light Emitting Diodes (LEDs), and 5 push-button switches. One push-button switch is associated with each LED, and is used to change the state of the circuit displayed (SET/RESET the flip-flop displayed by the LED). One register push-button (beneath the LED number 64 pushbutton) is used to select the RESET-function for all LED push-buttons in the register. Another register push-button (beneath the LED number 0 pushbutton) is used for Lamp-Test/Register-Clear operations. When this push-button is depressed, all LED lamps illuminate for a Lamp-Test. When this push-button is released, all circuits displayed by the register are CLEARed (RESET). The bottom push-button on all other register display circuit devices is unused, and has no effect on the status display.









MV4515

Figure 4-1. B 6900 MDP Cabinet Display and Control Panels

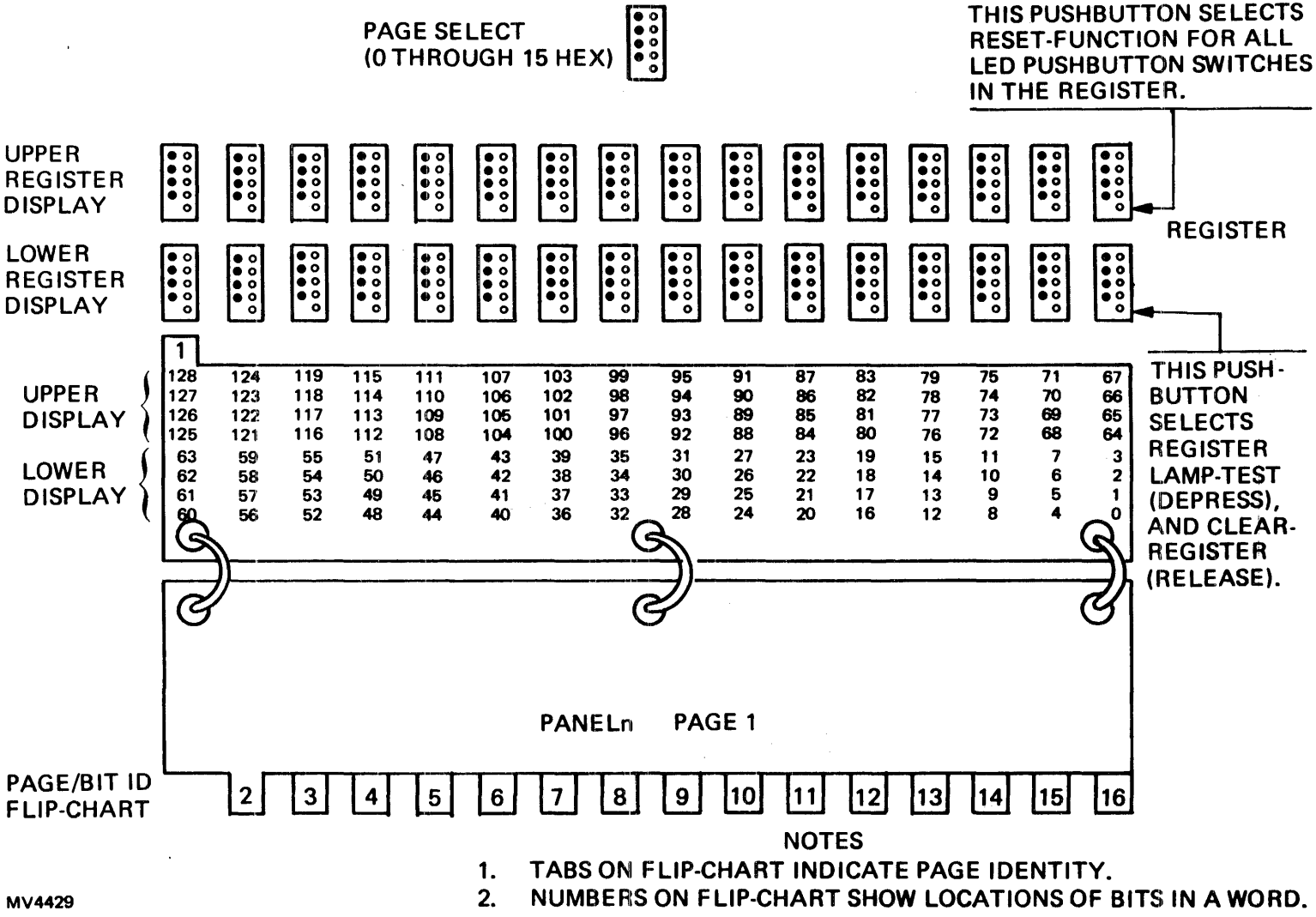


Figure 4-2. B 6900 Status Display Register

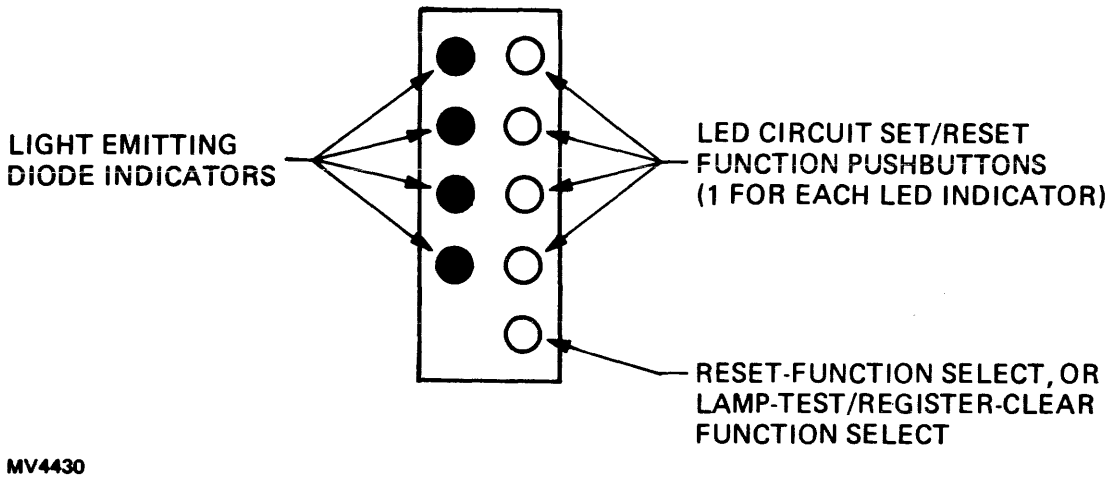


Figure 4-3. LED Indicator – Chip Circuit Display Device

If an LED is illuminated, the corresponding flip-flop is SET (or the logic signal is TRUE). If the LED is extinguished, the corresponding flip-flop is RESET (or the logic signal is FALSE).

Table 4-1 gives the general B 6900 system status displayed on each MDP display panel and display register. The major circuits displayed for each PAGE selection, both the upper and lower displays, are listed.

Table 4-1. B 6900 MDP Cabinet Status Display

Panel	Page	Upper Display	Lower Display
1	0	Top-Of-Stack A Register	Top-Of-Stack B Register
	1	Top-Of-Stack C Register	Program (P) Register
	2	Top-Of-Stack X Register	Top-Of-Stack Y Register
	3	Top-Of-Stack Z Register	Program Look-Ahead L Register
	4	A Register In Octal Notation	X Register In Octal Notation
	5	B Register In Octal Notation	Y Register In Octal Notation
	6	Card-Tester Logic	Card-Tester Logic
	14	Look-Ahead and Address Save Registers	Not Used
	15	System Status Display	Memory Address and Address Adder Sum Reg.
2	0	MLIP Logic	MLIP Logic
	1	MLIP Logic	MLIP Logic
	2	Memory Control Logic	Memory Control Logic
	3	Global Memory Logic	Memory-Tester Logic
	4	Event Logic	Event Logic
	5	Interrupt Controller Logic	Memory and Interrupt Controllers Logic
	6	Not Used	Program Controller Logic
	7	MLIP Logic (Time-Of-Day and Processor-Timer Logic)	Stack and Transfer Controllers Logic
	8	Arithmetic Controller (Family A) Logic	Arithmetic Controller (Families A and E) Logic
	9	Families C and D Logic	Memory Controller Logic
	10	Family U Logic (Families F, G, and H)	Not Used
	11	Families B and E Logic	Not Used

MDP DISPLAY PANEL ONE SIGNALS

Table 4-2 identifies the flip-flops and logic signals displayed on panel number 1 of the B 6900 MDP cabinet. If a signal mnemonic or flip-flop name is not listed in a particular bit of a PAGE, the bit-position is unused. Each PAGE display is provided as an upper-display (bits 64 through 127) and a lower-display (bits 0 through 63).

MDP DISPLAY PANEL TWO SIGNALS

Table 4-3 identifies the flip-flops and logic signals displayed on panel number 2 of the B 6900 MDP cabinet. If a signal mnemonic or flip-flop name is not listed in a particular bit of a PAGE, the bit-position is unused. Each PAGE display is provided as an upper-display (bits 64 through 127) and a lower-display (bits 0 through 63).

MDP DISPLAY SIGNAL DEFINITIONS

Table 4-4 lists in alpha-numeric sequence every B 6900 system status flip-flop and logic signal displayed by the MDP. A cross-reference to the display PANEL, PAGE, and BIT is given; and a definition of the Meaning or usage of each mnemonic signal is included.

The conventions used to define and describe the display logic signals in Table 4-4 are as follows.

[m:n]	This symbol defines a set of mnemonic-terms displayed in a single MDP display register. Mnemonic terms are grouped in sets only when sharing common logical-characteristics or performing a common function. The m character in this symbol identifies the most-significant bit of a set. The n character identifies the number of mnemonics in the set, including the most-significant bit. Signal or flip-flop mnemonics consist of alphanumeric characters. Variation of any alphanumeric character in a mnemonic identifies a unique signal or flip-flop. In a mnemonic set symbol, the character to the left of the colon may be any alphanumeric character, but the length character to the right of the colon is a numeric integer. Thus, signals or flip-flops in a set are represented by any mnemonic character difference, and the number of flip-flops or signals in a set is a numeric quantity.
n, or nn	This symbol is imbedded in or appended to a mnemonic term rather than a bit-designator. Each value of n (or nn) constitutes a separate mnemonic term. Mnemonics are grouped to show common logic functions.
Multiple Line Entries	Some bit-sets are displayed at more than a single MDP Display location. Where this condition exists, multiple line-entries for the same set of bits are given.
Split Line	Some bit-sets are not in consecutive-bit order in an MDP Display register; they are in random-bit order. Multiple line-entries are used to show where all bits in a bit-set are located in the display register.

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 1 of 9)

PAGE Zero Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
AROF				AR47	AR43	AR39	AR35	AR31	AR27	AR23	AR19	AR15	AR11	AR07	AR03
			AR50	AR46	AR42	AR38	AR34	AR30	AR26	AR22	AR18	AR14	AR10	AR06	AR02
			AR49	AR45	AR41	AR37	AR33	AR29	AR25	AR21	AR17	AR13	AR09	AR05	AR01
			AR48	AR44	AR40	AR36	AR32	AR28	AR24	AR20	AR16	AR12	AR08	AR04	AR00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Zero Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
				BR47	BR43	BR39	BR35	BR31	BR27	BR23	BR19	BR15	BR11	BR07	BR03
			BR50	BR46	BR42	BR38	BR34	BR30	BR26	BR22	BR18	BR14	BR10	BR06	BR02
			BR49	BR45	BR41	BR37	BR33	BR29	BR25	BR21	BR17	BR13	BR09	BR05	BR01
BROF			BR48	BR44	BR40	BR36	BR32	BR28	BR24	BR20	BR16	BR12	BR08	BR04	BR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 2 of 9)

PAGE One Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
				CR47	CR43	CR39	CR35	CR31	CR27	CR23	CR19	CR15	CR11	CR07	CR03
			CR50	CR46	CR42	CR38	CR34	CR30	CR26	CR22	CR18	CR14	CR10	CR06	CR02
			CR49	CR45	CR41	CR37	CR33	CR29	CR25	CR21	CR17	CR13	CR09	CR05	CR01
			CR48	CR44	CR40	CR36	CR32	CR28	CR24	CR20	CR16	CR12	CR08	CR04	CR00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE One Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
				PR47	PR43	PR39	PR35	PR31	PR27	PR23	PR19	PR15	PR11	PR07	PR03
	PSR2		PR50	PR46	PR41	PR38	PR34	PR30	PR26	PR22	PR18	PR14	PR10	PR06	PR02
	PSR1		PR49	PR45	PR41	PR37	PR33	PR29	PR25	PR21	PR17	PR13	PR09	PR05	PR01
PROF	PSR0		PR48	PR44	PR40	PR36	PR32	PR28	PR24	PR20	PR16	PR12	PR08	PR04	PR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 3 of 9)

PAGE Two Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
XROF				XR47	XR43	XR39	XR35	XR31	XR27	XR23	XR19	XR15	XR11	XR07	XR03
			XR50	XR46	XR42	XR38	XR34	XR30	XR26	XR22	XR18	XR14	XR10	XR06	XR02
			XR49	XR45	XR41	XR37	XR33	XR29	XR25	XR21	XR17	XR13	XR09	XR05	XR01
			XR48	XR44	XR40	XR36	XR32	XR28	XR24	XR20	XR16	XR12	XR08	XR04	XR00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Two Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
				YR47	YR43	YR39	YR35	YR31	YR27	YR23	YR19	YR15	YR11	YR07	YR03
			YR50	YR46	YR42	YR38	YR34	YR30	YR26	YR22	YR18	YR14	YR10	YR06	YR02
			YR49	YR45	YR41	YR37	YR33	YR29	YR25	YR21	YR17	YR13	YR09	YR05	YR01
			YR48	YR44	YR40	YR36	YR32	YR28	YR24	YR20	YR16	YR12	YR08	YR04	YR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 4 of 9)

PAGE Three Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
				ZR47	ZR43	ZR39	ZR35	ZR31	ZR27	ZR23	ZR19	ZR15	ZR11	ZR07	ZR03
			ZR50	ZR46	ZR42	ZR38	ZR34	ZR30	ZR26	ZR22	ZR18	ZR14	ZR10	ZR06	ZR02
			ZR49	ZR45	ZR41	ZR37	ZR33	ZR29	ZR25	ZR21	ZR17	ZR13	ZR09	ZR05	ZR01
			ZR48	ZR44	ZR40	ZR36	ZR32	ZR28	ZR24	ZR20	ZR16	ZR12	ZR08	ZR04	ZR00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Three Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
				LR47	LR43	LR39	LR35	LR31	LR27	LR23	LR19	LR15	LR11	LR07	LR03
			LR50	LR46	LR42	LR38	LR34	LR30	LR26	LR22	LR18	LR14	LR10	LR06	LR02
			LR49	LR45	LR41	LR37	LR33	LR29	LR25	LR21	LR17	LR13	LR09	LR05	LR01
LROF			LR48	LR44	LR40	LR36	LR32	LR28	LR24	LR20	LR16	LR12	LR08	LR04	LR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 5 of 9)

PAGE Four Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
AROF															
AR47	AR44	AR41	AR38	AR34	AR32	AR29	AR26	AR23	AR20	AR17	AR14	AR11	AR08	AR05	AR02
AR46	AR43	AR40	AR37	AR34	AR31	AR28	AR25	AR22	AR19	AR16	AR13	AR10	AR07	AR04	AR01
AR45	AR42	AR39	AR36	AR33	AR30	AR27	AR24	AR21	AR18	AR15	AR12	AR09	AR06	AR03	AR00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Four Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
XR47	XR44	XR41	XR38	XR35	XR32	XR29	XR26	XR23	XR20	XR17	XR14	XR11	XR08	XR05	XR02
XR46	XR43	XR40	XR37	XR34	XR31	XR28	XR25	XR22	XR19	XR16	XR13	XR10	XR07	XR04	XR01
XR45	XR42	XR39	XR36	XR33	XR30	XR27	XR24	XR21	XR18	XR15	XR12	XR09	XR06	XR03	XR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 6 of 9)

PAGE Five Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
BROF															
BR47	BR44	BR41	BR38	BR35	BR32	BR29	BR26	BR23	BR20	BR17	BR14	BR11	BR08	BR05	BR02
BR46	BR43	BR40	BR37	BR34	BR31	BR28	BR25	BR22	BR19	BR16	BR13	BR10	BR07	BR04	BR01
BR45	BR42	BR39	BR36	BR33	BR30	BR27	BR24	BR21	BR18	BR15	BR12	BR09	BR06	BR03	BR00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Five Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
YR47	YR44	YR41	YR38	YR35	YR32	YR29	YR26	YR23	YR20	YR17	YR14	YR11	YR08	YR05	YR02
YR46	YR43	YR40	YR37	YR34	YR31	YR28	YR25	YR22	YR19	YR16	YR13	YR10	YR07	YR04	YR01
YR45	YR42	YR39	YR36	YR33	YR30	YR27	YR24	YR21	YR18	YR15	YR12	YR09	YR06	YR03	YR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 7 of 9)

PAGE Six Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
P26F	P22F	P18F	P14F	P10F	P06F	P02F	P38C	P34C	P30C	P26C	P22C	P18C	P14C		P06C
P25F	P21F	P17F	P13F	P09F	P05F	P01F	P37C	P33C	P29C	P25C	P21C		P13C	P09C	P05C
P24F	P20F	P16F	P12F	P08F	P04F		P36C	P32C	P28C		P20C	P16C	P12C	P08C	P04C
P23F	P19F	P15F	P11F	P07F	P03F		P35C		P27C	P23C	P19C	P15C	P11C	P07C	
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Six Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
P52F	P48F	P44F	P40F	P36F	P32F	P28F	P38S	P34S	P30S	P26S	P22S	P18S	P14S	P10S	P06S
P51F	P47F	P43F	P39F	P35F	P31F	P27F	P37S	P33S	P29S	P25S	P21S	P17S	P13S	P09S	P05S
P50F	P46F	P42F	P38F	P34F	P30F		P36S	P32S	P28S	P24S	P20S	P16S	P12S	P08S	P04S
P49F	P45F	P41F	P37F	P33F	P29F	P39S	P35S	P31S	P27S	P23S	P19S	P15S	P11S	P07S	P02S
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 8 of 9)

PAGE Fourteen Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
LA19	LA15	LA11	LA07	LA03							ADD19	ADD15	ADD11	ADD07	ADD03
LA18	LA14	LA10	LA06	LA02							ADD18	ADD14	ADD10	ADD06	ADD02
LA17	LA13	LA09	LA05	LA01							ADD17	ADD13	ADD09	ADD05	ADD01
LA16	LA12	LA08	LA04	LA00							ADD16	ADD12	ADD08	ADD04	ADD00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Fourteen Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-2. B 6900 MDP Panel One Signal Display (Sheet 9 of 9)

PAGE Fifteen Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
								AROF	PSR2	STRA	STRB	STRC	STRD	STRE	SHLT
								BROF	PSR1	STRF	STRG	STRH	STRJ	STRK	ICFF
								PROF	PSR0	VARF	EDIT	TEEF	VECF	IIHF	LROF
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Fifteen Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
MA19	MA15	MA11	MA07	MA03						MSM19	MSM15	MSM11	MSM07	MSM03	
MA18	MA14	MA10	MA06	MA02						MSM18	MSM14	MSM10	MSM06	MSM02	
MA17	MA13	MA09	MA05	MA01						MSM17	MSM13	MSM09	MSM05	MSM01	
MA16	MA12	MA08	MA04	MA00						MEM16	MEM12	MEM08	MEM04	MEM00	
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 1 of 12)

PAGE Zero Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
HASL			MLIE	FLG3	MSWR	MRA3	SPTS1	MSP7	MSP3	RIEN3	R119	R115	R111	R107	R103
STCH		STMX	R3NG	FLG2		MRA2	SPTS0	MSP6	MSP2	RIEN2	R118	R114	R110	R106	R102
SCWE		FLGE1	MAHF	FLG1	MRAE	MRA1	MSP9	MSP5	MSP1	RIEN1	R117	R113	R109	R105	R101
SCCE		FLGE0	FLG4	FLG0	MRA4	MRA0	MSP8	MSP4	MSP0	RIEN0	R116	R112	R108	R104	R100
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Zero Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
			PENF	BRQF	PSC3	FST3	DST3	AGNT	STEN	R2EN3	R219	R215	R211	R207	R203
	GSP2	PAS2	PAD2	PSCE1	PSC2	FST2	DST2	CSEL	OUTF	R2EN2	R218	R214	R210	R206	R202
	GSP1	PAS1	PAD1	PSCE0	PSC1	FST1	DST1	INRQ	ASEL	R2EN1	R217	R213	R209	R205	R201
	GSP0	PAS0	PAD0	PSC4	PSC0	FST0	DST0	EMRQ	TERM	R2EN0	R216	R212	R208	R204	R200
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 2 of 12)

PAGE One Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
SPAR3	SPAR7		LPMX3	STS3	LP15	LP11	LP07	LP03	MINH	R3MX3	R319	R315	R311	R307	R303
BRST	SPAR6	HDPH/	LPMX2	STS2	LP14	LP10	LP06	LP02	BI2	R3MX2	R318	R314	R310	R306	R302
SPAR1	SPAR5	SPAR9	LPMX1	STS1	LP13	LP09	LP05	LP01	BI1	R3MX1	R317	R313	R309	R305	R301
SPAR2	SPAR4	SPAR8	LPMX0	STS0	LP12	LP08	LP04	LP00	BI0	R3MX0	R316	R312	R308	R304	R300
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE One Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
MMEN	MMBF	MM7F	MM3F						MDS3	MFS3	MOVR	MR15	MR11	MR07	MR03
MMPD	MMAF	MM6F	MM2F						MDS2	MFS2	MERQ	MR14	MR10	MR06	MR02
DBIT	MM9F	MM5F	MM1F						MDS1	MFS1	MIRQ	MR13	MR09	MR05	MR01
MAIM	MM8F	MM4F	MM0F					MRDY	MDS0	MFS0	MR16	MR12	MR08	MR04	MR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 3 of 12)

PAGE Two Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
RQTB	RQT7	RQT3	RQRB	RQR7	RQR3	CSC4	CAPFE	CAPFD	CAPFC	CAPFB	CAPFA	ABRF		HAR3	LACF
RQTA	RQT6	RQT2	RQRA	RQR6	RQR2	CSC3	SPM2	SRL2	ATEF	MAOF	WAIT	CARQ		HAR2	SNAP
RQT9	RQT5	RQT1	RQR9	RQR5	RQR1	CSC2	SPM1	SRL1	TRYF	RDFF	LOG2	MI51R		HAR1	IHCP
RQT8	RQT4	RQT0	RQR8	RQR4	RQR0	CSC1	SPM0	SRL0	CHGO	CINF	LOG1	MI48	PTGO	HAR0	IVAF
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Two Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
IMCF3	CIOF3	IMCF2	CIOF2	IMCF1	CIOF1	IMCF0	CIOF0	WSTF3	MRSF3	ICW3	MSW3	GS2F	ICNF	GT2F	GOAF
PS2F3	CAOF3	PS2F2	CAOF2	PS2F1	CAOF1	PS2F0	CAOF0	WSTF2	MRSF2	ICW2	MSW2	GS1F	GRDF	GT1F	GOBF
PS1F3	WCCF3	PS1F2	WCCF2	PS1F1	WCCF1	PS1F0	WCCF0	WSTF1	MRSF1	ICW1	MSW1	GS0F	GABF	GT0F	GAOF
PS0F3	PEDF3	PS0F2	PEDF2	PS0F1	PEDF1	PS0F0	PEDF0	WSTF0	MRSF0	ICW0	MSW0		CRFF		EGTM
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 4 of 12)

PAGE Three Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
GAINA	IGUEB	IGUEA	GCLER	ABRG	GSTRT	GUEX	GINV	GCHB	GSCX					SPMA3	SPMB3
GAINB	IGWEB	IGWEA	GBC2	IGXF	GLOAD	GWEX	GAEX	GCHA	GBS2					SPMA2	SPMB2
GEINA	IGREB	IGREA	GBC1	GAINT	GHALT	GREX	GAOX	TOUT	GBS1					SPMA1	SPMB1
GEINB		GARCS	GBC0	GEINT	GAOR	IHGT	GABX	TRIG	GBS0					SPMA0	SPMB0
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Three Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
BYR19	BYR15	BYR11	BYR07	BYR03	MTST	JV1	CMPE	CBPW	CB4W	CBPR	CB4R	IT10	IT06	IT02	ECSF
BYR18	BYR14	BYR10	BYR06	BYR02	TV2	JV0	TABT	WEFW	CF3W	WEFR	CB3R	IT09	IT05	IT01	EXTI
BYR17	BYR13	BYR09	BYR05	BYR01	TV1	OMCK	ALTWC	CB6W	CB2W	CB6R	CB2R	IT08	IT04	IT00	INTV
BYR16	BYR12	BYR08	BYR04	BYR00	TV0	CMTR	MI51W	CB5W	CB1W	CB5R	CB1R	IT07	IT03		INTE
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 5 of 12)

PAGE Four Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
EV21	EJCMP	EV13	EV09	EV05	EV01	ECT7	ECT3	ICOR	EJC11	EJC07	EJC03	HLTD	EST7	EOP3	EVCT
EV20	EV16	EV12	EV08	EV04	CCSF	ECT6	ECT2	MPBI	EJC10	EJC06	EJC02	ILHD	EST6	EOP2	ETED
EV19	EV15	EV11	EV07	EV03	MEVF	ECT5	ECT1	MIAI	EJC09	EJC05	EJC01	LODS	EST5	EOP1	EEDT
EV18	EV14	EV10	EV06	EV02	HOEF	ECT4	ECT0	ESTP	EJC08	EJC04	EJC00	LAVF	EST4	EOP0	EVAR
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Four Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
	WMMF	AMMF						JCS11	JCS07	JCS03			SRS3	OSR3	VCTS
PLK1	RMMF	AIMF						JCS10	JCS06	JCS02			SRS2	OSR2	TEDS
PLK0	WIMF	MEXI		HALT				JCS09	JCS05	JCS01			SRS1	OSR1	EDTS
PSOP	RIMF			ARPT				JCS08	JCS04	JCS00			SRS0	OSR0	VARS
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 6 of 12)

PAGE Five Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
RDCBA	RDREA	CKB4A		CKB4B					STAR	SCNR	SCAN	STB2	ADDR	EREN7	EREN3
ADREA	ADSEA	CKB3A		CKB3B				GNTR	INPW	STAP	RUNI	STB1	BURE	EREN6	EREN2
STOF	CKB6A	CKB2A	CKB6B	CKB2B			CAM3	INAGA	STUF	LOPE		STB0	RCPE	EREN5	EREN1
STIS	CKB5A	CKB1A	CKB5B	CKB1B				RDMEA	INALA			CMPR	PCPE	EREN4	EREN0
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Five Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
LRAP	LRIG	IML2	ABR1	INCT	INF+1	INFF	MPXI	DR31	DR27	DR23	DR19	DR15	DR11	DR07	DR03
LRIL	LRGN	IML1	ABEI	MEWT	SEC+2	ALSB	MPXB	DR30	DR26	DR22	DR18	DR14	DR10	DR06	DR02
LRAR	LAER	IML0	ILDM	BDST	SEC+1		MPXG	DR29	DR25	DR21	DR17	DR13	DR09	DR05	DR01
LRDM	OPTF		SEIN	ABIT	AYER	RTRY		DR28	DR24	DR20	DR16	DR12	DR08	DR04	DR00
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 7 of 12)

PAGE Six Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Six Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
CPA8	ICRF	CPIR1	CTIR	SSR2	WPIR	QP8F	QP4F	STMC	EDDT						
CPA4	ICCF	CPIR0	CSR2	SSR1	SECF	QP7F	QP3F	JP02							
CPA2	FWFF	WPTF	CSR1	SSR0		QP6F	QP2F	JP01							
CPA1	PRVA	WBCF	CSR0		VSJK	QP5F	QP1F	JP00							
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 8 of 12)

PAGE Seven Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
CP23	CP19	CP15	CP11	CP07	CP03	TD35	TD31	TD27	TD23	TD19	TD15	TD11	TD07	TD03	TOD3
CP22	CP18	CP14	CP10	CP06	CP02	TD34	TD30	TD26	TD22	TD18	TD14	TD10	TD06	TD02	TOD2
CP21	CP17	CP13	CP09	CP05	CP01	TD33	TD29	TD25	TD21	TD17	TD13	TD09	TD05	TD01	TOD1
CP20	CP16	CP12	CP08	CP04	CP00	TD32	TD28	TD24	TD20	TD16	TD12	TD08	TD04	TD00	TOD0
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Seven Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
BZ62	BZ61	YZ62	YZ61		TOA3		TOM3		DIS3		JS4F	SOIF			
AZ63	CZ63	XZ63	ZZ63		TOA2		TOM2		DIS2		JS3F	QS3F			
AZ62	CZ62	XZ62	ZZ62	TOA5	TOA1	TOM5	TOM1	DIS5	DIS1		JS2F	QS2F			
AZ61	CZ61	XZ61	ZZ61	TOA4	TOA0	TOM4	TOM0	DIS4	DIS0		JS1F	QS1F			
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 9 of 12)

PAGE Eight Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
TA3F	SA3F	JA7F	JA3F	EXAI	QA7F	QA3F	SM03	SM04			NLZ3	HR15	HR11	HR07	HR03
TA2F	SA2F	JA6F	JA2F	KA2F	QA6F	QA2F	SM02				NLZ2	HR14	HR10	HR06	HR02
TA1F	SA1F	JA5F	JA1F	KA1F	QA5F	QA1F	SM01	PSCF			NLZ1	HR13	HR09	HR05	HR01
TA0F	SA0F	JA4F	JA0F	KA0F	QA4F	QA0F	SM00	CMPF		NLZF	NLZ0	HR12	HR08	HR04	HR00
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Eight Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
BETB	NZTB	HRTB1	EXSB	B1TB	B8TB	ADSB	SPC1	BX02	AX02	YR-3	SC3F	SCEF	ICR7	ICR3	B46D
YETB	ZDTB	HRTB2	ECRI	Y1TB	Y8TB	CCNS	DPC1	BX01	AX01	YR-2	SC2F	ICRE	ICR6	ICR2	A46D
AETA	NZTA	HRTA1	A1TA	A2TA	A4TA	CCR3	C175	BX00	AX00	YR-1	SC1F	BXSE	ICR5	ICR1	BDPD
XETA	ZDTA	HRTA2	X1TA	X2TA	X4TA	CCL3	DPOV	YX00	XX00	XR--1	SC0F	DISX	ICR4	ICR0	ADPD
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 10 of 12)

PAGE Nine Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
TC3F	JC7F	JC3F	QC8F	QC4F	CRNCF		ACL7	ACL3		TD3F	JD7F	JD3F		QD8F	QD4F
TC2F	JC6F	JC2F	QC7F	QC3F	SASG		ACL6	ACL2		TD2F	JD6F	JD2F	QDBF	QD7F	QD3F
TC1F	JC5F	JC1F	QC6F	QC2F	QCZ2	ACM5	ACL5	ACL1		TD1F	JD5F	JD1F	QDAF	QD6F	QD2F
TC0F	JC4F	JC0F	QC5F	QC1F	QCZ1	ACM4	ACL4	ACL0		TD0F	JD4F	JD0F	QD9F	QD5F	QD1F
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Nine Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
ERZ9	ERZ8	DRSS	Z812	DFSX	ACT8	BRS7	BRS3	IRS7	IRS3	COUT	Z6T8	LL03			
CRF03	DRF13	DRF4	Z811	MSOR2	CPTR	BRS6	BRS2	IRS6	IRS2	CZIN	Z6L8	LL02			
CRF02	DRF12	DRF3	Z810	MSOR1	ECMF	BRS5	BRS1	IRS5	IRS1	SUBT	Z6T9	LL01			
CRF01	DRF11	DRF2	Z809	MSOR0	CRIC	BRS$	BRS0	IRS4	IRS0	Z6L9	LL04	LL00			
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 11 of 12)

PAGE Ten Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
TU8F	EEND	RETF		JU3F	SSZ2	SI08	DI08	QU4F	DGSF	SOPF			TFFF	EQVF	QUDF
TU4F	FINI	RTNF	JU6F	JU2F	SSZ1	SI04	DI04	QU3F	LHFF	UPDF			TFOF		QUCF
TU2F	EXSF	NVLF	JU5F	JU1F	DSZ2	SI02	DI02	QU2F	RPZF	SRRF			OFFF		QUBF
TU1F	EXPF	MPOP	JU4F	JU0F	DSZ1	SI01	DI01	QU1F	XROF	DPRF			FLTF	EXTF	QUAF
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Ten Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-3. B 6900 MDP Panel Two Signal Display (Sheet 12 of 12)

PAGE Eleven Upper-display

127	123	119	115	111	107	103	99	95	91	87	83	79	75	71	67
TE3F	JBCF	JE3F	QE3F	SMVF	LC2F	SF3F	MP35		TB3F	JB3F	QB4F				
TE2F	JE6F	JE2F	QE2F	MPYF	LC1F	SF2F	DBZF		TB2F	JB2F	QB3F				
TE1F	JE5F	JE1F	QE1F	SUBF	LC0F	SF1F	FNWF		TB1F	JB1F	QB2F				
TE0F	JE4F	JE0F	QE0F	LC3F	DPFF	SF0F	QE4F		TB0F	JB0F	QB1F				
124	120	116	112	108	104	100	96	92	88	84	80	76	72	68	64

PAGE Eleven Lower-display

63	59	55	51	47	43	39	35	31	27	23	19	15	11	07	03
60	56	52	48	44	40	36	32	28	24	20	16	12	08	04	00

Table 4-4. B 6900 Display Signals (Sheet 1 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
ABEI	2	5	50	An Abort Interrupt Controller Logic Signal
ABIT	2	5	44	The Abort Interrupt Logic Signal
ABRF	2	2	79	The Abort Memory Cycle Flip-flop
ABR1	2	5	51	The Abort Clock Save Logic Signal
ACLn [7:8]	2	9	[99:8]	Least-significant 8-bits of Address Couple For NAMC Operators
ACMn [5:2]	2	9	[101:2]	Most-significant 2-bits of Address Couple For NAMC Operators
ACT8	2	9	43	Address Couple To Z8-Bus Logic Signal
ADDnn [19:20]	1	14	[83:20]	The Memory Controller Save Address Register
ADDR	2	5	75	Address-Adder Residue Error Flip-flop
ADPD	2	8	0	Double Precision Operand In A Register Logic Signal
ADREA	2	5	126	The Memory Controller Address Retry Logic Signal
ADSB	2	8	39	The Mantissa Adder Subtract Mode Signal
ADSEA	2	5	122	The Memory Controller Read Data Single-Bit Signal
AETA	2	8	61	The A Register Exponent To A Side Of Exponent Adder Signal
AIMF	2	4	54	The Access IC Memory Flip-flop
ALSB	2	5	38	The Allow Family Strobe Logic Signal
ALTWC	2	3	33	The Memory Tester Alternate Worst Case Signal
AMMF	2	4	55	The Access Main Memory Flip-flop
AROF	1	0	27	The A Register Is Occupied Flip-flop
	1	4	127	
	1	14	94	
ARPT	2	4	44	The Anti-Repeat Flip-flop

Table 4-4. B 6900 Display Signals (Sheet 2 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
ARnn [50:51]	1	0	[114:51]	The Top-Of-Stack A Register
	1	4	[126:51]	
ASEL	2	0	25	MLIP Address Select Flip-flop
AXnn [02:3]	2	8	[27:3]	The A Mantissa 1-Octade Extension Register
AYER	2	5	40	The EVENT Logic Any Memory Error Signal
AZ6n [3:3]	2	7	[62:3]	The Bit-field Transfer (From A-Register) To Z6-Bus Signals AZ61 Transfers Bits [50:11] AZ62 Transfers Bits [39:20] AZ63 Transfers Bits [19:20]
A1TA	2	8	49	The A Register Mantissa To A Input Of Mantissa Adder Logic Signal
A2TA	2	8	45	2 Times A-Register Mantissa To A Side Of Mantissa Adder
A4TA	2	8	41	4 Times A-Register Mantissa To A Side Of Mantissa Adder
A46D	2	8	2	The A-Register Sign-Bit Change Delay Logic Signal
BDPD	2	8	1	The B-Register Contains A Double Precision Operand Logic Signal
BDST	2	5	45	The Maintenance Processor Test Logic Signal
BETB	2	8	63	The B-Register Exponent To B Side Of Exponent Adder Logic Signal
BIn [2:3]	2	1	[90:3]	The MLIP Byte Index Register
BROF	1	0	60	
	1	5	127	The Top-Of-Stack B-Register Occupied Flip-flop
	1	15	93	
BRQF	2	0	47	The MLIP BURST Request Flip-flop
BRST	2	2	126	The MLIP BURST Flip-flop

Table 4-4. B 6900 Display Signals (Sheet 3 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location Panel	Page	Bits	Meaning or Usage
BRSn [7:8]	2	9	[39:8]	The Memory Address Base Register Select Logic BRS0 Selects PBR Register BRS1 Selects SBR BRS2 Selects DBR BRS3 Selects TBR BRS4 Selects S Register BRS5 Selects SNR BRS6 Selects PDR BRS7 Selects TEMP
BRnn [50:51]	1 1	0 5	[50:51] [126:51]	The Top-Of-Stack B Register
BURE	2	5	74	The Bus Residue Error Flip-flop
BXSE	2	8	13	The B Side Of Mantissa Adder Logic Signal
BXnn [02:3]	2	8	[31:3]	The B Mantissa 1-Octade Extension Register
BYRnn [19:20]	2	3	[63:20]	The Memory Tester BYPASS Register Logic
BZ61	2	7	59	The Transfer From B-Register To Z6-Bus (Bits [50:11]) Logic Signal
BZ62	2	7	63	The Transfer From B-Register To Z6-Bus (Bits [39:20]) Logic Signal
B1TB	2	8	47	The B-Register Mantissa To B Input Of Mantissa Adder Logic Signal
B46D	2	8	3	The B-Register Change Sign-Bit Delay Logic Signal
B8T8	2	8	43	8 Times The B-Register Mantissa To B Input Of The Mantissa Adder Logic Signal
CAM3	2	5	98	Memory Controller Error Bit
CAOF0	2	2	34	Memory Controller Priority Occupying Port Number Zero Logic
CAOF1	2	2	43	Memory Controller Priority Occupying Port Number One Logic
CAOF2	2	2	50	Memory Controller Priority Occupying Port Number Two Logic

Table 4-4. B 6900 Display Signals (Sheet 4 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
CAOF3	2	2	58	Memory Controller Priority Occupying Port Number Three Logic
CBnR [P:8]	2	3	[23:8]	Memory-Tester READ Data Check-Bit Field
CBnW [P:8]	2	3	[31:8]	Memory-Tester WRITE Data Check-Bit Field
CKBnA [6:6]	2	5	[121:6]	Memory Controller Data Check-Bit Register
CKBnB [6:6]	2	5	[113:6]	Memory Controller Data Check-Bit Register
CMPE	2	3	35	The Memory Tester Compare Error Flip-flop
CMPF	2	8	92	The Arithmetic Controller Relational Operator Compare Flip-flop
CMPR	2	5	76	The Compare Residue Flip-flop
COUT	2	9	23	The Address Adder Cary-out Flip-flop
CPAn [8:4]	2	6	[63:4]	The CPU Clock-Counter Low-Order Flip-flops
CPIRn [1:2]	2	6	[55:2]	The PIR Word-Boundary Crossed Register
CPTR	2	3	36	Comparator Mode Enable Refresh Signal (Factory Use Only)
CPnn [23:24]	2	7	[127:24]	The High-order 24-Bits Of The MLIP CPU-Timer Register
CRFOn [3:3]	2	9	[62:3]	IC Memory Address Display Register Group-Card Select Logic CRFO1 Selects Group A Card CRFO2 Selects Group B Card CRFO3 Selects Group C Card
CRIC	2	9	40	Clear IC Memory Address Register Flip-flop
CRNCF	2	9	107	Interrupt Controller PIR And PBR Register Values Not Consistent Flip-flop
CRnn [50:51]	1	1	[114:51]	The Top-Of-Stack C Register
CSC [4:4]	2	2	[103:4]	Memory Controller Logic Requestor Sequence Counter
CSEL	2	0	30	The MLIP Channel Select Flip-flop
CSRn [2:3]	2	6	[50:3]	The Count Syllable Register

Table 4-4. B 6900 Display Signals (Sheet 5 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
CTIR	2	6	51	TIR Register Word-Boundary Crossed Flip-flop
CZIN	2	9	22	**Address Adder Carry-in Signal**
CZ6n [3:3]	2	7	[58:3]	Transfer Gate Signals To The Z6-Bus (From the C-Register) CZ61 Transfers [50:11] CZ62 Transfers [39:20] CZ63 Transfers [19:20]
C175	2	8	33	The Carry-in Signal To Bit-75 Of The Mantissa Adder
DBZF	2	11	98	The Destination Bit Zero Flip-flop
DFSX	2	9	47	Bit-8 Index Portion Of Address Couple Value
DGSF	2	10	91	A Logical Flip-flop Used By Family U For String Operators
DISX	2	8	12	Disable Extensions Flip-flop (Force To Zero)
DISn [5:6]	2	7	[29:6]	Transfer Controller Displacement Register
DInn [8:4]	2	10	[99:4]	The Family U Destination Index Byte Register
DPCI	2	8	34	The Double Precision Carry-in/Borrow-bit To The Mantissa Adder Logic
DPFF	2	11	104	The Double Precision Scale Right Multiplier Flip-flop
DPOV	2	8	32	The Double Precision Gating Override Logic Signal
DPRF	2	10	84	The Family U Destination Read Only Control Flip-flop
DRFnn [13:3]	2	9	[58:3]	The Display Address Card Group Select Logic Signals DRF11 Selects Group A Cards DRF12 Selects Group B Cards DRF13 Selects Group C Cards
DRFn [4:3]	2	9	[54:3]	The Display Address Register Select Signals
DRnn [31:32]	2	5	[31:32]	The Memory Address Display Register (D-Register) Select Logic Levels
DSTn [3:4]	2	0	[35:4]	The MLIP Delayed Status Register
DSZn [2:2]	2	10	[105:2]	The Family U Destination Byte Size Register

Table 4-4. B 6900 Display Signals (Sheet 6 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
ECMF	2	9	41	The Enable Comparator Mode Refresh Flip-flop (For Factory Use Only)
ECSF	2	3	3	The EVENT Logic Freeze Parameters Flip-flop
ECTn [7:8]	2	4	[103:8]	The EVENT Logic Counter
EDDT	2	6	27	The EVENT Detected Flip-flop
EDIT	1	15	80	The EDIT Mode Flip-flop
EDTS	2	4	2	The EDIT Mode Save Flip-flop
EEDT	2	4	65	The EVENT Logic EDIT Mode Signal
EEND	2	10	123	The End Of Enter EDIT Mode Cycle Control Flip-flop
EGMT	2	2	0	Global Memory Control Logic Flip-flop
EJCMP	2	4	123	The EVENT Logic Micro-module J-Count Select Signal
EJCnn [11:12]	2	4	[91:12]	The EVENT Logic Micro-module J-Count Register
EMRQ	2	0	29	The MLIP I/O Emergency BURST Request Flip-flop
EOPn [3:4]	2	4	[71:4]	The EVENT Logic Operator-Code Register
EQVF	2	10	71	The Family U Equivalent Control (Sum Equal To Zero) Flip-flop
ERENn [7:8]	2	5	]71:8]	The PROM-Card-Error (CPU Card Location) Register
ERZ8	2	9	59	The Residue-Error On Z8-Bus Signal
ERZ9	2	9	63	The Residue-Error On Z9-Bus Signal
ESTn [7:4]	2	4	[75:4]	The EVENT Logic Strobe Register
ETED	2	4	66	The EVENT Logic Table EDIT Mode Logic Signal
EVAR	2	4	64	The EVENT Logic VARIANT-Mode Logic Signal
EVCT	2	4	67	The EVENT Logic VECTOR-Mode Logic Signal
EVnn [21:20]	2	4	[127:4] [122:16]	The EVENT Logic EVENT Register

Table 4-4. B 6900 Display Signals (Sheet 7 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
EXAI	2	8	111	The Exponent Add Initiate Flip-flop
EXSB	2	8	51	The Exponent Adder Subtract Function Flip-flop
EXTF	2	10	68	The Family U External Sign Bit Flag Control Flip-flop
EXTI	2	3	2	The Interrupt Controller External Interrupt Signal
FINI	2	10	122	The Family U End-Of-Edit Cycle Control Flip-flop
FLGEn [1:2]	2	0	[118:2]	The MLIP Flag-Enable Signal Register
FLGn	2	0	[111:4]	The MLIP Flag Register
FLTF	2	10	72	The Family U Float Control Flip-flop
FNWF	2	11	97	The Final-Word Flip-flop
FSTn [3:4]	2	0	[39:4]	The MLIP Fast Status Signal Register
FWFF	2	6	57	The First-Word Fetch Flip-flop
GABF	2	2	9	A Global Memory (MC III) Control Flip-flop
GABX	2	3	96	The Global Memory Access Begin Logic Signal
GAEX	2	3	98	The Global Memory Address Error Signal
GAINA	2	3	127	The Global Alarm Interrupt Flip-flop
GAINB	2	3	126	Not Used In B 6900
GAINT	2	3	109	The Global Alarm Interrupt Signal
GAOF	2	2	3	A Global Memory (MC III) Control Signal
GAOR	2	3	104	The Global Access Obtained Return Flip-flop
GARCS	2	3	116	The Global Memory All Rows And Columns Clear Signal
GBCn [2:3]	2	3	[114:3]	The Global Sequence Control Register
GBSn [2:3]	2	3	[90:3]	The Global Clear Sequence Control Register
GCHA	2	3	94	The Global Memory Cycle Control Signal
GCHB	2	3	95	Not Used In B 6900
GCLER	2	3	115	The Global Clear Signal

Table 4-4. B 6900 Display Signals (Sheet 8 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
GEINA	2	3	125	The Global External Interrupt Flip-flop
GEINB	2	3	124	Not Used In B 6900
GEINT	2	3	108	The Global External Interrupt Signal
GHALT	2	3	105	The Global HALT Function Signal
GINV	2	3	99	The Global Invalid Request Signal
GLOAD	2	3	106	The Global LOAD Function Signal
GNTR	2	5	94	The Global Memory Not Ready Flip-flop
GOAF	2	2	3	A Global Memory (MC III) Control Flip-flop
GOBF	2	2	2	A Global Memory (MC III) Control Flip-flop
GRDF	2	2	10	A Global Memory (MC III) Control Signal
GREX	2	3	101	The Global Memory Read-error Signal
GSCX	2	3	91	The Global Scan-Control Signal
GSPn [2:3]	2	0	[58:3]	The MLIP Global-Priority Save Register
GSnF [2:3]	2	2	[15:3]	A Global Memory (MC III) Control Register
GSTRT	2	3	107	The Global START Function Signal
GTnF [2:3]	2	2	[7:3]	A Global Memory (MC III) Control Register
GUEX	2	3	103	The Global Memory Data Uncorrectable-Error Signal
GWEX	2	3	102	The Global Memory Write-Error Signal
HALT	2	4	45	The CPU HALT Function Logic Signal
HARn [3:4]	2	2	[71:4]	The Memory Controller Hold Address For Return Signal Register
HASL	2	0	127	The MLIP RAM Memory Is Initialized Signal Flip-flop
HDPH/	2	1	118	The MLIP Micro-Module Not Held Logic Signal

Table 4-4. B 6900 Display Signals (Sheet 9 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
HLTD	2	4	79	The CPU Is Halted Flip-flop
HOEF	2	4	104	The EVENT Logic HALT-ON-EVENT Signal
HRTAn [2:2]	2	8	[53:2]	The Exponent Adder A-Side Input Holding Register
HRTBn [2:2]	2	8	[54:2]	The Exponent Adder B-Side Input Holding Register
HRnn [15:16]	2	8	[79:16]	The Arithmetic Controller Holding Register
ICCF	2	6	58	The Program Controller Increment CPIR And CTIR Normal Control Flip-flop
ICFF	1	15	65	The Interrupt Controller Running Flip-flop
ICOR	2	4	95	The EVENT Logic Inhibit Memory-Correction-Cycle Signal
ICRE	2	8	14	The Input-Convert Register Enable Flip-flop
ICRF	2	6	59	The Program Controller Increment CPIR And CTIR (Remember) Control Flip-flop
ICRn [7:8]	2	8	[11:8]	The Input-Convert-Operation Register
ICWn [3:4]	2	2	[23:4]	The Memory Controller IC Memory Refresh Function Delay (For MSU Signal) Register
IGHT	2	3	100	The Global Memory Inhibit Global-Timer Signal
IGREA	2	3	117	The Global Memory Read-Error Flip-flop
IGREB	2	3	121	Not Used In B 6900
IGUEA	2	3	119	The Global Memory Uncorrectable-Error Interrupt Signal
IGUEB	2	3	123	Not Used In B 6900
IGXF	2	3	110	The Inhibit Global Crosspoint Flip-flop
IHCP	2	2	65	The Inhibit Setting CHGO And PTGO Flip-flop
IIHF	1	15	68	The Interrupt Controller Inhibit Interrupts (Control State) Flip-flop
ILDM	2	5	49	The Interrupt Load Micro Program Logic Signal

Table 4-4. B 6900 Display Signals (Sheet 10 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
ILHD	2	4	78	The Inhibit Look-Ahead Logic Flip-flop
IMCn [3:4]	2	2	39	The Initiate Memory Cycle Control Signals To Memory Ports 0, 1, 2, And 3
	2	2	47	
	2	2	52	
	2	2	63	
IMLn [2:3]	2	5	[55:3]	The Consecutive Interrupt Counter For Detecting SUPERHALT Conditions
INAGA	2	5	93	Global Memory Invalid Address-Bit Error Signal
INALA	2	5	88	Local Memory Invalid Address Signal
INCF	2	2	11	A Global Memory (MC III) Control Logic Signal
INCT	2	5	47	The Inconsistent P3 Parameter Signal
INFF	2	5	39	The Inhibit Fetch Flip-flop
INF+1	2	5	43	The Inhibit Fetch Flip-flop Delayed 1 Clock-pulse Logic Signal
INPW	2	5	90	The Invalid Program-Word Flip-flop
INRQ	2	0	29	The MLIP Interrupt Request Flip-flop
INTE	2	3	0	The Interval-Timer Enable Signal
INTV	2	3	1	The Interval-Timer Error Flip-flop
IRSn [7:8]	2	9	[31:8]	The Memory Address Read Index Register Select Signals IRS0 Selects PIR IRS1 Selects SIR IRS2 Selects DIR IRS3 Selects TIR IRS4 Selects LOSR IRS5 Selects BOSR IRS6 Selects F IRS7 Selects BUF
ITnn [10:11]	2	3	[15:11]	The Interval-Timer Register
IVAF	2	2	64	The Invalid Memory Address Flip-flop

Table 4-4. B 6900 Display Signals (Sheet 11 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
JAnF [7:8]	2	8	[119:8]	The Family A Sequence Count (J-Count) Register
JBCF	2	11	123	The Family E J-Count Bus Control Flip-flop
JBnF [3:4]	2	11	[87:4]	The Family B Sequence Count (J-Count) Register
JCSn [11:12]	2	4	[31:12]	The EVENT Logic J-Count Save Register
JCnF [7:8]	2	9	[123:8]	The Family C Sequence Count (J-Count) Register
JDnF [7:8]	2	9	[83:8]	The Family D Sequence Count (J-Count) Register
JEnF [6:7]	2	11	[122:7]	The Family E Sequence Count (J-Count) Register
JPnF [02:3]	2	6	[30:3]	The Program Controller Sequence Count Register
JSnF [4:4]	2	7	[19:4]	The Stack Controller Sequence Count Register
JUnF [6:7]	2	10	[114:7]	The Family U Sequence Count (J-Count) Register
JVn [1:2]	2	3	[39:2]	The Memory-Tester Logic Sequence Counter
KAnF [2:3]	2	8	[110:3]	The Family A K-Counter Logic
LACF	2	2	67	This Flip-flop Not Used On The B 6900 System
LAER	2	5	57	The Look-Ahead Logic Memory Error Signal
LAVF	2	4	76	The Look-Ahead Valid Flip-flop
LAnn [19:20]	1	14	[127:20]	The Look-Ahead Logic Memory Address Register
LCnF [3:4]	2	11	[108:4]	The Loop-Count Register
LHFF	2	10	90	A Family U Logical Flip-flop
LLnn [04:5]	2	9	[16:5]	The Lexicographical Level Register
LODS	2	4	77	The Load (Source) Select Flip-flop
LOGn [2:2]	2	2	[81:2]	The Memory Controller Error Control Register
LOPE	2	5	85	The Loop-Timer Error Flip-flop
LPMXn [3:4]	2	1	[115:4]	The MLIP Longitudinal Parity Register To MX-Bus Gating Signal Register

Table 4-4. B 6900 Display Signals (Sheet 12 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
LPnn [7:8]	2	1	[99:8]	The MLIP Longitudinal Parity Register
LRAP	2	5	63	The Look-Ahead Logic Address Parity Signal
LRAR	2	5	61	The Look-Ahead Logic Address Residue Signal
LRDM	2	5	60	The Look-Ahead Logic Read Data Timer Signal
LRGN	2	5	58	The Global Memory Not Ready For Look-Ahead Logic
LRIG	2	5	59	The Global Memory Invalid Address For Look-Ahead Signal
LRIL	2	5	62	The Local Memory Invalid Address For Look-Ahead Signal
LROF	1	3	60	The Look-Ahead Register (L Register) Occupied Flip-flop
LRnn [50:51]	1	3	[50:51]	The Look-Ahead Register (The Next Sequential Program Code Word in The Current Segment)
MAIM	2	1	60	The Micro-module Address To Input Multiplexor Flip-flop
MAOF	2	2	86	The Memory Access Obtained Flip-flop
MAnn [19:20]	1	15	[63:20]	The Memory Address Register
MDSn [3:4]	2	1	[27:4]	The MLIP Maintenance Display Status Register
MERQ	2	1	18	The MLIP Emergency Request (POLL-REQUEST From MLI) Signal
MEVF	2	4	105	The EVENT Logic Multiple EVENT Flip-flop
MEWT	2	5	46	The Families Memory Cycle Wait Time Signal
MEXI	2	4	53	The Mask External Interrupt Signal
MFSn [3:4]	2	1	[23:4]	The MLIP Maintenance Fast Status Register
MIAI	2	4	93	The EVENT Logic Mask Invalid Address Interrupt Signal
MINH	2	1	91	The MLIP Memory Inhibit Logic Signal
MIRQ	2	1	17	The MLIP Maintenance Interrupt Request Flip-flop
MI48	2	2	76	The Memory Controller Memory Protect Bit

Table 4-4. B 6900 Display Signals (Sheet 13 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
MI51R	2	2	77	The Memory Controller Read Data Word-Parity Bit (Odd)
MI51W	2	3	32	The Memory Controller Write Data Word-Parity Bit (Odd)
MMEN	2	1	63	The Micro-Module Enable Signal
MMPD	1	1	62	The Micro-Module Parity Disable For First Clock-pulse Signal
MMnF [B:12]	2	1	[59:12]	The Micro-Module Address (MLIP Entry-Vector) Signals
MOVR	2	1	19	The MLIP Maintenance Override Signal
MPBI	2	4	94	The EVENT Logic Mask Presence Bit Interrupt Signal
MPOP	2	10	116	The Micro Program Control Flip-flop
MPXB	2	5	34	The MLIP BURST Logic Signal
MPXG	2	5	33	The MLIP Access Granted (To CPU Memory Bus For A BURST Memory Cycle) Signal
MPXI	2	5	35	The MLIP Initiate BURST Request Signal (Remembered)
MPYF	2	11	110	The Scale-Right Multiply (Times Ten) Raised To The Value Of The Scale-Factor Enable Signal
MP35	2	11	99	The Scale-Right Multiplied By Third/Fifth Octade Signal
MRAE	2	0	105	The MLIP Memory Address Register Enable Flip-flop
MRAn [4:5]	2	0	[104:5]	The MLIP Memory Register Address
MRDY	2	1	28	The MLIP Maintenance Ready Flip-flop
MRSn [3:4]	2	2	[27:4]	The Local Memory Refresh Control Signal Register (For Ports 0, 1, 2, And 3)
MRnn [16:17]	2	1	[16:17]	The MLIP Maintenance Data Register
MSORn [2:3]	2	9	[46:3]	The Address Adder Sum Of Residue Bits MOSOR0 Is Residue Bit-1 MOSOR1 Is Residue Bit-2
MSMnn [19:20]	1	15	[23:20]	The Address Adder Sum Register
MSPnn [9:10]	2	0	[97:10]	The MLIP Micro-Stack Pointer Register

Table 4-4. B 6900 Display Signals (Sheet 14 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
MSWR	2	0	107	The MLIP Micro-Stack Write (From MLIP Register Number 1) Control Signal
MSWn [3:4]	2	2	[19:4]	The Memory Controller Select Write Signal Register For Ports 0, 1, 2, And 3
MTST	2	3	43	The Memory-Tester Test-Mode Control Flip-flop
NLZF	2	8	84	The Number Of Leading Zeroes Register Control Flip-flop
NLZn [3:4]	2	8	[83:4]	The Number Of Leading Zeroes Register
NVLF	2	10	117	The Family U Not Valid Control Flip-flop
NZTA	2	8	57	The Add NLZ (Number Of Leading Zeroes) To A-Input Of Exponent Adder Control Signal
NZTB	2	8	59	The Add NLZ (Number Of Leading Zeroes) To B-Side Of Exponent Adder Control Signal
OFFF	2	10	73	The Family U Overflow Control Flip-flop
ONCK	2	3	37	The CPU One-Clock Control Signal
OPTF	2	5	56	The Optional Adapter Test Flip-flop (Maintenance-Mode)
OSRn [3:4]	2	4	[7:4]	The EVENT Logic OP-CODE Save Register
OUTF	2	0	26	The MLIP Output Flip-flop
PADn [2:3]	2	0	[50:3]	The MLIP Port Address Register
PASn [2:3]	2	0	[54:3]	The MLIP Port Address Save Register
PCPE	2	5	72	The CPU PROM-Card Parity Error (Card-Location) Register
PEDFn [3:4]	2	2	32	The Memory Controller Parity Error Disable Control Signals Register (TO Local Memory Ports 0, 1, 2, And 3)
	2	2	40	
	2	2	51	
	2	2	56	
PENF	2	0	51	The MLIP Port Enable Flip-flop
PLKn [1:2]	2	4	[62:2]	The Clock-Control Phase-Lock Register

Table 4-4. B 6900 Display Signals (Sheet 15 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location Panel	Page	Bits	Meaning or Usage
PROF	1	1	60	The Program Controller Program-Register (P-Register)
	1	15	92	Occupied Flip-flop
PRVA	2	6	55	The PROF And VARF Flip-flops Are Valid Logic Signals
PRnn [50:51]	1	1	[50:51]	The Program Controller Program-Word (P-Register)
PSCF	2	8	93	The Pseudo-Call On Family A Flip-flop
PSCn [3:4]	2	0	[43:4]	The MLIP Priority-Sequencer Count Register
PSRn [2:3]	1	1	[58:3]	The Program Controller Program Syllable Register
	1	15	[90:3]	
PS0Fn [2:3]	2	2	[38:3]	The Memory Controller Port Sequence Count Register For Local Memory Port Zero
PS1Fn [2:3]	2	2	[46:3]	The Memory Controller Port Sequence Count Register For Local Memory Port One
PS2Fn [2:3]	2	2	[54:3]	The Memory Controller Port Sequence Count Register For Local Memory Port Two
PS3Fn [2:3]	2	2	[62:3]	The Memory Controller Port Sequence Count Register For Local Memory Port Three
PTGO	2	2	72	The Memory Controller Port Go (To Complete A Memory Cycle) Signal
PnnC [38:7]	1	6	[99:7]	
[30:6]	1	6	[91:6]	
[23:6]	1	6	[84:6]	The Card-Tester Pin Clear Register
[16:6]	1	6	[77:6]	
[09:6]	1	6	[70:6]	
PnnF [26:26]	1	6	[127:26]	The Card-Tester Pin Register
[52:26]	1	6	[63:26]	
PnnS [39:36]	1	6	[36:36]	The Card-Tester Pin-SET Register
P02S	1	6	0	The Card-Tester Pin-SET Signal For Pin Number 2
QAnF [7:8]	2	8	[107:8]	Family A Logical Control Flip-flops
QBnF [4:3]	2	11	[83:4]	Family B Logical Control Flip-flops

Table 4-4. B 6900 Display Signals (Sheet 16 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location Panel	Page	Bits	Meaning or Usage
QCZn [2:2]	2	9	[106:2]	The Family C Size-Save Register QCZ1 Saves Size-1 QCZ2 Saves Size-2
QCnF [8:8]	2	9	[115:8]	The Family C Logical Control Flip-flops
QDnF [B:12]	2	9	[74:12]	The Family D Logical Control Flip-flops
QEnF [4:5]	2 2	11 11	[115:4] [96:1]	The Family E Logical Control Flip-flops
QPnF [8:8]	2	6	[39:8]	The Program Controller Logical Control Flip-flops
QSnF [3:3]	2	7	[14:3]	The Stack Controller Logical Control Flip-flops
QUAF	2	10	64	The Family U Invalid Operation Control (QF01) Flip-flop
QUBF	2	10	65	The Family U Presence-Bit Control (QF02) Flip-flop
QUCF	2	10	66	The Family U Memory-Protect Control (QF03) Flip-flop
QUDF	2	10	67	The Family U Segmented-Array Control (QF04) Flip-flop
QUnF [4:4]	2	10	[95:4]	The Family U Logical Control Flip-flops
RCPE	2	5	73	The RAM-Card Parity-Error Flip-flop
RDCBA	2	5	127	The Memory Controller Read-Data Check-Bit Signal
RDFF	2	2	85	The Memory Controller Read Phase Flip-flop
RDMEA	2	5	92	The Memory Controller Multiple-Bit Error Signal
RDREA	2	5	123	The Memory Controller Address-Retry Signal
RETF	2	10	119	The Family U Return To Using Operation Control Flip-flop
RIENn [3:4]	2	0	[87:4]	The MLIP Enable Bit Signals For Register Number 1
RIMF	2	4	56	The EVENT Logic Read IC Memory Flip-flop
RMMF	2	4	58	The EVENT Logic Read Main Memory Flip-flop
RPZF	2	10	89	A Family U Logical Flip-flop

Table 4-4. B 6900 Display Signals (Sheet 17 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
RQRn [B:12]	2	2	[115:12]	The Memory Controller Request Address Register
RQTn [B:12]	2	2	[127:12]	The Memory Controller Request Address Trap Register
RTNF	2	10	118	The EVENT Logic Re-entrant From Interrupt Controller Flip-flop
RTRY	2	5	36	The Memory Controller Retry Flip-flop
RUNI	2	5	82	The Running Indicator Signal
R1nn [19:20]	2	0	[83:20]	The MLIP R-1 Register
R2En [3:4]	2	0	[23:4]	The MLIP Register-2 Bit-Enable Signal Register
R2nn [19:20]	2	0	[19:20]	The MLIP R-2 Register
R3MXn [3:4]	2	1	[87:4]	The MLIP R-3 Register Gated To The MX-Bus Control Signal Register
R3nn [19:20]	2	1	[83:20]	The MLIP R-3 Register
SASG	2	9	106	The Save Segmented-Bit Flip-flop
SAnF [3:4]	2	8	[123:4]	The Family A T-Register Save Register
SCAN	2	5	83	The Global Memory SCAN Command Signal
SCCE	2	0	124	The MLIP Status-Change Command Enable Signal
SCEF	2	8	15	The Scale Count-Enable Flip-flop
SCNR	2	5	87	Not Used In B 6900
SCWE	2	0	125	The MLIP Status-Change Write Enable Signal
SCnF [3:4]	2	8	[19:4]	The Scale Count Register
SDIS	2	5	124	The Interrupt Controller Syllable Dependent Interrupt Signal
SECF	2	6	42	The Syllable Execute Complete Level Save Flip-flop
SEC+1	2	5	41	The Syllable Execute Complete Level Delayed 1 Clock-pulse Signal

Table 4-4. B 6900 Display Signals (Sheet 18 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
SEC+2	2	5	42	The Syllable Execute Complete Level Delayed 2 Clock-Pulses Signal
SEIN	2	5	48	The Syllable Execute Complete Interrupts Enable Signal
SFnF [3:4]	2	11	[103:4]	The Scale-Factor Register
SHLT	1	15	67	The SUPERHALT Flip-flop
SInn [08:4]	2	10	[103:4]	The Family U Source-Byte Index Register
SMVF	2	11	111	The Enable Scale-Right PROM (generates TOA, TOM, And DIS Values)
SMnn [04:5]	2	8	[99:5]	The Steering And Mask Register (Generates Family A TOA, TOM, And DIS Values)
SNAP	2	2	66	The SNAP Mode Flip-flop (Used During Maintenance Testing)
SOIF	2	7	15	The Stack Overflow Interrupt Flip-flop
SOPF	2	10	87	The Family U Source-Pointer Equals An Operand Control Flip-flop
SPARn [9:9]	2	2	127	
	2	2	[126:6]	MLIP Spare Flip-flops
	2	2	[117:2]	
SPCI	2	8	35	The Single Precision Carry-in (Or Borrow) Signal To The Mantissa Adder Logic
SPMAn [3:4]	2	3	[71:4]	The Single-Pulse Mode A For Memory Port n Register SPMA0 Selects Port 0 SPMA1 Selects Port 1 SPMA2 Selects Port 2 SPMA3 Selects Port 3
SPMBn [3:4]	2	3	[67:4]	The Single-Pulse Mode B For Memory Port n Register (see SPMAn Signals For Port IDs)
SPMn [2:3]	2	2	[98:3]	Spare Flip-flops (Not Used)
SRLn [2:3]	2	2	[94:3]	The Sum-Of-Residue Of The Address In The LAR Register
SRRF	2	10	85	The Family U Source Pointer Read Only Flip-flop

Table 4-4. B 6900 Display Signals (Sheet 19 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
SRSn [3:4]	2	4	[11:4]	The EVENT Logic Strobe Save Register
SSRn [2:3]	2	6	[47:3]	The Syllable Save Register (Of The Syllable That Initiated A Table-EDIT-Mode Operation)
SSZFn [2:2]	2	10	[107:2]	The Family U Source Size Register
STAP	2	5	86	The Memory Controller Address Parity Error Signal
STAR	2	5	91	The Memory Controller Store Address Residue Signal
STBn [2:3]	2	5	[79:3]	The Stack Register (Indicates Where A Read-Data Word Was Placed In The Stack)
STCH	2	0	126	The MLIP Status-Change Signal
STEN	2	0	27	The MLIP Strobe-Enable Flip-flop
STMC	2	6	31	The Program Controller Is Cycling Signal (A SECL Signal Occurred, Or A Program-Branch To A Non-Present Program-Word Is To Be Executed)
STMX	2	0	118	The MLIP Status (MINH And BIn Gated To The MX-Bus) Signal
STOF	2	5	125	The Stack Overflow Signal
STRA	1	15	87	The Family A Strobe Flip-flop
STRB	1	15	82	The Family B Strobe Flip-flop
STRC	1	15	78	The Family C Strobe Flip-flop
STRD	1	15	74	The Family D Strobe Flip-flop
STRE	1	15	70	The Family E Strobe Flip-flop
STRF	1	15	85	A Family U (Family F) Strobe Flip-flop
STRG	1	15	81	A Family U (Family G) Strobe Flip-flop
STRH	1	15	77	A Family U (Family H) Strobe Flip-flop
STRJ	1	15	73	A Family C (Family J) Strobe Flip-flop
STRK	1	15	69	A Family C (Family K) Strobe Flip-flop

Table 4-4. B 6900 Display Signals (Sheet 20 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
STSn [3:4]	2	1	[111:4]	The MLIP Status Save Register
STUF	2	5	89	The Stack Underflow Flip-flop
SUBF	2	11	109	The Family E Last-Octade (Of Shift-Register Multiplication) Was A Subtract Logic Signal
SUBT	2	9	21	The Address Adder Subtract Function Flip-flop
TABT	2	3	34	The Memory-Tester Test All Bits Signal
TAnF [3:4]	2	8	[127:4]	The Family A T-Register
TBnF [3:4]	2	11	[91:4]	The Family B T-Register
TCnF [3:4]	2	9	[127:4]	The Family C T-Register
TDnF [3:4]	2	9	[87:4]	The Family D T-Register
TEDS	2	4	2	The Table-Save Flip-flop
TEEF	1	15	76	The Table-EDIT Mode Flip-flop
TERM	2	0	24	The MLIP Terminate Flip-flop
TEnF [3:4]	2	11	[127:4]	The Family E T-Register
TFFF	2	10	75	The Family U String-Operation True/False Comparison Flip-flop
TFOF	2	10	74	The Family U True/False Flip-flop (TFFF) Occupied Flip-flop
TOAn [5:6]	2	7	[45:6]	The Transfer Controller Top-Of-Aperture Register
TODn [3:4]	2	7	[67:4]	The MLIP Time-Of-Day Register (The 4 Low-Order Bits)
TOMn [5:6]	2	7	[37:6]	The Transfer Controller Top-Of-Mask Register
TOUT	2	3	93	The Global Memory Timeout Signal
TRIG	2	3	92	The Global Memory Trigger (Start) Global Timer Signal
TRYF	2	2	89	The Memory Controller Address Retry Flip-flop

Table 4-4. B 6900 Display Signals (Sheet 21 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
TUnF [8:4]	2	10	[127:4]	The Family U T-Register
TVn [2:3]	2	3	[42:3]	The Memory-Tester Test Vector Register
UPDF	2	10	86	The Family U Update Control Flip-flop
VARF	1	15	84	The VARIANT Mode Flip-flop
VARS	2	4	0	The Variant Save-Bit Flip-flop
VCTS	2	4	3	The Vector Save-Bit Flip-flop
VECF	1	15	72	The VECTOR Mode Flip-flop
VSJK	2	6	40	The Vector Strobe Save/Store Flip-flop
WAIT	2	2	82	The CPU General-Purpose Delay Flip-flop
WBCF	2	6	52	The Program Controller Word Boundary Crossed Flip-flop
WCCFn [3:4]	2	2	33	The Memory Controller Clear/Write Function Control Signals To Ports 0, 1, 2, And 3
	2	2	41	
	2	2	49	
	2	2	57	
WIMF	2	4	57	The Write IC Memory Flip-flop
WPIR	2	6	43	The Write PIR (On Return From Table Mode) Flip-flop
WPTF	2	6	53	The Write PIR Or TIR Flip-flop
WSTn [3:4]	2	2	[31:4]	The Memory Controller Write Control Signals To Memory Ports 0, 1, 2, And 3
XETA	2	8	60	The X-Register Exponent To The A-Side Input Of The Exponent Adder Gating Signal
XROF	2	10	88	The X-Register Occupied Flip-flop
XR-1	2	8	20	The X Register Low-order Bit (Input Conversion)
XRnn [50:51]	1	2	[114:51]	The Top-Of-Stack X Register
	1	4	[62:51]	
XX00	2	8	24	The X-Register Exponent 1-Bit Extension Signal

Table 4-4. B 6900 Display Signals (Sheet 22 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
XZ6n [3:3]	2	7	[54:3]	The X-Register Gating Signals To The Z6-Bus XZ61 Gates [50:11] XZ62 Gates [39:20] XZ63 Gates [19:20]
X1TA	2	8	48	The X-Register Mantissa To A-Side Input Of Mantissa Adder Gating Signal
X2TA	2	8	44	2-Times The X-Register Mantissa To A-Side Input Of Mantissa Adder Gating Signal
X4TA	2	8	40	4-Times The X-Register Mantissa To A-Side Input Of Mantissa Adder Gating Signal
YETB	2	8	62	The Y-Register Exponent Gated To The B-Side Input Of The Exponent Adder Gating Signal
YR-1 [3:3]	2	8	[23:3]	The Y-Register Mantissa 1-Octade Extension Register
YRnn [50:51]	1 1	2 5	[50:51] [62:51]	The Top-Of-Stack Y Register
YX00	2	8	28	The Y Exponent 1-Bit Extension Signal
YZ6n [2:2]	2 2	7 7	51 55	The Y-Register Mantissa Gated To The Z6-Bus Signals YZ61 Gates Bits [50:11] YZ62 Gates Bits [19:20]
Y1TB	2	8	46	The Y-Register Mantissa To The B-Side Input Of The Mantissa Adder Gating Signal
Y8TB	2	8	42	8-Times The Y-Register Mantissa To The B-Side Input Of The Mantissa Adder Gating Signal
ZDTA	2	8	56	Literal 1310 Gated To The A-Side Input Of The Mantissa Adder Signal
ZDTB	2	8	58	Literal 1310 Gated To The B-Side Input Of The Mantissa Adder Signal
ZRnn [50:51]	1	3	[114:51]	The Top-Of-Stack Z Register
ZZ6n [3:3]	2	7	[50:3]	The Z-Register Mantissa Gated To The Z6-Bus Gating Signals ZZ61 Gates Bits [50:11] ZZ62 Gates Bits [39:20] ZZ63 Gates Bits [19:20]

Table 4-4. B 6900 Display Signals (Sheet 23 of 23)

Flip-Flop or Signal Mnemonic	MDP Display Location			Meaning or Usage
	Panel	Page	Bits	
Z6L8	2	9	18	A 20-Bit Index Field On The Z6-Bus Gated To The Z8-Bus (Index Field From The Top-Of-Stack Gated To Address Adder)
Z6L9	2	9	20	A 20-Bit Index Field On The Z6-Bus Gated To The Z9-Bus (Index Field From The Top-Of-Stack Gated To Address Adder)
Z6T8	2	9	19	A 20-Bit Base Address On The Z6-Bus Gated To The Z8-Bus (A 20-Bit Base Address From A Top-Of-Stack Register Gated To The Address Adder)
Z6T9	2	9	17	A 20-Bit Base Address On The Z6-Bus Gated To The Z9-Bus (A 20-Bit Base Address From A Top-Of-Stack Register Gated To The Address Adder)
Z8nn [12:4]	2	9	[51:4]	1-Bit Of The INDEX Portion Of An Address Couple Z809 Is Index Bit-9 Z810 Is Index Bit-10 Z811 Is Index Bit-11 Z812 Is Index Bit-12

B 6900 SYSTEM CONTROL PANEL

Figures 4-1 and 4-4 show the location of the B 6900 System Control Panel. This control panel (refer to Figure 4-5) contains switches, indicators, and controls used for the entire B 6900 system. The following paragraphs define the function of each control or indicator on the panel, and briefly describe the system actions performed as a result of operating a control.

HALT

This pushbutton switch is used to cause the CPU data processor to halt, if it is executing machine language operator codes. The data processor completes the currently executing operator code and stops before the next operator code in sequence is executed. The direct result of depressing and releasing this pushbutton is to SET the HALT flip-flop. When the HALT flip-flop is SET, the SECL signal ending the current operator SETs the HALTED flip-flop, which prevents the Program Controller from setting the Strobe flip-flop for the next operator in sequence. Nothing happens if the data processor is already halted and the HALT pushbutton is depressed and released.

CHLT

Conditional Halt (CHLT) is selected when the CHLT toggle-switch is in the CHLT (UP) position, and is disabled when the CHLT toggle-switch is in the DOWN position.

CHLT functions in a manner similar to the HALT pushbutton. However, it stops the system only when a HALT (DF) operator code is executed. If the CHLT switch is in the CHLT (UP) position and a DF operator code executes, the HALT flip-flop SETs

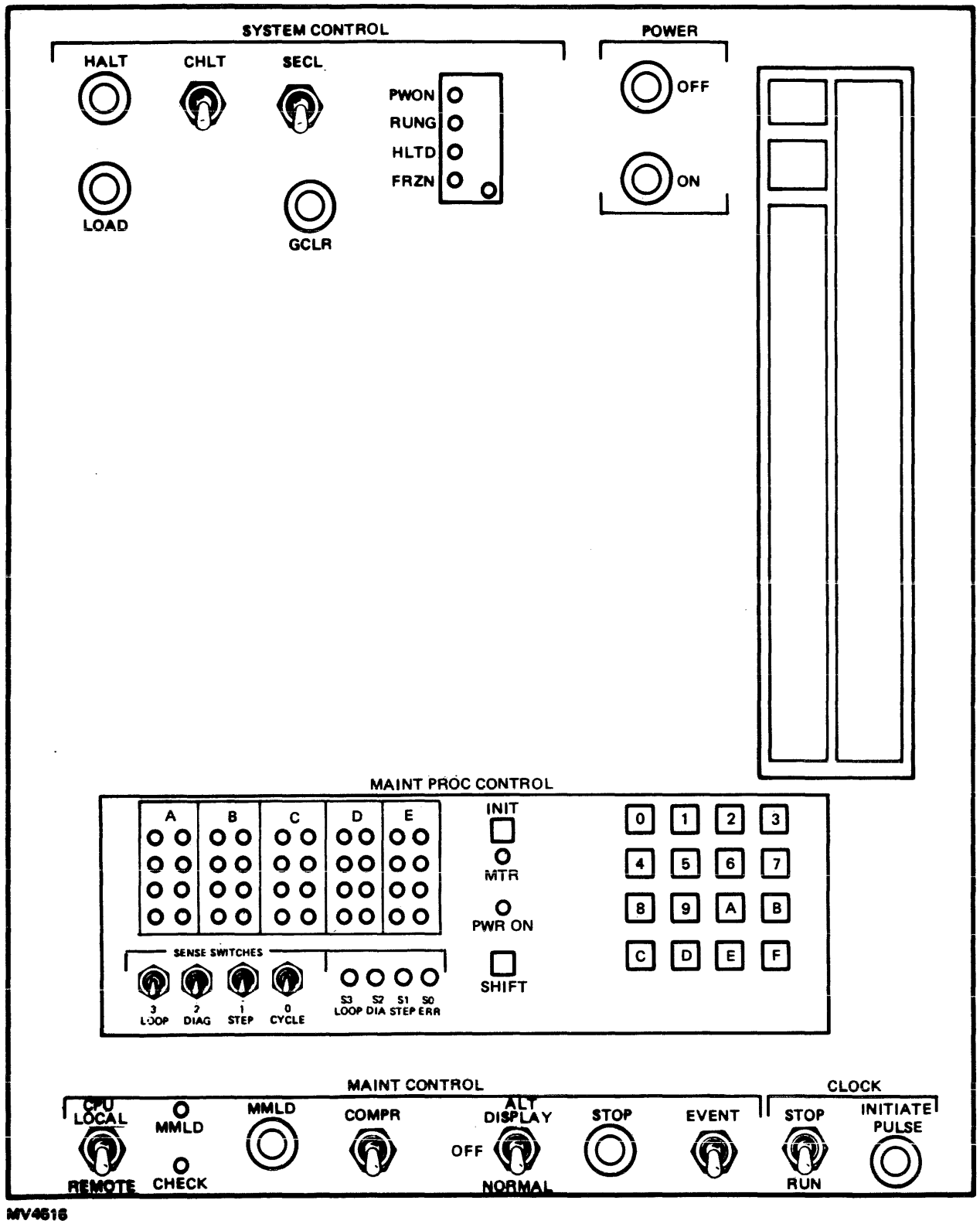
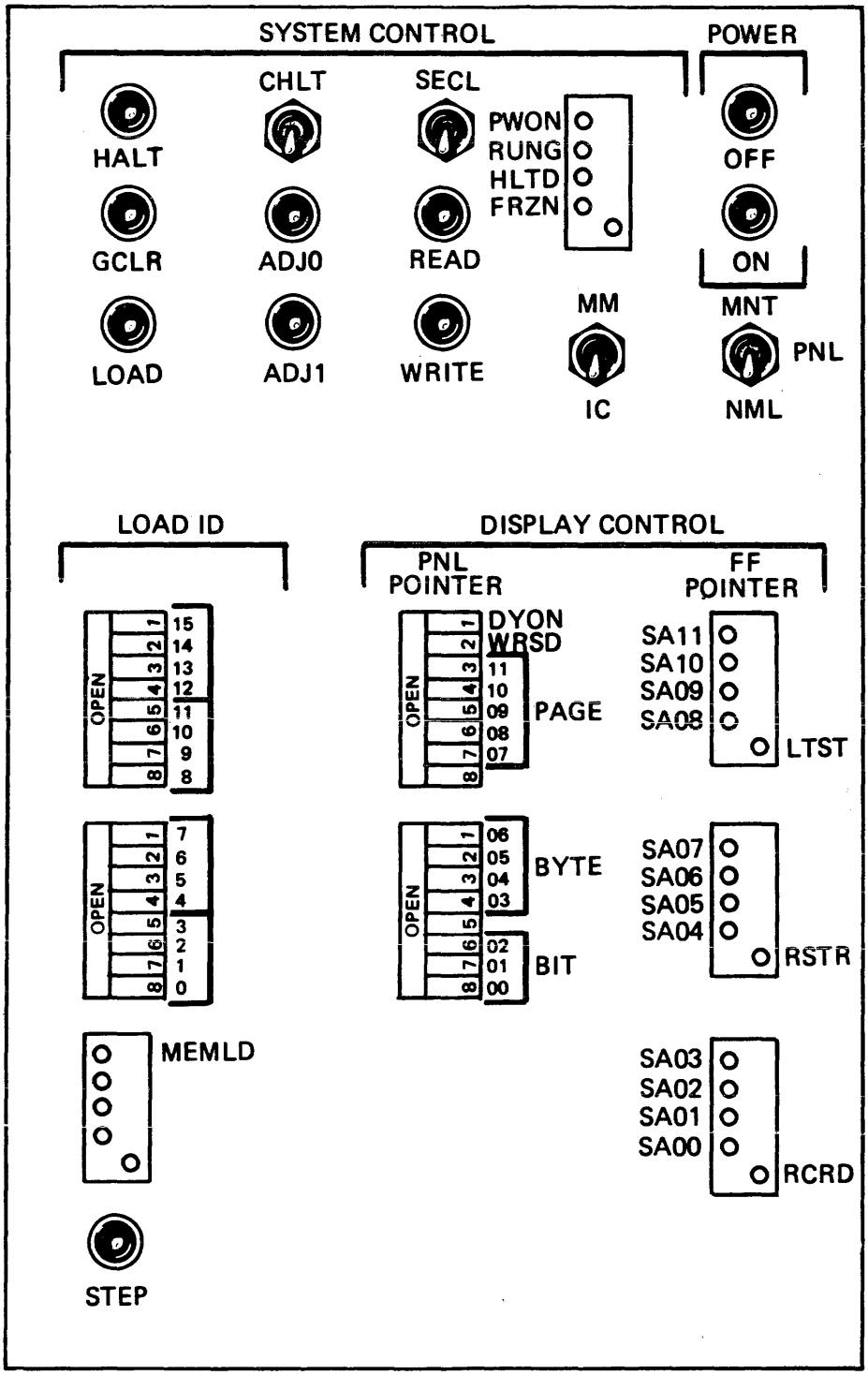


Figure 4-4. Maintenance Control Panels in an IODC Cabinet



MV4434

Figure 4-5. System Control Panel

and the system stops as though the HALT pushbutton had been depressed and released. If any other CPU operator is executed with the CHLT switch in position, the HALT flip-flop does not SET and the system does not stop. The CHLT logic is independent of any other system halt logic, and functions regardless of the position of the HALT pushbutton or any other system halt control.

SECL

The SECL toggle-switch is a HALT function selector, the same as the CHLT toggle-switch (see CHLT above). When the switch is in the SECL (UP) position, the SECL control circuit is enabled; when the switch is in the DOWN position, the SECL circuit is disabled.

If the SECL circuit is enabled the data processor halts at the end of each machine language operator sequence, when the SECL signal is TRUE. If the SECL circuit is disabled the data processor does not halt because the SECL signal went TRUE; however, it may halt at SECL signal time because another halt logic circuit is enabled. The SECL halt logic circuit is independent of all other halt logic circuits.

GCLR

The General Clear pushbutton is used to cause the logic circuits of the B 6900 system to be initialized. Generally, clearing a logic circuit causes it to be RESET (go to the ZERO or FALSE condition). However, some logic circuits such as the HDPH/ logic signal of the MLIP are initialized or cleared to the SET (or TRUE) condition.

LOAD and MEMLD

The LOAD pushbutton switch is used in conjunction with the MEMLD status display selector to cause a LOAD sequence of operations by the Maintenance Processor. A LOAD sequence is a firmware program routine from a flexible diskette that is present in the Maintenance Processor RAM memory. When the LOAD pushbutton is depressed and released, a system main memory LOAD operation or a Maintenance Processor RAM memory sequence is performed, depending on the state of the MEMLD status display. If MEMLD is SET, a system main memory LOAD sequence is performed. If MEMLD is RESET, a Maintenance Processor RAM memory LOAD sequence is performed.

If MEMLD is SET (status indicator illuminated) and the LOAD pushbutton is depressed, the B 6900 system is general cleared as when the GCLR pushbutton (see GCLR above) is depressed and released. When the LOAD pushbutton is released, the Maintenance Processor executes the LOAD routine sequence present in its RAM memory. The LOAD sequence causes a program file to be loaded into system main memory from a predefined system peripheral device. After a program file is loaded into memory, the data processor can fetch and execute code from the program file.

The program code loaded into memory may be the system software Master Control Program (MCP) or another system executive program. The LOAD pushbutton sequence is thus capable of initializing various B 6900 executive programs into operation. A choice between loading the MCP or some other executive program is made by proper selection of the predefined system peripheral device from which the program file is loaded. The LOAD ID switches, defined in a subsequent paragraph of this section, are used to select the peripheral device.

If the B 6900 system is operating and a LOAD sequence is to be performed, the system must be halted before the LOAD pushbutton is depressed and released. The HALT pushbutton (see HALT above) is used to halt the system in preparation for a LOAD operation.

If MEMLD is RESET (status indicator extinguished) and the LOAD pushbutton is depressed and released, a data file from a known peripheral device is loaded into the Maintenance Processor RAM memory. At the conclusion of the LOAD operation, the logic of the Maintenance Processor branches to the beginning address in RAM memory and halts. A subsequent initialization of the Maintenance Processor control logic causes the program in the RAM memory to be executed.

ADJ0 and ADJ1

The ADJ0 and ADJ1 pushbuttons are used to cause stack adjustments by the CPU Stack Controller logic. The ADJ0 pushbutton, when depressed and released, causes valid data words present in the Top-of-Stack A(X) and B(Y) registers to be pushed down into the memory portion of the stack. The ADJ1 pushbutton, when depressed and released, causes the stack to be adjusted until the top word in the stack is present in the Top-of-Stack A(X) register.

READ

The READ pushbutton, when depressed and released, causes the Memory Controller logic of the CPU to perform a READ memory cycle. If the MMIC switch is in the MM (UP) position, the READ operation is performed in system main-memory. If the MMIC switch is in the IC (DOWN) position, the READ operation is performed on a CPU IC Memory Address register.

WRITE

The WRITE pushbutton, when depressed and released, causes the Memory Controller logic of the CPU to perform a WRITE memory cycle. If the MMIC switch is in the MM (UP) position, the WRITE operation is performed to system mainmemory. If the MMIC switch is in the IC (DOWN) position, the WRITE operation is performed on a CPU IC Memory Address register.

POWER ON/OFF

The POWER ON and POWER OFF pushbuttons initiate power sequences in the B 6900 Central Power Supply cabinet. If the B 6900 system is not powered up (source input-power is present at the input to the System Power Supply cabinet) and the POWER ON pushbutton is depressed and released, then the Central Power Supply cabinet logic performs a power-up sequence. If the B 6900 system is already powered up when the POWER ON pushbutton is depressed and released, nothing happens.

If a B 6900 system is powered up and the POWER OFF pushbutton is depressed and released, then the Central Power Supply cabinet logic performs a power-off sequence. If the B 6900 system is already powered down when the POWER OFF pushbutton is depressed and released, nothing happens.

PNL

The Panel toggle switch (PNL) selects the B 6900 units to which the System Control Panel interfaces. If the PNL switch is in the Normal (NML, DOWN) position, the System Control Panel interfaces to the CPU cabinet and also to the Maintenance Processor module. If the PNL switch is in the Maintenance (MNT, UP) position, the System Control Panel only interfaces to the Maintenance Processor.

The Maintenance Processor is interfaced to the CPU cabinet by a Host Control Port interface cable. System Control Panel switches use the Host Control Port interface to initiate and control functions in the CPU cabinet. When the PNL switch is in the MNT (UP) position, System Control Panel signals are prevented from using the Host Control Port interface cable; consequently, CPU function control switches on the System Control Panel are inoperative. Maintenance Processor control functions of the System Control Panel are operational when the PNL switch is in either position. Central Power Control functions are also functional when the PNL switch is in either position.

CONTROL STATUS

Control Status display devices indicate the status of control logic signals, as follows.

PWON STATUS

The PWON indicator is illuminated when the cabinets of the B 6900 are receiving source power from the CPS cabinet. The PWON indicator is extinguished when the CPS is not supplying source power to the B 6900 cabinets.

RUNG STATUS

The RUNG indicator is illuminated when the Running-timer circuit is timing. If the Running-timer times-out, the RUNG indicator extinguishes.

HLTD STATUS

The HLTD indicator is illuminated when the CPU Halted flip-flop is SET. If the Halted flip-flop is RESET, the HLTD indicator is extinguished.

FRZN STATUS

The Frozen (FRZN) indicator is illuminated when the CPU clock is stopped, an EVENT Mode or Maintenance Mode condition. If the CPU clock is running the FRZN indicator is extinguished.

MEMLD STATUS

The MEMLD status indicator is used in conjunction with the LOAD control push-button (see LOAD above). MEMLD is SET by depressing and releasing the push-button corresponding to the MEMLD indicator LED. The LED illuminates when MEMLD is SET. If the LED is already illuminated and the pushbutton is depressed and released, nothing happens.

The MEMLD indicator is RESET by simultaneously depressing the pushbutton corresponding to the LED indicator and the bottom pushbutton on the LED indicator circuit device. When MEMLD is RESET the LED extinguishes. If MEMLD is already RESET and both pushbuttons are depressed and released, nothing happens.

LTST

The Lamp Test (LTST) pushbutton is used to test for faulty LED indicator circuits in the MDP display registers. When the LTST pushbutton is depressed, all LEDs in the MDP display registers are illuminated. When the LTST pushbutton is released, the MDP display register LEDs return to indicating system status conditions.

RCRD and RSTR

The RCRD pushbutton is used to cause the current displayed status of the B 6900 system to be recorded in the MDP display RAM memory. Recording the status in the RAM occurs when the RCRD pushbutton is depressed and released.

Depressing and releasing the RSTR pushbutton causes the B 6900 system status stored in the MDP display RAM to be restored as the current state of all displayed logic circuits.

The RCRD/RSTR pushbuttons are typically utilized to perform a maintenance operation on the B 6900 system. Before the maintenance operation is performed, the normal system operational state is recorded by means of the RCRD pushbutton. Before resuming normal system operations, the state of the system is restored by means of the RSTR pushbutton.

DISPLAY CONTROL

The Panel Pointer (PNL POINTER) logic contains 2 rocker-switch control devices. These rocker-switches are used to cause a particular flip-flop in the CPU cabinet to SET, similar to the way the MDP display register SET logic works. However, the PNL POINTER logic can only SET (not RESET) one CPU flip-flop at a time. In addition, the PNL POINTER Logic can be used to translate a hex CPU flip-flop address value to its corresponding MFIU address line value. When the PNL POINTER switches are used, the address value of the switches is translated to MFIO address line signals, which are displayed by the SAnn LED display devices.

Two ROCKER switches are used as control logic signals for the DISPLAY CONTROL logic. Rocker switch DYON must be ON to enable the MDP display panel logic. When DYON rocker switch is OFF the MDP display panel logic is disabled. The WRSD rocker switch connects the DISPLAY CONTROL logic to the Host Control Port interface logic. When WRSD is in the OFF position, the display control logic is disconnected from the CPU HCP interface logic; and the MDP display logic cannot SET or RESET CPU flip-flops.

If the WRSD rocker switch is OFF, a CPU flip-flop address can be translated from the PAGE, BYTE, and BIT notation used by the MDP logic to the equivalent CPU cabinet backplane address value. This equivalent address value is displayed in the SAnn LED circuits. The DYON switch is OFF for translation operations.

LOAD ID

The LOAD ID rocker switches are used to identify a peripheral unit through which a LOAD function can be performed (see LOAD, above). There are 2 rocker switch devices, each of which contains 8 switches. The switches are numbered from 0 through 15, and the value of a switch number denotes the binary significance of the switch in determining the LOAD peripheral unit identity number. The switches have significance as follows:

	Switch Number	Binary Weight
TOP ROCKER SWITCH	15	32768
	14	16384
	13	8192
	12	4096
	11	2048
	10	1024
	09	512
	08	256
BOTTOM ROCKER SWITCH	7	128
	6	64
	5	32
	4	16
	3	8
	2	4
	1	2
	0	1

The peripheral device identified by the LOAD ID rocker switches must be a proper I/O device type (magnetic tape, head-per-track disk, disk pack, or card reader) controlled by a DLP device present in IODC zero. The IODC base module must be properly configured to include the I/O device number represented by the LOAD ID rocker switches.

B 6900 MAINTENANCE CONTROL PANEL

Figures 4-1 and 4-4 show the location of the Maintenance Control Panel. This panel (Refer to Figure 4-6) contains switches and indicators used for maintenance operations on the B 6900 system. The following paragraphs define the function of each switch and indicator on the panel, and briefly describe the system actions performed as a result of operating a control.

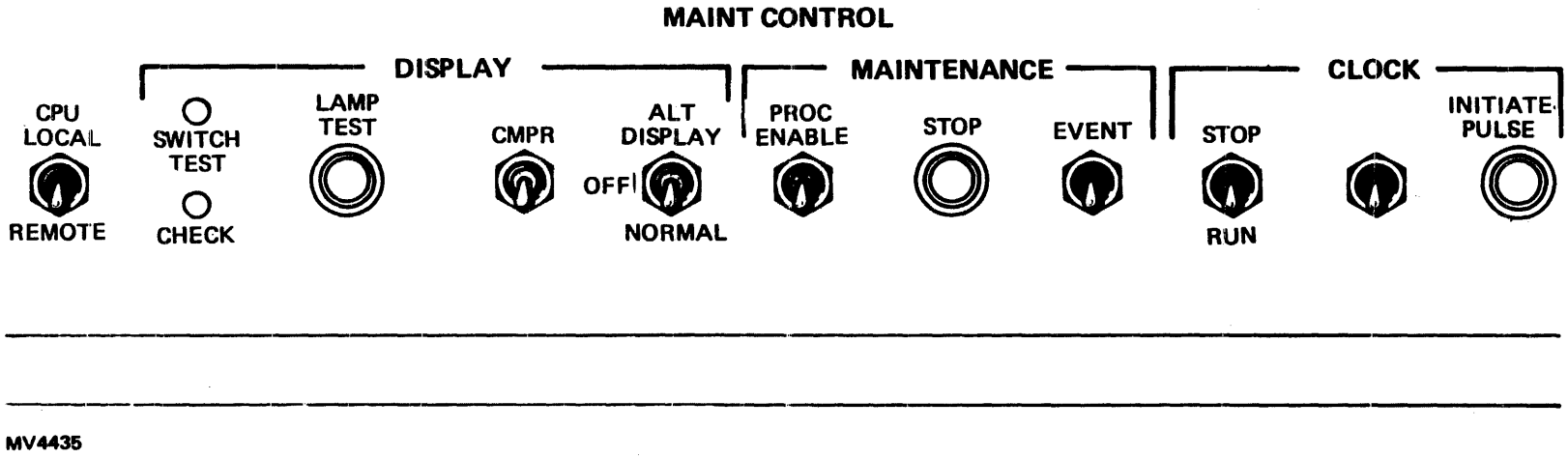


Figure 4-6. System Maintenance Control Panel

CPU LOCAL/REMOTE

The CPU LOCAL/REMOTE switch is used to select on-line system operation or local-unit system operation. In the REMOTE (DOWN) switch position, on-line operation is selected. In the LOCAL (UP) position, local-unit operation is selected.

During local-unit operations the CPU Memory Controller cannot perform Global memory accesses. Peripheral subsystems such as Reader/Sorters and/or Data Communications that use Global memory resources cannot function when the switch is in the LOCAL (UP) position. When the switch is in the REMOTE (DOWN) position Global memory accesses are performed; therefore, subsystems that utilize Global memory resources are fully functional.

SWITCH TEST

The SWITCH TEST indicator illuminates when any pushbutton switch for an MDP display register is depressed. The indicator is extinguished when no pushbutton switch for an MDP display register is depressed. This indicator detects shorted pushbutton switch circuits that remain closed when the pushbutton is released.

CHECK

The CHECK indicator illuminates when a fault is present in the CPU during a maintenance operation. The CHECK indicator extinguishes when a system genera-clear operation is performed.

The CHECK indicator is also used as a Maintenance Processor flag that illuminates when a Confidence Test detects a fault condition. The system may perform in a normal manner after the CHECK indicator flag is illuminated. However, the fact that a CHECK condition occurred is significant for subsequent maintenance operations. Therefore, the indicator remains illuminated until a system general-clear operation is performed.

LAMP TEST

The LAMP TEST pushbutton, when depressed, causes all lamps and LEDs in the MDP to illuminate. This pushbutton is used as a test for burned-out lamps or LEDs. The pushbutton is spring-loaded and returns to the OFF position when released.

CMPR

This toggle switch has three positions. When the switch is in Center/DOWN position, Normal display mode is selected. When the switch is in the CMPR (UP) position, Comparator display mode is selected.

Comparator mode operations, reserved for factory-use only, are not used for normal system operations. The CMPR switch is placed in the OFF (DOWN) position and remains in that position.

DISPLAY

This toggle switch has three positions. In the NORMAL (DOWN) position the MDP display logic is enabled and the status of the CPU is displayed in the MDP display registers. In the Alternate (ALT, UP) position, the status of another CPU is displayed in the MDP display registers. The ALT position is normally used for factory tests with a comparator to display the alternate CPU status, and is not used otherwise. The OFF (MIDDLE) position disables the MDP display logic, and no status is displayed in the display registers.

PROC ENABLE

The Processor Enable (PROC ENABLE) toggle-switch is used to select maintenance mode (in which the MP controls the HDP interface bus to the CPU) or to select normal mode (in which the CPU controls the HDP interface bus to the MDP). When the switch is in the PROC ENABLE (UP) position the MP logic controls the HDP interface bus, and maintenance mode is selected. When the switch is in the OFF (DOWN) position the CPU controls the HDP interface bus, and normal mode operations are selected.

STOP — The STOP pushbutton, when depressed, unconditionally stops the CPU in EVENT mode (refer to EVENT, below). The pushbutton is spring-loaded to the OFF position.

EVENT — The EVENT toggle-switch selects EVENT mode operations or normal mode operations. When the switch is in the EVENT (UP) position the CPU EVENT logic is enabled. When the switch is in the OFF (DOWN) position normal system operations are enabled, and EVENT logic is disabled.

CLOCK STOP/RUN — The Clock STOP/RUN toggle switch selects whether clock pulses are continuous or stopped. When the switch is in the RUN (DOWN) position, CPU clock-pulses are free-running; but are subject to stoppage by maintenance mode or EVENT mode FROZEN logic. When the switch is in the STOP (UP) position clock-pulses are prevented from being distributed to the CPU cabinet logic circuits.

INITIATE PULSE — The INITIATE PULSE pushbutton is used to cause a single clock pulse to be emitted when the STOP/RUN switch is in the STOP (UP) position. Each time the pushbutton is depressed and released, one clock pulse is emitted to the CPU logic circuits. The pushbutton is spring-loaded to the OFF position.

The undefined switch located between the STOP/RUN switch and the INITIATE PULSE pushbutton is unused in a B 6900 system.

B 6900 MAINTENANCE PROCESSOR CONTROL PANEL

Figures 4-1 and 4-4 show the location of the Maintenance Processor Control Panel. This panel (refer to Figure 4-7) contains switches and indicators used to control the operation of the B 6900 Maintenance Processor. The following paragraphs define the function of each control or indicator and briefly describes the system actions performed as a result of operating a control.

BANK LAMPS — The BANK LAMPS consist of 5 sets of indicators labeled A, B, C, D, and E. Each set contains 8 indicator lamps, numbered 0 through 7. The BANK LAMPS are connected programatically to logic circuits of the Maintenance Processor, and are used to display the status or value of the circuit to which they are presently connected.

The instantaneous indication of the BANK LAMPS depends on the current operating sequence and status of the Maintenance Processor, and is therefore too varied to define here. For precise technical data on various BANK LAMP indications consult the B 6900 Maintenance Processor FETM, Form No. 5011307.

INIT — The Initialize (INIT) pushbutton is used to clear and initialize the Maintenance Processor logic circuits. When the pushbutton is depressed and released, the Maintenance Processor begins to execute MP microcode instructions present in its ROM memory. The MP ROM memory contains a set of primitive MP instructions used to initiate all subsequent MP operations. If a fault condition occurs during the initialization processes the MP logic stops and displays its status in the BANK LAMPS. The interpretation of the BANK LAMP indications during the initialization is defined in the B 6900 Maintenance Processor FETM, Form No. 5011307.

MTR — The Maintenance Test Routine indicator lamp is used to indicate that the MP is performing a self-diagnostic test. The MTR signal level is used to select 1-of-2 areas of MP PROM memory. The other area of PROM memory contains the MP initialization microcode.

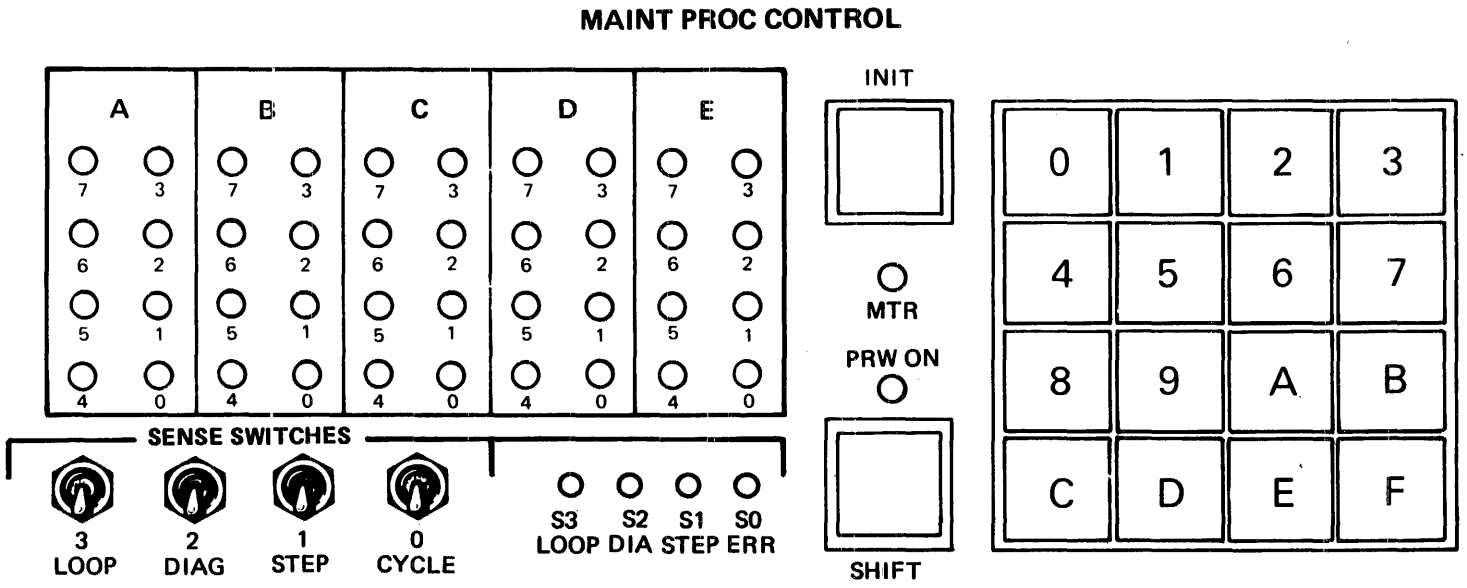


Figure 4-7. Maintenance Processor Control Panel

PWR ON	The PWR ON indicator lamp is used to indicate that source input power is applied to the input of the Maintenance Processor.
SHIFT	The SHIFT pushbutton is used to expand the number of key positions for the keyboard, from 16 to 32 positions. If the SHIFT key is not depressed, the value of the keyboard selects the corresponding position from among the first set of 16 positions. If the SHIFT key is depressed, the value of the keyboard selects the corresponding position from among the second set of 16 positions.
KEYS 0 THROUGH F	The 16 key (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F) keyboard is used to enter data into the MP memory.
LOOP, DIAG, STEP, and CYCLE SENSE SWITCHES	These 4 toggle switches are used to exercise control over the major functional processes of the MP. The logic flow of the MP senses the positioning of these switches and alters MP processes accordingly. The meaning and use of these switches is defined in the B 6900 Maintenance Processor FETM, Form No. 5011307.
S0, S1, S2, and S3	These 4 indicators that occur during the execution of test routine programs indicate errors (ERR), and other MP operating conditions (LOOP, DIA, and STEP).

B 6900 SOFT DISPLAY

An Operator Display Terminal (ODT) device may be used for system status display. A firmware executive program (Soft Display) is activated during B 6900 system initialization, and this program causes system status to be formatted and displayed on the ODT peripheral device screen.

The Soft Display firmware executive program may be utilized in B 6900 systems that have an MDP cabinet installed. Such systems have 2 methods of displaying system status. Systems that do not have an MDP cabinet have only the Soft Display method for system status display.

B 6900 Soft Display Program Control

The Soft Display program is selected for execution by entering and transmitting a "Y" input message on the ODT screen, when the initialization program displays the "AWAITING A/T" output message. This output message indicates that the firmware Executive program is at the Command level, and that it is ready to receive an input Command message. The "Y" input message is a Command input which specifies that the Soft Display program is to be executed. In response to the "Y" input message, a "Soft Display" output message flashes, and a list of current Soft Display Command names are displayed on the ODT screen.

The Soft Display program is a control mechanism through which various system display and control functions are initiated. The following paragraphs describe and define the input messages used to cause a Soft Display Command to be executed.

ODT SCREEN FORMAT

A B 6900 system Operator Display Terminal (ODT) screen can display up to 25 lines of data, and each line contains 80 alphanumeric character columns. The character display lines of the ODT screen are numbered from top-to-bottom, with line-1 the upper-most line on the screen, and line-25 the bottom-most line. Line-1 through line-24 are usable lines on which data can be displayed. Line-25 is reserved for ODT subsystem status report and command displays, and is not used for normal data display. The left-most character position on a line of the screen is column-1, and the right-most character position is column 80.

The Soft Display program controls the ODT screen format during a Soft Display program function. Line-1 and line-2 contain Soft Display program commands. Line-3 is reserved for Soft Display program error displays. Line-4 and line-5 are used as a buffer for additional Soft Display commands. Additional commands in the command buffer execute only when called by specific commands present on line-1 or line-2.

Soft Display commands on lines 1, 2, 4, and 5 of the ODT screen are used to specify B 6900 system control functions to be performed by the Soft Display program, and also to specify various B 6900 system status to be displayed on the ODT screen.

The Soft Display control program takes advantage of the ODT screen programmable intensity feature, when displaying B 6900 system status. This screen feature allows data to be displayed at 2 different levels of intensification, or brilliance. Display status data is brilliantly intensified if it is TRUE or HIGH (the binary-1 condition), and is moderatley intensified if it is FALSE or LOW (the binary-0 condition).

ODT SCREEN COMMAND STRUCTURE AND OPERATION

The Soft Display program displays a list of valid Soft Display command names on the ODT screen in response to a "Y" input at the executive level (see B 6900 SOFT DISPLAY above). The user of the Soft Display program must construct a string of syntactically correct Soft Display commands on line-1 and line-2 (plus optional commands on line-4 and line-5).

After a string of Commands is constructed, depressing and releasing the <XMIT> key of the ODT keyboard causes the Soft Display program to execute the commands present in the Command string.

The Soft Display program executes commands present on line-1 and line-2 in the order of occurrence. The order of occurrence is from the left-most command to the right-most command on line-1, followed by the left-most command through the right-most command on line-2. Soft Display commands on line-4 and line-5 are executed in the same order as line-1 and line-2, but are not executed unless specific commands present on line-1 and/or line-2 direct that commands in the command buffer be executed.

The Soft Display program checks each command for proper syntax before the command is executed. If no syntax error is found the command is executed, and this sequence is repeated for the next command in the command string. If a syntax error is found, error data is displayed on line-3 of the ODT screen and the Soft Display program immediately terminates without executing the command that contained the syntax error or subsequent commands in the command string.

The Soft Display program continues executing commands until a syntax error occurs, until all commands in the command string have been executed, or until an <END> command is executed. When any (one) of these events occurs the Soft Display program completes by returning control of the Maintenance Processor to the executive level of operation. A subsequent operation of the Soft Display program must be initiated by means of another "Y" input message on the ODT screen.

The B 6900 Maintenance Processor saves a copy of all Soft Display command strings, so that they can easily be repeated as often as the program user desires. The syntax of the Soft Display program commands provides a method for recovering the former contents of a command string for subsequent Soft Display program operations (see <SAVE> and <RETURN> syntax diagrams).

SYNTAX DIAGRAM RULES

The syntax for constructing valid Soft Display commands are presented in the following "Railroad diagrams." These diagrams yield valid Soft Display command formats when they are followed along the forward direction indicated by arrowhead symbols. The optional characteristics of a valid command statement are given in semantic discussions of the diagrams.

Soft Display program commands are entered on line-1 or line-2 of the ODT screen in the order that they are to be executed. A blank space separates consecutive commands on a line, and commands are not split across line-boundaries.

SOFT DISPLAY COMMAND CATEGORIES

Table 4-5 lists the 4 general categories of Soft Display commands that are used. The following paragraphs describe these 4 categories.

SYSTEM — System commands provide a method for activating system control logic circuits, in the same way that MAINT/EVENT commands invoke the maintenance or event control logic (see MAINT/EVENT command).

MAINT/EVENT — Maintenance logic or EVENT logic commands cause circuit control devices to SET or RESET. By setting or resetting a circuit control device, the corresponding maintenance circuit or EVENT logic circuit is activated or deactivated. These commands allow the Soft Display program to invoke programmatic maintenance or event logic activation as if manual control switches had been positioned by the system user.

FAMILIES — Family commands are used to activate CPU function status display signals and levels. A Soft Display Family command mnemonic implies the collection and formatting of status display signals for a particular CPU function. Activation of a Family CPU status display causes the collected and formatted CPU status to be displayed on the ODT screen.

FUNCTIONS — Function commands provide programmatic methods to activate CPU or MDP circuits that handle data. The syntax for a Function command provides a mechanism for supplying required input and/or handling any resulting output data.

Table 4-5. B 6900 Soft Display Command List

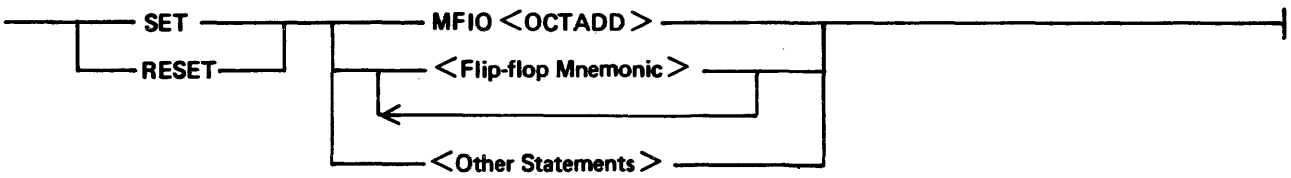
System Commands	MAINT/EVENT Commands	Families Commands		Functions Commands		
ARCS	AAIF	A	MEMCON	ADD	FAMILY	REVERS
HALT	ALTF	ARICON	MEMPRT	BRIGHT	HELP	SAVE
PULSE	CHLT	B	MEMTST	CAPTUR	INFO	SMEAR
STEP	CPTF	C	MMOD	CLRIC	INSERT	STATUS
STOP	CSTP	CPU	PROGCL	CLRMM	NZDATA	USRFAM
	EVNT	D	U	DEL	PROGRM	WAIT
	LOCL	E	UFAM	DIFF	RDHDP	WRIC
	NOSTEP	ERRORS	XREFCL	DO-UNTIL	RDIC	WRMM
	OCTAL	EVENT		DUMP	RDMM	**
	SAFE	GLOBAL		END	RESTOR	--
	SECL	INTCON		EXEC	RETURN	++
		I/O				

SOFT DISPLAY PROGRAM GENERAL COMMANDS

The Soft Display program command structure includes General Commands which do not conform to any one of the previously defined command categories. These general commands are used basically to change the state of particular system flip-flops or the contents of registers, without disturbing the state of other circuit devices. These commands add a dimension of choice and selectivity to the power of the Soft Display program. The General Commands of the Soft Display program are defined in the following paragraphs.

<SET> and <RESET> COMMANDS

The Soft Display program logic executes SET or RESET instructions present in a command string. A SET/RESET instruction causes the CPU flip-flops identified by the SET/RESET instruction to go to the SET or RESET state.



MV4517

SEMANTICS

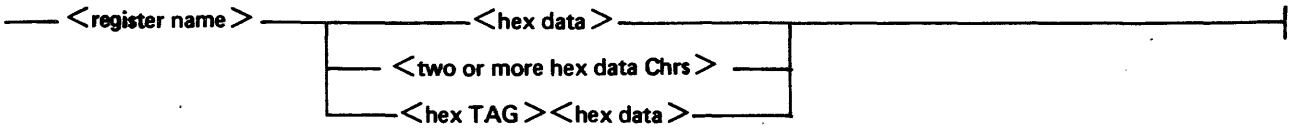
The SET option of this Command causes the flip-flop devices named to SET. The RESET option of this Command causes the flip-flop devices named to RESET.

The "MFIO <OCTADD>" option allows CPU flip-flops that are not defined as a Soft Display register flip-flop to be SET/RESET. Octal addresses for CPU flip-flops are obtained from the MDP Display Fault Lists. MDP Display Fault Lists are part of Test and Field documentation delivered from the factory with the hardware.

The SET/RESET <Flip-flop Mnemonic> option allows multiple flip-flops to be SET or RESET by a single Command. This option is used to control MAINT/EVENT and System Command flip-flops.

<REGISTER> COMMANDS

The Soft Display program contains a list of register names that are frequently used for displaying system status or for control functions. A REGISTER Command is used to replace the contents of a register.



MV4518

SEMANTICS

A <register name> must be register listed in the Soft Display program register name list. If the contents of a register not present in the Soft Display name list is to be replaced, then the SET/RESET Command must be used (see <SET> Or <RESET> above). Table 4-6 is a list of the Command register names defined in the Soft Display program.

The <register name> <hex TAG> <hex data> Command option is used to replace the contents of Top-of-Stack registers and their TAG fields. These registers each contain 1 hex TAG character and 12 hex data characters. If <hex data> contains more than 12 hex characters, the right-most hex characters (in excess of 12) are truncated and lost. If <hex data> contains fewer than 12 hex characters, the hex data is placed in the register, right-justified; and register bits not filled from <hex data> are SET to leading zeroes. The register TAG is filled from the <hex TAG> character. One blank character space separates <register name>, <hex TAG>, and <hex data> in this Command format.

The <register name> <hex data> Command option is used to replace the contents of all Soft Display program defined registers other than Top-of-Stack registers. If the significant binary bits in <hex data> exceeds the binary bit capacity of the <register name> register, an error condition is detected and reported on line-3 of the ODT screen. When an error condition is detected the Soft Display program immediately terminates. If the number of significant binary bits present in <hex data> is less than the binary bit capacity of <register name>, the register is filled by binary bits from <hex data>, right justified. Binary bits of the register not filled from <hex data> are RESET to leading zeroes.

If the <register name> <2 or more hex data Chrs> Command option is used, the TAG-field of <register name> is zeroed. This option allows a TAG-field for a Top-of-Stack register to be RESET to zero without altering the <hex data> contents of the register.

Table 4-6. Soft Display Register Names (Sheet 1 of 6)

Soft Display Register Name	Circuit Displayed	Display Meaning or Usage
A	AR[50:51]	Displays the HEX value of the Top-of-Stack A register
ACL	ACL[7:8]	Displays the least significant 8-Bits of a NAMC operator address value
ACM	ACM[5:2]	Displays the most significant 2-Bits of a NAMC operator address value
ADSV	ADD[19:20]	Displays the value of the Address-Save register
AX	AX[02:3]	Displays the value of the A mantissa extension register
AZ	AZ[63:3]	Displays the value of the AZ6n signals, which are used to transfer a specific field from a transmitter register into a receiver register, by means of the Stack Controller Z6 bus
B	BR[50:51]	Displays the HEX value of the Top-of-Stack B register
BI	BI[2:3]	Displays the value of the MLIP byte index register
BRS	BRS[7:8]	Displays the contents of the CPU IC memory base read select register signals
BX	BX[2:3]	Displays the contents of the B mantissa 1-octade extension register
BYR	BYR[19:20]	Displays the memory-tester logic BYPASS register contents
C	CR[50:51]	Displays the contents of the Top-of-Stack C register
CA	CAOF[2:3]	Displays the Memory Controller Port priority occupying status signals
CBR	CB[6:6]R	Displays the Memory Controller READ-data check-bit code value
CBW	CB[6:6]W	Displays the Memory Controller WRITE data check-bit code value
CI	CIOF[3:4]	Displays the Memory Controller Port priority occupying signals for channel B memory requestor
CKBA	CKB[6:6]A	Displays the Memory Controller data check-bit code value for requestor A
CKBB	CKB[6:6]B	Displays the Memory Controller data check-bit code value for requestor B
CP	CP[23:24]	Displays the value of the high-order 24 bits in the processor timer register
CPA	CPA[8:4]	Displays the value of the CPU clock counter circuit
CSC	CSC[4:4]	Displays the sequence count value for the Memory Controller requestor logic
CSR	CSR[2:3]	Displays the value of the Program Controller count syllable register

Table 4-6. Soft Display Register Names (Sheet 2 of 6)

Soft Display Register Name	Circuit Displayed	Display Meaning or Usage
DI	DI[8:4]	Displays the value of the string operator destination index byte register
DIS	DIS[5:6]	Displays the value of the Transfer Controller displacement register
DRF	DRF[4:3]	Displays the value of the IC memory display register address select bits
DST	DST[3:4]	Displays the value of the MLIP delayed status register
DSZ	DSZ[2:2]	Displays the value of the string operation destination byte size register
ECT	ECT[7:8]	Displays the value of the EVENT logic counter
EJC	EJC[11:12]	Displays the value of the EVENT logic micro-module J-count (sequence) register
EOP	EOP[3:4]	Displays the EVENT logic operator code register value
EREN	EREN[7:8]	Displays the value of the CPU PROM Card location register
EST	EST[7:4]	Displays the value of the EVENT logic strobe register
FST	FST[3:4]	Displays the value of the MLIP fast status signal register
GBC	GBC[2:3]	Displays the value of the Global sequence-control register
GBS	GBS[2:3]	Displays the value of the Global clear sequence-control register
GPS	GSP[2:3]	Displays the value of the MLIP Global priority save register
GS	GS[2:3]F	Displays the value of the Global memory control signal register
GT	GT[2:3]F	Displays the value of the Global memory control signal register
HAR	HAR[3:4]	Displays the value of the Memory Controller hold address for return register
HR	HR[15:16]	Displays the value of the Arithmetic Controller holding register
HRTA	HRTA[2:2]	Displays the value of the Arithmetic Controller exponent adder A-side input holding register
HRTB	HRTB[2:2]	Displays the value of the Arithmetic Controller exponent adder B-side input holding register
ICR	ICR[7:8]	Displays the value of the Input-Convert operation register

Table 4-6. Soft Display Register Names (Sheet 3 of 6)

Soft Display Register Name	Circuit Displayed	Display Meaning or Usage
ICW	ICW[3:4]	Displays the value of the Memory Controller IC memory REFRESH function delay (for MSU signal) register
IMCF	IMC[3:4]	Displays the value of the initiate cycle control signals to the 4 CPU local memory port adapters
IML	IML[2:3]	Displays the value of the Interrupt Controller counter used for detecting SUPERHALT conditions
IRS	IRS[7:8]	Displays the value of the CPU IC memory index register READ select signals
IT	IT[10:11]	Displays the value of the interval-timer register
JA	JA[7:8]F	Displays the value of the Family A sequence-count (J-count) register
JB	JB[3:4]F	Displays the value of the Family B sequence-count (J-count) register
JC	JC[7:8]F	Displays the value of the Family C sequence-count (J-count) register
JCS	JCS[11:12]	Displays the value of the EVENT logic J-count save register
JD	JD[7:8]F	Displays the value of the Family D sequence-count (J-count) register
JE	JE[6:7]	Displays the value of the Family E sequence-count (J-count) register
JP	JP0[2:3]	Displays the value of the Program Controller sequence-count register
JS	JS[4:4]F	Displays the value of the Stack Controller sequence-count register
JU	JU[6:7]F	Displays the value of the Family U sequence-count (J-count) register
JV	JV[1:2]	Displays the value of the Memory-tester logic sequence counter
KA	KA[2:3]F	Displays the value of the Family A K-counter
L	LR[50:51]	Displays the value of the Program Controller look-ahead register
LAR	LA[19:20]	Displays the value of the Program Controller look-ahead address register
LC	LC[3:4]F	Displays the value of the Family E loop-count register
LL	LL0[4:5]	Displays the value of the Program Controller lexicographical level register
LP	LP[15:16]	Displays the value of the MLIP longitudinal parity register
MAR	MA[19:20]	Displays the value of the Memory Controller memory address register

Table 4-6. Soft Display Register Names (Sheet 4 of 6)

Soft Display Register Name	Circuit Displayed	Display Meaning or Usage
MDS	MDS[3:4]	Displays the value of the MLIP maintenance display status register
MFS	MFS[3:4]	Displays the value of the MLIP maintenance fast status register
MM	MM[B:12]	Displays the value of the micro-module address register (entry-vectors)
MR	MR[16:17]	Displays the value of the MLIP maintenance data register
MRA	MRA[4:5]	Displays the value of the MLIP memory register address (for MLIP RAM memory)
MSM	MSM[19:20]	Displays the value of the Memory Controller address-adder sum-register
MSOR	MSOR[2:3]	Displays the value of the address adder sum-of-residue register
MSP	MSP[9:10]	Displays the value of the MLIP micro-stack (MLIP RAM memory) pointer register
MSW	MSW[3:4]	Displays the value of the Memory Controller select-WRITE control signals to the local memory port adapters
NLZ	NLZ[3:4]	Displays the value of the Arithmetic Controller number of leading zeroes register
OSR	OSR[3:4]	Displays the value of the EVENT logic operator code save register
P	PR[50:51]	Displays the contents of the Program Controller program-code register
PAD	PAD[2:3]	Displays the value of the MLIP port address register
PAS	PAS[2:3]	Displays the value of the MLIP port address save register
PEDF	PEDF[3:4]	Displays the value of the Memory Controller parity-error-disable control signals to the local memory port adapters
PSC	PSC[4:5]	Displays the value of the MLIP priority sequencer count register
PSR	PSR[2:3]	Displays the value of the Program Controller program syllable register
RQR	RQR[9:10]	Displays the value of the Memory Controller request address register
RQT	RQT[9:10]	Displays the value of the Memory Controller request address trap register
R1	R1[19:20]	Displays the value of the MLIP R1 register
R2	R2[19:20]	Displays the value of the MLIP R2 register

Table 4-6. Soft Display Register Names (Sheet 5 of 6)

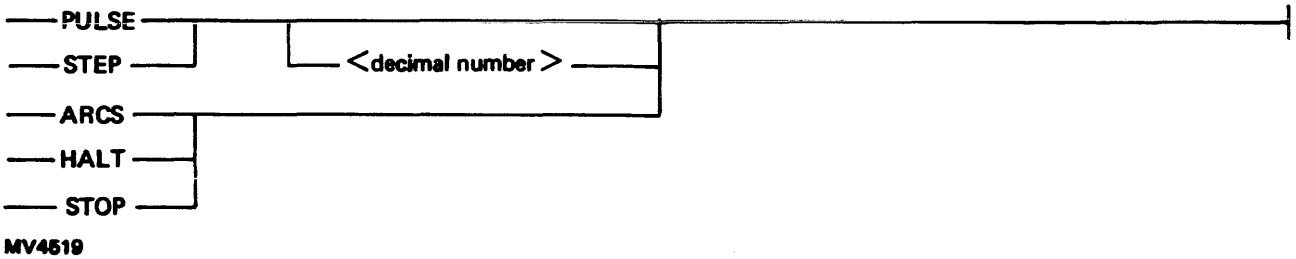
Soft Display Register Name	Circuit Displayed	Display Meaning or Usage
R3	R3[19:20]	Displays the value of the MLIP R3 register
SA	SA[3:4]F	Displays the contents of the Family A T-register
SC	SC[3:4]F	Displays the contents of the Family E scale count register
SF	SF[3:4]F	Displays the value of the Family E scale factor register
SI	SI[08:4]	Displays the value of the Family U source byte index register
SM	SM[04:4]	Displays the contents of the Family A steering-and-mask register (regenerates TOA, TOM, and DIS values)
SPMB	SPMB[3:4]	Displays the value of the Memory Controller single-pulse mode control signals for local memory port adapters
SRL	SRL[2:3]	Displays the value of the Memory Controller sum-of-residues register (for the address present in the LAR register)
SRM	SPM[2:3]	Displays the value of spare flip-flops (unused) in the Memory Controller logic
SRS	SRS[3:4]	Displays the value of the EVENT logic strobe save register
SSR	SSR[2:3]	Displays the value of the EVENT logic syllable save register
SSZ	SSZF[2:2]	Displays the value of the Family U source size register
STB	STB[2:3]	Displays the value of the Stack Controller stack register (shows where a READ-data word was placed in the stack)
STS	STS[3:4]	Displays the value of the MLIP status-save register
TA	TA[3:4]F	Displays the value of the Family A T-register
TB	TB[3:4]F	Displays the value of the Family B T-register
TC	TC[3:4]F	Displays the value of the Family C T-register
TD	TD[3:4]F	Displays the value of the Family D T-register
TE	TE[3:4]F	Displays the value of the Family E T-register
TOA	TOA[5:6]	Displays the value of the Transfer Controller Top-of-Aperture register
TOD	TD[35:36]	Displays the value of the Time-of-Day register

Table 4-6. Soft Display Register Names (Sheet 6 of 6)

Soft Display Register Name	Circuit Displayed	Display Meaning or Usage
TODC	TOD[3:4]	Displays the value of the low-order 4-bits of the Time-of-Day register
TOM	TOM[5:6]	Displays the value of the Transfer Controller Top-of-Mask register
TU	TU[8:4]F	Displays the value of the Family U T-register
TV	TV[2:3]	Displays the value of the Memory-tester test vector register
WCCF	WCCF[3:4]	Displays the value of the Memory Controller CLEAR/WRITE function control signals to local memory port adapters
WSTF	WST[3:4]	Displays the value of the Memory Controller WRITE function control signals to local memory port adapters
X	XR[50:51]	Displays the contents of the Top-of-Stack X register
Y	YR[50:51]	Displays the contents of the Top-of-Stack Y register
YRM	YR-[3:3]	Displays the value of the Arithmetic Controller 1-octade extension register
Z	ZR[50:51]	Displays the contents of the Top-of-Stack Z register

SYSTEM CONTROL COMMANDS

System Control Commands are used to initiate system control functions. A B 6900 system MDP cabinet contains manual switches that can be used to initiate system control functions. If a B 6900 system does not have an MDP cabinet, the corresponding Soft Display program Commands must be used to initiate these system control functions.



SEMANTICS

The semantics for System Control Commands are given in the following paragraphs.

<PULSE> COMMAND

A PULSE Command is used to specify the number of clock pulses to be issued to system logic circuits. If an optional <decimal number> is included in the Command, <decimal number> clock pulses are issued. If <decimal number> is not included a single clock pulse is issued. A PULSE Command implies that normally free-running system clock pulses are controlled. Before a PULSE Command is used, the control of System clock pulses is implemented by use of MAINT/EVENT Commands.

<STEP> COMMAND

A STEP Command is used to specify the number of steps (operations or functions) the CPU is to perform. If the <decimal number> option is included as part of the Command, <decimal number> steps are performed. If a <decimal number> is not included, a single step is performed. One STEP is counted each time the CPU Program Controller SECL signal goes to a TRUE level. The CPU Program Controller responds to MAINT/EVENT Conditional Halt logic and STEP Commands simultaneously. If a Conditional Halt occurs while a STEP <decimal number> Command is in process, the CPU does not halt; instead, the STEP Command causes steps beyond the Conditional Halt to be performed.

<ARCS> COMMAND

The ARCS (All Rows and Columns Signal) Command is used to cause the system logic circuits to be initialized. Most B 6900 circuit devices are initialized to the cleared (binary zero) state; however, some circuit devices are initialized to the SET (binary one) state. IODC bases are not initialized when an ARCS Command executes unless the MAINT/EVENT SAFE condition is RESET (see SAFE). If SAFE is SET and an ARCS Command executes, the MLIP logic HASL flip-flop is SET and the PSC register is initialized to a value of 9. If SAFE is RESET and an ARCS Command executes, the MLIP logic HASL flip-flop is RESET and the PSC register is cleared to binary zero.

<HALT> COMMAND

The HALT Command is used to cause the HALT and HALTED flip-flops in the CPU to be SET. When the HALTED flip-flop is SET, a STEP Command is required to resume system operations.

<STOP> COMMAND

The STOP Command is used to cause the STOP logic signal of the Host Control Bus (HCB) interface to be TRUE. When the STOP signal is TRUE the system status display of the CPU logic is not updated by the Maintenance Processor. This Command effectively disables the HCB interface between the Maintenance Processor and the CPU cabinet.

MAINTENANCE AND EVENT CONTROL COMMANDS

EVENT/MAINT Commands are used to SET/RESET control flip-flops for B 6900 system EVENT Mode operation and Maintenance Mode operation. The syntax for these Commands is presented in the <SET> and <RESET> COMMANDS subsection. This subsection describes system actions that result when these Commands are executed by the Soft Display program.

A STATUS Command (see FUNCTIONS COMMANDS) can be used to cause the current state of EVENT/MAINT control flip-flops to be displayed on the ODT screen.

<AAIF> COMMAND

This Maintenance Control Command invokes the Address Analyze feature of the Maintenance Processor (MP) logic. When invoked, Address Analyze monitors main-frame addresses (PANEL PAGE and BYTE) on the HCP interface bus (that connects the MP to the CPU). If a CPU adapter module is addressed on the HCP bus address lines, the DOUT (data out used to SET a CPU flip-flop) line is forced to zero. In this way, the Address Analyze feature prevents the MP from setting/resetting CPU adapter flip-flops.

<ALTF> COMMAND

This Maintenance Control Command invokes the Alternate Display control (see B 6900 MDP CABINET MAINTENANCE CONTROL PANEL). This Command is used only for factory maintenance operations.

<CHLT> COMMAND

This Maintenance Control Command invokes the Conditional Halt logic (see B 6900 SYSTEM CONTROL PANEL).

<CPTF> COMMAND

This Maintenance Control Command invokes the Comparator Display mode logic (see B 6900 MDP CABINET MAINTENANCE CONTROL PANEL). This Command is used only for factory maintenance operations.

<CSTP> COMMAND

This Maintenance Control Command invokes the Clock STOP logic (see B 6900 MAINTENANCE CONTROL PANEL).

<EVNT> COMMAND

This Maintenance Control Command invokes the Maintenance EVENT logic (see B 6900 MDP CABINET MAINTENANCE CONTROL PANEL).

<LOCL> COMMAND

This Maintenance Control Command invokes the CPU LOCAL/REMOTE logic (see B 6900 MAINTENANCE CONTROL PANEL).

This Command invokes a Soft Display program toggle control, which replaces a STEP sequence with a Soft Display program halt sequence. (See FUNCTIONS COMMANDS, <NOSTEP> COMMAND description.)

<OCTAL> COMMAND

The Display Control Command invokes octal display format for all registers, main memory, and CPU IC memory. Memory addresses display in hexadecimal regardless of whether octal format is invoked.

<SAFE> COMMAND

This Maintenance Control Command invokes a Protected CPU operating mode for operations of the Soft Display control program. SAFE mode is the default mode of the Soft Display program and must be RESET to operate in a CPU mode that is not protected. SAFE mode prevents the Soft Display program from performing Commands that destroy the operating system environment or require subsequent system initialization.

If SAFE is SET (TRUE), and the HLTD flip-flop is RESET, the following MAINT/EVENT Commands are not allowed to execute. Attempting to execute one of these Commands results in a Soft Display program error being detected.

1. SET EVNT
2. SET LOCL
3. SET CSTP

If SAFE and RUNI are SET and HLTD is RESET, the following Functions Commands and System Control Commands are not allowed to execute. Attempting to execute one of these Commands results in a Soft Display program error being detected.

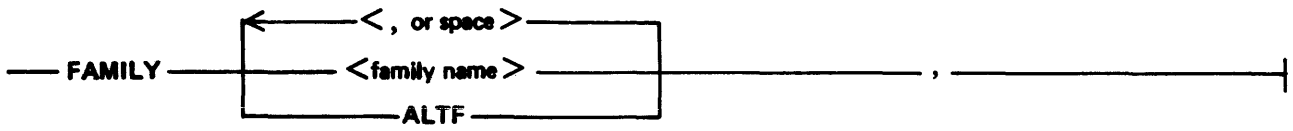
1. ARCS (System Control Command)
2. CAPTUR (Functions Command)
3. CLRIC (Functions Command)
4. CLRMM (Functions Command)
5. DUMP (Functions Command)
6. EXEC (Functions Command)
7. PROGRM (Functions Command)
8. RDIC (Functions Command)
9. RDMM (Functions Command)
10. RESTOR (Functions Command)
11. SMEAR (Functions Command)
12. WRIC (Functions Command)
13. WRMM (Functions Command)

<SECL> COMMAND

This Command invokes the Syllable Execute Complete Level (SECL) HALT logic of the CPU (see the B 6900 SYSTEM CONTROL PANEL).

FAMILIES CONTROL COMMANDS

Families Commands are Soft Display program Commands that cause B 6900 circuit status to be displayed on the ODT screen. The status displayed by use of a Family Command is presented in fixed format. Multiple family circuits status can be displayed by use of a single Family Command.



MV4820

SEMANTICS

The word FAMILY is required as the first word of a Families Command. Multiple <family name> and ALTF phrases may be used in a Families Command. If multiple phrases are included, they must be separated from each other by means of a blank space or a comma. Families Commands are terminated by a semicolon character.

The <family name> phrase defines a particular set of circuit status to be displayed. The format of a particular <family name> status display is fixed by the logic of the Soft Display program and cannot be varied by the program user. The following are the only Families Command <family name> phrases that can be used in a proper Command.

A	This <family name> causes the CPU Program Controller Family A logic status to be displayed on the ODT screen. Family A logic circuits control arithmetic operations in the CPU.
ARICON	This <family name> causes the status of the CPU Arithmetic Controller circuits to be formatted and displayed on the ODT screen.
B	This <family name> causes the CPU Program Controller Family B logic status to be displayed on the ODT screen. Family B logic circuits control logical, and field/bit manipulation operations in the CPU.
C	This <family name> causes the CPU Program Controller Family C logic status to be displayed on the ODT screen. Family C logic circuits control program subroutine and branching operations in the CPU.
CPU	This <family name> causes the operating system and addressing environment status of the current program segment to be displayed on the MDP screen.
D	This <family name> causes the CPU Program Controller Family D logic status to be displayed on the ODT screen. Family D logic circuits control program literal values, memory operations, and Top-of-Stack register operations in the CPU.
E	This <family name> causes the CPU Program Controller Family E logic status to be displayed on the ODT screen. Family E logic circuits control scaling and input data conversion operations in the CPU.

ERRORS	This <family name> causes most (not all) CPU error status flip-flops and registers to be formatted and displayed on the ODT screen.
EVENT	This <family name> causes the CPU EVENT logic status to be displayed on the ODT screen.
GLOBAL	This <family name> causes the Global memory port control logic status of the CPU to be displayed on the ODT screen.
INTCON	This <family name> causes the CPU Interrupt Controller logic status to be displayed on the ODT screen.
I/O	This <family name> causes the CPU MLIP control logic status to be formatted and displayed on the ODT screen.
MEMCON	This <family name> causes the CPU Memory Controller logic status to be displayed on the ODT screen.
MEMPRT	This <family name> causes the Memory Controller port control logic status to be displayed on the ODT screen.
MEMTST	This <family name> causes the Memory Tester control logic status to be displayed on the ODT screen.
MMOD	This <family name> causes the CPU micro module control logic status to be displayed on the ODT screen.
PROGCL	This <family name> causes the CPU Program Controller logic status to be displayed on the ODT screen.
U	This <family name> causes the CPU family U control logic (subfamilies F, G, and H) status to be displayed on the ODT screen.
UFAM	This <family name> causes the User Family set of logic signals and levels to be formatted and displayed on the ODT screen. The User Family consists of as many as 150 flip-flops and/or registers, that have been defined by Soft Display program Commands (see FUNCTIONS COMMANDS USERFAM, ADD, DEL, FAMILY, and INSERT).
XFERCL	This <family name> causes the CPU Transfer Controller logic status to be displayed on the ODT screen.

The <ALTF> option of a FAMILIES Command is used only in the factory, for system comparator station operations. Use of this <family name> causes the ALT DISPLAY control flip-flop to toggle (see B 6900 MDP CABINET MAINTENANCE CONTROL PANEL).

FUNCTIONS COMMANDS

FUNCTIONS Commands provide for the use of Soft Display macro-commands. A macro-command is a series of micro-commands, such as FAMILIES, EVENT/MAINT Commands and other macro-commands that are executed in a predetermined command sequence. Macro-commands add power to the Soft Display program because complex system operations can be performed by use of a single FUNCTIONS Command. A complex operation example is writing into system memory where both an Address-value and data must be present in the CPU logic circuits before the operation is initiated.

FUNCTIONS Commands include instructions used to control the ODT screen display during Soft Display program operations. FUNCTIONS Commands are also used to initialize and control the display of FAMILIES Command logic, to initiate Interrupt Controller memory-dump procedures, and to write user-devised machine language codes or user-defined data words into system memory.

The Soft Display program FUNCTIONS Commands are as follows:

<ADD> COMMAND

<BRIGHT> COMMAND

SEMANTICS

The BRIGHT FUNCTIONS Command causes the ODT screen to be brilliantly illuminated for non-ZERO register and flip-flop status displays. This Command enables the highlighting feature of the ODT video screen. If this feature is not enabled, all status displays are of the same intensity, regardless of whether the state of the device is HIGH (TRUE) or LOW (FALSE). When this feature is enabled, non-ZERO (TRUE) states are displayed brilliantly and ZERO (FALSE) states are displayed with normal intensity. The contrast between normal intensity and brilliant intensity makes it easier to distinguish the current state condition of a logic signal.

<CAPTUR> COMMAND

CAPTUR

MV4523

SEMANTICS

The CAPTUR FUNCTIONS Command causes the Maintenance Processor (MP) to capture the current CPU logic display state in the MP RAM memory. The CPU must be halted to capture its current state in MP RAM memory. The CAPTUR Command provides the first part of a method for interrupting the CPU to execute a Soft Display Command sequence. The second part of this method is provided by use of a RESTOR Command, which is defined later in this section.

CAUTION

Care must be taken when halting a CPU to execute Soft Display CAPTUR and RESTOR Commands. Halting the CPU by stopping the emission of CPU clock pulses may cause a Data Processor operator code to be captured in mid-sequence. Executing Soft Display STEP or PULSE Commands while an operator-code is captured results in stepping the CPU micro module address, causing a mismatch between the logical state and sequence count of the captured operator-code. This condition results in unpredictable CPU behavior upon resuming the execution of the captured CPU operator-code.

A CPU operation must not be restored and resumed unless the CPU is halted with the CPU Program Controller SECL signal at a TRUE level. This prevents unpredictable CPU behavior upon resuming the execution of a restored CPU function, regardless of the Soft Display Commands used while the CPU logic state was captured.

The current logical state of a CPU can be compared to a captured CPU logical state by means of a Soft Display FUNCTIONS DIFF Command.

<CLRIC> COMMAND

CLRIC

MV4524

SEMANTICS

The CLRIC is a macro-command that writes zeroes in all Data Processor IC memory address registers. The process flow for this Command is as follows:

a. CAPTUR the CPU logical state

b. ARCS (clear) the CPU

c. Write zeroes in each IC memory address Display register D0 through D31

d. ARCS (clear) the CPU

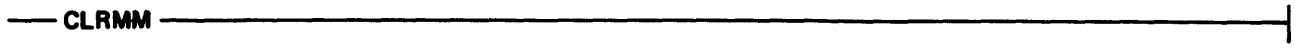
e. Write zeroes in each IC memory address Base and Index register, by means of the control signal CRIC logic

f. ARCS (clear) the CPU

g. RESTOR the CPU logical state

The CRRIC Command uses CPU EVENT logic signals EV1, EV6, EV8, and EV11 to control the various processes of this macro-command.

<CLRMM> COMMAND



SEMANTICS

The CLRMM Command is a macro-command that writes zeroes into system memory word addresses. If the Soft Display program is operating in SAFE mode (the default mode), the first main memory address written is word 1FF hex (511 decimal). If the Soft Display program is not operating in SAFE mode, the first main memory address written is word ZERO. The first 511 memory word addresses are not written when SAFE mode is in effect because these word addresses contain code needed for system HALT/LOAD operations.

A CLRMM Command writes all zeroes in successive ascending memory addresses, until an invalid memory address interrupt occurs. An invalid memory address occurs for an interrupt condition, or after the last memory address is written. The CLRMM Command terminates by displaying the last memory address value on the ODT screen.

The process flow for this macro-command is as follows:

a. CAPTUR the current CPU state

b. If SAFE mode SET memory address to 1FF hex; otherwise, RESET memory address to zero

c. RESET Top-of-Stack X register (and TAG) to all zeroes

d. Use CPU EVENT logic to detect an invalid memory address condition (EV19, and EV20)

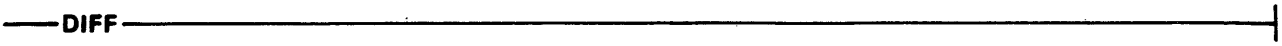
e. Use CPU Memory Tester logic (TVN = 3) to overwrite data from the X register into successive memory addresses

f. Upon detection of an invalid memory address, wait one second, and then display the last address value on the ODT screen.

g. RESTOR the prior CPU state

<DEL> COMMAND

<DIFF> COMMAND

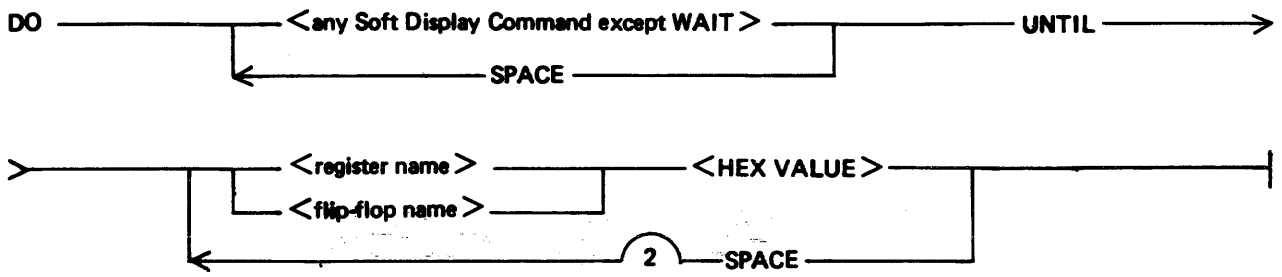


MV4527

SEMANTICS

The DIFF Command compares the current CPU logical state to a prior CPU logical state. The prior logical state of the CPU was saved in the Maintenance Processor RAM memory by executing a Soft Display CAPTUR Command before executing the DIFF Command. The names of all CPU logic signals that are not equal are displayed on the ODT screen.

<DO-UNTIL> COMMAND



MV4528

SEMANTICS

The DO-UNTIL FUNCTIONS Command allows a Soft Display program user to specify a number of Soft Display Commands that are to be repeated until a specified logical condition is satisfied. A DO-UNTIL Command must be entirely present on line-1 and/or line-2 of the ODT screen; that is, no part of this command syntax may be present on line-4 or line-5 (the Command Buffer).

Soft Display Commands (except WAIT FUNCTIONS Commands) are listed between the required DO and UNTIL words of this Command syntax. The Commands listed are separated by spaces, and the entire Command group is essentially a user-devised Soft Display macro-command. The Soft Display program executes the command parts of this macro in the order of occurrence, from left-to-right. Each Command in the macro is executed at least one time. Each time the entire macro is completed, the Soft Display program evaluates the conditions for terminating the macro-command. If no termination condition is TRUE, the macro-command is repeated from the beginning. If a termination condition is TRUE, the macro-command is terminated.

The Soft Display program user must specify 1, 2, or 3 conditions for terminating execution of this macro-command. A condition is a particular value in a Soft Display register, or it is a flip-flop being in a particular state (1 or 0). A condition is specified by means of a <register name> <hex value> or <flip-flop name > <hex value> phrase, following the UNTIL word entry. If 2 or 3 conditions are specified, they are separated by commas. The railroad diagram shows that the line containing the comma may be traversed only 2 times. Thus, a maximum of three termination conditions may be specified.

<DUMP> COMMAND

DUMP

MV4529

SEMANTICS

The DUMP FUNCTIONS Command forces a software memory-dump procedure to be executed as a Soft Display program sequence. This Command assumes that the software operating system was initialized before the start of Soft Display program operations, and that system main-memory has not been cleared by a Soft Display program Command. If any part of this assumption is false, execution of the DUMP Command results in a Soft Display program error condition and no memory-dump operation is performed.

The DUMP Command is a macro-command. The process flow of this macro is as follows:

a. STOP the CPU

b. Save the values in the CPU Top-of-Stack A and B registers, and the states of AROF and BROF flip-flops

c. Save the value in the CPU Lexacographical Level register

d. CAPTUR the logical state of the CPU

e. General Clear (ARCS) the CPU circuits

f. SET a word of program-code in the CPU P register, as follows: "3 AE4014B0B0AB"

g. Restore the saved values of the A register, B register, Lexacographical Level register, and AROF and BROF flip-flop states

h. STEP the CPU

When the CPU is stepped (see h above), a memory-dump procedure located in memory-address D0 +14 (referenced in the program-code word) is entered and executed. Logical conditions of the CPU that might prevent the memory-dump procedure operation (such as a CPU SUPERHALT, which stops system clocks distribution) are removed by General Clearing the CPU. The CAPTUR function saves the CPU state conditions present at the beginning of the DUMP Command. Thus, CPU state conditions at the time of the memory-dump are still available after the memory-dump, they are saved in the Maintenance Processor RAM memory.

<END> COMMAND

MV4630

SEMANTICS

The END FUNCTIONS Command is used to terminate operation of the Soft Display program, and return control of B 6900 system operations to the Maintenance Processor Executive program. If a maintenance test routine was in process when the Soft Display program was initiated, the test routine is resumed at the point where it was interrupted.

NOTE

Care must be exercised when a test routine is resumed by executing a Soft Display END Command. If the Soft Display program executed any of the following Commands, the test routine may no longer be valid.

1. CLRMM
2. WRMM
3. CLRIC
4. WRIC
5. PULSE
6. STEP
7. ARCS

Any macro-command that exercises one of these Commands may also cause a resumed test routine to be invalid.

<EXEC> COMMAND

MV4631

SEMANTICS

The EXEC FUNCTIONS Command initiates the CPU to execute a program-code sequence in system memory. The data value parts of the EXEC Command syntax are hexadecimal values used to establish the operating system addressing environment.

The PBR data part of this Command establishes the initial value of the CPU IC memory Program Base Register (PBR). The PBR hexadecimal value must not exceed 5 hexadecimal characters in length. If PBR is less than 5 hex characters in length, the value is placed in the PBR register, right-justified, and unspecified high-order bits are filled with leading zeroes.

The Program Syllable Register (PSR) and Program Index Register (PIR) part of this Command establishes the initial values of the CPU IC memory Program Index Register, and the CPU hardware Program Syllable Register. These two values are concatenated to form a single Command syntax 4-character hexadecimal value part. The 3 high-order bits of the hexadecimal value are the initial value for the PSR register, and the low-order 9-bits are the initial value for the IC memory PIR register. The PIR bits are placed in the PIR IC memory register, right-justified, and the unspecified high-order bits are filled with leading zeroes.

The Lexacographical Level (Lex Level) part of this Command establishes the initial value of the Lexacographical Level register. The Lex Level part contains 2 hexadecimal characters, and the least-significant 5-bits of the 2 characters are used to fill the Lex Level register.

The EXEC Command is a macro-command that performs the following listed functions.

a. CAPTUR the CPU logic signal state

b. General clear (ARCS) the CPU logic

c. Place the <Lex Level> part value in the Lexacographical Level hardware register of the CPU

d. SET the following hexadecimal program-code word in the CPU P register; "3 A2****DFDFDF", where "****" is the <PSR and PIR> part value

e. Place the <PBR> part value in the Program Base Register of the CPU

f. Initialize the EVENT logic to execute one pass through the program-code or stop on a SECL signal (and INFF/), Syllable Dependent Interrupt, or Alarm interrupt

g. Prepare the CPU to accept a STEP Command signal, but emit the STEP signal

NOTE

The EXEC Command performs a BRANCH UNCONDITIONAL operator to begin executing the program code. The EXEC Command establishes all of the prerequisite functions to execute the BRANCH UNCONDITIONAL operator, but does not actually perform the branching operation. A STEP Command subsequent to the EXEC Command is required to execute the program code.

<FAMILY> COMMAND

The FAMILY Command is defined in previous paragraphs entitled Families Control Commands, Section 4, of this manual.

<HELP> COMMAND

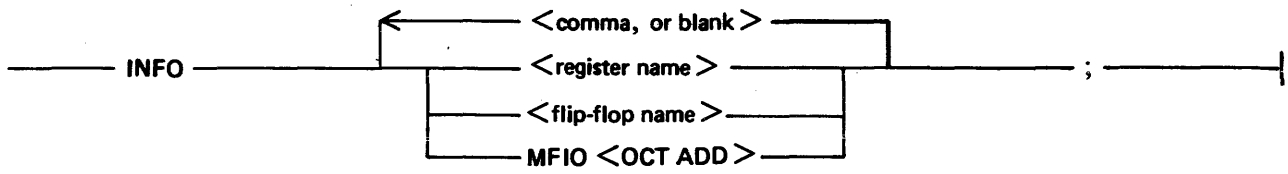
HELP

MV4532

SEMANTICS

This FUNCTIONS Command causes the list of Commands and FAMILY names to be displayed on the ODT screen.

<INFO> COMMAND



MV4533

SEMANTICS

This FUNCTIONS Command provides a method of displaying data about a <register name>, <flip-flop>, or MFIO <OCT ADD> on the ODT screen. Multiple <register name>, <flip-flop name> and MFIO <OCT ADD> items may be included in a single <INFO> Command; however, a comma or blank character must separate an item from other items in the same Command. A semicolon character must be used to terminate an <INFO> Command.

The data displayed on the ODT screen about each register, flip-flop, or MFIO octal address in an INFO Command is as follows:

a. The type (either register or flip-flop)

b. The current value/state of the register, or flip-flop

c. The number of bits (in <register name>, or the number of flip-flops present at MFIO <OCT ADD>

The MFIO <OCT ADD> option of an INFO Command allows the Soft Display program user to select a specified CPU card package to be displayed on the ODT screen. A maximum of 8 flip-flops of a selected CPU card-package are displayed on the ODT screen. This Command can be used to display the status of CPU flip-flops that do not have Soft Display program names associated with their physical location within the CPU logic.

<INSERT> COMMAND

MV4534

<NOSTEP> COMMAND

MV4535

SEMANTICS

A NOSTEP Command controls the execution of a consequent Soft Display program macro-command. A NOSTEP Command causes a NOSTEP POINT to be present in the sequence of the macro-command, where a STEP sequence normally occurs. When the macro-command is executed, a "WAITING FOR STEP" output message is displayed on the ODT screen, and the Soft Display program stops. The macro-command waits at its NOSTEP POINT until a STEP Command is entered at the ODT keyboard.

When a SET NOSTEP Command is executed, program toggle "NOSTEP" is SET. When the macro-command containing the NOSTEP POINT is executed, the "NOSTEP" toggle is RESET. Commands following the macro-command in the command buffer are not executed. A STEP Command ODT input at a NOSTEP POINT causes the waiting macro-command to be executed, after which the Soft Display program returns to its initialization point and waits for new commands to be inserted into the command buffer.

The macro-commands that respond to the use of a SET NOSTEP Command are as follows:

1. CLRIC
2. CLRMM
3. DUMP
4. EXEC
5. RDHDP
6. RDIC
7. RDMM
8. WRIC
9. WRMM

<NZDATA> COMMAND

NZDATA

MV4536

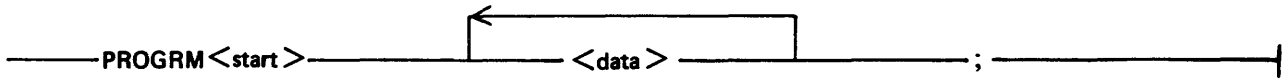
SEMANTICS

The NZDATA FUNCTIONS Command causes the Soft Display program to scan all CPU register and flip-flop display status data. All registers that contain non-ZERO data (in the TAG-field or data fields) are listed on the ODT screen. All flip-flops that are SET (TRUE) are also listed on the ODT screen.

NOTE

There are more than 1800 registers and flip-flops in a B 6900 CPU cabinet to be scanned for listing on the ODT screen by this Command. Therefore, it takes a significant time interval for this Command to complete its sequences.

<PROGRM> COMMAND



MV4537

SEMANTICS

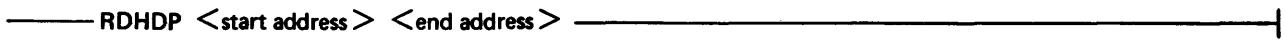
This FUNCTIONS Command writes <data> words of program code into memory beginning at memory address <start>. Each program-code wore written into memory contains a Command-supplied TAG-Field = 3 value. The use of this Command allows a Soft Display program user to write a machine-language program in memory. A subsequent EXEC Command is used to initiate execution of the user-supplied program-code words in memory.

The <start> beginning memory address is a 1-to-5 character hexadecimal number, that defines the absolute memory address of the first word of program code. This hexadecimal address value is right-justified. If it contains fewer than 5 hex characters, unspecified high-order memory address bits are filled with leading zeroes.

The <data> field is a string of hexadecimal program code. The PROGRM Command automatically segments the hex string into program-code words of 12 hexadecimal characters (6 program-code syllables), and inserts a TAG-field value of 3, hexadecimal. The address is incremented +1 each time a program-code word is written into memory, Thus, a PROGRM Command writes multiple successive code words into memory, until all program-code syllables are present in a contiguous memory area.

A PROGRM Command is a macro-command. The sequences of this macro save the state of the CPU logic as the beginning sequence of the Command, and restore the CPU logical condition as the terminating sequence of the Command.

<RDHDP> COMMAND



MV4538

SEMANTICS

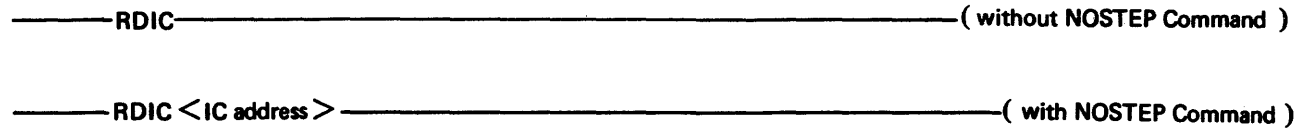
The RDHDP FUNCTIONS Command is used to cause the contents of the MLIP RAM memory to be displayed on the ODT screen. This Command can cause a maximum of 21 words of MLIP RAM memory data to be read for display on the ODT screen. The first RAM memory address to be read by a RDHDP Command is specified by the <start address> part. The final RAM memory address to be read is specified by the <end address> part. If the difference between the start address and the end address is greater than 21 decimal, only the 21 addresses that begin in <start address> are read and displayed. If <end address> is less than <start address>, a Soft Display program error is detected, and no RAM addresses are read or displayed. If <start address> and <end address> are equal, 1 RAM address is read and displayed.

The B 6900 MLIP RAM memory contains 3FF hexadecimal (1024 decimal) addresses. The RDHDP Command utilizes the MLIP micro-stack pointer for addressing RAM, thus, all MLIP RAM addresses can be accessed. The MLIP RAM data is read into the MLIP R1 register; and from the R1 register status display update, it is displayed on the ODT screen.

If either the <start address> or <end address> values in a RDHDP Command are greater than 3FF hexadecimal, a Soft Display program error is detected and the RDHDP Command is not executed.

<RDIC> COMMAND

There are 2 different formats for the RDIC Soft Display Command. One format is used when a NOSTEP Command precedes the RDIC Command in the command buffer. The second format is used without a preceding NOSTEP Command in the command buffer. The syntax for both are given, and then the SEMANTIC discussion defines the differences between the 2 formats.



MV4539

SEMANTICS

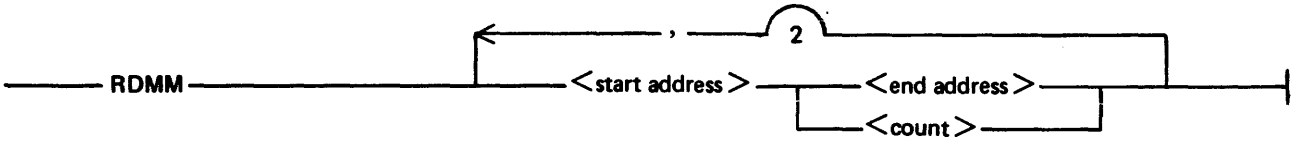
The RDIC FUNCTIONS Command causes the contents of the CPU IC memory address registers to be displayed on the ODT screen. The display on the ODT screen contains the IC memory address register mnemonic, the hexadecimal address of the IC memory register, the address value detected while the register is being read.

If the RDIC Command is used with the NOSTEP toggle RESET, a beginning IC memory address is not used. This RDIC Command syntax format causes all 48 IC memory address registers to be read and displayed. The sequences of the RDIC macro-command read the 8 INDEX IC memory address registers, the 8 BASE IC memory address registers, and the 32 DISPLAY IC memory address registers in hexadecimal register address order.

If the RDIC is used with the NOSTEP toggle SET, an <IC address> entry is required by the RDIC Command syntax. The <IC address> contains a 2-character hexadecimal value which defines the register address of the first IC memory address register to be read and displayed. All IC memory address registers with a register address value higher than <IC address> are also read and displayed.

An error flag is indicated by an "*ERR*" message, which is displayed adjacent to the corresponding IC memory address register data on the ODT screen. This error flag is present if an IC memory address register has not been written into since system power was applied to the CPU cabinet circuits. It is also displayed if a read data error or residue error is detected during the read sequence of the RDIC Command.

<RDMM> COMMAND



MV4540

SEMANTICS

The RDMM FUNCTIONS Command causes the contents of system memory to be read and displayed on the ODT screen. An RDMM Command requires at least one <start address> and <end address> or <count> address-range. There may be 3 address ranges present in an RDMM Command, with ranges separated from each other by commas. An RDMM Command can cause up to 63 memory words, from 1 to 3 different address-ranges in memory, to be read and displayed on the ODT screen.

If a single address-range is present in an RDMM Command, a maximum of 21 memory words and their TAG-Fields are read and displayed on the ODT screen, in a single column ODT display format.

If 2 address ranges are present in an RDMM Command, the first range can display from 1-to-21 memory address words and their TAG-Fields, in the left-hand column of the ODT screen display. The second range can display up to 42 memory address words and their TAG-Fields in the middle and right-hand columns of the ODT screen display.

If 3 address ranges are present in an RDMM Command, each range can display up to 21 memory words and their TAG-Fields, in 3 columns of display on the ODT screen. The left most column displays the words from the first range; the center column displays the words from the second range; and the left most column displays the words from the last range.

An address range consists of a <start address> and <end address> pair, or a <start address> and <count> pair. A <start address> field contains 1-to-5 hexadecimal characters, which define the first absolute memory address word to be displayed on the ODT screen. The value of <start address> is placed in the CPU memory address register, right justified. If there are fewer than 20-bits in a <start address> value, the unspecified high-order address register bits are filled with leading zeroes. If there are more than 20-bits present in <start address> a Soft Display program error is detected, and the RDMM Command and any subsequent Commands in the command buffer are not executed.

An <end address> is similar to a <start address>. It contains 1-to-5 hexadecimal characters which are handled in the manner described in the preceding paragraph for a <start address> value. The value of an <end address>, relative to its paired <start address>, determines the number of memory words displayed on the ODT screen. If <end address> is equal to <start address>, 1 memory word is displayed. If <end address> is greater than <start address>, then as many as 21 memory words are displayed for address-range 1 or 3, and as many as 42 memory words are displayed for address-range 2.

A <count> value contains 2 hexadecimal characters, and determines how many memory words, including the paired <start address> word, are to be displayed on the ODT screen. A <count> value must be less than the value of its paired <start address>; otherwise, the Soft Display program treats the <count> value as an <end address>. If a <start address> value is less than 16 hexadecimal (22 decimal), an <end address> value is used instead of a <count> value.

The sequences of a RDMM macro-command, which save the status of the CPU logic at the start of the RDMM Command, also restore the CPU to the saved status condition at the end of the Command flow.

<RESTOR> COMMAND

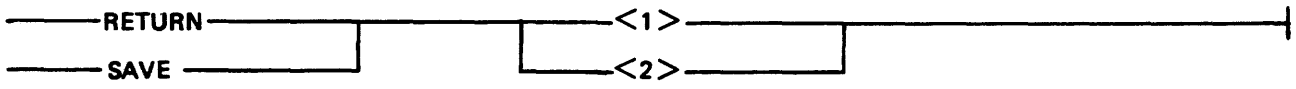
RESTOR

MV4541

SEMANTICS

The RESTOR FUNCTIONS Command is used to restore a CPU logical state condition that was captured before restoration. A prior CPU state condition is present in the Maintenance Processor RAM memory, and a RESTOR Command causes the CPU to assume the same state as that in MP RAM memory. A CPU operation cannot be resumed by means of a RESTOR Command if CPU clock pulses are emitted in the CPU while a CPU logical state condition is saved in MP RAM memory. A clock pulse in the CPU causes the micro module address count logic to be incremented, thus a saved CPU logical state condition is no longer synchronized with the current CPU micro module address.

<RETURN> and <SAVE> COMMANDS



MV4542

SEMANTICS

SAVE/RETURN FUNCTIONS Commands are used to preserve the contents of the Soft Display program command buffer, and to recall the preserved contents to the command buffer for subsequent command executions.

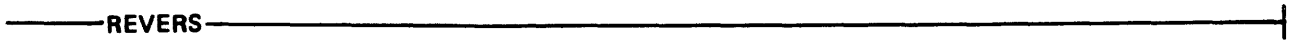
A SAVE <1> Command causes line-1 of the ODT command buffer to be copied on line-4 of the ODT screen. A SAVE <2> Command causes line-2 of the ODT command buffer to be copied on line-5 of the ODT screen.

A RETURN <1> Command causes line-4 of the ODT screen to be copied on line-1 of the ODT screen, which is the top-line of the command buffer. A RETURN <2> Command causes line-5 of the ODT screen to be copied on line-2 of the ODT screen, which is the bottom line of the command buffer.

SAVE/RETURN Commands may be executed when they are located on either line-1 or line-2 of the ODT screen.

Saved buffers of Commands can be reused in either of 2 ways. The ODT cursor can be positioned at line-4 and the XMIT key depressed to execute Commands in the saved buffers. The ODT line-delete feature can be used to rotate lines 4 and 5 to the line 1 and 2 positions, and then depressing the XMIT key from the homed position.

<REVERS> COMMAND

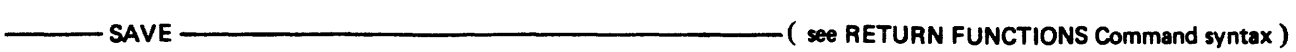


MV4543

SEMANTICS

The REVERS FUNCTIONS Command causes the ODT screen video display mode to go to the default video mode setting. The default video mode is typical display mode, where the ODT screen brilliant display feature is not used. In this default display mode all flip-flop states and all register values are displayed with normal video intensity, regardless of the non-ZERO or ZERO state.

<SAVE> COMMAND



MV4544

<SMEAR> COMMAND

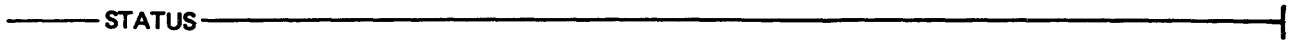
MV4545

SEMANTICS

The SMEAR FUNCTIONS Command is a macro-command which utilizes the CPU Memory Tester logic to smear a user-supplied data word in memory. The values of the <tag> and <data> items define the user-supplied data word. The value of the <start> and <bypass> items define the memory address-range into which the user-supplied data word is written. The Command cycles through all inclusive memory addresses until an invalid memory address interrupt is sensed, or until the Memory Tester logic completes the test. Upon completing, the SMEAR Command causes the last memory address written to be displayed on the ODT screen.

This macro-command saves the current logical state of the CPU at the beginning of its operation sequences, and restores the CPU state as the last sequence of the Command.

<STATUS> COMMAND



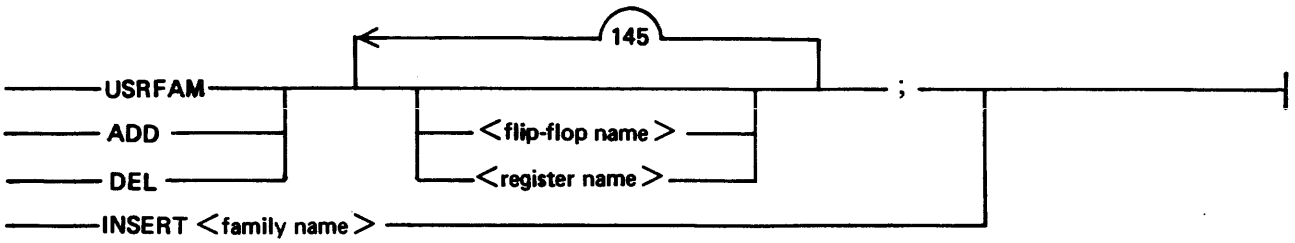
MV4546

SEMANTICS

The STATUS FUNCTIONS Command causes the state of 10 system control circuits to be displayed on the ODT screen. The 10 system control circuits are as follows:

1. AAIF – Maintenance Processor Address Analyze Feature
2. ALTF – Alternate/Normal System Display Select
3. CHLT – Conditional Halt Status
4. CPTF – Comparator Select Status
5. CSTP – Clock Stop Logic Status
6. EVNT – EVENT Mode Flip-flop State
7. LOCL – CPU Local/Remote Status
8. OCTAL – Octal Data Dispiay Select Status
9. SAFE – Safe Mode Status
10. SECL – Syllable Execute Complete Level Status

<USERFAM> COMMAND



SEMANTICS

The USRFAM FUNCTIONS Commands provide a method for creating and updating a user-defined display Family. System state conditions are parts of user-defined Families, and are displayed on the ODT screen when a UFAM FAMILIES Command is executed.

A USRFAM Command causes a user family array to be created and to be initialized. The user family array is a buffer area that contains space for up to 145 flip-flop and register names. When a USRFAM Command is executed the array is created, and any <flip-flop name> or <register name> items specified are placed in the array, in the order of their appearance in the USRFAM Command syntax. If there are no <flip-flop name> or <register name> items specified by a USRFAM Command syntax, the array is initialized to a cleared state.

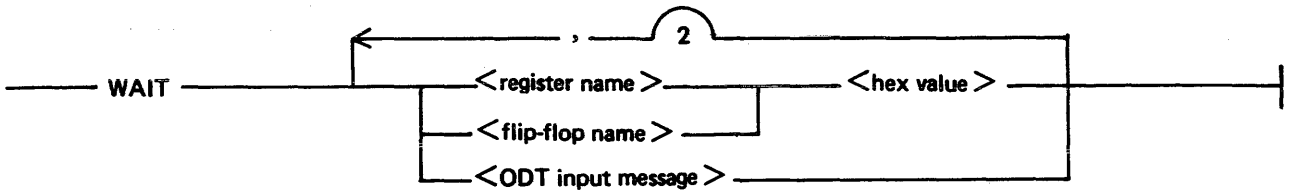
The ADD Command is used to put additional <flip-flop name> and <register name> items in a previously initialized array. Items added to the array are placed at the end of the array list, in the order of appearance in the ADD Command syntax. If an ADD Command is used but the user family array was never initialized by a USRFAM Command, a Soft Display program error is detected and the ADD Command and all subsequent Commands in the command buffer will not be executed.

The DEL (Delete) Command is used to remove <flip-flop name> and <register name> items from the user family array. If a DEL Command is used but the user family array was never initialized by a USRFAM Command, a Soft Display program error is detected and the DEL Command and all subsequent Commands in the command buffer are not executed.

An INSERT <family name> Command is used to add a Family display to the user defined family array list. If an INSERT Command is used but the user family array was never initialized by a USRFAM Command, a Soft Display program error is detected and the INSERT Command and all subsequent Commands in the command buffer will not be executed.

If a <flip-flop name> or a <register name> is not a valid Soft Display program recognized name, an error condition is detected. Moreover, the Command syntax that contains the name and all subsequent Commands in the command buffer are not executed.

<WAIT> COMMAND



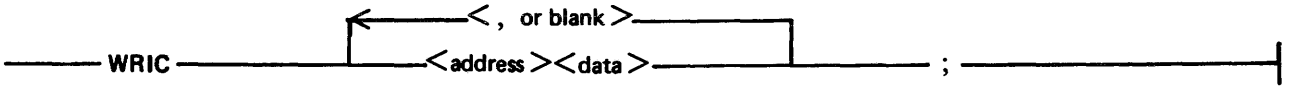
MV4548

SEMANTICS

A WAIT FUNCTIONS Command provides a method for temporarily stopping the execution of Soft Display Commands until a preselected condition is present. From 1 to 3 conditions are specified as the basis for resuming the execution of subsequent Commands in the command buffer. If more than 1 condition is listed, commas are used to separate the conditions.

A condition consists of the name of a logic circuit and a selected hexadecimal value for the selected circuit, or of an ODT input message. If a selected condition is a <register name>, the condition is present when the current value of <register name> exactly matches the <hex value> specified for that condition. When a <flip-flop name> is used as a condition, the <hex value> is <1> for the TRUE condition and <0> for the FALSE condition. If <ODT input message> is used as a condition, depressing the keyboard XMIT pushbutton causes the condition to be present, and any data transmitted by depressing the XMIT pushbutton is discarded.

<WRIC> COMMAND

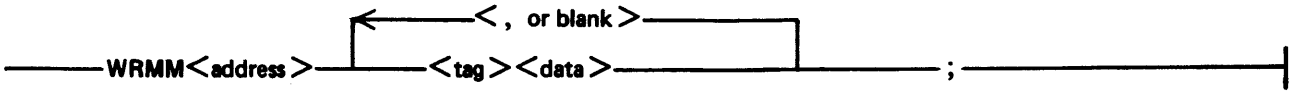


MV4549

SEMANTICS

The WRIC FUNCTIONS Command is used to establish the value of a CPU IC memory address register. Multiple IC memory address register values can be established by use of a single WRIC Command. The <address> item identifies the particular IC memory address register into which the corresponding <data> item is to be written. Both the <address> and <data> items are 5-character hexadecimal values.

<WRMM> COMMAND

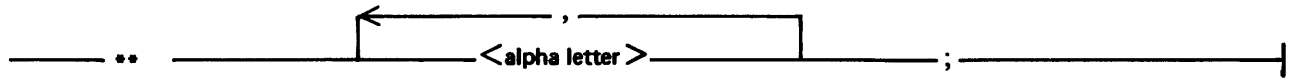


MV4550

SEMANTICS

The WRMM FUNCTIONS Command is used to write user-supplied memory words into successive main memory addresses. The <address> item specifies the beginning main memory address into which the first word of data is written. Each memory word contains a 1-character hexadecimal <tag> field value, followed by a blank space, followed by a 12-character hexadecimal <data> field value. Multiple, successively addressed memory words may be written into main memory by use of a single WRMM Command. Memory word tag and data-field pairs are separated from other word tag and data-field pairs by commas or blank characters.

<**> COMMAND



MV4551

SEMANTICS

An "**" FUNCTIONS Command causes the state of all flip-flops and registers whose names begin with <alpha letter> to be displayed on the ODT screen. Multiple <alpha letter> items may be used in a single "**" Command.

<--> and <++> COMMANDS

MV4567

SEMANTICS

The "--" FUNCTIONS Command causes the CPU arithmetic logic to perform a subtraction function, and displays the difference value on the ODT screen. The <hex value 2> is subtracted from <hex value 1>.

The ODT displays "<hex value 2> -- <hex value 1> = difference".

The "++" FUNCTIONS Command causes the CPU arithmetic logic to perform an addition function, and displays the sum value on the ODT screen.

The ODT displays "<hex value 1> ++ <hex value 2> = sum".

SECTION 5

SYSTEM CONCEPT

GENERAL

The B 6900 system consists of a central processing unit, a central power cabinet, a maintenance display processor cabinet, Input/Output Data Communications (IODC) cabinets and the associated peripheral equipment for input/output. This section generally defines the overall system hardware operation.

The central processing unit (CPU) is the heart of system operations in the B 6900 system; therefore, while other units of the system are discussed in this section, the main thrust is to describe the units that are parts of the CPU cabinet. The three main parts of the CPU cabinet are as follows:

a. The data processor (DP)

b. The Message Level Interface Processor (MLIP)

c. The memory control (MC)

DATA PROCESSOR

The data processor part of the CPU produces the objective results of a program by performing the necessary arithmetic and logical functions of the program flow.

The data processor contains two major divisions: the functional resources and operator algorithms (Figure 5-1). The functional resources are referred to as the "hardcore" of the processor.

The functional resources are the event logic, the micro-program module, the top of stack registers, the address adder, the MLIP, and six controllers. The operator algorithms are a group of six families of operators. The operator algorithms provide the logic required to control the functional flow of the program.

OPERATOR FAMILIES

The operator families and functional controllers are linked by 11 busses (bus Z1 through Z6, and Z8 through Z12). These busses provide for data movement and signal routing within the processor (see Figure 5-2).

A bus is a group of wires used to transmit signals from one place to another. The busses within the transfer controller are etched on a single card connecting the same bit of all "hard registers" together; that is, bit 1 of registers A, B, C, X, Y and Z are all on the same physical card.

The operators are grouped into six groups called the operator families (Figure 5-1). The grouping of related operators into families minimizes the logic required in the processor. The six families of operators with a brief purpose for each are:

a. Family A OPS (Arithmetic Operators).

b. Family B OPS (Logical Operators).

c. Family C OPS (Subroutine Operators).

d. Family D OPS (B 6900 Word Oriented Operators).

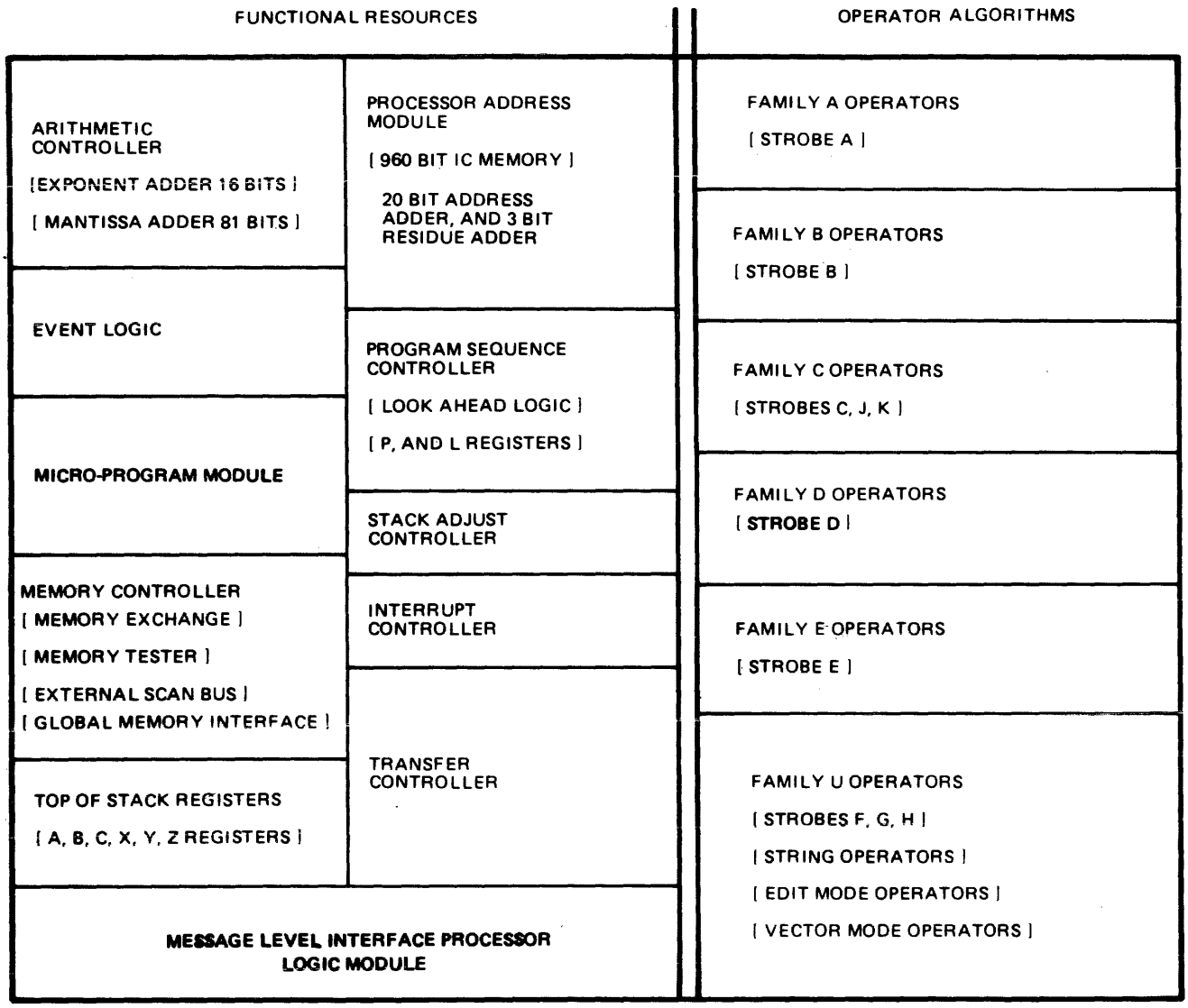


Figure 5-1. B 6900 CPU Organization

e. Family E OPS (Scaling Operators).

f. Families F, G, H OPS (String Operators).

PROGRAM CONTROLLER

The program controller (see Figure 5-2) controls the program flow in the data processor. The program controller determines when the P register contains machine language operators to be executed, which syllable of code is to be executed next, when to replace the contents of the P register and L register, and the source location of the data used to replace the contents of the P register and L register. The P register is considered to contain valid program code only if the Program Register Occupied Flip-Flop (PROF) is set.

The Program Syllable Register (PSR) serves as a pointer to the next syllable to be executed from the P register.

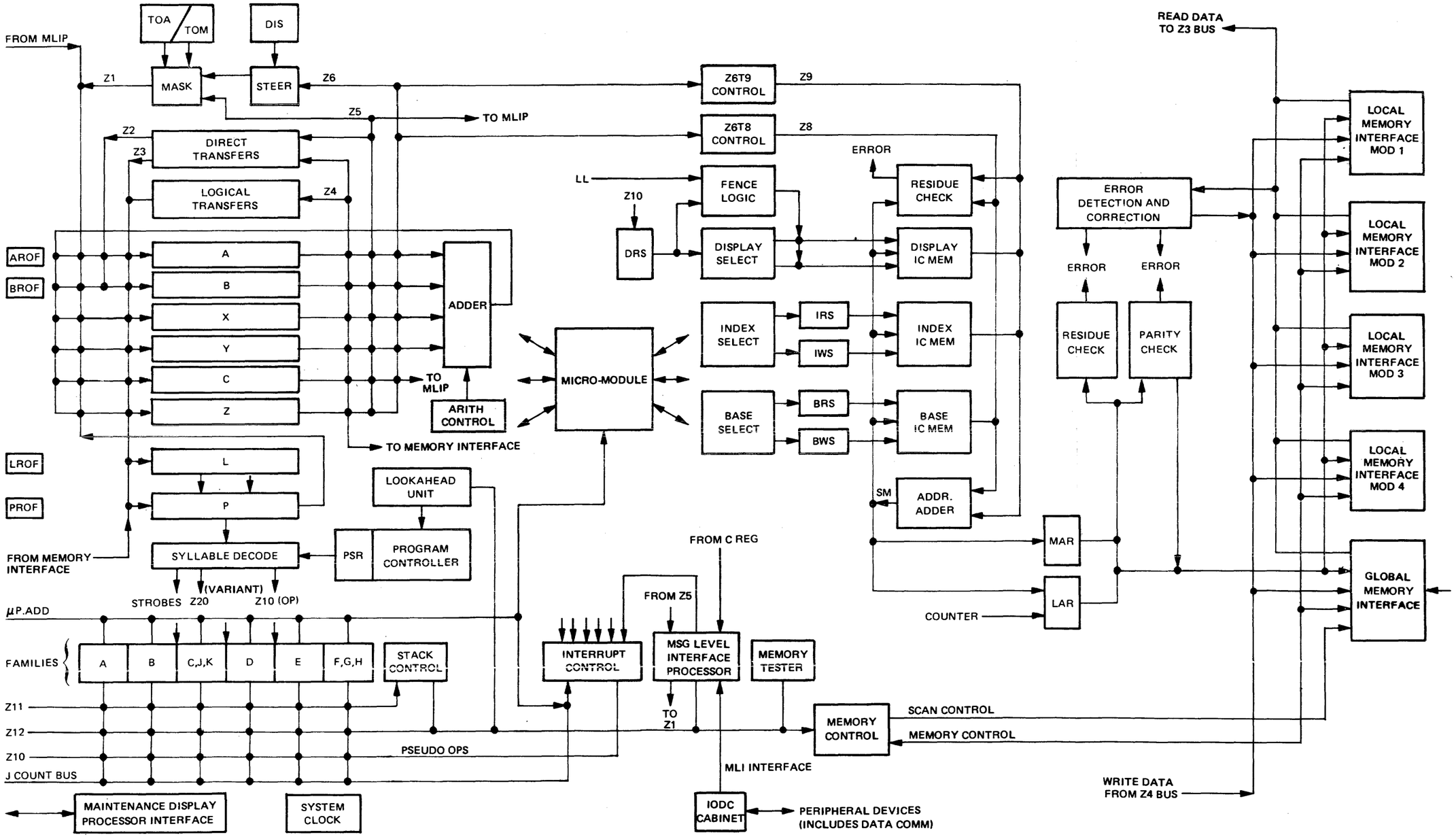


Figure 5-2. B 6900 CPU Block Diagram

Look Ahead Logic

A look ahead function is implemented by provision of the L register and the associated L Register Occupied Flip-flop (LROF). The function of the look ahead logic is to overlap as far as possible the fetching of code from main memory. In look ahead mode, L acts as a buffer against the P register, such that code is executed from P while L gets the next code word. Code addresses are initially formed by adding the value of the Program Base Register (PBR) to the value of the Program Index Register (PIR). Code addresses are maintained in the look ahead logic in the Look Ahead Address Register (LAR).

In certain modes, the normal sequential code execution, as affected by the look ahead logic, is undesirable and therefore inhibited. Such cases are branch instructions, subroutine entries and exits (or returns), and table edit mode operations. In the first two cases, new values of PBR and PIR are presented to the program controller, and are used as described. In table edit mode, look ahead logic is totally inhibited, and the program controller uses the Table Base Register (TBR) and the Table Index Register (TIR) to form the table mode edit operator code address. Only the P register is used to contain edit mode table operator code (and not the L register). In table edit mode operations, the TIR address register is updated by the program controller, as required.

Integrated Circuit (IC) Memory

The B 6900 system data processor maintains the procedure addresses of the program currently being executed in the data processor. These procedure addresses are maintained in a group of address registers commonly identified as IC memory address registers (see Figure 5-2). The IC memory address registers are classified as display address, base address, and index registers.

There are 32 display address registers (labeled D0 through D 31) in the data processor. A display register number corresponds to a lexicographical programming level, and locates the absolute local memory base address of the process stack (the MSCW of the stack) for all current programming levels. The maximum number of programming levels (lexicographical levels) in a procedure is fixed by the number of display address registers available in the data processor. The number of programming levels in a procedure is limited to 30, because programming level zero is required for the MCP, and programming level one is required for the segment descriptor index. The bottom of a stack is identified by the address located in the BOSR register, which was identified earlier in this manual. The top of a stack is identified by the address located in the S register, which was also identified earlier in this manual (refer to Section 3).

The following eight base address registers are in the data processor:

Base Register Number	Base Register Name	Register Usage
0	PBR	The base address of the program code segment.
1	SBR	The base address of string source data.
2	DBR	The base address of string destination data.

Base Register Number	Base Register Name	Register Usage
3	TBR/BUF2	The base address of table program code, or alternatively a temporary buffer for storing an address value.
4	S	The address of the top word in the current stack.
5	SNR	The stack number register. The stack number is used to contain a vector value for locating the current stack descriptor. The vector value is an index on the address of the stack vector descriptor for locating the stack descriptor.
6	PDR	The program dictionary register. This register is used to contain the address of the base of the current program code segment descriptor in memory.
7	TEMP	The temporary register. This register is a general purpose register used to store addresses temporarily

The following eight index address registers are in the data processor:

Index Register Number	Index Register Name	Register Usage
0	PIR	The program index register. The program index value is an index on the base address contained in the PBR register. The sum of PBR and PIR is the absolute address of the word of program code that is presently in the P register.
1	SIR	The source index register. The source index value is an index on the base address that is contained in the SBR register. The sum of SBR plus SIR defines the address of a word of source data for string operations.
2	DIR	The destination index register. The destination index value is an index on the base destination register. The sum of DBR plus DIR defines the address of a word of destination data for string operations.
3	TIR/BUF3	The table index register. The table index value is an index on the address that is contained in the TBR register. The sum of TBR and TIR defines the address of the word containing the micro-operators in the table code. When this address register is not being used for table type operations, it is alternatively used (as BUF3) for temporary storage of other address values.
4	LOSR	The limit of stack register. This register contains the upper stack boundary address for the current procedure. This register limits the size of the stack.

Index Register Number	Index Register Name	Register Usage
5	BOSR	The bottom of stack register. This register contains the lower boundary address for the current stack.
6	F	The F register. This register contains the address of the last MSCW for the current process stack in memory. The F register and the display register that corresponds to the present lexicographical level contain the identical address value.
7	BUF	The buffer address register. The buffer is used to temporarily store addresses.

Address Adder and Residue Test Logic

The address adder is a shared mechanism through which all addresses used within the B 6900 system are manipulated. Figure 5-2 shows this mechanism, with associated data paths and data integrity residue generation and check blocks.

All traffic to and from the IC memory is conducted through the address adder (or the Z8 and Z9 busses) to the address adder. Data integrity within all of these blocks is maintained by modulo three residue checking. This guarantees to detect any single bit error, and some multiple bit errors that occur in IC memory, or the address adder. An error in the modulo three residue generation circuit or in the residue check circuit is also detected.

Any addressing error in the address adder or in the residue check circuit is a fatal condition, and results in an "abort" type interrupt condition.

TRANSFER CONTROLLER

The transfer controller (see Figure 5-2) has two major sections: a hard register section referred to as stack registers for data and program information, and an internal data transfer section. Six busses, Z1 through Z6, are used for the normal data movement to and from the hard registers. Z1, Z2, and Z3 are input busses to these register contents are never are output busses. The capacity of each bus is 51 bits.

Three special busses are used for arithmetic operations (see Figures 5-3 and 5-6).

Stack Registers

Each information register has 51 bit positions (see Figure 5-3). Registers A, B, C, X, Y, and Z are for information handling during program flow. Registers P and L contain B 6900 program words. The P and L registers contents are never written into memory.

The Z3 and Z4 busses provide for bi-directional data flow between the hard registers and memory or the multiplexor.

The A and B registers are the top of stack registers, and X and Y are normally second-word information registers for double-precision operands. Registers C and Z are general purpose registers which provide temporary storage during operator execution.

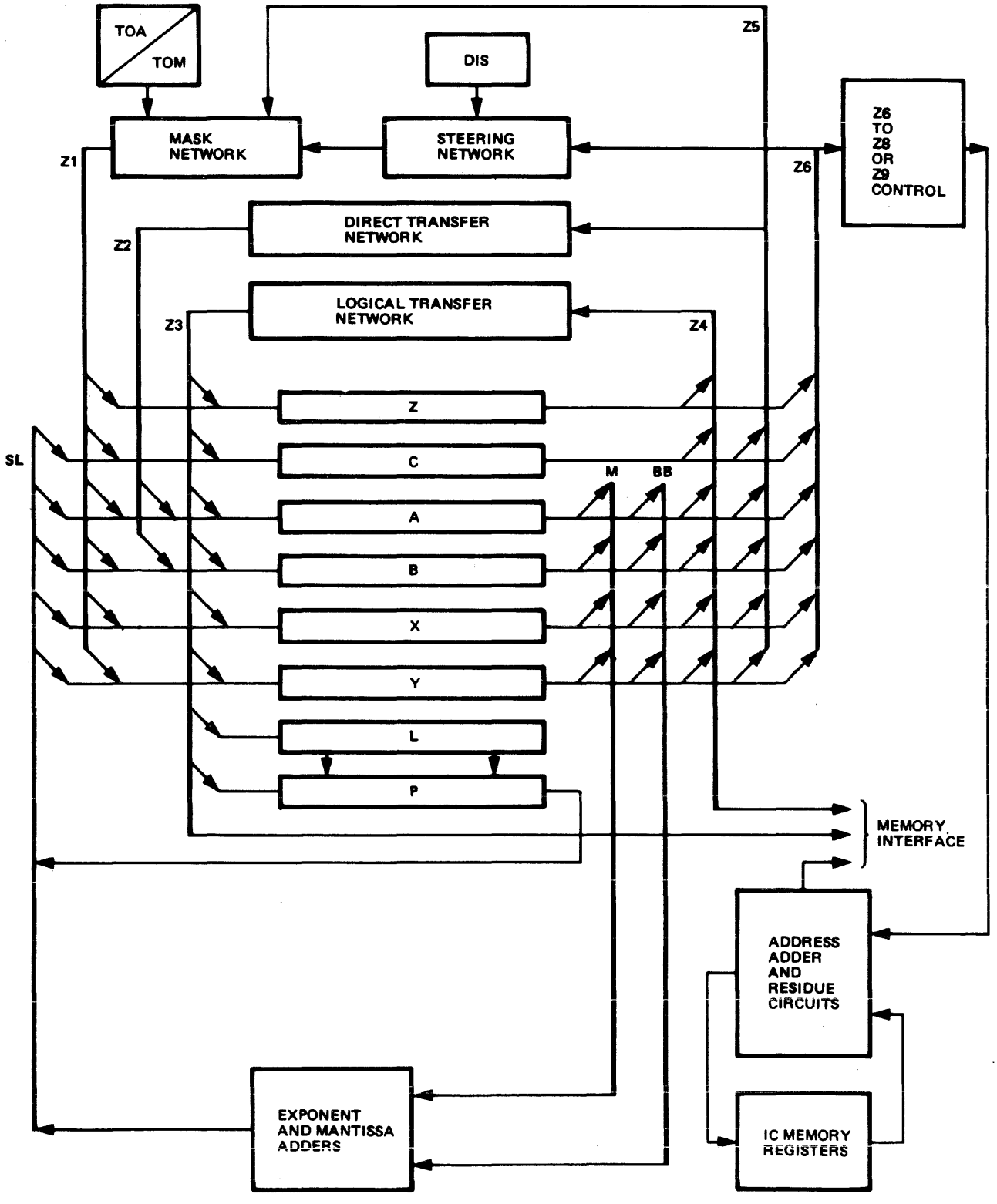


Figure 5-3. Internal Data Transfer Section

Internal Data Transfer Section

The internal transfer section (see Figure 5-3) permits the following data transfers between stack registers:

a. A direct, full-word transfer path using the Z5 and Z2 busses.

b. A logical transfer path to create the results of the family B (logical) operators, using the Z4 and Z3 busses. The logical transfer path also provides one additional full word transfer path between registers.

c. A steering and mask network providing a field displacement between stack registers using the Z6 and Z1 busses.

d. A transfer path to the address adder by means of the Z6 to Z8 or Z9 busses. This path extracts one of four fields, [39:20], [36:16], [19:20] or [13:14], from a stack register during execution of operator syllables.

e. A data movement path to and from the high speed adder by means of the AA, BB, and SL busses.

Mask and Steering

The mask and steering network moves bit fields from register to register by means of the Z6 and Z1 busses. All bits are transferred to and from the busses in parallel. Two pointers (TOA/TOM) set up a "window" defining the upper and lower limit of the bits being transferred to the accepting data register. A displacement register (DIS) shifts the bits to the right, 0 to 47 bits from the position previously held in the sending data register. The three controls used to steer and mask are as follows:

1. TOA. The highest bit position of the accepting field (highest bit of the window).

2. TOM. The highest bit position to be inhibited on the transfer (lowest bit of the window).

3. DIS. A right shift of the bits through the steering matrix.

Registers TOA, TOM, and DIS are set by the operator families or other controllers.

Mask and Steering Example

Assume the C register contains a stuffed indirect reference word (SIRW) and it is necessary to extract the STKNR (stack number) field (bits 45:10), and place these bits into the index field of the C register. The logic sets the window TOA := 29, TOM := 19, as shown in Figure 5-4. The displacement register is set to 16: DIS := 16. The actual starting bit of the field is calculated as: TOA + DIS = 29 + 16 = 45.

All Bits in the C register are gated to the Z6 bus. The bits (except TAG) are then shifted 16 places to the right with only the bits that align with the window appearing on the Z1 bus. The Z1 bus is then gated to the C register, with the masked field destroyed or retained; if the masked field is to be retained, the C register must be gated onto the Z5 bus as "prior content".

If no register is gated on the Z5 bus during a Z1 bus to Z6 bus transfer, the masked field is cleared.

In the example shown in Figure 5-4, a field of ten bits is transferred from one field location in the C register, to another field location in this same register. Because the STKNR field of the C register lies outside of the receiving field range, bits 45:10 are cleared, and bits 29:10 will contain the STKNR value at the conclusion of the example operation. Bit fields 47:18, and 19:20 of the C register are cleared and only 50:03 remain unchanged.

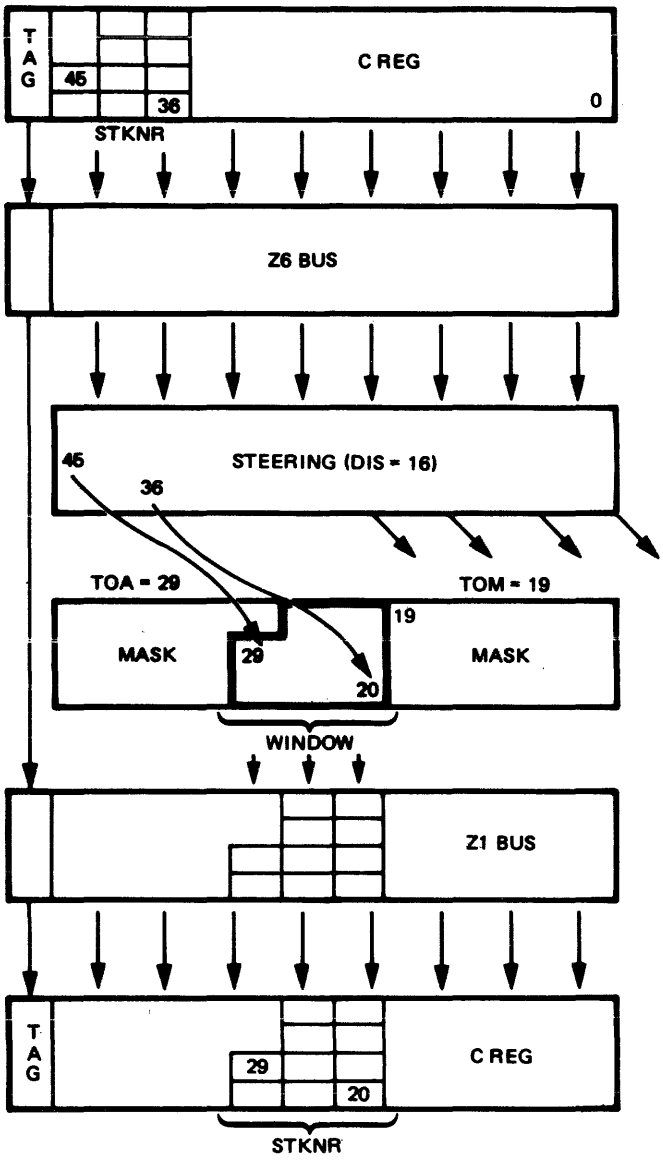


Figure 5-4. Mask and Steering

Stack Controller

The B 6900 provides automatic stack adjustment as required by the operators. These requirements are supplied to the stack controller on the Z11 bus from the operator families and other functional controllers.

The stack controller manipulates data between main memory and the A and B registers during both the pop-up and push-down cycles. The X and Y registers are included in the adjustment cycles when double-precision operands are involved.

A typical program stack is shown in Figure 5-5. The stack controller determines whether a push-up or push-down cycle will be initiated. All other Controllers remain idle until an adjust complete signal is sent to the controller that initiated the adjustment.

ARITHMETIC CONTROLLER

The arithmetic controller (see Figure 5-6) is a functional controller between the stack registers (A, B, C, X, Y and Z) and the exponent and mantissa adders. This controller is enabled by the family A operators and other operator families that require the use of these facilities.

Exponent and Mantissa Adders

Figure 5-6 shows the logical path of data flow to and from the exponent and mantissa adders. The exponent adder is composed of a 16-bit full adder/subtractor circuit, and the mantissa adder is composed of an 81-bit full adder/subtractor circuit. The inputs to the two adder circuits, and the outputs from the adder circuits, are directed from and to the stack hardware registers by the arithmetic controller.

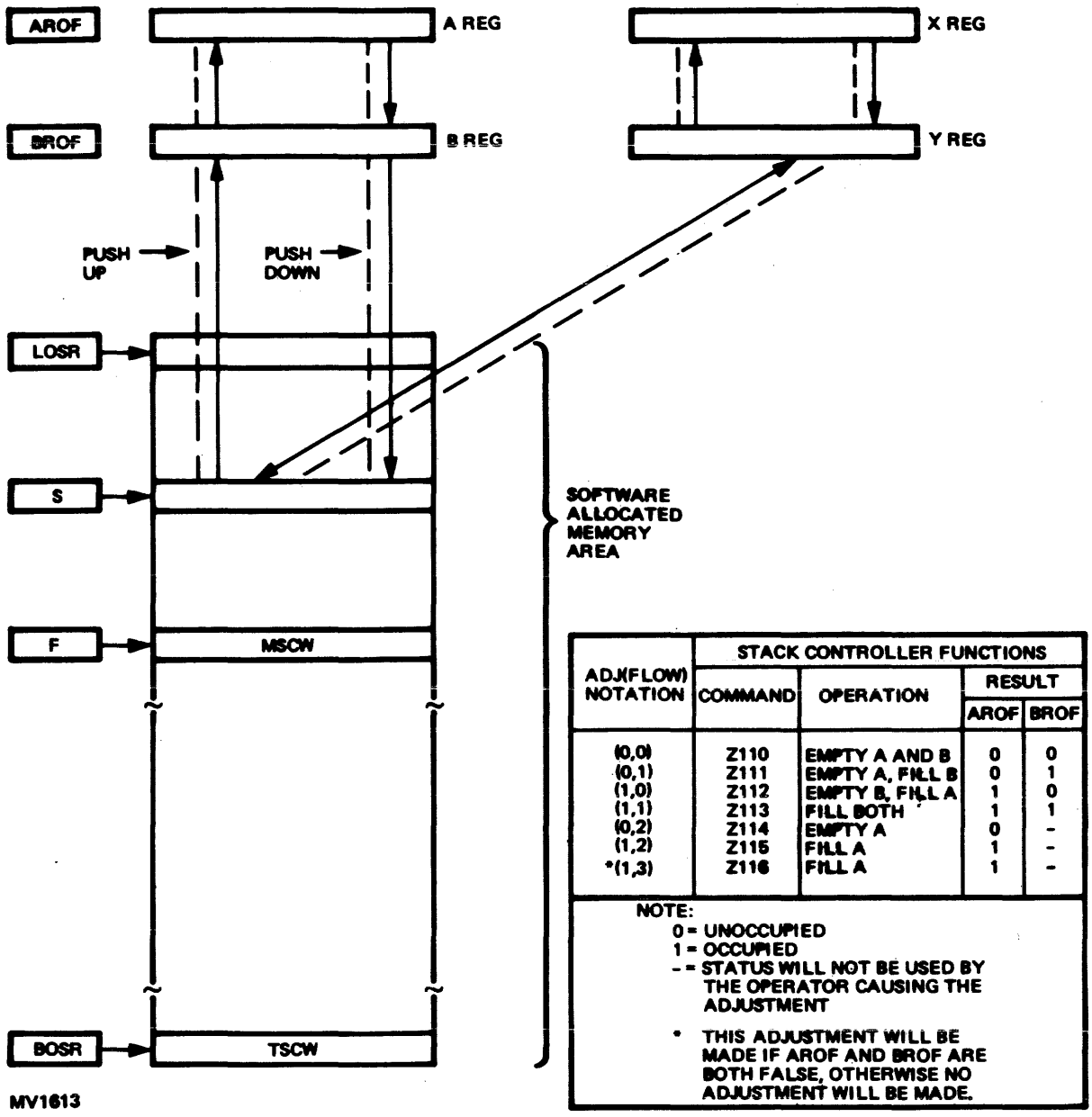
The arithmetic controller and the two adder circuits are capable of performing complete double precision mathematics in one continuous synchronized operation. The arithmetic controller gates both the exponent and mantissa portions of both halves of a double precision operand to the two adder circuits in a single operational step. Exponent adder operations are only performed during multiply or divide functions and for mantissa alignments.

Each of the two adder circuits consist of an A input (AA), a B input (BB), and a C (SL) resultant output. During a double precision ADD (80) operation, the A input to the mantissa adder consists of the 78-bits of the mantissa field from the double precision operand in the A and X registers. The B inputs to the two adders for a double-precision ADD operation are the same as the A inputs, but are derived from the B and Y registers. After the inputs to the two adders have been routed to the adder inputs by the arithmetic controller, the ADD operation is performed in one step. After the ADD algorithm is completed, the resultant sum of the two numbers is routed by the arithmetic controller back to the proper stack register(s).

INTERRUPT CONTROLLER

The Interrupt Controller of the B 6900 CPU recognizes certain types of system interrupts, automatically causes the currently running program to halt, and ENTERs into the Interrupt Handling Procedure of the Master Control Program (MCP). The Interrupt Handling Procedure takes actions required because of the interrupt, and then automatically RETURNs to the program that was halted when the interrupt was sensed. Thus, interrupt handling in a B 6900 system is a dynamic process that is initiated automatically when an interrupt occurs, and terminates by resuming program processing at the point where the interrupt was sensed.

The actions of the Interrupt Controller Logic include collecting and formatting information about the nature of the interrupt that occurred. Before the MCP Interrupt Handling Procedure is ENTERed this information is placed in the Top-of-Stack. The MCP Interrupt Handling Procedure uses the information collected by the Interrupt Controller, to analyze the nature and cause of the interrupt that occurred, and to determine what action is to be taken because of the interrupt.



ADJ(FLOW) NOTATION	STACK CONTROLLER FUNCTIONS			
	COMMAND	OPERATION	RESULT	
			AROF	BROF
(0,0)	Z110	EMPTY A AND B	0	0
(0,1)	Z111	EMPTY A, FILL B	0	1
(1,0)	Z112	EMPTY B, FILL A	1	0
(1,1)	Z113	FILL BOTH	1	1
(0,2)	Z114	EMPTY A	0	-
(1,2)	Z115	FILL A	1	-
*(1,3)	Z116	FILL A	1	-

NOTE:
0 = UNOCCUPIED
1 = OCCUPIED
- = STATUS WILL NOT BE USED BY THE OPERATOR CAUSING THE ADJUSTMENT

* THIS ADJUSTMENT WILL BE MADE IF AROF AND BROF ARE BOTH FALSE, OTHERWISE NO ADJUSTMENT WILL BE MADE.

Figure 5-5. Hardware Stack Adjustment

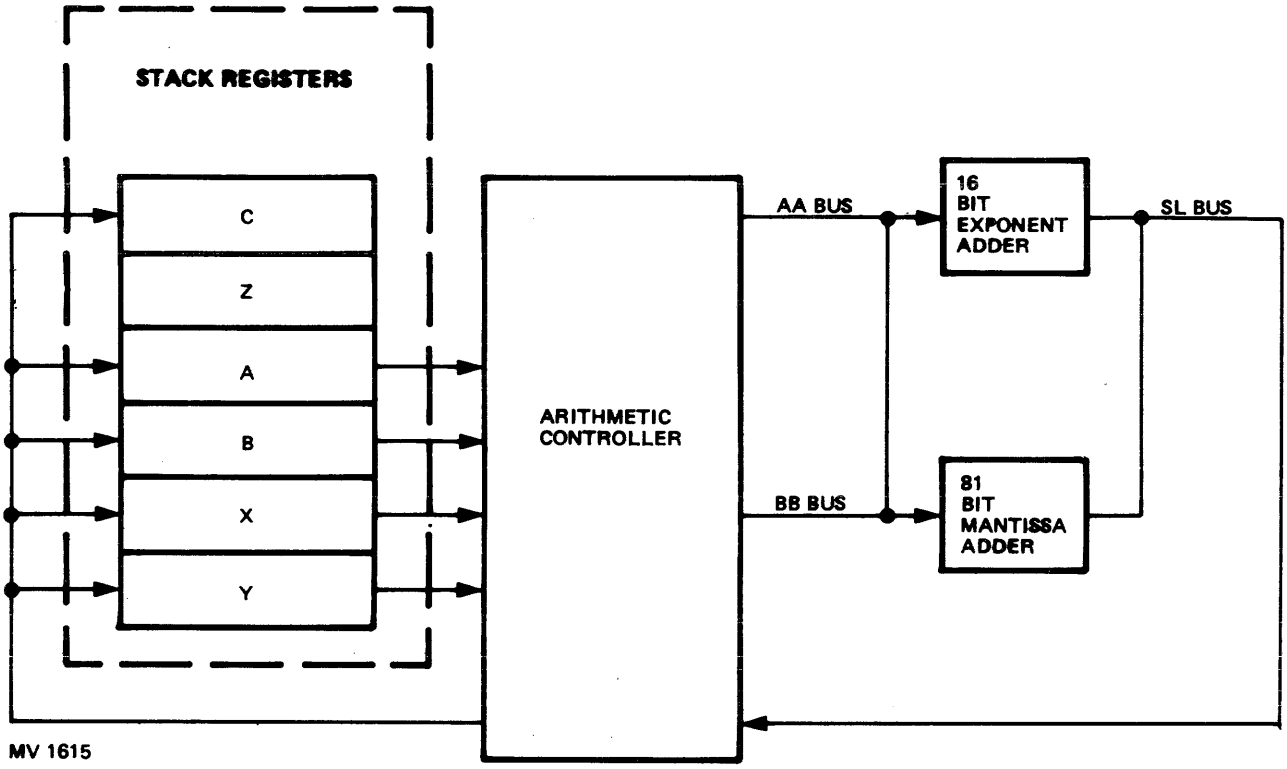


Figure 5-6. Arithmetic Control

INTERRUPT PARAMETER WORDS

Figure 5-7 shows three interrupt stack parameter words, an Indirect Reference Word (IRW) that points to the relative memory location of the MCP Interrupt Handling Procedure, and a Mark Stack Control Word (MSCW). These five words constitute an Interrupt Procedure Stack to be used by the Interrupt Handling Procedure. They are formed, after the interrupt condition is sensed and the currently running program is halted, by action of the Interrupt Controller in the Top-of-Stack registers. Note that the Interrupt Handler Procedure of the MCP has not yet been ENTERed. The words beneath the Interrupt Stack shown in the Figure are the Stack for the program that was halted when the interrupt was sensed.

The Interrupt Controller Logic pseudo-calls the ENTER operator flow, to initiate the Interrupt Handler of the MCP into operation. The ENTER operator flow uses the IRW in the interrupt stack, to find the PCW for the Interrupt Handler procedure. The ENTER operator flow also generates the Return Control Word (RCW), that points back to the procedure that was interrupted. This RCW is written in the interrupt stack, in the same memory word address that holds the IRW. At the conclusion of the Interrupt Handler procedure, the RCW is used to return control of the system to the procedure that was interrupted.

The Interrupt Controller logic causes a branch into the Interrupt Handler Procedure of the MCP by referencing the location of the IRW. This IRW points to the Program Control Word (PCW), which is always present in the MCP stack, at location D0 +3. Because the location of the PCW in the MCP stack is a fixed memory location, the Interrupt Controller logic can use an IRW to reference the PCW, no matter where a current program is in memory when an interrupt occurs.

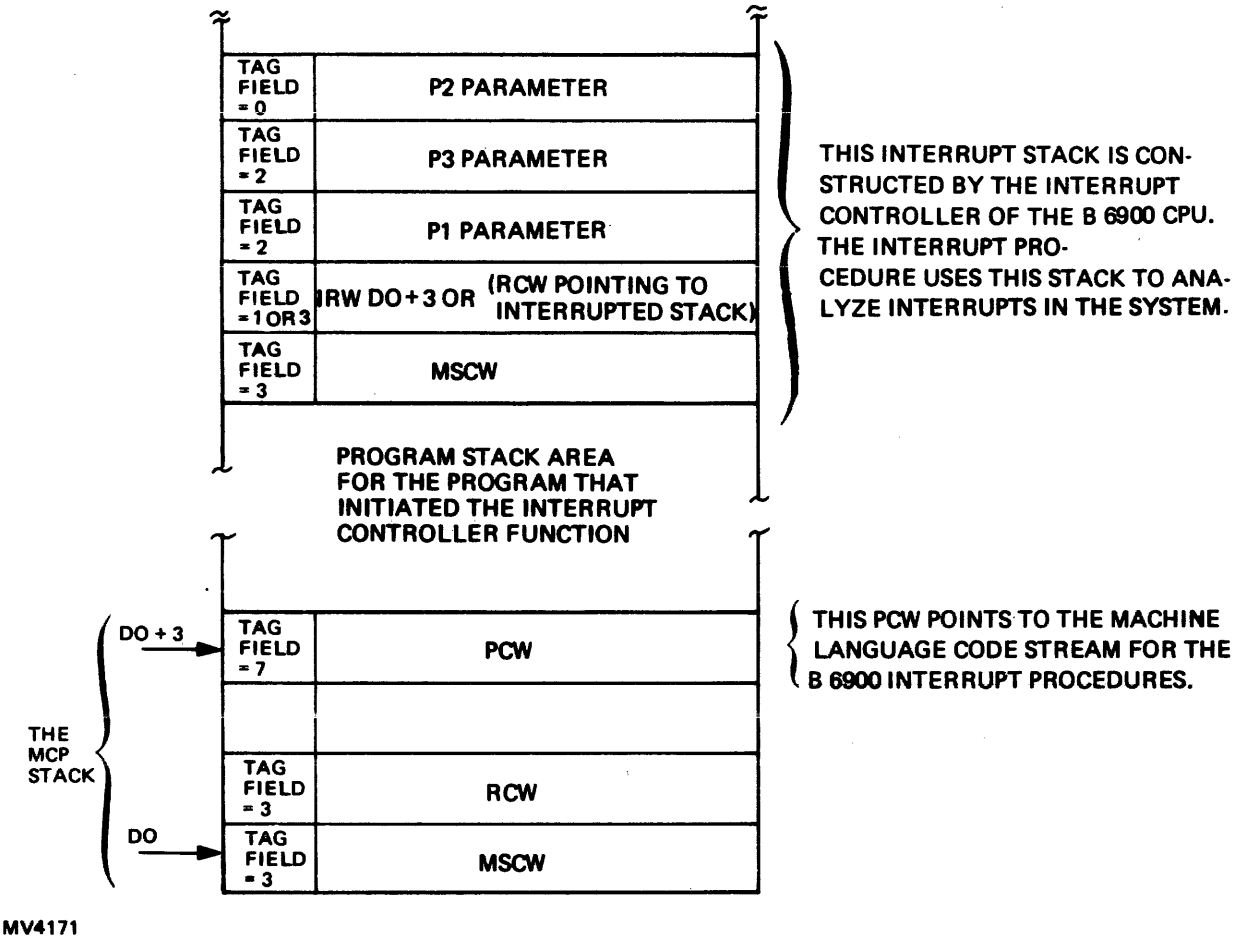


Figure 5-7. Interrupt Controller Stack Parameters

The three interrupt parameters in the Interrupt Stack are the data about the interrupt, that was collected by the Interrupt Controller. The P1 parameter word is formatted to identify the type of interrupt that occurred, the class of the interrupt, and the specific interrupt within the type and class that occurred. The P2 and P3 parameters contain specific information about the interrupt identified by the P1 parameter. The values of the three parameter words change with each particular interrupt type and class sensed by the Interrupt Controller.

The formats for various interrupt types and classes are given in the following order:

1. ALARM Interrupts.
2. HARDWARE Interrupts.
3. GENERAL CONTROL Interrupts.
4. EXTERNAL Interrupts.
5. SYLLABLE DEPENDENT Interrupts.

ALARM INTERRUPTS

Figures 5-8 through 5-11 define the Interrupt Stack parameter word layouts for ALARM Interrupts. Figure 5-8 shows the word layout of the P1 parameter for an ALARM Interrupt. Figures 5-9 and 5-10 show the variations in the P2 parameter for an ALARM Interrupt. Figure 5-11 shows the word layout for an ALARM Interrupt P3 parameter. Table 5-1 lists the fields in the P3 parameter that are used for each type of ALARM Interrupt.

ALARM INTERRUPT DESCRIPTIONS

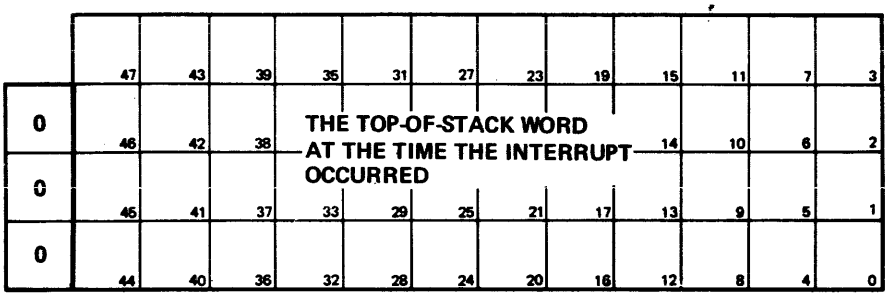
A description of each ALARM interrupt that can be detected by the B 6900 Interrupt Controller follows. These descriptions define the most likely reason for the occurrence of the interrupt and also describe the condition of the Top-of-Stack at the end of the Interrupt Controller logic operation. This ending condition is the state of the Top-of-Stack when the Interrupt Handling Procedure of the MCP is ENTERed. It also represents the condition present if the EVENT logic of the CPU is used to freeze the CPU on the occurrence of an ALARM interrupt.

	0 47	0 46	0 45	0 44	0 43	1 27	0 42	X 19	0 18	0 11	READ DATA MULT. BIT ERR.	INV. ADDR. LOCAL MEM. 3
0	0 40	0 39	0 38	0 34	0 33	0 28	0 32	X 18	0 14	0 10	MEM. ADDR. RESI-DUE 6	0 2
1	0 40	0 41	0 37	0 36	0 35	1 25	0 31	X 17	0 13	GL. MEM. NOT READY9	INV. PROG. WORD 5	MEM. ADDR. PARITY 1
0	0 44	0 40	0 39	0 32	0 28	0 24	0 30	0 16	0 12	INV. ADDR. GL. MEM. 8	STACK UNDER FLOW 4	LOOP TIMER 0

25 = 1 = ALARM INTERRUPT TYPE INTERRUPT
19 = RETRY FLIP-FLOP STATE
= 1 = RETRY FF IS SET
= 0 = RETRY FF IS RESET
18 = TYPE OF GLOBAL MEMORY ERROR (ONLY USED FOR GLOBAL MEMORY ERRORS)
= 1 = GLOBAL SCAN OPERATION ERROR
= 0 = GLOBAL MEMORY OPERATION ERROR
17 = P3 PARAMETER CONSISTENCY FLAG
= 1 = P3 PARAMETERS ARE INCONSISTENT
= 0 = P3 PARAMETERS ARE CONSISTENT
X = 1 OR 0

MV4172

Figure 5-8. ALARM Interrupt P1 Parameter Word Layout

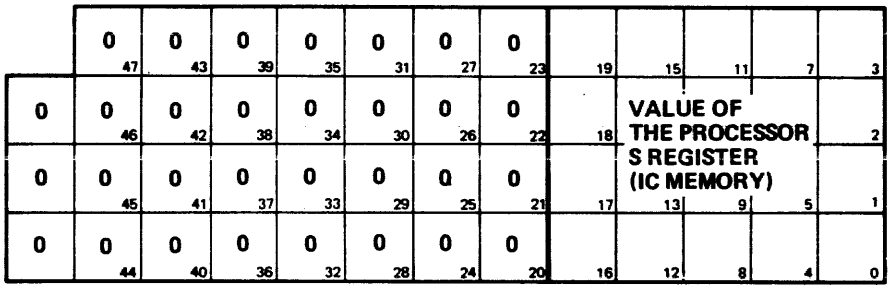


NOTE

A STACK UNDERFLOW ALARM INTERRUPT P2 PARAMETER IS A SEPARATE CONDITION FROM ALL OTHER ALARM INTERRUPTS. SEE THE SPECIAL P2 WORD FORMAT, WHICH IS DIFFERENT.

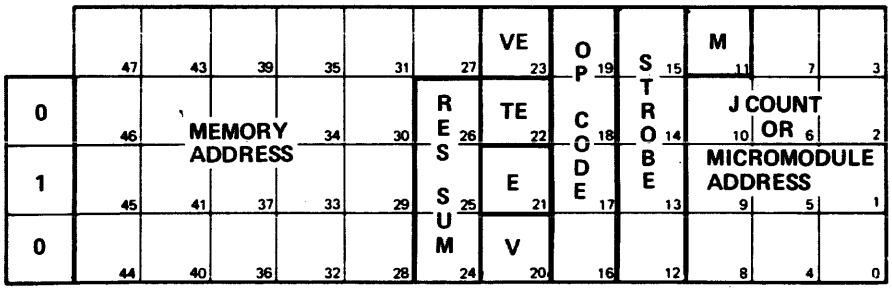
MV4173

Figure 5-9. ALARM Interrupt P2 Parameter Word Layout



MV4174

Figure 5-10. ALARM Interrupt Stack Underflow P2 Parameter Layout



RES SUM = RESIDUE OF ADDRESS
VE = VECTOR MODE
TE = TABLE EDIT MODE
E = EDIT MODE
V = VARIANT MODE
M = MODED
= 0 = J COUNT VALUE IS PRESENT
= 1 = MICROMODULE ADDRESS IS PRESENT

MV4175

Figure 5-11. ALARM Interrupt P3 Parameter Word Layout

Table 5-1. ALARM Interrupt P3 Parameter Fields Usage

Interrupt Type	Fields Present in the P3 Parameter Word
Loop Timer	Op Code, Strobe, J Count
Memory Address Parity	Address, OP Code, Strobe, J Count
Invalid Address Local	Address, OP Code, Strobe, J Count
Stack Underflow	Address, OP Code, Strobe, J Count
Invalid Program Word	OP Code, Strobe, J Count
Memory Address Residue	Address, OP Code, Strobe, J Count
Read Data Multiple-Bit	Address, OP Code, Strobe, J Count
Invalid Address Global	Address, OP Code, Strobe, J Count
Global Memory Not-Ready	Address, OP Code, Strobe, J Count

LOOP Interrupt

This interrupt is invoked if the Data Processor fails to provide a SECL signal within 2 seconds. This interrupt could occur if an attempt is made to execute an invalid operator code. If the interrupt occurs, the P1 parameter is left in the B register, the A register is cleared, and the Program controller PIR register is backed up.

Memory Address Parity Interrupt

This interrupt is invoked if the Memory Controller detects an even number of ADDRESS and CONTROL bits being transmitted between the Data Processor/MLIP and a system memory module. Should this interrupt occur, the P1 parameter is left in the B register, the A register is cleared, and the Program Controller PIR register backed up.

Invalid Address Local Interrupt

This interrupt is invoked by the Memory Controller if within 8 clock-periods it does not receive acknowledgement of a local memory request. Failure to acknowledge indicates an attempt to access a non-existent local memory module. Consequently, the P1 parameter is left in the B register, the A register is cleared, and the Program Controller PIR register is backed up.

Stack-Underflow Interrupt

This interrupt is invoked if during a stack adjustment operation the Stack Controller detects an attempt to change the value of the IC memory S register to a value that is less than that of the F register. If this interrupt occurs, the P1 parameter is left in the B register, the A register is cleared, and the Program Controller PIR register is backed up.

Invalid Program Word Interrupt

This interrupt is invoked if any of the following condiditions occur:

1. A word with a TAG not equal to 3 is placed in the P register for execution (except in TABLE EDIT Mode).
2. The VARIANT operator syllable (95) is followed immediately by another VARIANT operator syllable (95).
3. The Data Processor is in EDIT MODE and a family strobe for a family other than an EDIT MODE operator family is emitted.

If this interrupt occurs the P1 parameter is left in the B register, the A register is cleared, and the Program Controller PIR register is backed up.

Memory Address Residue Interrupt

This interrupt is invoked when the Memory Controller detects that an error is present in the MAR/LAR address registers. Residue checking is a method of detecting abnormalities in the Address Adder and/or the IC memory address registers. Any activity of the Address adder that results in the setting of a Residue Interrupt prevents a memory access cycle from being initiated by the Memory Controller.

Read Data Multiple-bit Interrupt

This interrupt is invoked when the Memory Controller detects more than a single bit in error during the ERROR DETECTION/ERROR CORRECTION part of a memory READ cycle operation. Multiple bits in error are not correctable; thus, when such an error is detected the Memory Controller causes an ALARM interrupt to occur.

Invalid Address-Global Interrupt

This interrupt is identical to the INVALID ADDRESS-LOCAL interrupt previously defined, except that the invalid address is for a Global memory module instead of a local memory module. Refer to the description of an INVALID ADDRESS-LOCAL interrupt.

Global Memory Not-ready Interrupt

This interrupt is invoked when a memory access is initiated on a Global memory module, and when the Global memory module does not properly respond to the control of the Memory Controller logic.

HARDWARE INTERRUPTS

Figures 5-12 through 5-14 define the Interrupt Stack parameter word layouts for HARDWARE Interrupts. Figure 5-12 shows the word layout of the P1 parameter, Figure 5-13 shows the word layout of the P2 parameter, and Figure 5-14 shows the word layout of the P3 parameter for a HARDWARE Interrupt. Table 5-2 lists the fields in the P3 parameter that are used for each type of HARDWARE Interrupt.

	0	0	0	0	0	1	0	X	0	0	0	ADDER RESI. ERR. 3
	47	43	39	35	31	27	23	19	15	11	7	
0	0	0	0	0	0	1	0	0	0	0	0	BUS RESI. ERR. 2
	46	42	38	34	30	26	22	18	14	10	6	
1	0	0	0	0	0	0	0	X	0	0	0	RAM CARD PAR. ERR. 1
	45	41	37	33	29	25	21	17	13	9	5	
0	0	0	0	0	0	0	0	0	0	COMP. RESI. ERR. 4		PROM CARD PAR. ERR. 0
	44	40	36	32	28	24	20	16	12	8	4	

26 = 1 = HARDWARE INTERRUPT TYPE INTERRUPT
19 = RETRY FLIP-FLOP STATE
= 1 = RETRY FF IS SET
= 0 = RETRY FF IS RESET
17 = P3 PARAMETER CONSISTENCY FLAG
= 1 = P3 PARAMETERS ARE INCONSISTENT
= 0 = P3 PARAMETERS ARE CONSISTENT
X = 1 OR 0

MV4176

Figure 5-12. HARDWARE Interrupt P1 Parameter Word Layout

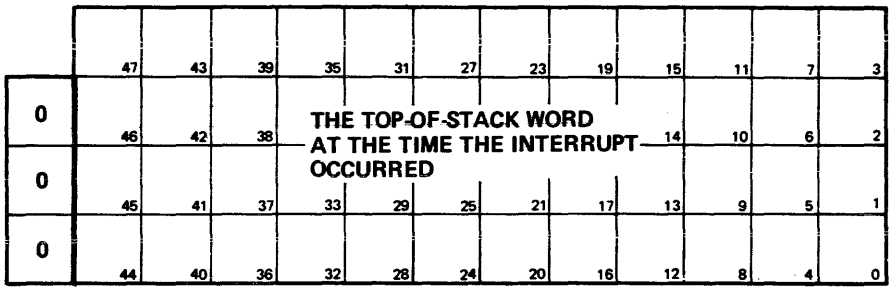


Figure 5-13. HARDWARE Interrupt P2 Parameter Word Layout

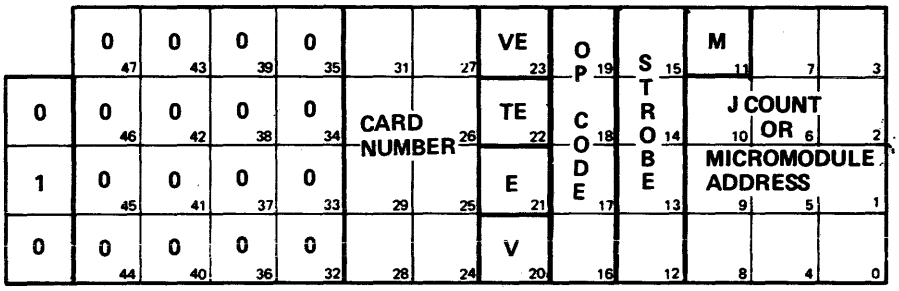


Figure 5-14. HARDWARE Interrupt P3 Parameter Word Layout

Table 5-2. HARDWARE Interrupt P3 Parameter Fields Usage

Interrupt Type	Fields Present in the P3 Parameter Word
PROM Card Parity	Card Number, OP Code, Strobe, J Count
RAM Card Parity	Card Number, OP Code, Strobe, J Count
Bus Residue	OP Code, Strobe, J Count
Compare Residue	OP Code, Strobe, J Count

HARDWARE INTERRUPT DESCRIPTIONS

A description of each HARDWARE interrupt that can be detected by the Interrupt Controller follows. These descriptions define the most likely reason for the occurrence of the interrupt. The Interrupt Stack parameter conditions are the same for a HARDWARE interrupt as those described previously for an ALARM interrupt (refer to the ALARM INTERRUPT DESCRIPTIONS subsection in this section).

PROM Card Parity Interrupt

The CPU contains many PROM component devices which are used to hold preselected microcodes and addresses. Each time a PROM device is addressed, the output of the PROM is tested for parity. If a PROM device parity error condition is detected by the test, a PROM Card Parity Interrupt is invoked. The parameters for the interrupt contain the address of the CPU card package on which the PROM parity condition was detected.

RAM Card Parity Error Interrupt

The CPU contains RAM memory devices. Each time a RAM memory device is accessed, the output of the device is tested for a parity error condition. If a RAM parity error condition is detected during the test, a RAM Card Parity Error interrupt is invoked. The interrupt parameters contain the CPU card package location of the RAM device that caused the parity error condition.

Bus Residue Interrupt

The Residue Generator card packages of the CPU test the residue bits from the Z8 and Z9 busses. These busses are inputs to the ADDRESS ADDER logic of the CPU. If an error condition is detected in the Bus Residue value(s), a Bus Residue interrupt is invoked, and the residue bit-value that causes the interrupt to be detected is placed in the Interrupt Stack parameters.

Adder Residue Interrupt

The CPU RESIDUE ADDER operates in conjunction with the CPU ADDRESS ADDER logic. The residues of the address inputs to the ADDRESS ADDER circuit, present at the RESIDUE ADDER inputs, are tested for residue value errors. If a residue value on the Z8 or Z9 contains a value error, an Adder Residue interrupt is invoked. The residue value that caused the value error to be detected is remembered by storing it in the Interrupt Stack parameters.

Compare Residue Interrupt

Residue values present at the input to the RESIDUE ADDER in the CPU are added, and a sum of residues is produced. At the same time that the RESIDUE ADDER is adding two residue values, the ADDRESS ADDER is adding the address values that correspond to the residue values in the RESIDUE ADDER. The output of the ADDRESS ADDER includes a new residue sum, which is the same as the sum of residues from the RESIDUE ADDER. The two new residue sums, one from the RESIDUE ADDER and the other from the ADDRESS ADDER, are compared. If the two new residues are not the same, a Compare Residue Interrupt is invoked. The residue comparator output is saved by placing its value in the Interrupt Stack parameters.

GENERAL CONTROL INTERRUPTS

Figures 5-15 through 5-17 define the Interrupt Stack parameter word layouts for GENERAL CONTROL interrupts. Figure 5-15 shows the P1 parameter, Figure 5-16 shows the P2 parameter, and Figure 5-17 shows the P3 parameter. Table 5-3 shows the fields that are present in the P3 parameter for each type of GENERAL CONTROL interrupt.

GENERAL CONTROL INTERRUPT DESCRIPTIONS

A description of each GENERAL CONTROL interrupt detected by the B 6900 Interrupt Controller follows. These descriptions define the most likely reason for the occurrence of the interrupts. The Interrupt Stack parameter conditions are the same for a GENERAL CONTROL interrupt as those described previously for an ALARM interrupt (refer to the ALARM INTERRUPT DESCRIPTIONS subsection in this section).

	47	43	39	35	31	27	23	19	15	11	7	3
	0	0	0	0	0	0	0	X	0	0	0	READ DATA RETRY
0	X 46	0 42	0 38	0 34	0 30	0 26	1 22	X 18	0 14	0 10	0 6	0 2
1	0 45	0 41	0 37	0 33	0 29	0 25	0 21	0 17	0 13	0 9	ADDR. RETRY 5	0 1
0	X 44	0 40	0 36	0 32	0 28	0 24	0 20	0 16	0 12	0 8	READ DATA CHECK BIT 4	READ DATA SINGLE BIT 0

22 = 1 = GENERAL CONTROL TYPE INTERRUPT

46 = PRESENCE BIT DURING VALUE CALL OPERATION FLAG
= 1 = VALUE CALL SYLLABLE EVALUATION IN PROCESS
= 0 = NO VALUE CALL SYLLABLE EVALUATION IN PROCESS

44 = VECTOR MODE OPERATION FLAG
= 1 = VECTOR MODE OPERATION IN PROCESS
= 0 = NO VECTOR MODE OPERATION IN PROCESS

19 = RETRY FLIP-FLOP STATE
= 1 = RETRY FF SET
= 0 = RETRY FF RESET

18 = TYPE OF GLOBAL MEMORY ERROR (ONLY USED FOR GLOBAL MEMORY ERRORS)
= 1 = GLOBAL SCAN OPERATION ERROR
= 0 = GLOBAL MEMORY OPERATION ERROR

X = 1 OR 0

MV4179

Figure 5-15. GENERAL CONTROL Interrupt P1 Parameter Word Layout

	47	43	39	35	31	27	23	19	15	11	7	3
0	46	42	38	THE VALUE OF THE TOP-OF-STACK WORD AT THE TIME THE INTERRUPT OCCURRED				18	14	10	6	2
0	45	41	37	33	29	25	21	17	13	9	5	1
0	44	40	36	32	28	24	20	16	12	8	4	0

MV4180

Figure 5-16. GENERAL CONTROL Interrupt P2 Parameter Word Layout

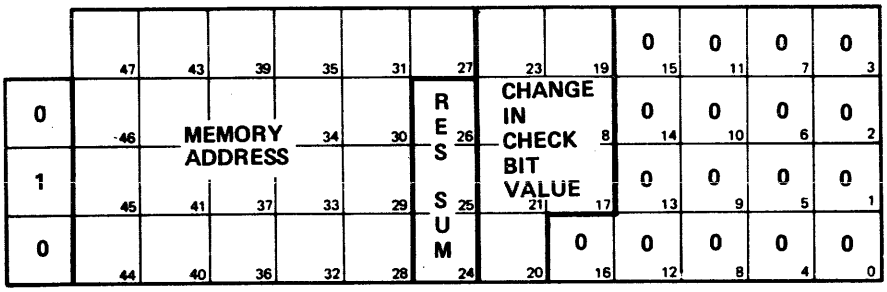


Figure 5-17. GENERAL CONTROL Interrupt P3 Parameter Word Layout

Table 5-3. GENERAL CONTROL Interrupt P3 Parameter Field Usage

Interrupt Type	Fields Present in the P3 Parameter Word
Read Data Single-bit	Address, RES SUM, Change In Check Bits
Read Data Retry	Address, RES SUM
Read Data Check-bit	Address, RES SUM, Change In Check Bits
Address Retry	Address, RES SUM

Read Data Single Bit Interrupt

The Read Data Single Bit interrupt is invoked when the Memory Controller ERROR DETECTION/ERROR CORRECTION circuit detects and corrects a single-bit error in memory READ data. The bit-in-error is corrected and the program in progress continues as if no error had been detected. As a result of the Interrupt Controller operation, data about the single-bit error is recorded in the SYSTEM/SUMLOG file. This information is used by maintenance personnel to anticipate and analyze potentially serious memory data failures (refer to multiple-bit error ALARM Interrupt subsection in this section).

Read Data Retry Interrupt

A Read Data Retry interrupt is invoked when the Memory Controller causes READ data to be restrobed onto the CPU/Memory Module interface bus. Restrobing of READ data onto the bus is caused by the Memory Port Control logic sensing a parity error on the interface bus. If restrobing the data on the bus corrects the parity error, a retry interrupt is invoked; otherwise, an ALARM Interrupt is invoked. Retry interrupts are used for system maintenance analysis, as described for single-bit errors above.

Read Data Check Bit Interrupt

A memory READ word contains eight bits that are used for an ERROR DETECTION/ERROR CORRECTION check code. If an error is detected in the check code during a memory READ operation and if no error is present in the READ data, then the Read Data Check Bit Interrupt is invoked. This error is written into the SYSTEM/SUMLOG the same as the Read Data Single Bit Interrupt, and the program in progress is continued as if no error had occurred (refer to Read Data Single Bit Interrupt above). If a READ data bit is also in error, then a multiple-bit error exists (see the ALARM Interrupt subsection of this section), and the ALARM Interrupt is invoked instead of the GENERAL CONTROL Interrupt.

The ERROR DETECTION/ERROR CORRECTION check codes of the B 6900 system are internal codes of the operating system. They are not available to a system user, except through use of the Memory Tester logic of the CPU.

Address Retry Interrupt

An Address Retry Interrupt is essentially the same as a READ Data RETRY Interrupt, except that it is invoked if a memory address is in error instead of a data bit in error (refer to Read Data Single Bit Interrupt subsection of this section). If the address retry is successful, the program in process is continued the same as though no error existed. If the address retry is not successful, then an ALARM Interrupt is invoked instead of the GENERAL CONTROL Interrupt.

EXTERNAL INTERRUPTS

Figures 5-18 through 5-20 define the Interrupt Stack parameter word layouts for EXTERNAL Interrupts. Figure 5-18 shows the P1 parameter, Figure 5-19 shows the P2 parameter, and Figure 5-20 shows the P3 parameter. The B 6900 system only utilizes one EXTERNAL Interrupt, which is the I/O Finished Interrupt.

	47	43	39	35	31	27	23	19	15	11	7	3
	0	0	0	0	0	0	0	0	0	0	1	0
0	0 46	0 42	0 38	0 34	0 30	0 26	0 22	0 18	0 14	0 10	0 6	0 2
1	0 45	0 41	0 37	0 33	0 29	0 25	0 21	0 17	0 13	0 9	0 5	0 1
0	0 44	0 40	0 36	0 32	0 28	0 24	1 20	0 16	0 12	0 8	1 4	1 0

MV4182

Figure 5-18. EXTERNAL Interrupt P1 Parameter Word Layout

	47	43	39	35	31	27	23	19	15	11	7	3
	0	0	0	0	0	0	0	0	0	0	0	0
0	0 46	0 42	0 38	0 34	0 30	0 26	0 22	0 18	0 14	0 10	0 6	0 2
0	0 45	0 41	0 37	0 33	0 29	0 25	0 21	0 17	0 13	0 9	0 5	0 1
0	0 44	0 40	0 36	0 32	0 28	0 24	0 20	0 16	0 12	0 8	0 4	0 0

MV4183

Figure 5-19. EXTERNAL Interrupt P2 Parameter Word Layout

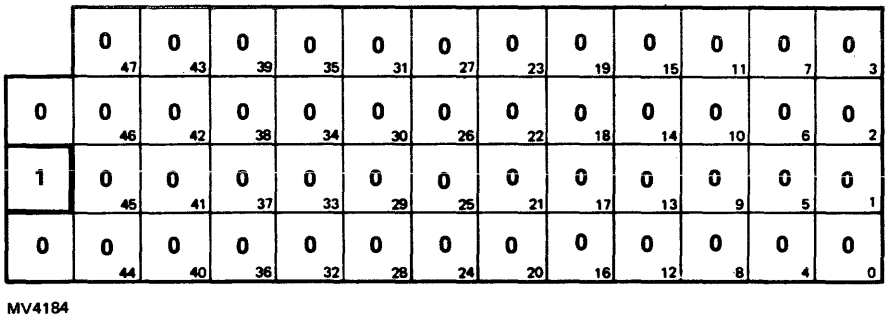


Figure 5-20. EXTERNAL Interrupt P3 Parameter Word Layout

I/O Finished Interrupt

An I/O Finished Interrupt is invoked at the conclusion of a peripheral device operation, when the IOCB for the I/O device operation specifies that such an interrupt is required. Word zero of the IOCB (the Control Word, CW) contains two bits which may specify that an I/O Finish Interrupt is required.

If bit-3 of the Control word in an IOCB is a binary 1, an I/O Finish Interrupt is required at the conclusion of the peripheral device operation.

If bit-2 of the Control Word in an IOCB is a binary 1, an I/O Finish Interrupt is required at the conclusion of the peripheral device operation.

SYLLABLE DEPENDENT Interrupts

Figures 5-21 through 5-27 define the Interrupt Stack parameter word layouts for SYLLABLE DEPENDENT Interrupts. Figure 5-21 shows the word layout of the P1 parameter. Figures 5-22 through 5-26 show the variations in the word layout of the P2 parameter. Figure 5-27 shows the word layout of the P3 parameter. Table 5-4 lists the fields in the P3 parameter that are used for each type of SYLLABLE DEPENDENT Interrupt.

SYLLABLE DEPENDENT Interrupt Classes

There are 2 classes of SYLLABLE DEPENDENT Interrupts. One class consists of interrupts where the Program Controller register values are consistent, after the interrupt is invoked by the Interrupt Controller. The other class consists of those interrupts where the Program Controller register values are not consistent after the interrupt is invoked. The Program Controller register values in question are the PBR, PIR, and PSR registers.

Consistent Program Controller register values are backed up to point at the beginning of the program operator code in process when the interrupt was detected by the Interrupt Controller. Inconsistent register values may or may not have been backed up in a consistent manner.

The P1 parameter word (Figure 5-21) indicates the class of a SYLLABLE DEPENDENT Interrupt.

SYLLABLE DEPENDENT Presence-Bit Interrupts

Presence Bit Interrupts are a special class of SYLLABLE DEPENDENT Interrupts. To make the B 6900 a "Virtual" system presence-bit interrupts are used in conjunction with Descriptor. The P1 parameter for a SYLLABLE DEPENDENT Interrupt (Figure 5-21) contains bits that identify the nature of a possible Presence-bit operation that was in process when the SYLLABLE DEPENDENT Interrupt was invoked.

	0 47	0 43	VS 39	0 35	0 31	0 27	X 23	X 19	0 15	INT. TIMER 11	BASE OF STACK 7	EXP. OVER FLOW 3
0	RT 46	0 42	0 38	0 34	0 30	0 26	0 22	0 18	0 14	SEG. ARRAY 10	INT. OVER FLOW 6	DIV. BY ZERO 2
1	RT 45	0 41	0 37	0 33	0 29	0 25	0 21	0 17	CONF. ERROR 13	SEQ. ERROR 9	INV. INDEX 5	INV. OPND 1
0	X 44	0 40	0 36	0 32	0 28	X 24	0 20	0 16	STACK OVER FLOW 12	PRES. BIT 8	EXP. UNDER FLOW 4	MEM. PROT. 0

	BIT 24	BIT 23		
	24	23	=	SYLLABLE DEPENDENT TYPE INTERRUPT
	0	1	=	PIR, PSR, & PBR VALUES ARE INCONSISTENT
	1	0	=	PIR, PSR, & PBR VALUES ARE CONSISTENT

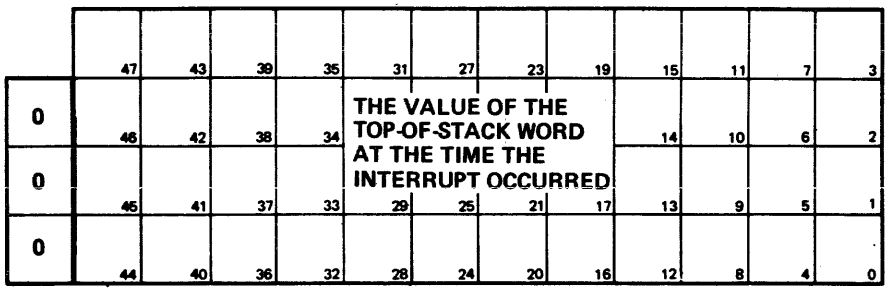
BIT 46	BIT 45	BIT 39		
46	45	39	=	PRESENCE BIT INTERRUPT PARAMETERS
0	0	1	=	VECTOR STACK CAUSED INTERRUPT, (PROCEDURE DEPENDENT) THE EXIT OPERATOR FLOW WAS USED TO ESCAPE FROM THE PRESENCE BIT INTERRUPT.
0	1	1	=	VECTOR STACK CAUSED INTERRUPT, (PROCEDURE DEPENDENT) THE RETURN OPERATOR FLOW WAS USED TO ESCAPE FROM THE PRESENCE BIT INTERRUPT.
1	0	0	=	VALUE CALL OPERATOR CAUSED INTERRUPT, (DATA DEPENDENT) THE EXIT OPERATOR FLOW WAS USED TO ESCAPE FROM THE PRESENCE BIT INTERRUPT.
1	1	0	=	VALUE CALL OPERATOR CAUSED INTERRUPT (DATA DEPENDENT) THE RETURN OPERATOR FLOW WAS USED TO ESCAPE FROM THE PRESENCE BIT INTERRUPT.

BIT 44		
	=	VECTOR MODE OPERATION FLAG BIT
	=	0 = VECTOR MODE OPERATION NOT IN PROCESS
	=	1 = VECTOR MODE OPERATION IN PROCESS

BIT 19		
	=	RETRY FLIP-FLOP STATE
	=	0 = RETRY FF RESET
	=	1 = RETRY FF SET
X	=	1 OR 0

MV4185

Figure 5-21. SYLLABLE DEPENDENT Interrupt P1 Parameter Word Layout



NOTE

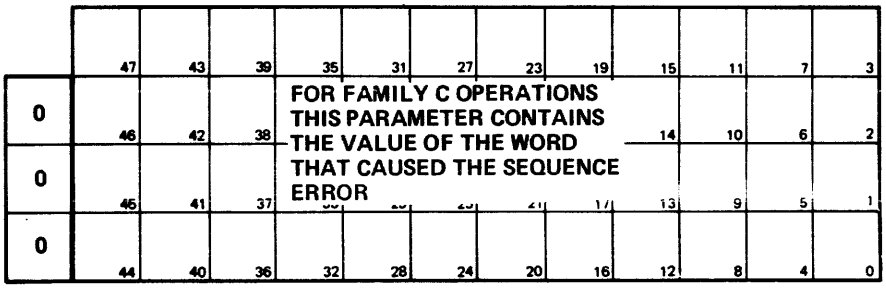
THE FOLLOWING SYLLABLE DEPENDENT INTERRUPTS USE A DIFFERENT P2 PARAMETER FORMAT:

- SEQUENCE ERROR INTERRUPT DURING FAMILY C OPERATIONS.
- SPLT OPERATOR (FAMILY K VARIANT 9543) INTERRUPT.
- JOIN OPERATOR (FAMILY K VARIANT 9542) INTERRUPT.
- SEQUENCE ERROR INTERRUPT FOR STRING OPERATIONS.

SEE SPECIAL P2 PARAMETER WORD LAYOUTS FOR THESE TYPES OF SYLLABLE DEPENDENT INTERRUPTS, WHICH FOLLOW.

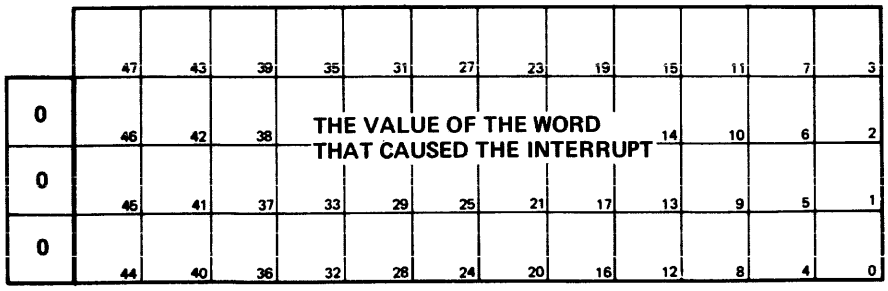
MV4186

Figure 5-22. SYLLABLE DEPENDENT Interrupt P2 Parameter Word Layout



MV4187

Figure 5-23. SYLLABLE DEPENDENT Sequence Error P2 Parameter Word



MV4188

Figure 5-24. SYLLABLE DEPENDENT SPLT (9543) Operator P2 Parameter

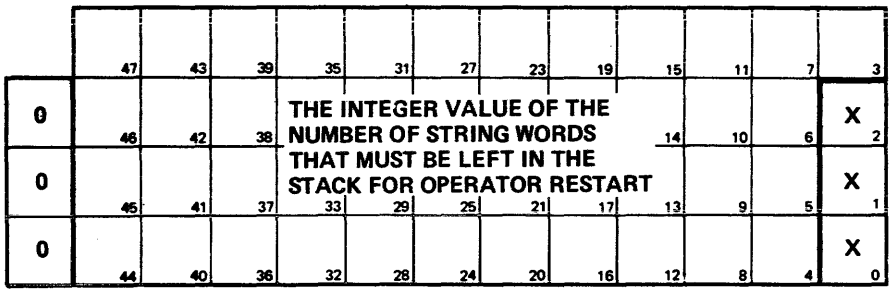


Figure 5-25. SYLLABLE DEPENDENT JOIN (9542) Operator P2 Parameter

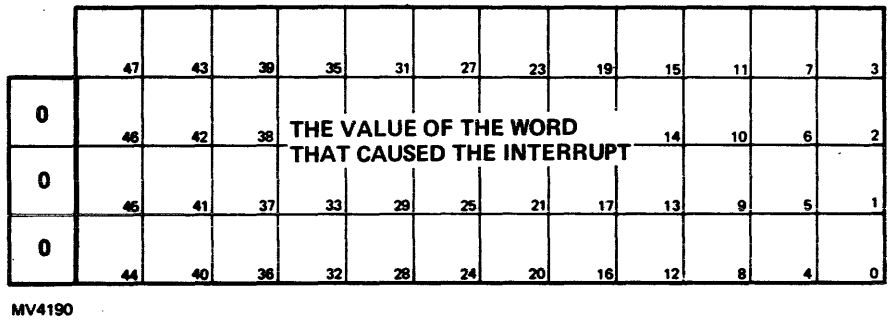
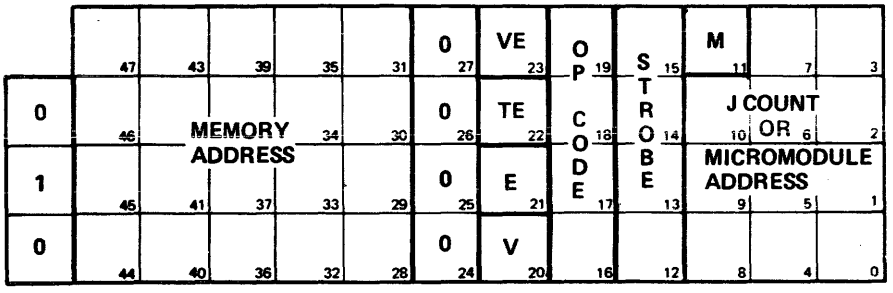


Figure 5-26. SYLLABLE DEPENDENT Segmented Array Interrupt P2 Parameter



VE = VECTOR MODE
TE = TABLE EDIT MODE
E = EDIT MODE
V = VARIANT MODE
M = MODED
= 0 = J COUNT VALUE IS PRESENT
= 1 = MICRO MODULE ADDRESS IS PRESENT

Figure 5-27. SYLLABLE DEPENDENT Interrupt P3 Parameter Word Layout

Table 5-4. SYLLABLE DEPENDENT Interrupt P3 Parameter Fields Usage

Interrupt Type	Fields Present in the P3 Parameter Word
Programmed Operator	OP Code, Strobe, J Count
Memory Protected	OP Code, Strobe, J Count
Invalid Operand	OP Code, Strobe, J Count
Divide By Zero	OP Code, Strobe, J Count
Exponent Overflow	OP Code, Strobe, J Count
Exponent Underflow	OP Code, Strobe, J Count
Invalid Index	OP Code, Strobe, J Count
Integer Overflow	OP Code, Strobe, J Count
Base of Stack	OP Code, Strobe, J Count
Presence Bit	OP Code, Strobe, J Count
Sequence Error	OP Code, Strobe, J Count
Segmented Array	OP Code, Strobe, J Count
Interval Timer	OP Code, Strobe, J Count
Stack Overflow	OP Code, Strobe, J Count
Confidence Error	OP Code, Strobe, J Count

SYLLABLE DEPENDENT INTERRUPT DESCRIPTIONS

A description of each type of SYLLABLE DEPENDENT Interrupt that can be detected by the B 6900 Interrupt Controller follows. These descriptions define the most likely reason for the occurrence of the interrupt, and also describe the condition of the Top-of-Stack at the end of the Interrupt Controller logic operation.

Programmed Operator Interrupt

A Programmed Operator Interrupt is invoked when the Program Controller executes an invalid operator code (see Primary Mode Operator NVLD, Code = FF). This Interrupt is used as a "communicate with system" instruction by a user program.

Memory Protect Interrupt

The memory Protect Interrupt is invoked by the Memory Controller logic, under the following conditions:

1. A STORE, OVERWRITE, READ/LOCK, or STRING TRANSFER operation is attempted using a Data Descriptor that has the READ ONLY bit (Bit-43) set. The operation is terminated prior to the memory access operation, leaving the Data Descriptor (the addressing word) in the A register.

2. A STORE operation is attempted into a memory word address that has the PROTECT-BIT (Bit-48) set. The PROTECT-BIT is detected in the "flashback word" in the C register and, when set, the WRITE operation is not performed. Instead, the original contents of the memory address (the flashback data) is restored in the memory address. The memory address word that was used to access the protected memory word is left in the A register.

Invalid Operand Interrupt

The Invalid Operand Interrupt is invoked when an operator tries to use the wrong type of CONTROL WORD or data word. B 6900 Operators test the TAG fields of all operands used, to insure that the words meet the necessary requirements for the particular type of operation performed. If an operand TAG does not meet the requirements, the FAMILY Control logic invokes the Invalid Operand Interrupt.

Divide-By-Zero Interrupt

The Divide-By-Zero Interrupt is invoked by the Arithmetic Controller, when a Divide operation is attempted with the Divisor equal to zero. The Divide operation is terminated prematurely, leaving the A register cleared and the P1 parameter in the B register. The Program Controller PSR and PIR registers are backed up to point at the Divide Operator code syllable.

Exponent Overflow Interrupt

The Exponent Overflow Interrupt is invoked by the Arithmetic Controller when the positive capacity of an EXPONENT field in an arithmetic operand is exceeded (refer to Exponent Underflow, below). The arithmetic operation in process when the interrupt is invoked is prematurely terminated. The A register is left cleared, and the P1 parameter is left in the B register.

Exponent Underflow Interrupt

The Exponent Underflow Interrupt is similar to an Exponent Overflow Interrupt, except for the value of the Exponent SIGN-BIT. The SIGN-BIT for an Exponent Underflow Interrupt is SET and, consequently, an Exponent Underflow Interrupt indicates that the negative capacity of an operand EXPONENT field has been exceeded (refer to the Exponent Overflow Interrupt subsection of this section).

Invalid Index Interrupt

The Invalid Index Interrupt is invoked when an attempt is made to index a memory address by less-than-zero, or by a value that is equal-to/greater-than the upper-bound (LENGTH) of a Descriptor. Invalid Index Interrupts can be invoked by various operator codes in Families A, B, or C.

Integer Overflow Interrupt

The Integer Overflow Interrupt is invoked when an attempt is made to integerize an operand, and when the integerized value would be greater than the maximum value for an integer. In general, this interrupt occurs during the exponent adjustment part of the Integerize algorithm.

The Family A Integerize operator is frequently pseudo-called by other Family operators. If another family pseudo-calls the Intergerize operator and the Integerize operator fails because of an Integer Overflow condition, the operator that pseudo-called the Integerize operation is terminated and the Integer Overflow Interrupt is invoked.

Bottom Of Stack Interrupt

A Bottom Of Stack Interrupt is invoked when a Family C EXIT or RETURN operator causes the operating program stack pointers to point at the base of the program stack area in memory. If this interrupt is invoked, the Return Control Word (RCW) used during the EXIT or RETURN operation is left in the A register.

Presence Bit Interrupt

A Presence Bit Interrupt is invoked when an attempt is made to access control information or data, and the information or data is not present in local or Global memory. All operator codes that use Data or String Descriptors to address memory have the ability to invoke this interrupt.

Special consideration is given to the class of a Presence Bit Interrupt, to determine whether it is Procedure Dependent or Data Dependent. The two classes of Presence Bit Interrupts require different handling for the Program Controller PIR and PSR register values. Therefore, there are essentially two ways to handle a Presence Bit Interrupt, depending on its class.

DATA-DEPENDENT PRESENCE BIT INTERRUPTS

A data-dependent Presence Bit Interrupt is invoked when the Data Processor is seeking to access data in the currently operating programs' procedure environment. The Presence Bit Interrupt procedure makes the absent data present in system memory, and then the interrupted program procedure is resumed. The Program Controller PIR and PSR register values are backed up by the interrupt procedure to point at the operator that invoked the Presence Bit Interrupt. When the interrupted program procedure is resumed, the operator that invoked the Presence Bit Interrupt is executed again; and this time, the data that was missing is present. Therefore, no Presence Bit Interrupt is invoked.

PROCEDURE-DEPENDENT PRESENCE BIT INTERRUPTS

A procedure-dependent Presence Bit Interrupt is invoked, (1) if the Data Processor is ENTERing into a new operating program procedure, or (2) is EXITing/RETURNing from/to an old operating program procedure and the program procedure is not present in system memory. A procedure-dependent interrupt is also invoked if the Data Processor attempts to access a non-present Segment Descriptor during a display update sequence.

The Presence Bit Interrupt procedure makes the absent segment code present in memory, and terminates by pseudo-calling the EXIT operator flow into operation. The EXIT operator sequences allow the Presence Bit Interrupt procedure to return control of CPU operations to the interrupted procedure with the segment code present in memory.

The selection of an EXIT or RETURN operator to escape from the interrupt handler procedure depends on whether or not a parameter (IRW or Data Descriptor) is left on top of the interrupted program procedure stack. Some types of operators require such a parameter if interrupted for a Presence Bit Interrupt, while others do not. If a parameter is to be left on the stack, then the RETURN operator is used to escape from the interrupt handler procedure; otherwise, the EXIT operator is used.

When a procedure-dependent Presence Bit Interrupt occurs during an ENTER, EXIT, or RETURN operator flow, the first operator of the absent code segment has not yet been fetched from memory. The initial PIR and PSR values from the control word that invoked the procedure (a PCW if the interrupt occurred during an ENTER operator, or an RCW if the interrupt occurred during an EXIT or RETURN operator) are saved in the RCW of the Presence Bit Interrupt stack. The EXIT operator (pseudo-called at the end of the interrupt procedure) references the RCW in the Presence Bit Interrupt stack. Therefore, initial PIR and PSR values of a procedure which is Presence Bit interrupted are saved across the intervening operation of the interrupt handling procedure, and are used when the interrupted procedure is resumed.

Sequence Error Interrupt

A Sequence Error Interrupt is invoked during an attempt to access a Mark Stack Control Word (MSCW), if the word accessed does not have a TAG value of 3 (hex). A Sequence Error implies that the stack linkage mechanism, or the stack history of the stack being accessed is in error. A Sequence Error may occur at various places in the flow of an operation, and may therefore occur before or after the place where the Program Controller register values of PBR, PIR, and PSR are adjusted. If the interrupt is invoked before the Program Controller values are adjusted, bit-24 of the P1 parameter is SET. If the interrupt occurs after the adjustments have been made, then bit-23 is SET in the P1 parameter, and bit-24 is RESET. Bit-23 and bit-24 are never SET at the same time in the P1 parameter word for a Sequence Error Interrupt.

Segmented Array Interrupt

A Segmented Array Interrupt is invoked by a string operator when the upper-limit boundary of an array vector is detected. Array vectors for string operations are divisions of data into groups (segments) of up to 256 data words, bounded by Memory Link words. A Memory Link word is a control word that has the Memory Protect, Bit-48, SET. A segment of string data is accessed by means of a Data Descriptor.

Each word read from memory during a string operation is checked to see whether bit-48 (Memory Protect bit) is SET. If bit-48 is SET, it implies that an upper-limit boundary memory link was accessed, and a Segmented Array Interrupt is invoked.

String operator interrupts leave a special value in the P2 parameter word of the interrupt stack. This P2 value is an integer number that defines how many data words (below the interrupt stack) must be left in memory to restart the string operation.

Restarting a string operation after a Segmented Array Interrupt has occurred implies a new segment of data words has been brought into memory. A data string upon which a string operation is performed may not conform to the 256 word limit for segmented array vectors. This condition, when present, requires that data from the previous segment be present in memory to restart the string operation.

Interval Timer Interrupt

An Interval Timer Interrupt is invoked when the Interval Timer times out. The Interval Timer circuit is located in the MLIP logic, and is initiated into operation by the execution of a Set Interval Timer (SINT, 9545) operator. This timer is used by the system software for time slicing operations. The interrupt from the Interval Timer allows the MCP to detect the end of one time slice, and to begin a new time slice. The SINT operator allows the MCP to control the length of a time slice, by presetting a time counter to a predetermined time increment count. The timer counts from the preset count to a maximum count, and then invokes the Interval Timer Interrupt.

Stack Overflow Interrupt

A Stack Overflow Interrupt is invoked when the IC Memory Address S register value is equal to the IC Memory Address LOSR register value. This interrupt is invoked because the currently operating procedure attempted to utilize more memory space for its program stack than was allocated for the program by setting the value of LOSR.

Confidence Error Interrupt

A Confidence Error Interrupt is invoked when the Confidence test routine is being executed, and a test failure or an error condition is detected by the Confidence test. The Confidence test is automatically initiated into execution when the Data Processor is in an IDLE state, and when a software procedure is not being processed. The occurrence of a Confidence Error Interrupt causes data about the test failure or error condition to be written into the SYSTEM/SUMLOG disk file. B 6900 maintenance personnel utilize the SUMLOG data to analyze system failures and repair system defects.

The Interrupt Controller formats the data for a Confidence-test failure or error into the interrupt stack parameters, and the Interrupt Handling procedure of the MCP writes the parameters into the SYSTEM/SUMLOG disk file.

String Operators

String operators control the character accessing, formatting, and editing capability of the B 6900 system. The string operators are comprised of the operators in strobes F, G, and H, which are grouped in a "super-family" designated family U. Family U operators share a common "T" register (operator code register), a common logical sequence counter, and a common group of logical flip-flops.

The most significant advantage from collecting all string operators into a single super-family is that the common logical functions that all string operators share are not duplicated in each family controller. For instance, all string operators require a method for accessing local memory and for addressing the characters of data within a memory word. A typical string operator must be capable of addressing a number of different words in memory in order to perform an editing operation on a string of data characters. Moreover, once the editing has been performed, the word must be stored in memory so that the same editing can be performed on other words of data. The logic circuits and operator functions required to perform this type of operation are common, and are thus collected into the single super-family U in the B 6900 system.

Memory Controller

The memory controller in the CPU (refer to Figure 5-2) services requests for access to memory resources of the system from the data processor, the look ahead logic, and the MLIP. These three modules are all located within the CPU cabinet, and share a common path to/from memory. Internal logic circuits of the memory controller establish when each of these three modules has priority for accessing system memory resources.

When the MLIP is processing an I/O operator and a need for a burst cycle exists, the MLIP has first priority for a memory access request. This condition causes the data processor to suspend its operation while the MLIP obtains access to memory. The data processor will suspend its operation until the MLIP completes its memory access. At the conclusion of the MLIP memory access operation, the data processor will continue its operations at the place where the suspension occurred.

The order of priority in accessing memory is MLIP, processor, and look ahead logic, in that order.

The memory controller logic has the capability to store two requests for access to memory. The storing of access requests consists of remembering which requests were received over the Z12 memory control bus. The memory controller examines the contents of the two request registers (RQR and RQT) to determine which request has the higher priority for the next access to memory.

The logic mechanism used by the memory controller to remember what memory requestor units require an access to memory consists of two request registers located in the A input logic to the memory control. When a request for a memory access is transmitted to the memory control, the request (bits D:14 on the Z12 bus) is stored in the RQT register (13:14). If RQT contains a request but RQR does not, the request in RQT is placed in RQR. This frees RQT to accept the next memory request in sequence. Each time a memory request is to be processed the memory controller will examine both the RQT and the RQR registers to determine which of two possible requests for access to memory has the higher priority. As one of the two possible memory requests are performed, the stored request information in the RQT register (or alternatively the RQR register) is reset to binary zeroes. This removes a request presently being executed from further contention for an access to memory, and frees the register that was reset to accept a new access request.

The memory controller monitors all memory requests for errors. If an error condition is detected during a memory bus operation, the memory controller will cause an interrupt to be present in the data processor interrupt controller. The memory controller passes parameters that describe the type of interrupt that occurred to the interrupt controller. The interrupt handling procedure of the MCP causes the interrupt parameters from the memory controller to be written in the SYSTEM/SUMLOG, thus preserving a record of memory errors.

Control State/Normal State

A B 6900 data processor has the ability to perform in either normal or control state. In control state, all external interrupts are inhibited, and a few privileged operators are enabled. The Inhibit Interrupt Flip-Flop (IIHF) must be set for processing to occur in control state.

The data processor switches to control state upon entering a procedure by means of a control state program control word (PCW).

MESSAGE LEVEL INTERFACE PROCESSOR

The Message Level Interface Processor MLIP module (see Figure 5-2) is essentially a peripheral device path control mechanism. The MLIP is semi-independent, and can initiate an I/O device path control function only in response to execution of a Communicate Universal I/O (CUIO) operator by the Program Controller. The MLIP logic proceeds in an independent manner after it is initiated by a CUIO operator, until the I/O device operation is terminated. The MLIP logic, if specified, causes an external interrupt in the Interrupt Controller logic upon termination of an I/O device operation.

The MLIP performs additional system functions, such as establishing the general environment for I/O path control, and system-timing functions. The I/O path control environment is established by use of path control logic circuits that can be SET/RESET by specific MLIP control logic. System-timing functions such as Time-of-Day and Interval-Timer logic are controlled by execution of specific system Program Controller operator codes. The MLIP also contains timer circuits which are automatic features of the I/O path control logic.

MLIP CONTROL OPERATIONS

MLIP operations are controlled by micro-code sequences contained in the micro-module. When an MLIP operator code is executed by the Program Controller, logic circuits in the MLIP generate a micro-module address, called an Entry Vector. Entry Vectors are sent to the address logic of the micro-module, where they select the first sequence address for a particular MLIP micro-code function.

The micro-code sequences for MLIP functions are subroutines and may or may not be executed, depending on current logical conditions present in the MLIP. Logic signals representing current logic conditions in the MLIP are present at the input addressing logic of the micro-module. The state of these logic signals is used to alter the sequential addressing of the micro-module. By altering the sequential addressing of the micro-module, various subroutines of MLIP control programs are entered into or returned from. The execution of an MLIP directing operator code by the Program Controller selects the particular MLIP function to be performed. Subsequently, MLIP logic conditions specify which subroutines of the MLIP function micro-code are performed.

I/O DEVICE CONTROL OPERATIONS

The MLIP is not the final control mechanism for the operation of a peripheral device; it is an intermediate control mechanism. System control of peripheral device operations is shared by the MLIP and a UIO Data Link Processor (UIO-DLP). The UIO-DLP device is the final control mechanism for operation of an I/O device. Once a UIO-DLP device is initiated into operation, the MLIP becomes transparent to the flow of peripheral data between the B 6900 system and the UIO-DLP device. However, the MLIP continues to control the interfaces between the UIO-DLP device and B 6900 system memory.

UIO-DLPs are semi-independent peripheral control devices. A UIO-DLP can only control a single type of I/O device and can only initiate a device into operation when the operation is specified by signal inputs from the MLIP. Once a UIO-DLP begins operation of a peripheral device, it proceeds under the control of its own internal logic. The UIO-DLP only communicates with the MLIP to send/receive peripheral device data and to report the status of the UIO subsystem when the UIO-DLP operation is terminated.

A MLIP communicates with UIO-DLP devices by means of a Message Level Interface (MLI) cable connection. The MLIP contains eight MLI ports which are used as channels for interfacing various organizations of IODC modules to the B 6900 CPU. The MLIP also contains extensive logic circuits to control communications over the MLI ports, and to establish priorities for use of the MLI interfaces between the IODC modules and UIO-DLP devices that are currently in use.

A communication between the MLIP and a UIO-DLP device must be initiated by the MLIP. Such a communication is interrupted while the UIO-DLP is performing its independent processes and is reestablished when it is necessary to pass data or result status between the MLIP and UIO-DLP. Either the MLIP or the UIO-DLP can initiate the resumption of an interrupted communication over the MLI interface.

MLIP SIMPLIFIED LOGIC CIRCUITS

Figure 5-28 is a simplified schematic of the MLIP module. The schematic shows the major circuits of the MLIP and, in general implies some of the relationships between these major circuits. The circuits and relationships of MLIP circuits defined in the following paragraphs of this manual can be better understood by referring to this figure.

MLIP INTERFACES

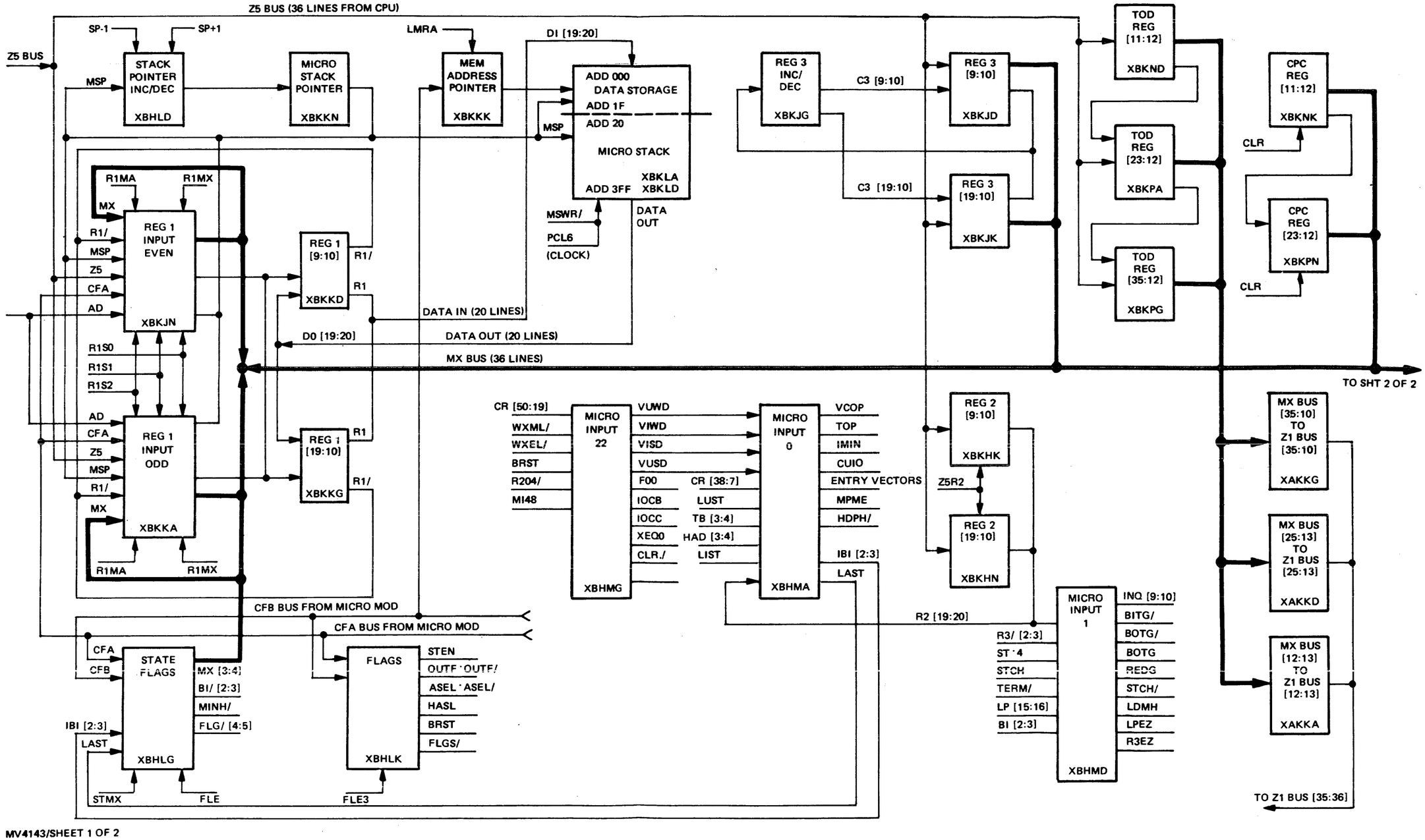
The MLIP has three interfaces to other modules of the B 6900 system. These interfaces connect the MLIP:

1. To the Data Processor.
2. To the micro-module.
3. To the IODC module(s).

The interface between the Data Processor and the MLIP includes the path between the MLIP and system memory. The Data Processor utilizes this path to communicate instructions and control data to the MLIP. The MLIP utilizes this path to access system memory (through the logic of the Transfer and Memory Controllers).

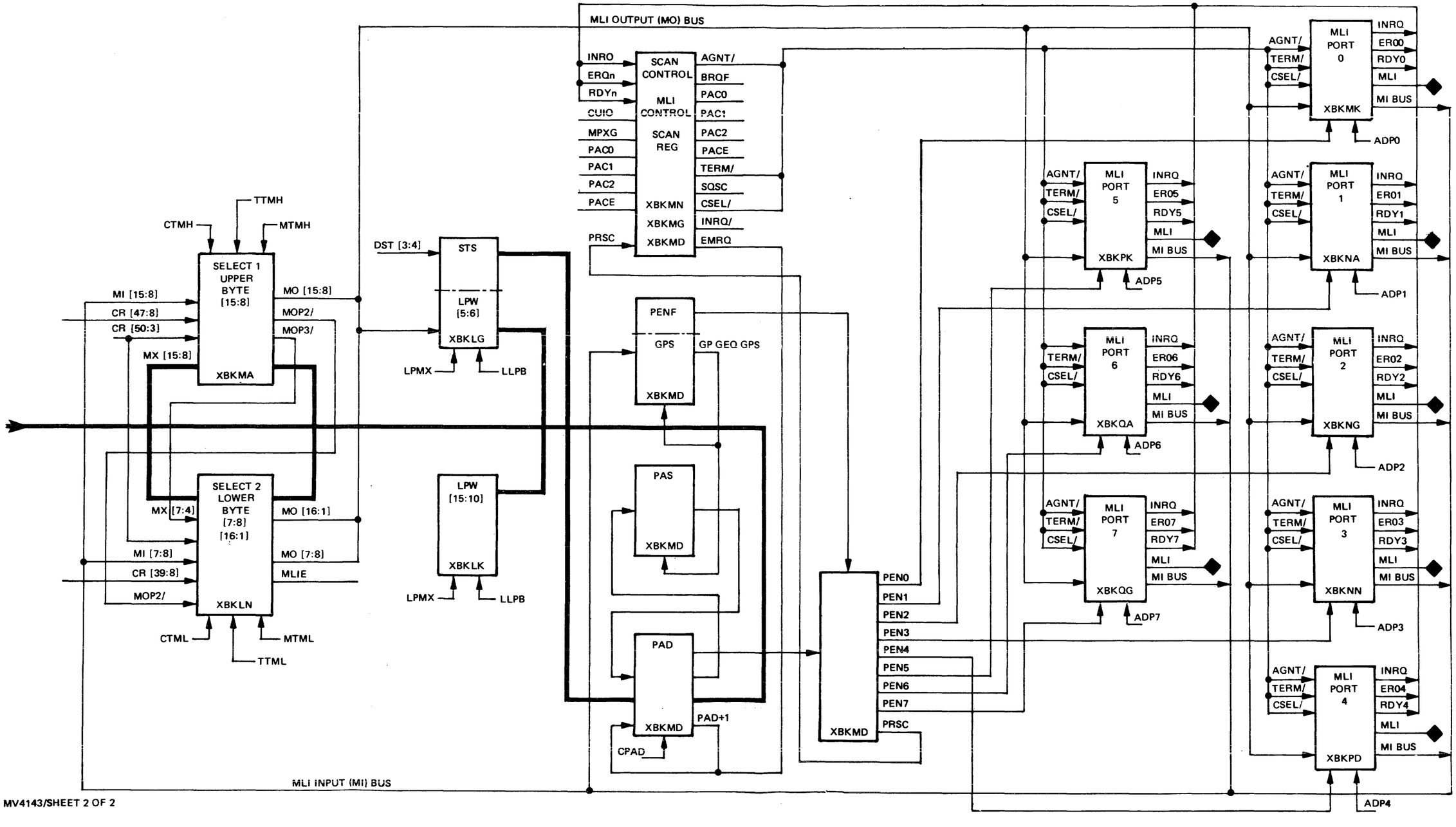
The interface between the micro-module and the MLIP module is used to send control signals, data, and micro-module addresses from the micro-module to the MLIP logic. The MLIP uses this interface to transmit Entry Vector addresses to the micro-module. Various MLIP logic signal levels are present at the address inputs to the micro-module, in addition to the standard interface connections.

The interface(s) between the MLIP and the IODC modules are Message Level Interface (MLI) interfaces. There are as many as eight separate MLI interfaces, which are used to provide communication paths between the MLIP module and the IODC modules of the system.



MV4143/SHEET 1 OF 2

Figure 5-28. MLIP Simplified Schematic (1 of 2)



MV4143/SHEET 2 OF 2

Figure 5-28. MLIP Simplified Schematic (2 of 2)

MLIP To Data Processor Interfaces

Figure 5-29 shows the interface between the MLIP and the Top-of-Stack registers of the CPU. The MLIP sends data to the Top-of-Stack over the 52-bit Z1 bus, and receives data from the Top-of-Stack over the 52-bit Z5 bus. A special 19-bit bus is used to transfer the TAG Field and the high-order 16-bits of data from the Top-of-Stack C register to the MLIP. The MLIP also has a control signal interface to the Interrupt Controller through which it can initiate an External Interrupt when an I/O operation is terminated. In addition, various logic signals from the Program, Memory, and Transfer Controllers, and signals from C and K Families are routed to the logic of the MLIP.

The Z1 and Z5 busses are shared by the Data Processor and the MLIP modules. The special 19-bit C register bus is not shared, and only transfers information in one direction, from the C register to the MLIP.

The logic of the Transfer Controller and the Memory Controller cause a connection between the Z1/Z5 busses and the Z3/Z4 busses for MLIP module memory operations. This connection is explained in greater detail later in this section.

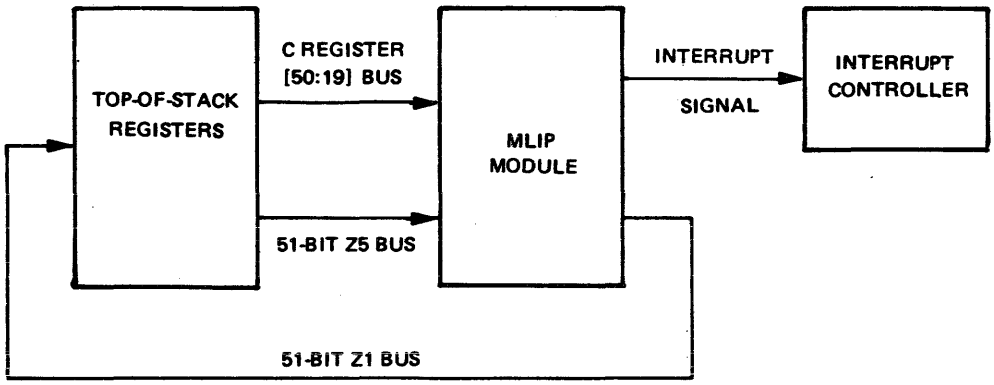


Figure 5-29. Interface Between MLIP and Top-of-Stack

MLIP To Micro-Module Interfaces

Figure 5-30 shows the interfaces between the micro-module and the MLIP. The interface between the MLIP and the micro-module includes an 8-bit Control Field A (CFAn) bus, a 5-bit Control Field B (CFBn) bus, and a special 12-bit micro-code address bus. The CFAn and CFBn busses only transfer data in a single direction, from the micro-module

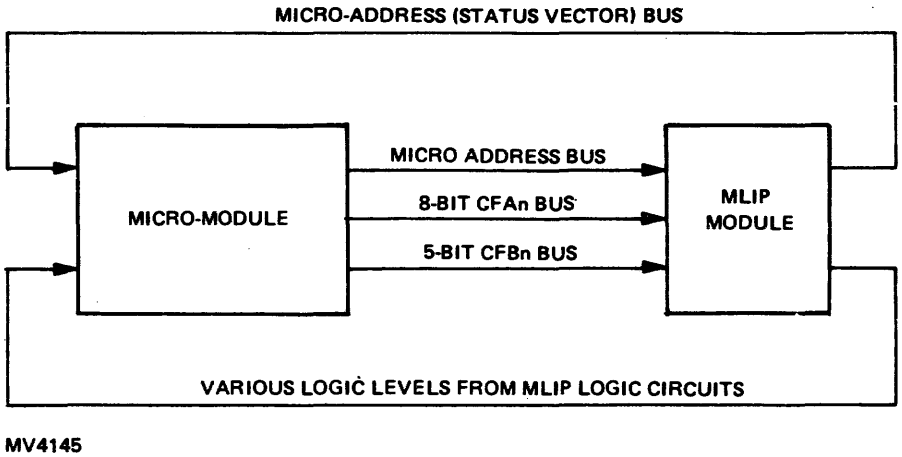


Figure 5-30. MLIP to Micro-Module Interfaces

logic to the MLIP. The 12-bit micro-code address bus is used to transfer Entry Vector data from the MLIP logic to the micro-module, and also to transfer micro-code sequence counts from the micro-module to the MLIP. These busses are internal logic circuits of the CPU cabinet and do not use external bus cable connections.

The CPU micro-module code contains the process flows for all MLIP functions. An MLIP function is a single complete MLIP operation that includes all the options, variations, and error-handling processes for the function. A program flow for an MLIP function varies dynamically; that is, the micro-code program takes branches within a process flow based upon the value of various logic signals which the micro-module receives from the MLIP module and/or the data processor module.

The MLIP receives the sequence flow address from the micro-module for the current MLIP operation sequence. The MLIP must know its sequence flow address in the micro-module so that in the event of an interrupt (caused by an error condition in the MLIP logic), the interrupt parameters contain the micro-code address of the point in the MLIP sequence flow at which the interrupt occurred.

In addition to the listed interfaces between the micro-module and the MLIP, various logic levels of the MLIP are also present at the address inputs to the micro-module. These levels are used to modify the next micro-module sequence address, thereby implementing the subroutine calling procedures of MLIP control micro-code.

An entry vector is the starting address in the micro-module for an MLIP operation sequence. Entry vectors are transmitted to the micro-module to select and start the operation of an MLIP control sequence. The value of the 12-bit entry vector determines which sequence is selected, and the occurrence of the entry vector on the bus determines when the operation sequence starts to execute.

MLIP To Peripheral Device Interfaces

The MLIP logic contains from 1-to-8 external cable interface port connections to the Universal I/O Base (IODC) modules (refer to Figure 5-31). At least one of the interface (MLI) ports must connect the MLIP module to an IODC module. Each MLI interface consists of a 25-signal cable connection.

Each MLI interface connection can conduct communications between the MLIP module and up to eight IODC modules. If multiple IODC Base modules are connected to an MLI interface, then the IODC modules are interconnected by extensions (Line Expansion Modules or LEMs) to the MLI interface bus.

Each IODC module can contain up to eight Data Link Processor (DLP) devices.

MLIP GENERAL OPERATING CHARACTERISTICS

The MLIP module receives STRC, STRK, VARF, and Family T-register value signals from the Data Processor module. The MLIP logic interprets the T-register values for each Variant Mode Family C and K operator, and detects operator codes that initiate MLIP micro-code functions.

The Variant operators that initiate MLIP micro logic are:

a.	Family C	95A7 RTOD	Read Time-of-Day
b.	Family K	9540 RCPC	Read Central Processor Count
c.	Family K	9541 RUNI	Set Running Timer
d.	Family K	9549 WTOD	Write Time-of-Day
e.	Family K	954C CUIO	Control Universal Input Output

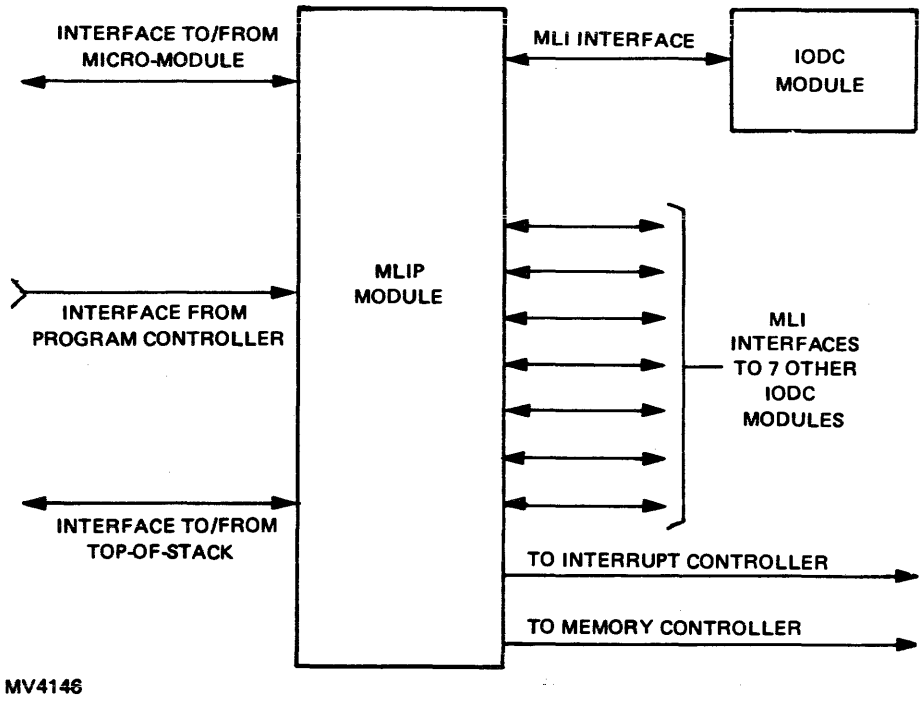


Figure 5-31. MLIP to Peripheral Subsystem Interface

When an operator code that initiates the MLIP logic is detected, the MLIP generates an Entry Vector into the micro-module. An Entry Vector is essentially a beginning address in the micro-module of the micro-code for the detected MLIP function. As the micro-module proceeds through the MLIP function code flow, it returns control information for the operation to the MLIP, in the proper sequence. Thus, the MLIP detects the requirements for its own functions and initiates the micro-module to the proper address (Entry Vector) for each of its functions.

Processor Timer Operation

Figure 5-28 shows that the MLIP contains Processor Timer logic circuits. When an RCPC operator is detected by the MLIP, the value of the Processor Timer logic is returned over the Z1 bus to a Top-of-Stack register. The value returned to the Data Processor is a 24-bit binary field that represents elapsed time in 2.4 microsecond increments up to about 40 seconds maximum. The count does not contain time that is expended by the Data Processor or the MLIP for accessing system memory. The system software uses this count value in computing billing costs for various users of the B 6900 system resources.

When an RCPC operator is detected, the 24-bit Processor Timer counter is RESET to a count of zero, and begins to count up to 2.4 microsecond increments. The counter is inhibited from counting up when signal RCPI is TRUE (while a CPU memory cycle is in process). If the counter is full (all bits contain binary 1's), it steps through the count of zero and continues counting.

Time-of-Day Operation

Figure 5-28 shows that the MLIP module contains Time-of-Day (TOD) logic. The TOD logic consists of a 36-bit counter that counts time in 2.4 microsecond increments. The counter can be initialized to any selected count value, after which it proceeds to increment the count value. The TOD counter cycles, so that counting does not stop when the counter is full. Instead, it counts through zero and continues.

When the MLIP module detects a Family K RTOD operator, the current value of the TOD counter is returned to a Data Processor Top-of-Stack register through the Z1 bus logic. When a WTOD operator is detected, a 36-bit value on the Z5 bus initializes the counter value of the TOD logic, after which counting continues at the new value.

Running Timer Operation

The Running Timer causes the Running Indicator to illuminate when the timer is counting (has not timed-out). The Running Indicator is used to show that the B 6900 system CPU is functioning.

This timer counts clock periods for 2.04 1+/- 0.16 seconds and then times-out, unless it is RESET. The System Running (SRUN) signal, from the micro-module to the MLIP, RESETs the timer. When SRUN goes TRUE, the Running Timer is RESET, thus beginning a new timing sequence. Under normal system operating conditions, the timer never times out; thus, the Running Indicator is continuously illuminated.

The Running Indicator is important for B 6900 system operations because during certain previleged types of operation, the system operator has no other way of knowing whether or not the system has halted. The CPU micro-module is functioning during previleged operations, and by use of the Running Indicator shows the true processing state of the system.

Other MLIP Timer Operations

The MLIP logic contains and operates other timing devices for the B 6900 system. These other timing circuits are defined and discussed separately because they are not triggered into operation directly, as a result of the MLIP decoding a Data Processor operator. These other timer functions of the MLIP logic are:

1. The LOOP timer.
2. The Interval Timer.
3. The Base Busy Timer.
4. The Ready Timer.

LOOP TIMER

The LOOP Timer is used to cause an ALARM type interrupt when the Data Processor operating program is detected to be trapped in a program operator flow. The operating program is trapped if a selected system condition does not occur before the LOOP Timer times-out.

The LOOP Timer counts clock pulses and times-out in 2.04 1+/- 0.16 seconds, unless it is RESET. The timer is RESET by any one of eight different conditions being present. When the timer is RESET, counting starts and continues until either the timer times-out or until another RESET occurs.

The system conditions that cause the timer to RESET are:

1. A family operator completes, and there are more family operators present in the P register waiting to be executed.
2. The MLIP logic receives control of the memory interface to access system memory.
3. The LOOP Timer RESET signal from the Data Processor is TRUE.
4. The Conditional Halt logic detects a Conditional Halt state to be TRUE.

5. The Data Processor is HALTed.

6. The Data Processor is performing an IDLE operator.

7. The Maintenance Display Processor (MDP) is scanning the state of CPU flip-flops to update the display or control of CPU logic signals.

8. A CPU LOAD function is in process.

INTERVAL TIMER

The Interval Timer circuit is used by the system software to cause a Syllable Dependent Interrupt condition after a given time interval has passed. The software operating system uses the interrupt from the Interval Timer as a key for interlacing software programs that are operating in a multi-processing environment.

The Interval Timer counts system clock pulses and times-out 500 1+/- 38.4 microseconds after the Start Interval Timer (STIT) signal triggers the timer into operation. When the timer times-out, the Interval Timer Interrupt (ITIN) signal is generated and returned to the Interrupt Controller logic.

BASE BUSY TIMER

The MLIP accounts for the fact that a IODC module may go "Busy" during an MLIP I/O control sequence over its MLI interface. If this condition occurs, it hangs the MLIP and suspends further system I/O operations until the MLIP is disengaged from the IODC module MLI interface.

The Base Busy Timer circuit provides the method for disconnecting the MLIP from an MLI with which it is hung. The Base Busy Timer limits the length of time such a condition can exist to 2.04 1+/- 0.6 seconds.

When the MLIP connects to one of its MLI ports, the Base Busy Timer is triggered into timing operation by signal BBTR. If the timer circuit times-out before another BBTR signal occurs (while the MLIP is still connected to the MLI) the Base Busy Time-Out (BBTO) signal is generated to cause an MLIP fault interrupt condition in the Interrupt Controller logic. The resultant Interrupt Controller operation disconnects the MLIP from the MLI to which it is connected.

READY TIMER

An MLIP accounts for the fact that a UIO-DLP module may be connected to another MLI and unable to respond to a POLL-TEST operation. Such a condition results in the IODC module returning a NOT READY Result Descriptor in response to the HDPs POLL-TEST operation sequence. When this condition occurs, the MLIP waits for the UIO-DLP to finish its current operation and respond to the POLL-TEST. If the UIO-DLP becomes READY, the MLIP proceeds to complete the POLL-TEST operation sequence and to initialize the UIO-DLP for a subsequent I/O operation.

The Ready Timer circuit is used to limit how long an MLIP waits for a UIO-DLP to respond to a POLL-TEST request. When the MLIP first attempts to execute the POLL-TEST request, the Ready Timer circuit is triggered into operation. If the timer circuit times out before the UIO-DLP responds to the POLL-TEST request, the MLIP aborts the POLL-TEST request, generates a NOT READY Result Descriptor for the UIO-DLP, and sends an I/O finished interrupt to the Interrupt Controller.

The Ready Timer circuit counts clock pulses from the time that the MLIP initiates the POLL-TEST request (triggered by signal RYTR), until the MLIP receives the READY (signal RDY..OK) response from the IODC, or times-out. The timer times-out 8.0 1+/- 0.6 milliseconds after it is triggered, unless the RDY..OK response is received.

Peripheral Device Operation

When the MLIP detects a family K CUIO operator, the logic circuits of the MLIP generate an entry vector to the micro-module, to start the operation of an MLIP Universal I/O device operation sequence. This type of MLIP operating sequence is defined in the following text.

During normal B 6900 system operations, the MLIP module operates to relieve the Master Control Program of the responsibility for controlling the operations of system I/O devices and controls. The MCP specifies by certain data in system memory:

a. The particular I/O device that is to be operated.

b. The particular type of operation that the device is to perform.

c. The expected result status that the I/O device is to return to the system.

d. The location of the data buffer in system memory that is to be used for the I/O device operation.

e. The maximum length of data records to be handled by the I/O device.

f. How many I/O operations are to be performed without an interruption to the system (providing that the I/O device or the MLIP does not encounter an error condition).

g. Where the I/O device result descriptor is to be stored in system memory.

h. The point in a series of I/O operations at which the attention of the MCP is to be obtained.

i. The particular path to be used to interface to the I/O device.

PRIORITY SEQUENCER OPERATIONS IN THE MLIP

The MLIP module contains Priority Sequencing logic circuits that act as the overall operational control for normal MLIP I/O operations. The Priority Sequencer logic (see Figure 5-32) controls the ordering of MLIP functions that originate from requests by the B 6900 software system, or requests that originate in the UIO peripheral subsystem.

The Priority Sequencer logic consists of a 5-bit counter which steps through the sequences that are conditioned by logic signals from various circuits in the MLIP. The Priority Sequences determine when the MLIP is to respond to a CUIO operation by the CPU, or when it responds to an input POLL-REQUEST operation by a IODC module. The Priority logic resolves the priority between a CUIO operator and a POLL-REQUEST that are present at the MLIP at the same time. In addition, the Priority Sequencer resolves priorities between simultaneous POLL-REQUESTs originating from two or more IODC modules, or between two or more UIO-DLPs within the same IODC module.

A POLL-REQUEST sequence for a IODC is required when a UIO-DLP in the IODC executes certain sequence counts of its control logic. IODC modules monitor the sequence counts of the UIO-DLPs located in the module, and strobe onto the MLI interface the sequence count of the highest priority UIO-DLP needing a POLL-REQUEST sequence. In addition to the sequence count, the IODC strobes the Global-priority value for the DLP onto the MLI interface. The MLIP logic monitors the sequence counts and Global-priorities present at its MLI ports, and generates an Entry Vector to the micro-module for a POLL REQUEST sequence to the highest priority UIO-DLP needing a POLL REQUEST sequence.

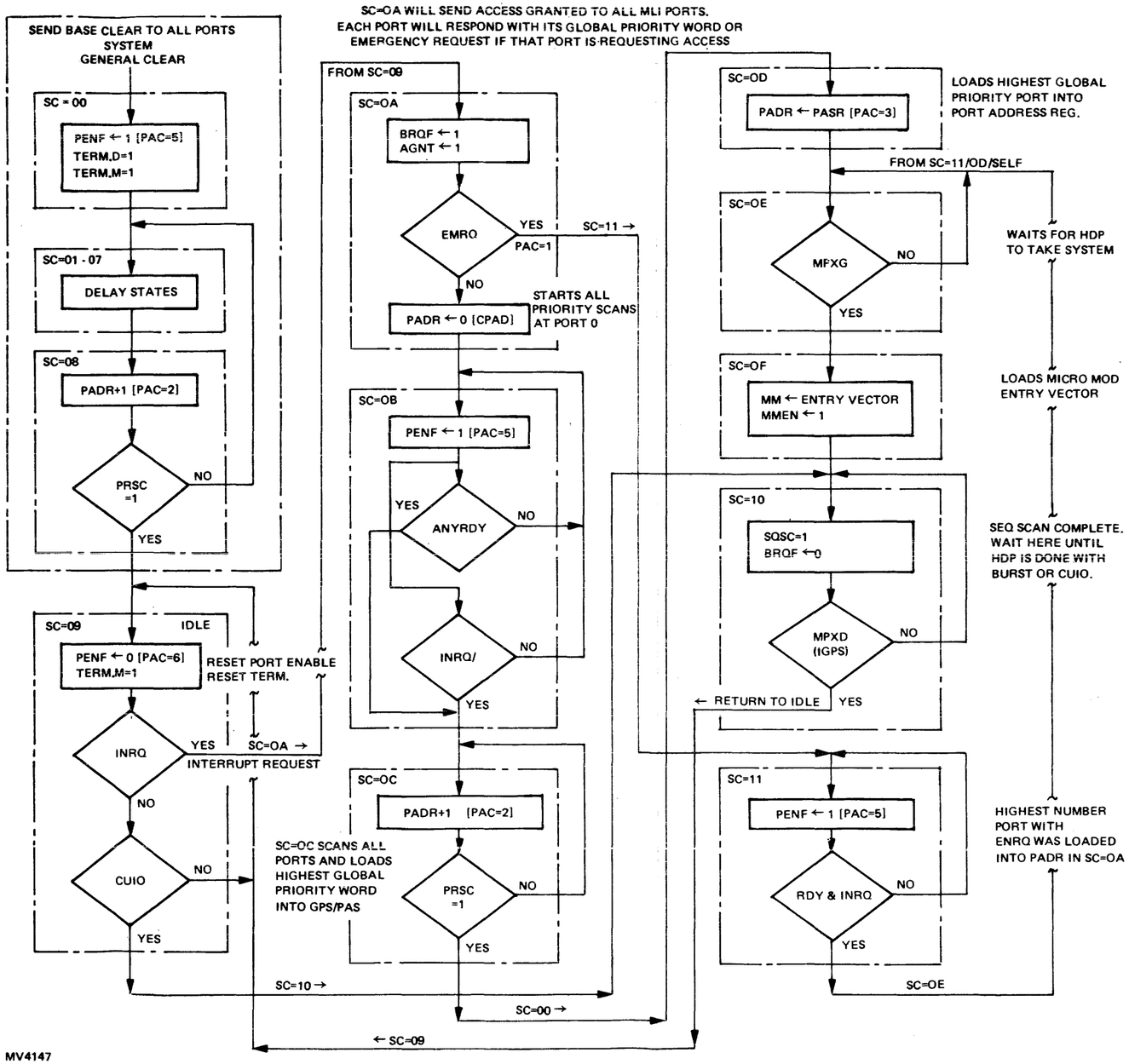


Figure 5-32. Priority Sequencer Sequences

I/O OPERATION INITIATION PROCESSES IN THE MLIP

The specifications for an I/O operation are located in system memory in fixed queues of information called I/O Control Blocks (IOCBs). An IOCB contains at least 15 consecutive words of memory and may contain more words for MCP software purposes. The first fifteen words in an IOCB contain I/O control data in fixed word/field formats (refer to Figure 5-33). The data in an IOCB is used by the MLIP to initiate and control a particular I/O operation. The MCP forms an IOCB and places the required data for an I/O operation in the word fields of the IOCB, before the MLIP is initiated by the CUIO operator.

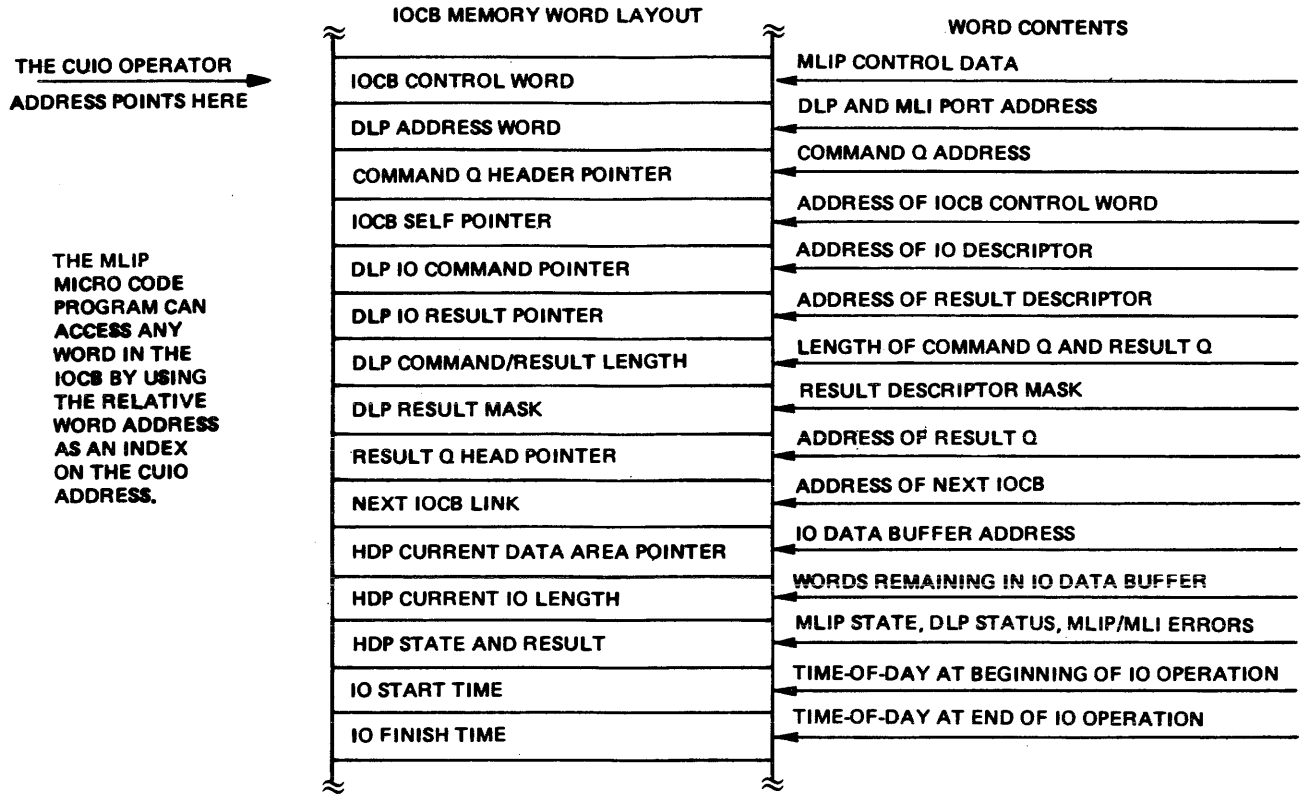


Figure 5-33. B 6900 IOCB Memory Word Layout

The IOCB area in system memory is used jointly by the MCP and the MLIP. After the MCP has created an IOCB and placed I/O control data in it, the MCP causes the Data Processor to execute a CUIO operator. The execution of the CUIO operator causes the MLIP to access the data in the IOCB and to begin the I/O device operation that is specified there. Once the MLIP is initiated into operation of an I/O device from data in an IOCB, that IOCB is controlled by the MLIP and not by the MCP.

A CUIO operator serves two purposes. It initiates the MLIP I/O control logic into operation. It also provides the address of the first word in the IOCB, where the MLIP finds control data for the I/O operation.

The MLIP, when directed, informs the MCP that an I/O operation is terminated by causing an I/O Finished External Interrupt to occur. This interrupt subsequently causes the MCP to examine the State And Result Word for the I/O operation, which is located in the IOCB. The MLIP writes a State And Result Word in the IOCB before a possible I/O Finished Interrupt is generated as part of the normal process for terminating peripheral device operations.

MLIP INITIATION OF THE COMMAND QUEUE STRUCTURE IN MEMORY

An I/O operation is initiated when the logic of the MLIP causes the IOCB to be linked into a Command Queue structure in system memory. A Command Queue (see Figure 5-34) is an organization of IOCBs that are scheduled to be initiated by the MLIP. The MLIP maintains the Command Queue structure for controlling the ordering of multiple I/O device operations in a dynamic system operating environment.

Linking an IOCB into a Command Queue structure is accomplished by inserting Next IOCB Links (memory address pointers) in all queue IOCBs. The Next IOCB Link of an IOCB points to the address of the Next IOCB in sequence in the Command Queue. If no other IOCB is present in the queue or if this is the last/only IOCB in the queue, then the Next IOCB Link word contains an operand with a value of zero.

The control logic of the MLIP scans the contents of all Command Queues periodically and attempts to initiate I/O devices for which IOCBs are present in the Command Queue. When an I/O device DLP is successfully initiated, the MLIP delinks the IOCB from the Command Queue by altering the NEXT LINK words of other IOCBs in the Queue. The NEXT LINK word of an IOCB that has been successfully initiated is replaced by an operand with an integer value of 1.

Horizontal Command Queue operations are an automatic subroutine of the MLIP INITIATE-DLP function. The MLIP logic automatically attempts to initiate the UIO-DLP device at the conclusion of the ENQUEUE-IOCB function. If a UIO-DLP is BUSY when the MLIP attempts to initiate it and if the Command Queue can be horizontally queued, then the MLIP ENQUEUE-HORIZONTAL sequence subroutine is invoked. If a UIO-DLP is not BUSY when the MLIP attempts to initiate it and if a successful connection to the UIO-DLP is completed, then the IOCB is delinked from the Command Queue by the INITIATE-DLP function as described previously.

Figure 5-35 shows how control-information inputs to the MLIP logic initiate an MLIP operation. The MICRO INPUT 0 block detects an MLIP function command from Program Controller input signals and causes an Entry Vector input to the micro-module. The memory address of an IOCB is present at the Z5-bus input to REGISTER 1 when the control signals from the Program Controller are present at MICRO INPUT 0.

The micro-module subsequently causes the MICRO OUTPUT block to access the IOCB in system memory, and obtain control information about the MLIP operation to be performed. The IOCB memory information appears on the Z5-bus inputs to REGISTERs 1, 2, and 3. IOCB memory data is fetched into REGISTER 1 and stored in the MLIP RAM memory where it is available for subsequent use by the MLIP logic. Memory information present in REGISTER 2 comes from the IOCB Command Word shown in Figure 5-36. The contents of REGISTER 2 are passed to the micro-module through the combined logic of MICRO INPUTs 0, 1, and 2. REGISTER 3 receives I/O LENGTH data from the LENGTH-Field of word 10 in the IOCB.

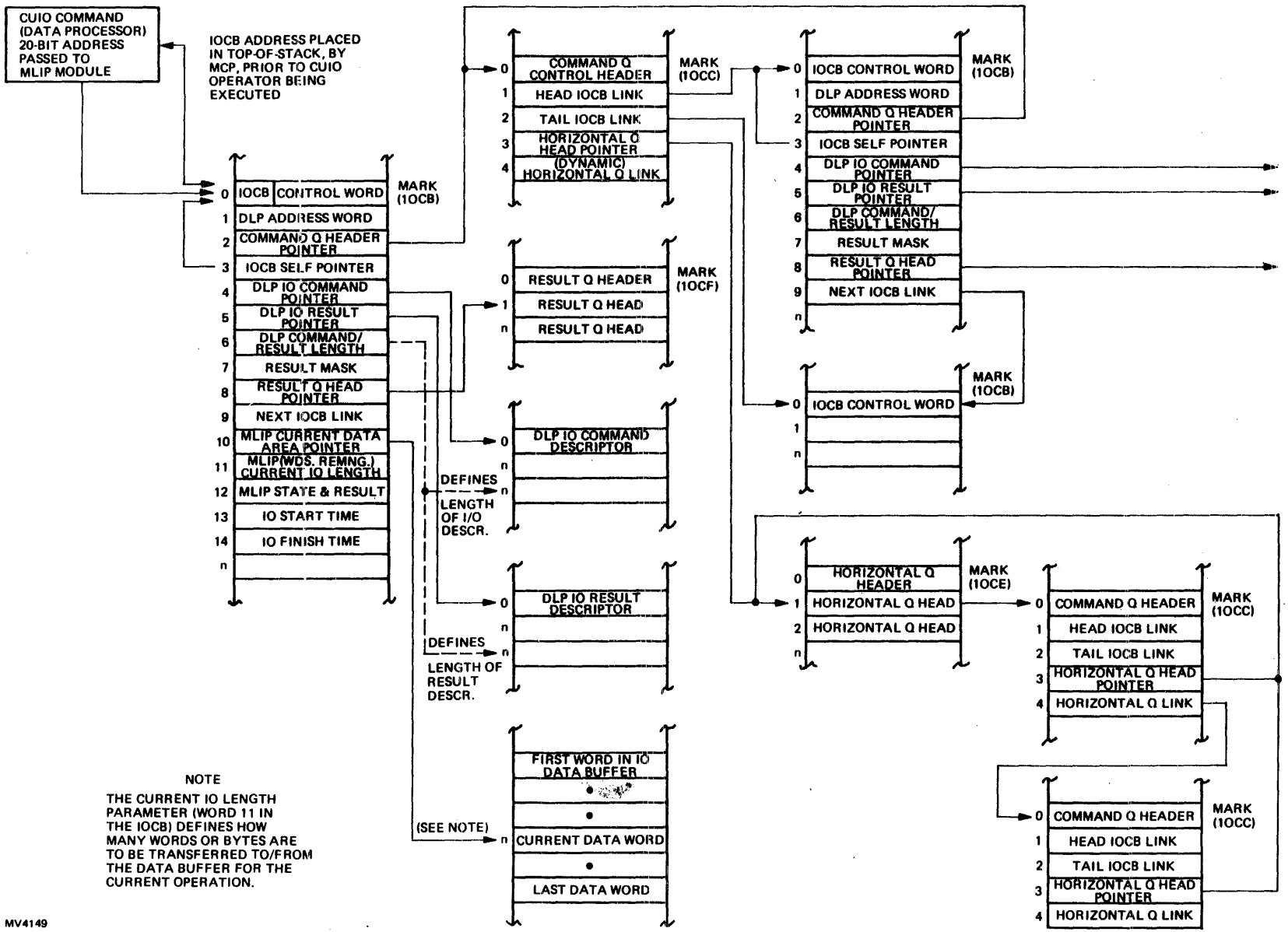


Figure 5-34. MLIP Command Queue Structures

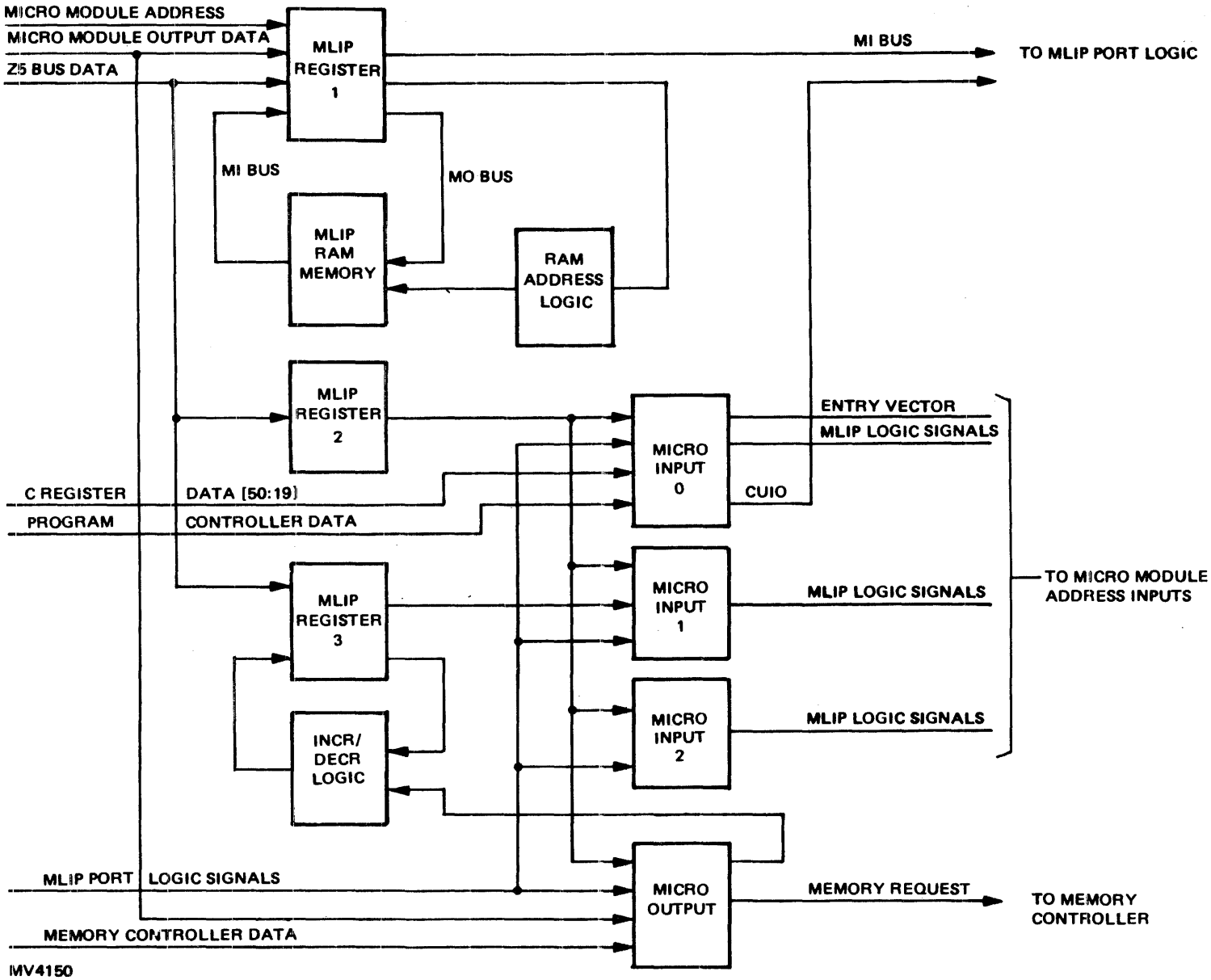


Figure 5-35. MLIP System Control Function Diagram

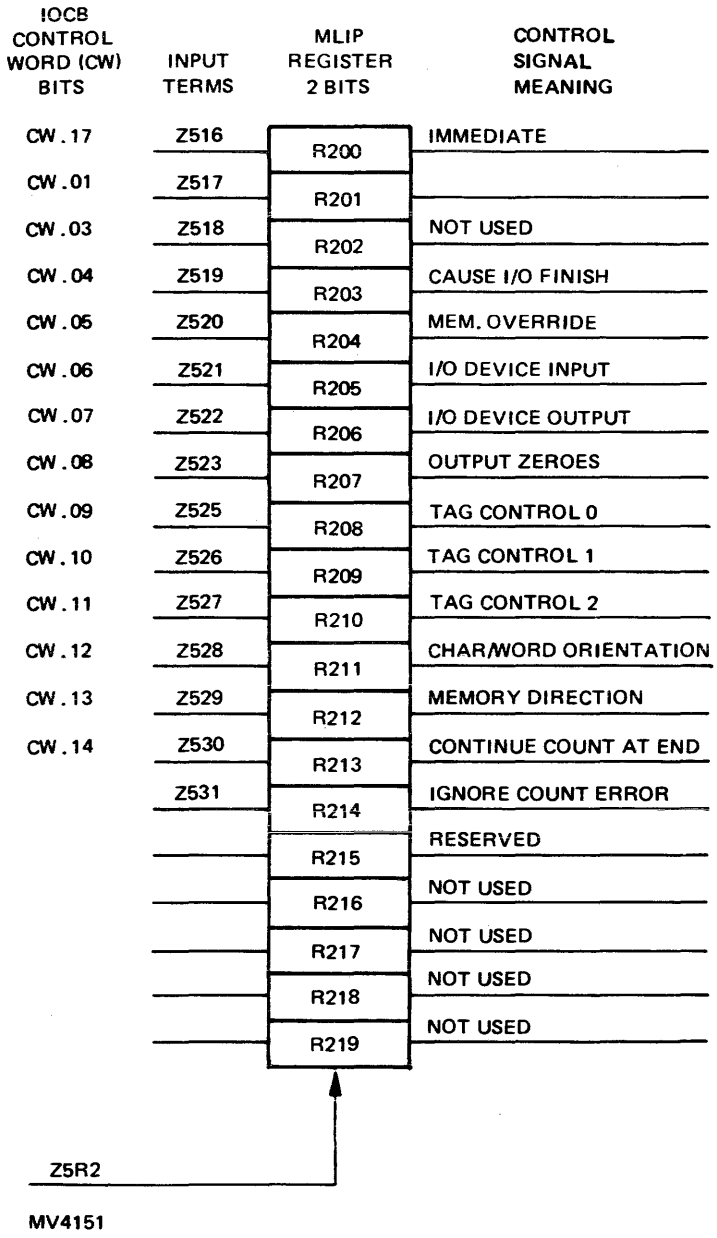


Figure 5-36. MLIP Register 2 Function Control Logic

Figure 5-37 shows how control information and data pass through the MLIP logic to/from the UIO-DLP subsystem. The CUIO signal comes from the MICRO INPUT 0 logic shown in Figure 5-35. The IOCB DLP ADDRESS comes from bits [10:3] of the DLP Address Word in the IOCB and is used to select 1 of 8 MLI Port Adapters. The MLIP initiates an I/O Command by passing control signals and data through the MLI SELECT logic to the selected MLI Port adapter. UIO-DLP Commands and output data are passed to the Port Adapters by means of the MX-bus and MO-bus. Input data and Result Status data are received from the UIO-DLP by means of the MI-bus and MX-bus. The MLIP logic creates interfaces between its MX-bus and the Z1/Z5 busses, and between the MX-bus and the special 19-bit C REGISTER bus.

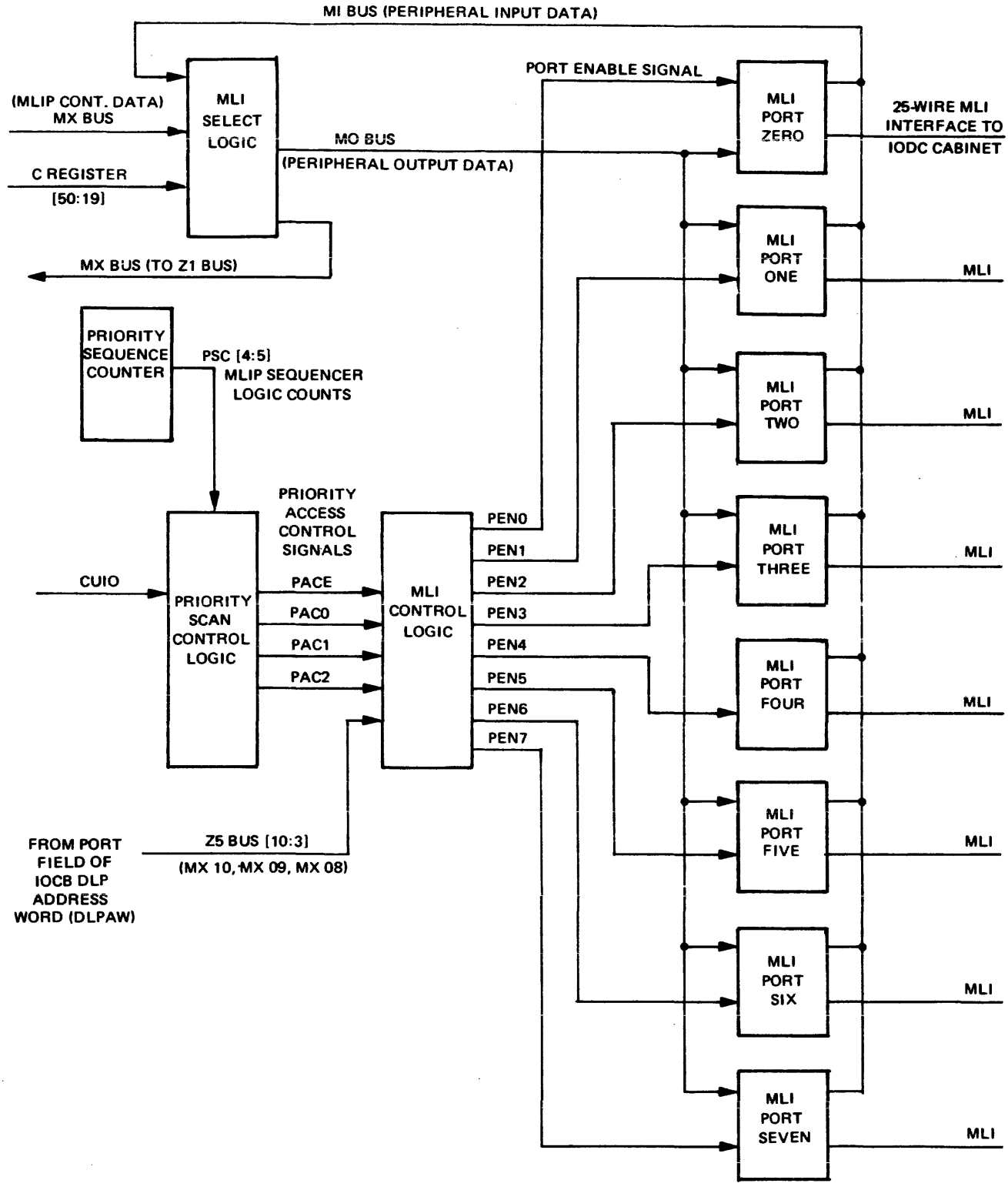


Figure 5-37. MLIP Port Control Function Diagram

MLIP RAM MEMORY OPERATIONS

The MLIP contains a Random Access Memory (RAM), used to store data that is pertinent to the current I/O device operation (see Figure 5-38). This RAM contains 1024 20-bit memory words which are divided into a Data Storage section of 32 words (in addresses 0-31), and a micro-stack section of 992 words (in addresses 32-1023).

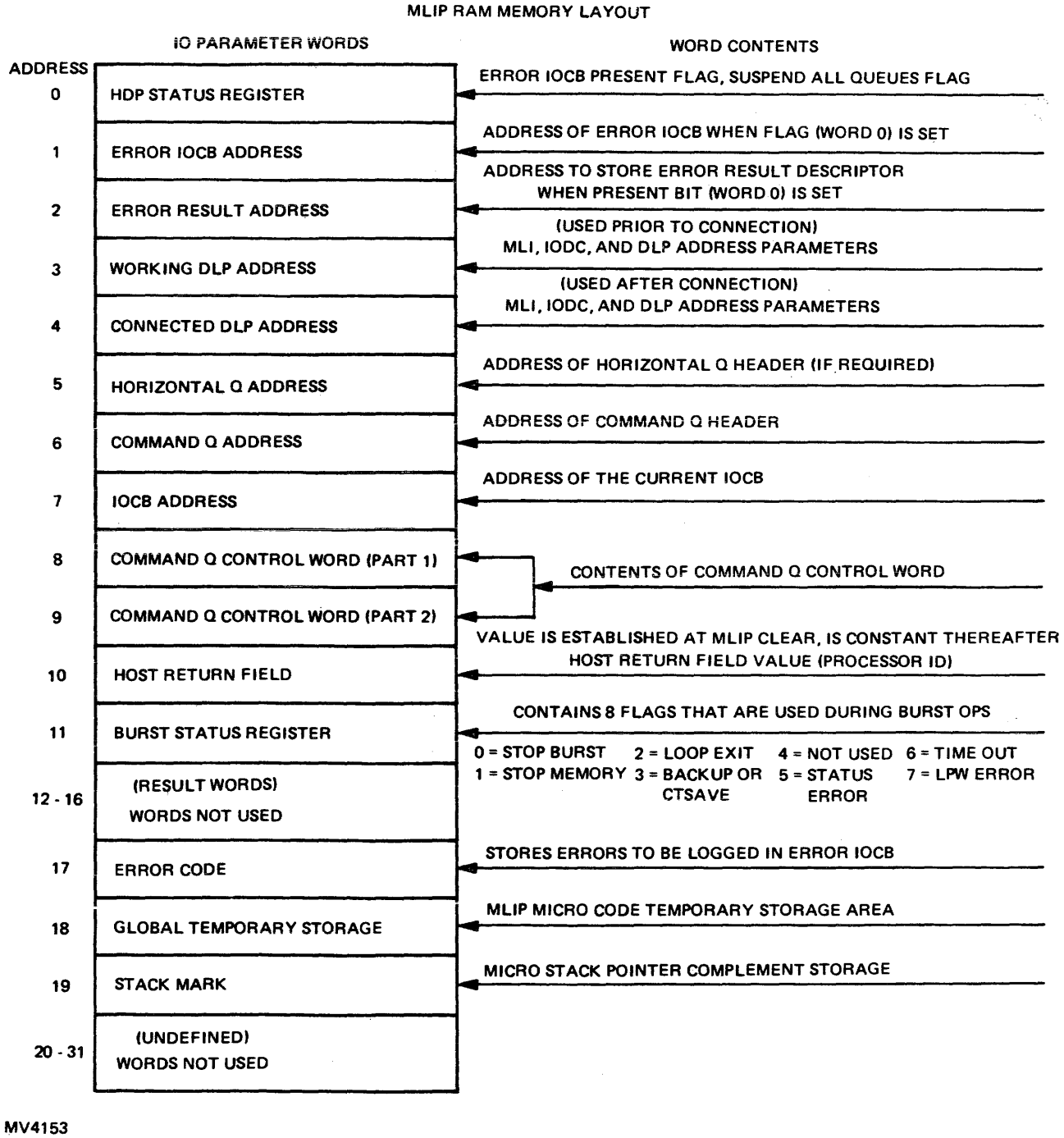


Figure 5-38. MLIP RAM Data Storage Section Word Layout

Micro-Stack Section of RAM Memory

The micro-stack section of the MLIP RAM is controlled by the micro-module unit and is used to store dynamic data that is needed during the execution of a micro-code function. The data in the micro-stack section of the RAM at any given instant depends on the requirements of the particular MLIP function in progress.

Data-storage Section of RAM Memory

The Data Storage Section of the MLIP RAM contains data needed for any current I/O operation that is in process. This section of the RAM is commonly called the "Register Section" because its information is in fixed format and is used in much the same way as if the RAM was a series of registers. Figure 5-38 shows the layout of the Data Storage section of the RAM.

RAM Memory Addressing

The MLIP contains two separate addressing circuits for the RAM memory. The Memory Storage Address (MSAn, see Figure 5-28) logic is used to address the first 32 RAM addresses (the Data Storage section of the RAM). The MSAn logic contains a 5-bit binary address field and can address only the first 32 addresses (0-31) of the RAM. This prevents the MSAn from being able to address the micro-stack section of the RAM.

The micro-stack Pointer logic (MSPn, see Figure 5-28) has a 10-bit binary address field and can access all addresses in the RAM. This address logic is used for accessing the micro-stack section of the RAM and can also access the Memory Storage section.

RAM Memory Functions

The MLIP contains only a single RAM memory that must be used for all I/O device operations. The MLIP logic establishes the contents of the RAM when a new I/O device operation is initiated, and must restore the data in the RAM before each subsequent sequence of an I/O operation.

The data in the RAM comes from the IOCB for an I/O device operation. The MLIP logic uses the memory address of the IOCB to access data which is loaded into the RAM. At the conclusion of an I/O device operating sequence, current operating data is written into the IOCB. Thus, for subsequent I/O device sequences, current data is restored in the RAM.

The MPC provides the MLIP with the IOCB memory address which is part of the CUID operator sequence, and which is used to initiate an I/O device. The IOCB memory address used to initiate an I/O device is provided to the MLIP by the MCP as part of the CUIO operator sequence. After an I/O device has been initiated, the IODC provides the MLIP with the IOCB address, as part of the POLL REQUEST sequence.

I/O DEVICE INTERFACE PROCESSES IN THE MLIP

Communications between the MLIP module and the IODC modules are separated into two types, depending on whether the MLIP initiates the interface, or a UIO-DLP module initiates the interface. An interface that is initiated by the MLIP module is a POLL-TEST operation, and an interface that is initiated by a UIO-DLP module is a POLL-REQUEST operation. The difference between a POLL-TEST and a POLL-REQUEST communication is the direction that information travels on the MLI interface. Extensive logic circuits are required, both in the MLIP module and the IODC modules, to control and discipline the communications conducted over the MLI interfaces.

An MLI is a disciplined communication path over which multiple two-way communications between the MLIP and the 64 possible UIO-DLP modules occur. Line-discipline (a built-in feature of the MLIP path control logic) is used to identify the particular MLI path used for a communication between the MLIP and a UIO-DLP device. This path identification is required because an MLI is a logic fan-out gate, with the MLIP at one end and as many as 64 UIO-DLP devices at the other end. This is the only way the MLIP has to associate the address of a UIO-DLP with an MLI interface port while determining the priority of simultaneous POLL-REQUESTs (inputs from more than one MLI).

MLIP CONNECT/DISCONNECT Sequences

When an MLIP is actively communicating with a UIO-DLP over an MLI interface, the two units are connected. An MLIP can only be connected to one UIO-DLP at any one time. Thus, all DLPs that are not connected, are disconnected. An MLIP must connect to a UIO-DLP to initiate a peripheral device operation, and must disconnect from that DLP, while the DLP controls the peripheral unit. A complete I/O device, operating sequence consists of a series of connect/disconnect sequences, some of which are initiated by the MLIP, and other that are initiated by the UIO-DLP module.

MLIP Polling Operations

The first communication between an MLIP and a IODC module must be a POLL-TEST sequence (proceed from the MLIP to the IODC module). A POLL-TEST must be executed first because an IODC module cannot address the MLIP until it contains DESCRIPTOR LINK data. During a POLL-TEST sequence, the MLIP passes DESCRIPTOR LINK data to the IODC Base module logic. Thereafter, the UIO-DLP can initiate a POLL-REQUEST sequence over the MLI interface. Until the POLL-TEST sequence has been performed, the IODC module does not contain valid DESCRIPTOR LINK data and, consequently, cannot initiate a POLL-REQUEST sequence over the MLI interface path to the MLIP.

POLL-REQUEST DESCRIPTOR LINK Usage

Figure 5-39 shows how data from an IOCB is used by the MLIP to initiate a POLL-TEST communication over the MLI interface. It also shows the DESCRIPTOR LINK data that is transmitted by the MLIP to the IODC during the POLL-TEST sequence. The HOST RETURN field of the data in the DESCRIPTOR LINK is the CPU PROCESSOR ID number, and identifies the MLI over which the MLIP communicates with the particular IODC module. The IOCB ADDRESS field of the DESCRIPTOR LINK is used by the MLIP to associate a POLL-REQUEST operation from a UIO-DLP with the peripheral device control data in an IOCB. The IOCB ADDRESS is the same address that the MLIP received from the Data Processor during the CUIO operator and subsequently used to acquire the I/O control data from the IOCB.

RESULT-STATUS For POLL TEST Operation

A POLL-TEST operation by the MLIP serves to establish that the particular I/O device to be initiated is present in the IODC Base, and that the device is available to perform the operation (is not already engaged performing some function). The availability of the UIO-DLP device is determined by the normal response of the IODC Base module to a POLL-TEST operation, which is to return the Result Status of the UIO-DLP device to the MLIP.

During a POLL-TEST operation sequence, the MLIP receives Result Status information from the UIO-DLP and IODC module. This Result Status is part of the normal MLI connection sequence and is used by the MLIP to verify that the UIO-DLP device addressed by the POLL-TEST operation is present in the IODC module, is not busy, and responds to the POLL-TEST communication in a satisfactory manner.

In order to receive POLL-TEST sequence Result Status over the MLI, the direction of communication over the MLI must be reversed; that is, data must pass from the IODC module to the MLIP. Line reversal of an MLI interface direction is called a LINE-TURNAROUND. A LINE-TURNAROUND may occur during any type of connection between the MLIP and a UIO-DLP, whenever the direction of data passing over the MLI must be reversed.

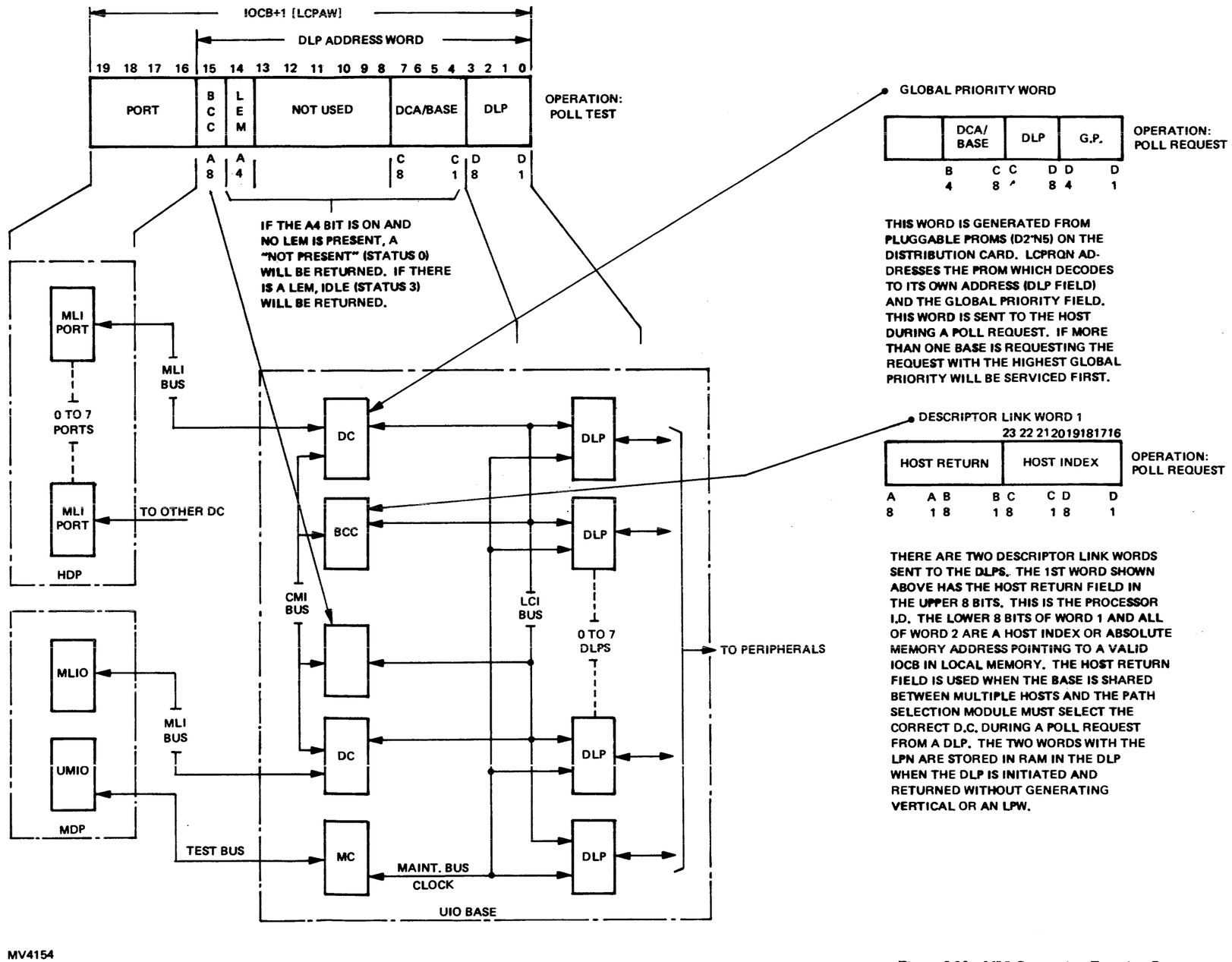


Figure 5-39. MLI Connection Function Between the MLIP and an IODC

Polling Operation Status Reporting

If the MLIP determines (from the DLP Result Status) that a requested I/O device operation cannot proceed, it causes an interrupt to the MCP. If the initiation of an I/O device operation is continued after the MLIP receives the Result Status from the UIO-DLP, then the MLIP saves and updates the status data for subsequent status reporting to the software operating system.

Regardless of how an I/O device operation is terminated, the MLIP reports status about the I/O operation to the software operating system. To transfer this information to the software operating system, the MLIP causes an MLIP STATE AND RESULT word to be written into the IOCB. The data in the STATE AND RESULT word is derived, in part, from the UIO-DLP Result Status data and, in part, from the MLIP Result Status logic circuits. In this way, when the system software examines the STATE AND RESULT word in the IOCB (after an interrupt from the MLIP), it is able to determine the status of the I/O device, UIO-DLP, MLI interface and the MLIP module.

Polling Operation BURST Data Sequence

When a UIO-DLP control is ready to transfer data between the peripheral device and the B 6900 system, the IODC module executes a POLL-REQUEST connection sequence of operations. This sequence recalls the attention of the MLIP module to the requested I/O device operation. To execute its POLL-REQUEST connection sequence or Polling Operation Line-Reversal on the MLI interface, the IODC module uses the DESCRIPTOR LINK data received from the MLIP module during the POLL-TEST sequence. The DESCRIPTOR LINK data contains the proper CPU MLI address, and also the address of the IOCB in system memory (the same address that the MLIP provided during the execution of the MLIP POLL-TEST connection sequence). Consequently, when the MLIP responds to a POLL-REQUEST sequence by an IODC module, it is able to reacquire all of the data about the I/O operation from the IOCB in memory.

MLIP MEMORY OPERATIONS

The MLIP module logic initiates two different types of memory request operations. The first type is to read control data from or to write control data into the IOCB. The second type is to transfer data between system memory and a peripheral device.

MLIP 51-Bit Memory Cycle Operations

The first type of memory operation (refer to Figure 5-40) is used during the initial part of an MLIP I/O device operation function, to ENQUEUE the IOCB, and to copy I/O control data from the IOCB into the MLIP RAM. This

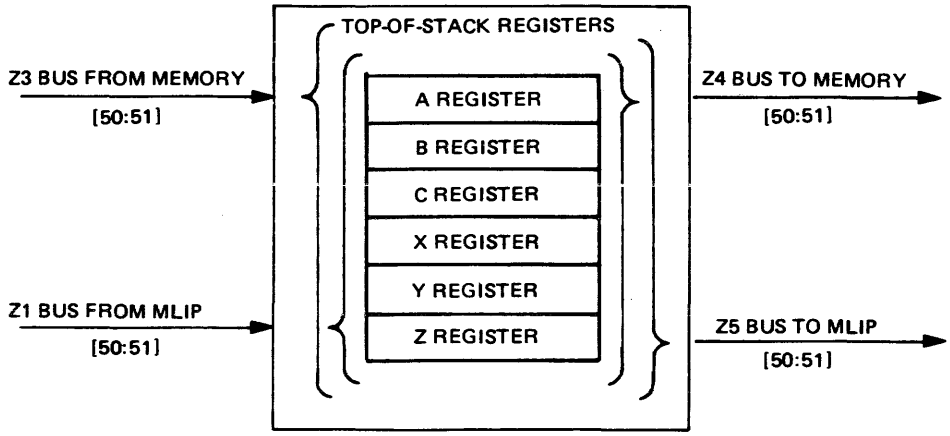


Figure 5-40. 51-Bit Memory Paths Between the MLIP and Memory Control

type of memory operation is also used to update the Command Queue IOCBs, during and after the operation of a peripheral device. Transfer Controller barrelshifting operations are not performed for this type of MLIP memory cycle operations.

MLIP BURST Memory Operations

The second type of memory operation, commonly called a BURST cycle, (refer to Figure 5-41), is used to transfer data to/from memory only during a peripheral device operation. BURST memory operations are more involved than other memory operations because the data to/from a peripheral device (on an MLI) is present in two-character increments, while a memory word contains six characters plus a word TAG field. This means that from one to four transfers of data over an MLI interface must be performed for each BURST memory operation performed.

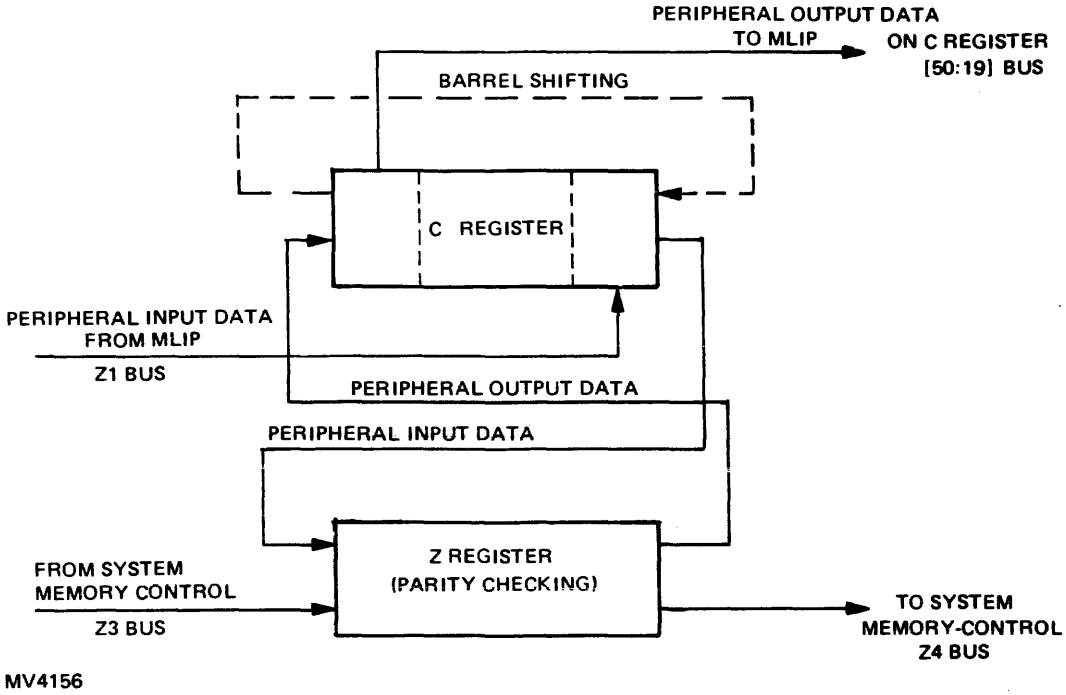


Figure 5-41. BURST Data Memory Paths Between the MLIP and Memory Control

Memory Operation Logic

All memory operations initiated by the logic of the MLIP use the hardware circuits of the Memory and Transfer Controllers. The MLIP shares a single path to system memory with the Data Processor, and the use of these two controller logic circuits is efficient because the Data Processor cannot use the controller logic when an MLIP memory request is being performed.

MLIP Memory Cycle Priority

Priority logic for the use of the memory access path is required. The Memory Controller logic establishes the priorities for the use of the memory path. Briefly stated, the MLIP has first priority for the use of the path to memory while it is performing BURST memory cycle operations. The Data Processor has priority for the use of the path at all other times.

MLIP Peripheral Data Format

Peripheral device data on an MLI interface to the MLIP module is in the form of two Extended Binary Coded Decimal Interchange Code (EBCDIC) characters only. No other character or byte format is used for peripheral data in the B 6900 system. Figures 5-42 and 5-43 show the formats of peripheral data and memory data words.

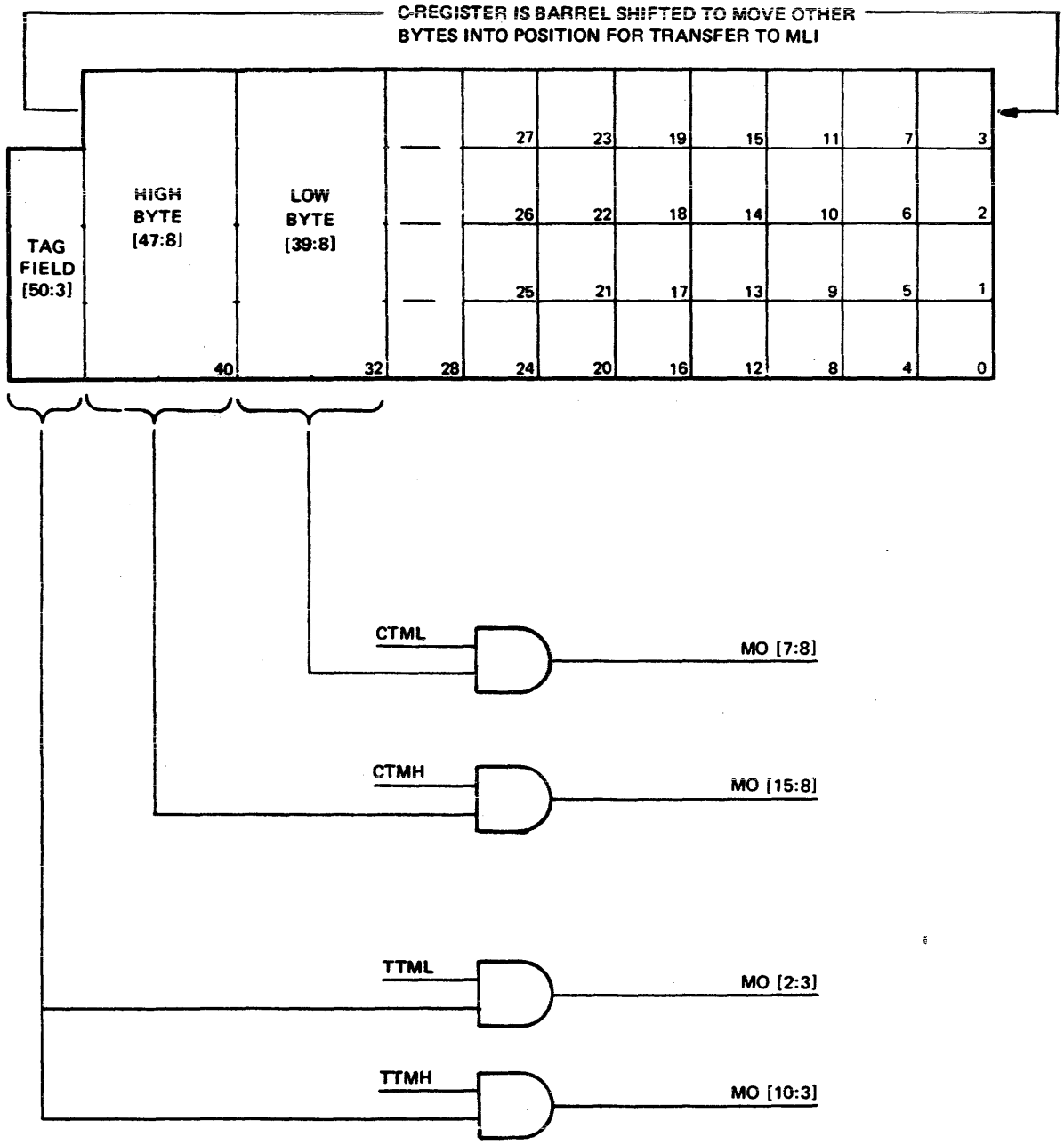


Figure 5-42. MLIP Peripheral Output Data Path From Top-of-Stack

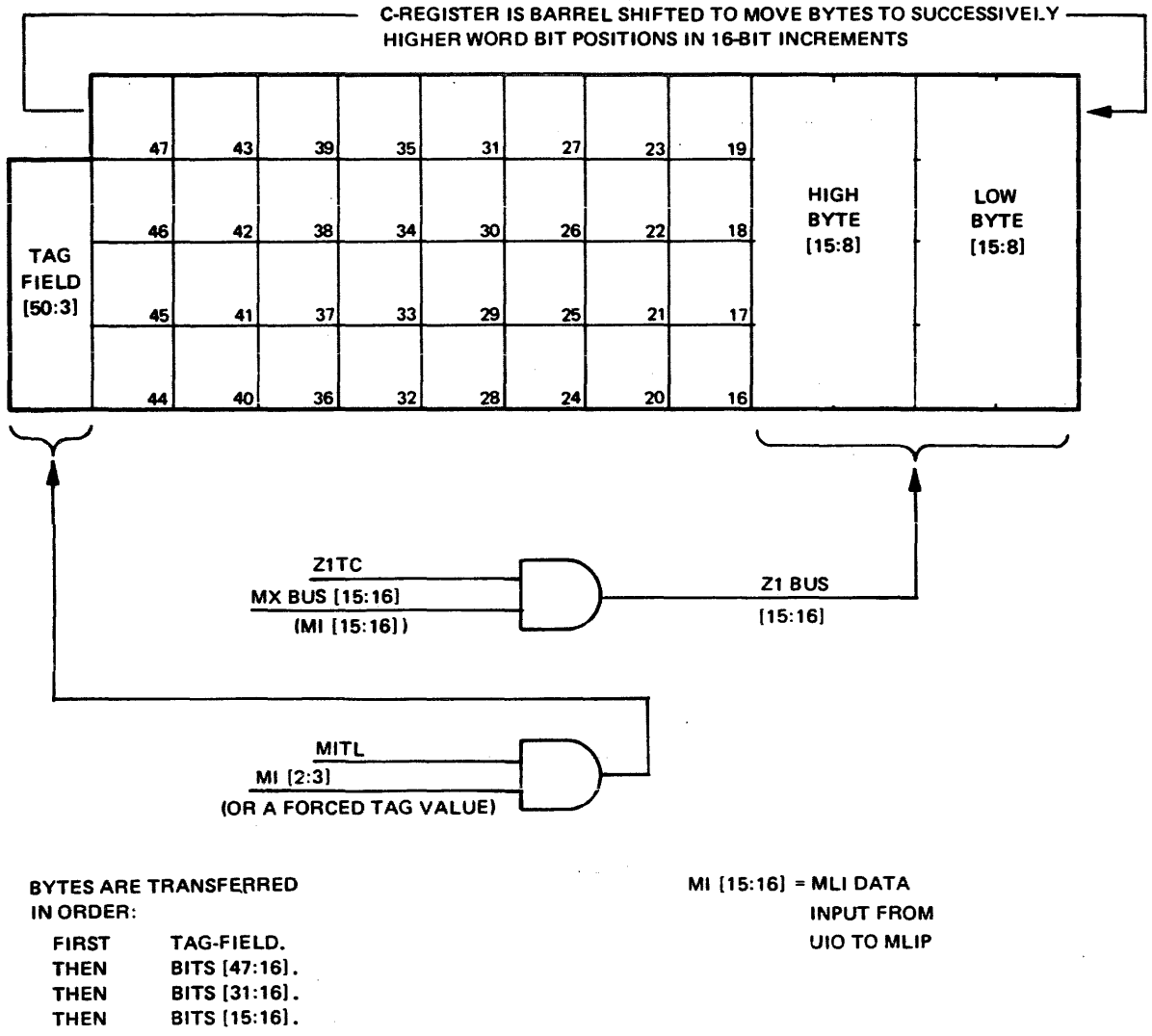


Figure 5-43. MLIP Peripheral Input Data Path to Top-of-Stack

The 16-bits of I/O data on an MLI interface are divided into two characters, (1) a high-order and (2) a low-order character. If only one character of I/O data is present on an MLI interface, it occupies the high-order character position.

MLIP Memory Word Format

B 6900 memory words contain from 1 to 6 EBCDIC characters with the most significant character of the word located in the 8 high-order bit positions. This means that peripheral data from an MLI interface must be barrelshifted so that a BURST memory word represents the same significance as that of the peripheral device data. The first (most significant) peripheral data character is located in the 8 high-order bits of the first BURST data memory word, and so forth.

Figures 5-44 and 5-45 show the logic circuits of the MLIP used to transfer data between peripheral devices and system memory.

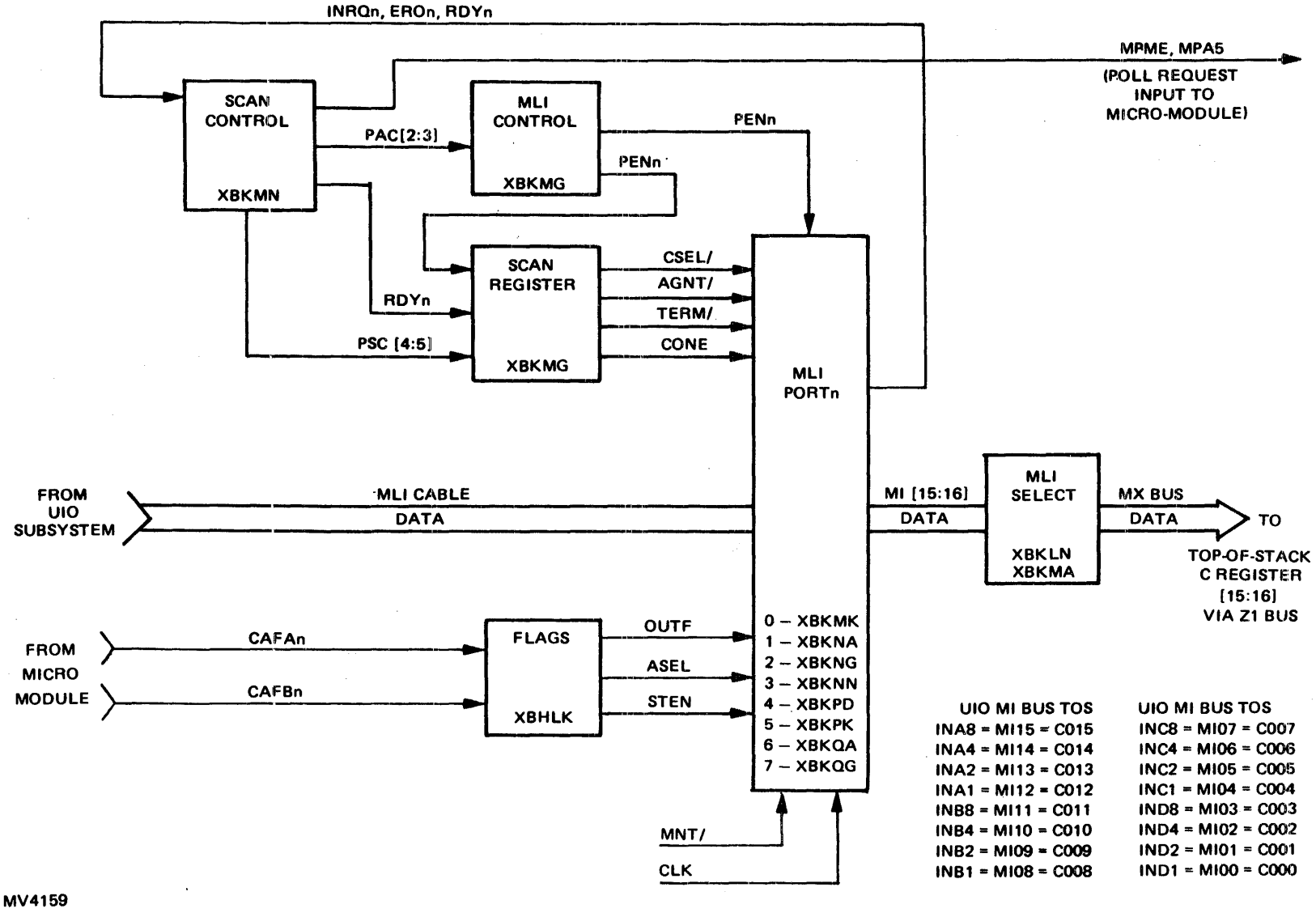


Figure 5-44. Input Peripheral Data and MLIP Control Logic

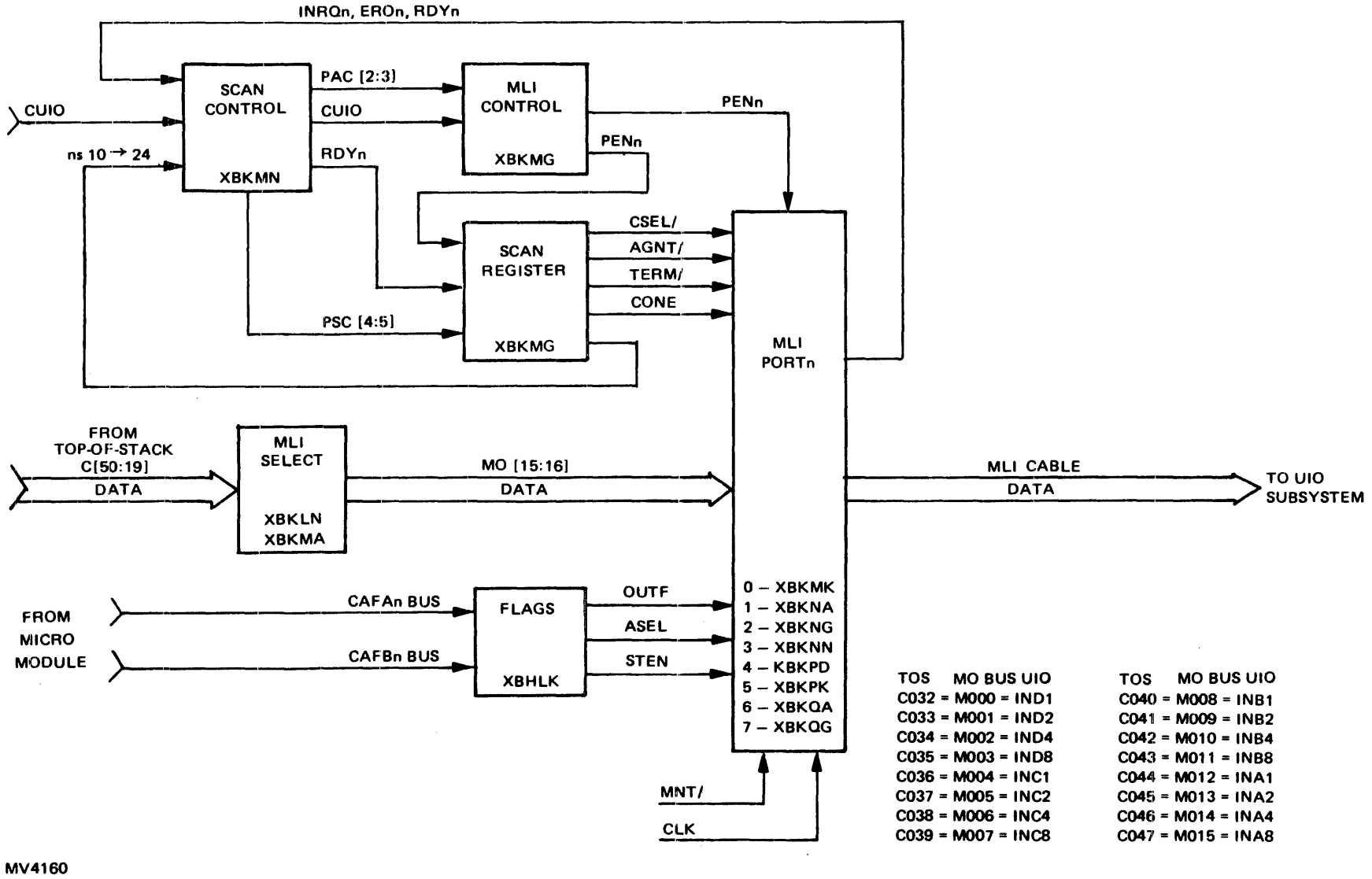


Figure 5-45. Output Peripheral Data and MLIP Control Logic

MLIP Barrelshifting

Barrelshifting is a Transfer Controller function performed by the Memory Controller, in conjunction with an MLIP BURST memory cycle. Barrelshifting consists of rotating a BURST data word around in the Top-of-Stack register, while selectively transferring 16-bit increments of the BURST data word to/from the MLIP logic. Barrelshifting allows a BURST memory word to be reformatted into the 16-bit increments (bytes) required for the MLI interface. Conversely, barrelshifting also allows 6-character B 6900 system memory word formats to be constructed from standard 16-bit peripheral device data formats.

The MLIP does not contain a buffer for peripheral device data. Instead, it uses the Top-of-Stack C and Z registers as a data buffer. These two registers are used as an I/O data buffer only during BURST memory cycle (and barrelshifting) operations. Each UIO-DLP contains a data buffer that is used for the peripheral devices connected to that particular UIO-DLP.

I/O DEVICE OPERATION TERMINATION PROCESS

Every I/O device operation terminates with the UIO-DLP returning result status data about the I/O operation to the MLIP module, over the MLI. The MLIP micro-code control program utilizes the information contained in the DLP result status to formulate a Result Descriptor. The MLIP causes the Result Descriptor to be written into the memory location specified by the I/O Result Pointer (word five of the IOCB). The result status returned to the MLIP from a UIO-DLP is variable length (in bytes) depending on the type of peripheral device controlled by the UIO-DLP. A Result Descriptor is also variable length, and the MCP specifies the number of bytes contained in a particular I/O device Result Descriptor (word six of the IOCB).

IOCB RESULT AND STATE Word Usage

The MLIP forms an MLIP RESULT AND STATE word which it writes into word Twelve of the IOCB. The RESULT AND STATE word contains the general status of an I/O operation including the result status from the UIO-DLP, the status of the MLI, the status of the MLIP logic, the status of memory operations initiated by the MLIP, and the STATE of the MLIP micro-code program sequence. The RESULT AND STATE word describes the entire I/O operation status and identifies the location of any fault or error that occurred during the operation sequences.

After the MLIP logic has completed the Result Descriptor and MLIP RESULT AND STATE words, the I/O operation is complete. The MLIP then proceeds to link the IOCB into a Result Queue. The memory address of the Result Queue into which the IOCB is linked is specified by word five of the IOCB. The current IOCB is always linked into the tail of the Result Queue.

The software operating system specifies when the normal completion of an I/O operation is to cause an IO Finish Interrupt (in word zero of the IOCB). It also specifies whether or not software attention is required at the conclusion of the I/O operation. If either of these conditions are specified, the MLIP causes an I/O Finish Interrupt in the Interrupt Controller at the termination of the I/O operation.

MLIP Error Handling

If a hardware failure or program error is detected during an I/O device operation, the MLIP causes an appropriate HARDWARE or ALARM Interrupt to be initiated by the Interrupt Controller logic. Hardware failures or program errors are detected by the MLIP, MLI interface, IODC module, or Memory Controller logic (during BURST memory cycles). All of these circuits report any error conditions sensed to the MLIP, and the MLIP logic initiates the Interrupt Controller logic into operation.

The software operating system handles I/O error interrupts the same as I/O Finish Interrupts. However, while a normal I/O finish interrupt is performed only if the IOCB requests such an interrupt, an interrupt caused by an error condition is performed unconditionally.

If a HARDWARE or ALARM Interrupt condition is detected by the logic of the MLIP, an Error-IOCB is completed. The completion of an Error-IOCB by the MLIP logic is a subroutine function of the micro-module. This micro-code subroutine is executed as part of the procedure for terminating the I/O device operation.

MEMORY ORGANIZATION

The memory resources of the B 6900 system (see Figure 5-2) are organized so that only one storage module of memory may be accessed at any one time. The memory resources of the system consist from 128K to 1024K words of memory. Local memory may contain all 1024K words. Global memory may consist of that portion of 1024K words that are not local to the CPU.

Memory Addressing

A memory word consists of 60 parallel bits of data that are present at one of the memory module interfaces to the memory exchange. These 60 bits are further divided into a parity bit, 51 data bits, and eight error detection/error correction bits.

Associated with a local memory word are 17 memory address bits and 12 memory function control bits. These bits define a local memory operation to be performed, such as a READ operation or a WRITE operation, and specify the proper word address in the memory module at which the memory operation is to be performed.

The memory exchange logic utilizes the three high-order bits of the memory address field to select a memory module. The low-order 17-bits of the memory address are used to identify a specific word address within the memory module.

Global memory is selected when local memory is not addressed for a valid memory function, or for a global system control function. Local memory is selected for a memory operation when a valid memory function is defined, and one of the four local memory ports is configured with an identical module selection code as that contained in the high-order 3-bits of the memory address. If none of the four local memory ports is configured identically to the high-order 3-bits of the memory address code, then global memory is defined, and the global memory port responds to the memory request. A global system control operation is defined by a special configuration of control bits in the memory requestor logic and is executed only as a result of a scan command operator execution in the CPU processor logic. Global scan operations are defined later in this section.

A memory storage module contains 128K words of continuous memory storage addresses (see Figure 5-46). A 20-bit binary address field is used to select a memory module and a specific word address within the module (see Figure 5-46). The low order 17-bits of the 20-bit address field select one word of the 128K words within a memory module. The high-order 3-bits of the 20-bit memory address field are used to select one of four local memory modules or global memory. A local memory storage module is synonomous to one of the local memory ports of the memory exchange.

In addition to address and information data, the memory interface bus also transmits control information between the memory control and the memory module or Global memory. This control information directs the memory operation that will be performed by the memory module, such as WRITE or READ functions.

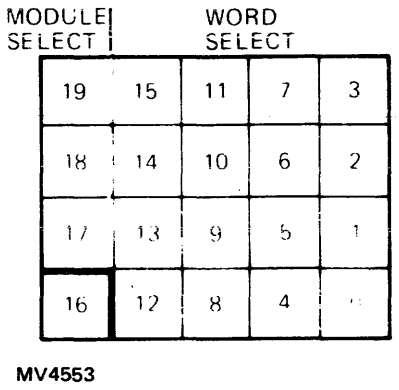


Figure 5-46. Memory Address Decoding

For local memory modules, the control signals include the Initiate Memory Cycle (IMC) timing signal, and a 3-bit memory function code that is comprised of the Read Modify Write (RMW), Write Cycle Conditional (WCC), and the Parity Error Disable (PED) control signals. The significance of these control signals is discussed in the portion of this section entitled Local Memory Port Interface Control Logic. The control signals present at the global memory interface port are discussed in the portion of this section entitled Global Memory Port Interface Control Logic.

Global Memory and Global System Control

A B 6900 system can be interfaced to a global system through the global memory port of the CPU memory controller. When a B 6900 system is connected to a global system, it is part of the global system to which it is connected, and is subject to the rules for global system operation.

A global system may contain a single B 6000 system, in which case the global system is only an extension of the memory resources of the B 6000 system, and no global control is utilized. A global system may contain several B 6000 systems in which case, global system control is utilized.

Global system control is utilized to organize and control the application of the processor elements (B 6000 systems) in the global system. It is also utilized to control the dedication of the global memory resources among the various processor elements of the global system.

Global system control is dynamic in nature. Reorganization and reallocation are functions of the Master Control Program(s) that are operational at any given instant in time. The hardware cabinets of the global system are the Global Memory Module (GMM), which contains the logic circuits for both global system control and global memory control, and memory module cabinets. Figure 5-47 shows the hardware logical organization of a GMM, including the system control interfaces and the memory control interfaces and modules.

GLOBAL SYSTEM ORGANIZATION

Global systems (see Figure 5-47) provide for multiple processors (systems) and multiple memory modules to be coupled together in global networks. Such multiple systems are dynamically controlled by the software operating system. Dynamic control consists of defining and controlling the paths of communication between the processors that are present, and between the processor organizations and the memory module resources. Because the global system is dynamic, its structure is subject to change based upon the instantaneous requirements of the software operating system.

Two distinct types of architecture are involved in a global system: the physical organization of the various processors and memory modules (GMMs), and the logical organization of the global system. To understand the global system, the physical and logical structure of the system must be thought of as separate dimensions of the same entity.

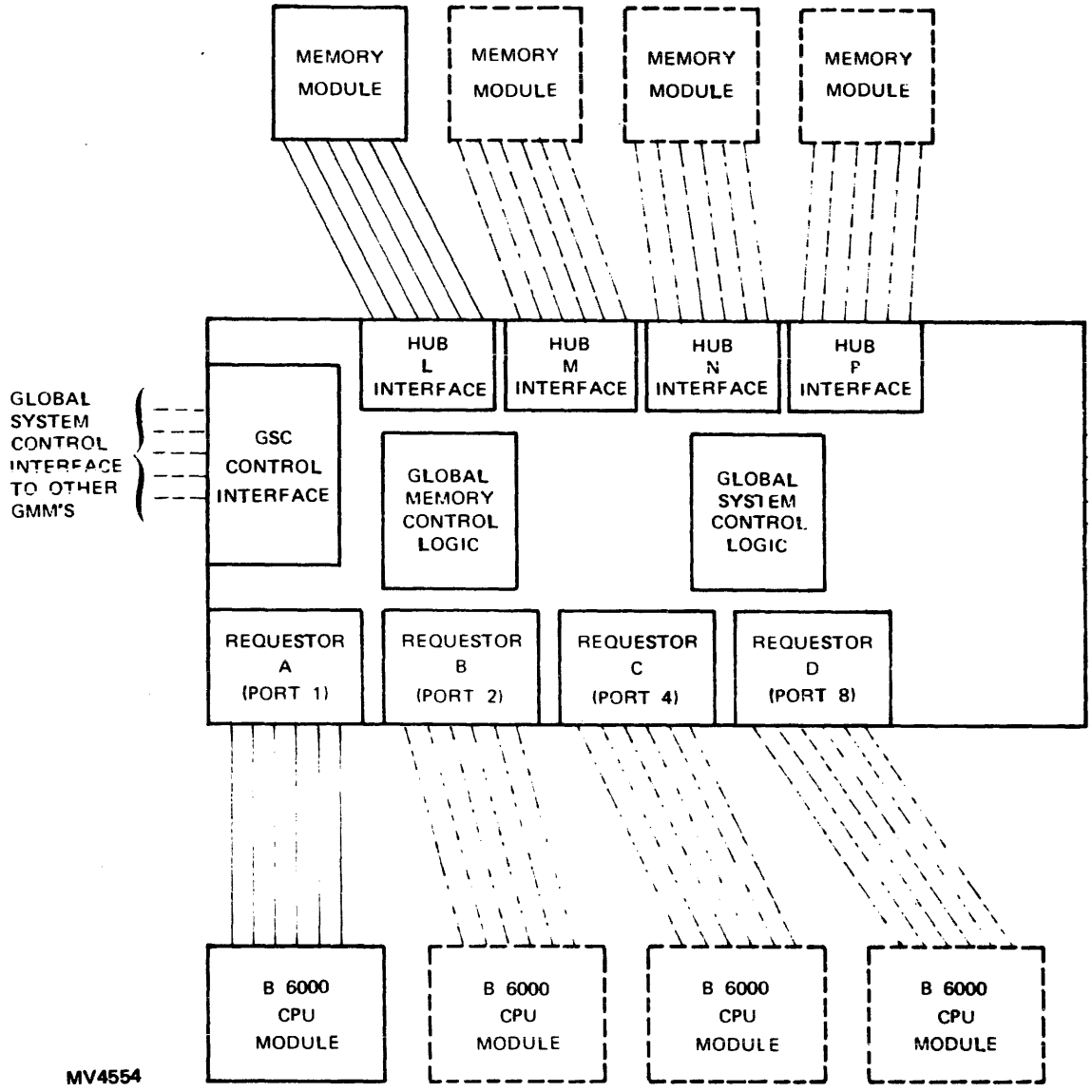


Figure 5-47. Global Memory Module (GMM) Organization

Physical Structure

The physical structure of global system components includes the number of type of cabinets (processors, GMM, and memory modules) and also the manner and order in which the system components are interfaced with each other. This structure defines the constraints and limits under which the software operating system can act to dynamically control the global system. It dictates which global units may be coupled, and also how the memory resources can be utilized within a subsystem. Figure 5-48 shows a global system that contains all of the cabinet types required of a global system in its most elementary form, plus those global system components that may be added to the elementary global system without adding cabinets that are used solely for expanding the global system capabilities.

Elementary Global System Requirements

A global system must contain at least one processor, one GMM, and one memory module. If a global system contains one GMM cabinet, it may also contain up to four processors and up to four memory modules. Figure 5-48 shows a

single GMM cabinet, which is connected to four processors and four memory modules. If a global system only contains one GMM cabinet, then a global system control bus (multi-cabinet adapter) is no required because all of the logic for global system control is present in the GMM cabinet. A separate interface is required for each processor and each memory module that is connected to a GMM cabinet.

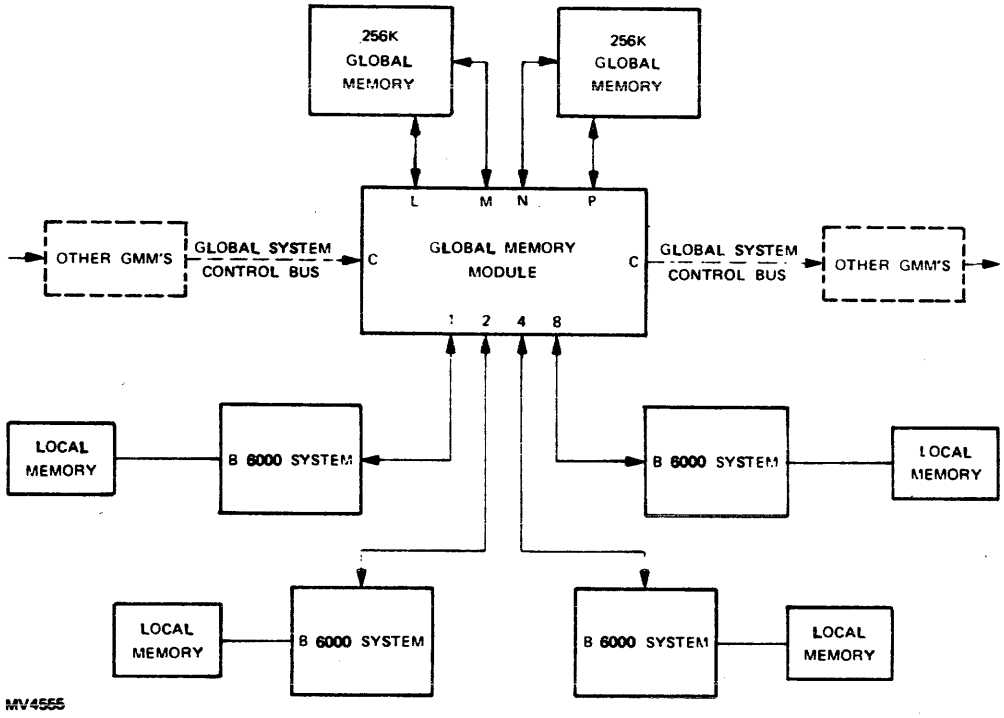


Figure 5-48. Global System Interfaces

Logical Structure

Logical structure is that organization of communication paths within a global network that defines the global system(s) present at any given instant. "Communication paths" do not pertain to the memory resources of the global network, but rather, to the processor resources of the network only.

Each processor within a global network may be named; each processor that is part of a global system must have a name. The name consists of two parts: system name and processor mask field. The combination of the two parts of a processor name identifies a global system and a specific processor within the global system.

Processors within a global system are organized in a master/slave relationship. A master processor is logically above its slave processor(s), and for each slave processor there can only be one master processor. A processor that is slave to another processor may also have a processor which is slave to it.

Processor Addressing in a Global System

A processor within a global network may be identified by two different types of identification. The first type of identification is the physical port location of a GMM to which the processor is interfaced.

The second type of processor identification is a naming convention controlled by the software operating system(s). This type of organization is used to associate a processor with other processors in a global system. This type of organization allows the software operating system to dynamically couple processors into groupings, without regard to the physical organization of the GMM cabinet that is part of the global network.

Port Identification Addressing

A processor port identification is comprised of up to four 4-bit numeric digits. The actual number of 4-bit digits used for processor port identification depends on the number of GMM cabinets that are part of the global network. The most significant digit of the port identification number represents the port connection to the GMM cabinet.

Logical Naming Identification

The naming convention used by the software operating system to identify the processors in a global system is the logical structure of the system, and bears no required resemblance to the physical port identities of the processors that are part of the global network. This organization allows the software operating system to determine what types of global systems the global network contains, and which processors are members of global systems.

A system logical name consists of 12 binary bits which form three hexadecimal digits. The most significant digit in a system name is the left-most digit of the name, and identifies the upper-most level of the global system. The middle digit of the system name identifies the middle level of the global system. The least significant digit identifies the lowest or bottom level of the global system.

The processors present at each level of a global system are identified by a 12-bit mask field, which is appended to the system name. Each of the 12-bits in the mask field (bits zero through eleven) identifies one of the 12 processors that may be present at any one level of the global system. A mask field for a particular processor may contain only one bit. The proper addressing for a processor in the global system or subsystem includes the name of that system or subsystem and also a mask in which 1-bit is set.

It is impossible to have a third (bottom) level global subsystem without also having a corresponding second (middle) level subsystem. A global system that only contains one level is the top level. A global system that contains two levels includes the top and middle levels. Only three levels of global system name are permitted.

System Memory Interface

The system memory interface consists of a one-by-five exchange that is used to interface the B 6900 CPU to the memory resources of the B 6900 system. The five memory storage module interfaces are designated as ports number zero through three (local memory) and the global memory port. Figure 5-49 shows the organization of the requestor interfaces and the port interfaces to the system memory control.

MEMORY REQUESTOR

Figure 5-50 shows the path used in the data processor to access the system memory control. This path is controlled by the memory controller, through use of the Z12 bus. All data written into memory from the data processor or MLIP is routed to the system memory interface exchange by means of the Z4 bus. All data read into the data processor, MLIP, or look ahead logic is routed from the system memory port interface to the Z3 bus. Address information is routed from the memory address register or look ahead address register by means of an internal memory address bus.

Figure 5-51 shows how information, address, and control data are routed internally within the requester logic of the memory exchange. This figure also shows how port selection is made within the exchange module, by means of the port select logic.

Figure 5-51 shows the PACK (port acknowledge) control bus. This bus has a true level if a local memory port interface is selected by the port select logic. If a local memory port is not selected (PACK/is true) and a valid request is present in the requestor logic, then the global memory port is selected by default.

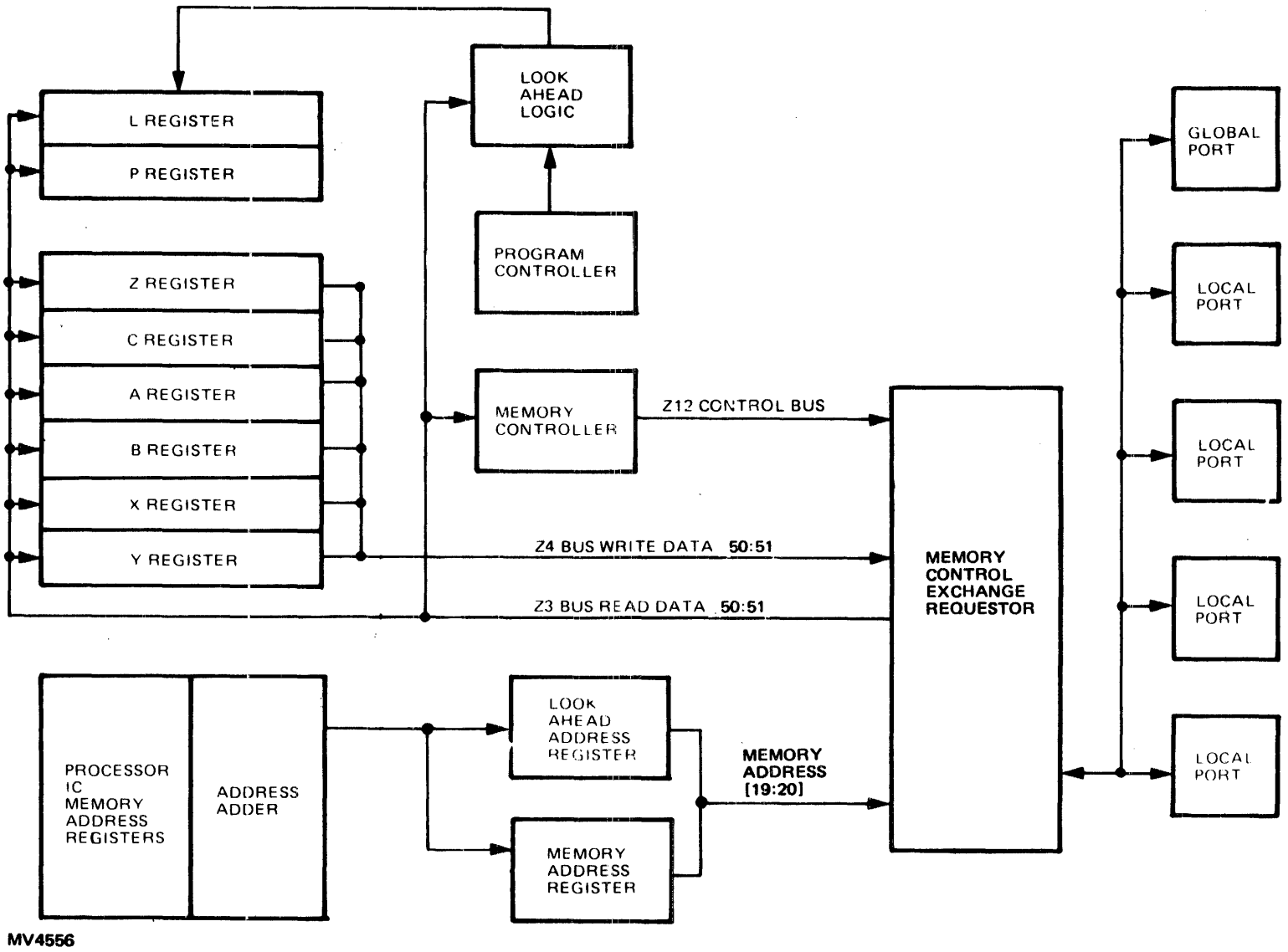


Figure 5-49. Memory Control Block Diagram

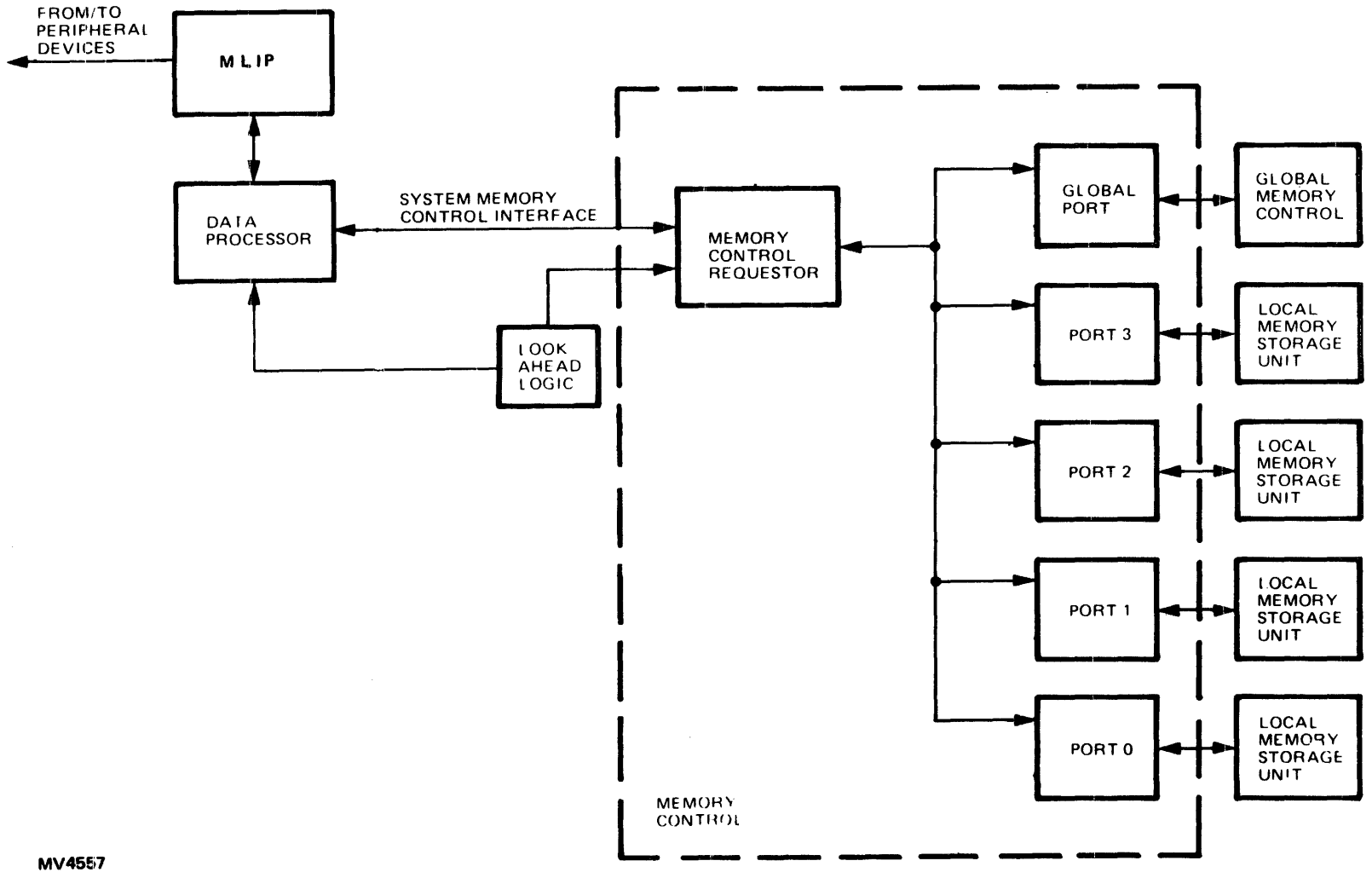


Figure 5-50. Data Processor to Memory Control Exchange Transfer Path

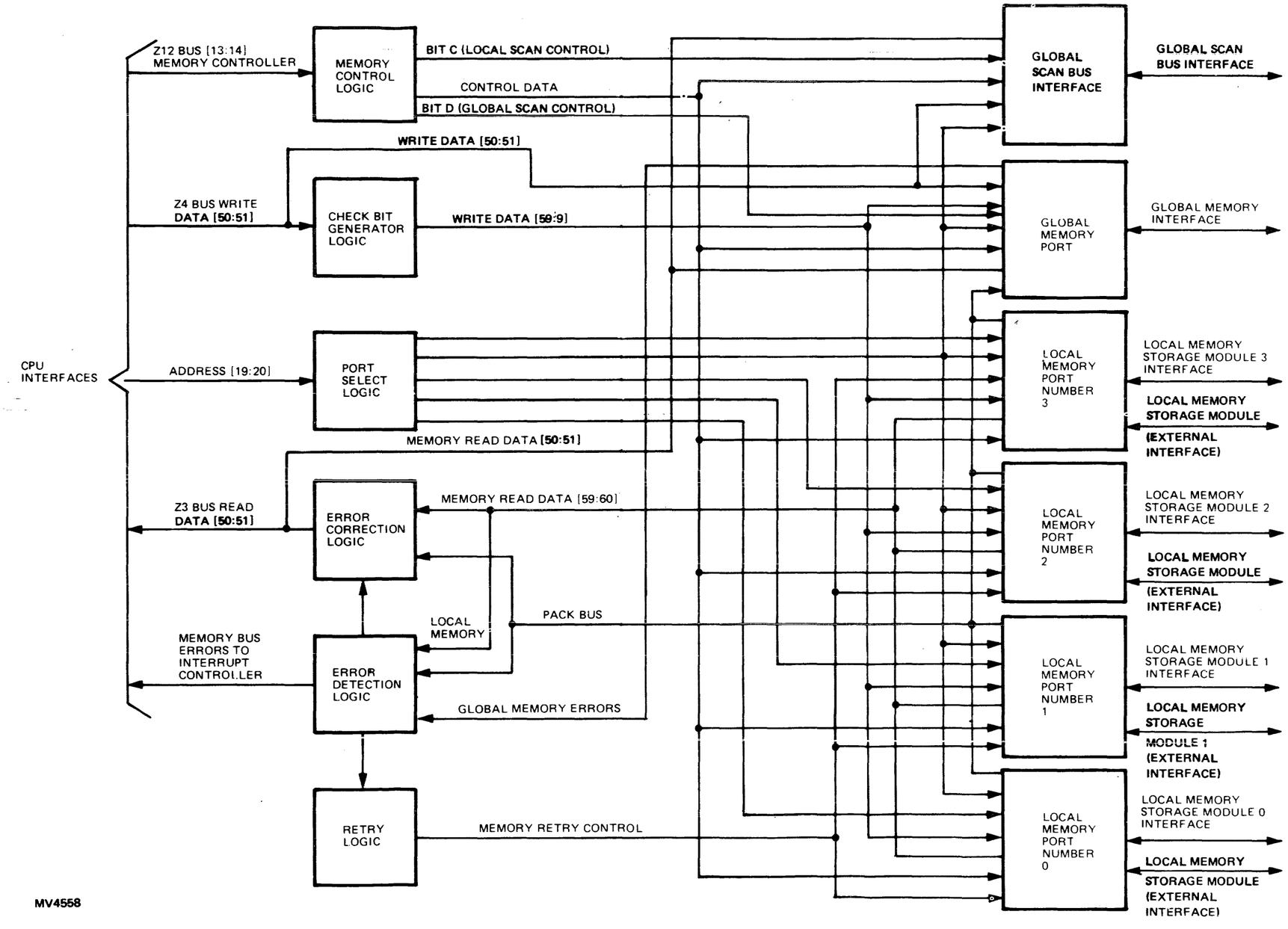


Figure 5-51. Memory Exchange Interface Functional Block Diagram

The 14-bits of the memory control Z12 bus are identified as follows:

Bit Field	Meaning and Usage
5:6	The register select field. This field identifies the data processor register that is to receive the data for a memory READ operation, or the data processor register from which data is to be written into memory for a memory WRITE operation. Bit zero is used to select register Z Bit one is used to select register Y Bit two is used to select register X Bit three is used to select register C Bit four is used to select register B Bit five is used to select register A
9:4	The request field. This field identifies the type of memory operation to be performed.

Bit: 9	8	7	6	Operation to be Performed
0	0	0	1	Protected WRITE with flashback to C register
0	0	1	0	Clear WRITE
1	0	1	0	Overwrite with flashback to C register
0	1	0	0	READ
0	0	1	1	Protected WRITE with no flashback

A,B	The look ahead request field. When bit A is true, the request originates in the look ahead logic. If bit A is false, the request originates in the data processor/MLIP. Bit B is used to specify which register in the data processor is to receive the data input from memory when a look ahead memory cycle is completed. If bit B is true, the data is to be placed in the L register of the data processor. If bit B is false, the data is to be placed in the P register of the data processor.
C	Bit C is not used by a B 6900 system.
D	The global scan bit. If bit D is true, the operation to be performed through the global memory port is a global scan operation instead of a global memory operation. If bit D is false, the operation to be performed is a memory operation instead of a global scan operation.

Memory Error Detection and Correction

The memory requestor logic contains error detection/correction logic circuits (refer to Figure 5-52). Each time a memory WRITE operation is performed, 8-bits of error detection check code are generated by the error detection circuits and appended to the memory write data. The total number of bits written in memory during a WRITE operation is 60-bits, of which 52 data bits are write data from the CPU, and the other 8-bits are the error detection check code.

During memory READ operations, the error detection check bits (which were written into memory during the memory WRITE operation) are tested for bit errors in the data word received from the memory storage unit. If a single bit of a memory read data word is in error, the error correction circuit corrects the bit in error. If more than a single bit in the memory read data word is in error, the error is not correctable, but the error detection circuit detects a multiple bit data error. All single bit and multiple bit data errors are reported to the data processor interrupt handling procedure, and are logged in the SYSTEM/SUMLOG.

Memory Retry

The memory control performs memory RETRY operations under certain conditions.

The memory control performs a memory RETRY operation if the memory module detects a parity error in the address and control data that is transmitted from the CPU cabinet to the memory module cabinet over the port interface. This RETR consists of performing the entire memory cycle over again. If the retry of the memory cycle is successful, then the memory controller causes the interrupt controller to make an entry in the SYSTEM/SUMLOG that indicates a RETRY operation occurred, and the memory operation proceeds in a normal manner. If the RETRY operation is not successful (a second parity error is detected in the memory address and control data), then the memory cycle is aborted, and the memory controller causes an alarm interrupt to be recorded in the SYSTEM/SUMLOG. The procedure that caused the memory cycle which was aborted is terminated because of the memory parity error.

The memory control also performs a RETRY operation if the memory control senses a parity error in the read data that is transmitted from the memory module cabinet to the CPU cabinet. This RETRY operation consists of causing the read data in the storage module read latches to be transmitted to the CPU cabinet a second time. A second memory cycle is not performed by the storage module. The results of successful RETRY operations are reported in the same way that a successful address and control retry is reported.

If the RETRY operation for a parity error in the read data is not successful, then an error correction memory cycle is initiated. The entries made in the SYSTEM/SUMLOG as a result of an error correction memory cycle were described previously in this section.

The memory control does not perform RETRY operations for parity errors in the write data transmitted from the CPU cabinet to the memory module cabinet.

Only one RETRY operation will be attempted for each memory operation.

Global Memory

Global memory provides a path through which one B 6900 system may control the operations of another B 6900 system (global system control operations), and also provides a path up to 512K words of global memory. The system control functions of the global scan bus and the global memory functions share a common interface path through the channel A global memory port of the B 6900 system. A global memory request from the B 6900 system and a global scan operation cannot be processed simultaneously.

A global memory request is identical to a local memory request. The method used to distinguish between local and global memory operations was defined previously in this section (Memory Organization), and is a function of module addressing.

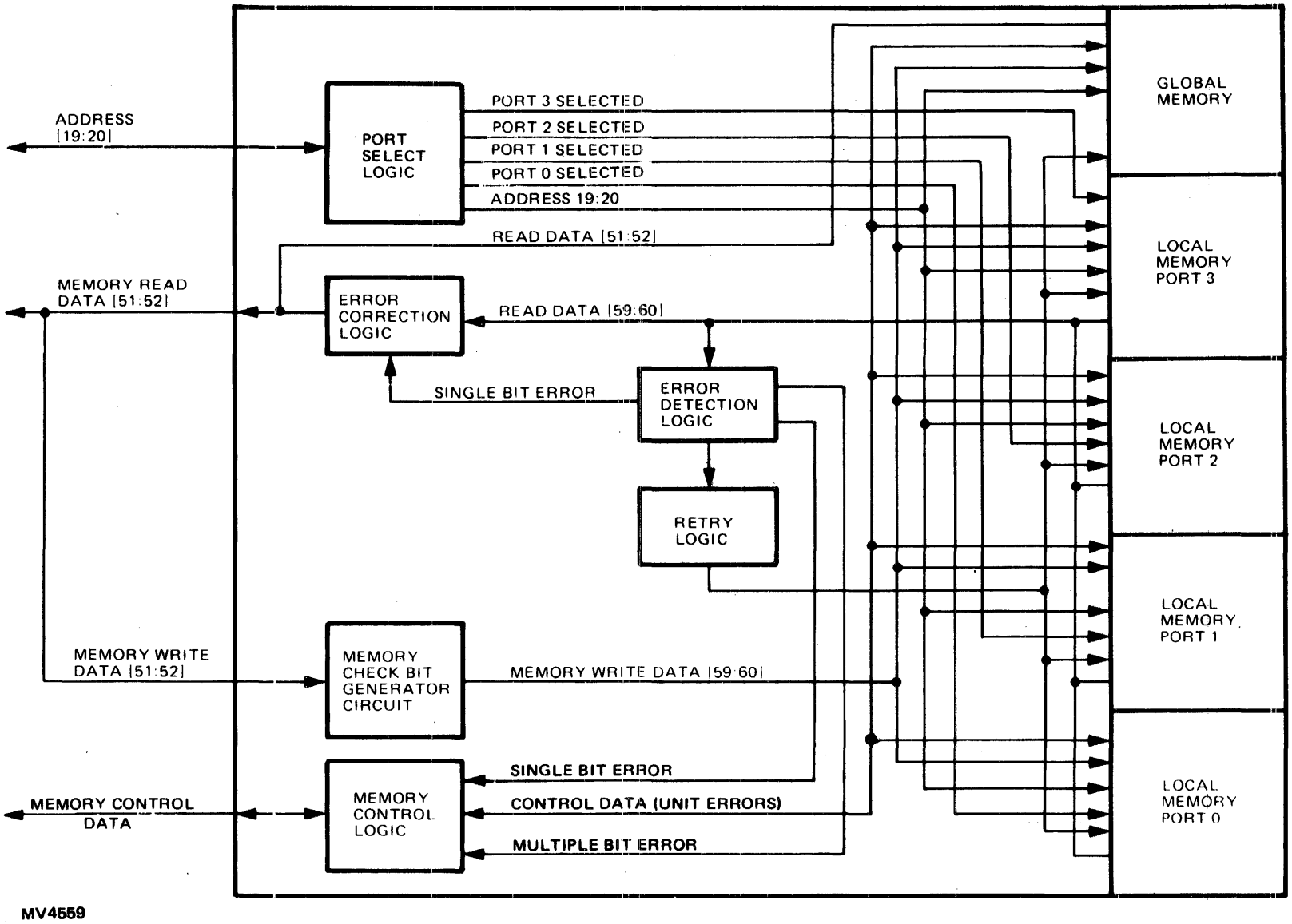


Figure 5-52. Error Detection Correction Logic

Global System Control (Scan) Operations

Global scan operations are common with global memory operations only in that they both use the global memory path to communicate with the global system. The global scan operations are of two types, SCAN-IN and SCAN-OUT.

Global SCAN-OUT

A global SCAN-OUT operation is performed when a SCNO operator is executed from the data processor P register. The distinction between a global SCAN-OUT operation and other SCAN-OUT operations is the contents of the scan function word present in the A register when the SCNO operator is executed. If the SCNO function word contains hexadecimal B in bits 19:4, then a global scan function is defined. The destination of a global SCAN-OUT data word (in the global memory module cabinet) is the response buffer. The SCAN-OUT data word is located in the B register of the data processor at the start of the global SCAN-OUT operation, and defines the function to be performed by the global memory control module. The format of the SCAN-OUT function word and of the SCAN-OUT data word is shown in Figure 5-53.

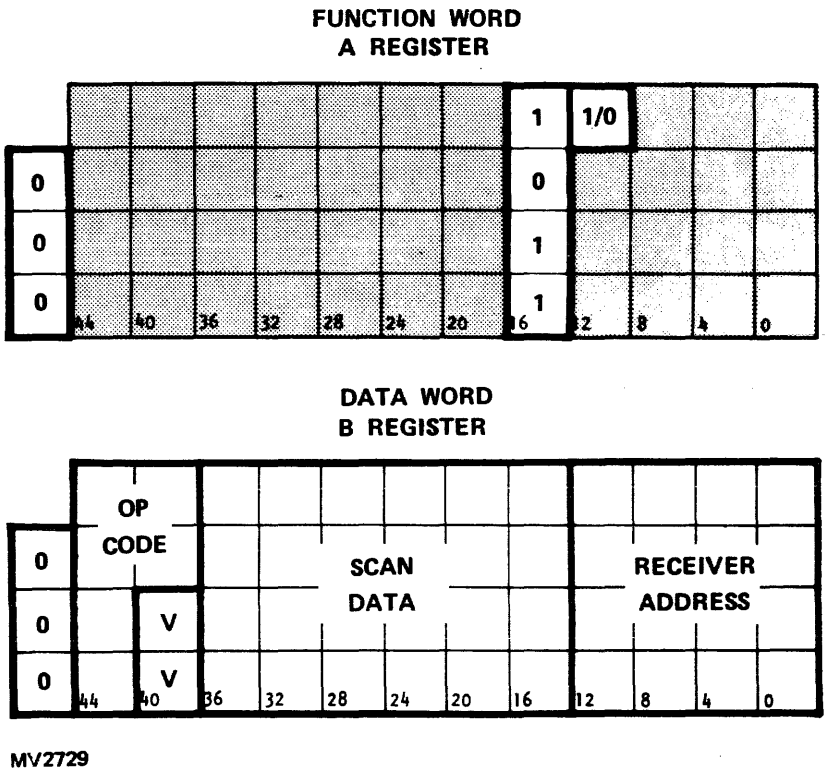


Figure 5-53. Global Scan Function And Data Word Format

Global SCAN-IN

A global SCAN-IN function is similar to a global SCAN-OUT function. The difference between the two types of global scan functions is that the global SCAN-IN function is performed when a SCNI operator is executed from the data processor P register, and a SCAN-OUT function is performed when a SCNO operator is executed from the data processor P register. If bit 15 of the function word (for a global SCAN-IN function) is a binary zero, then the source of the SCAN-IN word is the response buffer in the global memory control logic. If bit 15 is a binary one, then the source of the SCAN-IN word is the message buffer in the global memory control logic. Bit 15 is not used for a global SCAN-OUT type operation, because the destination of the data word is always to the response buffer in the global memory control logic.

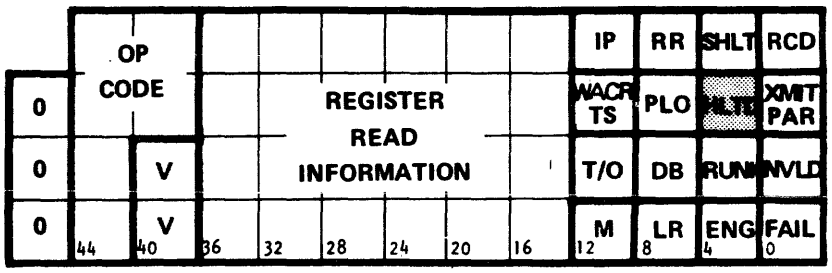
Typical Global System Control Operation

If two B 6900 systems communicate with each other by means of the global scan bus, the system that transmits a message executes a global SCAN-OUT operation, and thus places a global scan data word in the response buffer of its global memory control. The receiver B 6900 system receives an interrupt from its memory controller, and executes a SCAN-IN of the contents of the global memory control message buffer. The contents of the global memory message buffer is partially the data word that was scanned out by the transmitter B 6900 system. The global memory control of the receiver B 6900 system returns a word of data that describes the results of the global scan function to the response buffer of the transmitter global memory control. The transmitter B 6900 system may then SCAN-IN the contents of its response buffer, and thus know the status of the completed global scan operation.

The response word received by a transmitter B 6900 system at the conclusion of a global scan operation has two formats, depending on whether or not an error occurred during the global scan operation. Figure 5-54 shows the format of the word present in the transmitter response buffer when no errors were encountered during the global scan operation. Figure 5-55 shows the word in the response buffer when an error was encountered during the global scan operation.

Global system control functions are specified by the contents of the scan-data word that is present in the B register at the start of a global system control operation. There are 32 different global system control functions that may be specified by the contents of the OP CODE field in the data word. These functions are divided into five classes as follows:

OP CODE Field Value [47:6]	Class	Global System Control Function
000001	1	HEYU
000010	1	HEYALL
000011	1	ARE YOU THERE (PN)
000100	1	WHERE ARE YOU
000101	1	TRANSFER i
000110	1	SHARE WRITE i
000111	1	SHARE READ i
111000	2	HALT
111001	2	CLEAR
111010	2	LOAD
111100	2	START
111101	2	ZAP
010000	3	I AM
010001	3	WHAT IS MY NAME
010010	3	WHAT IS MY NUMBER
010011	3	RESET MY LR

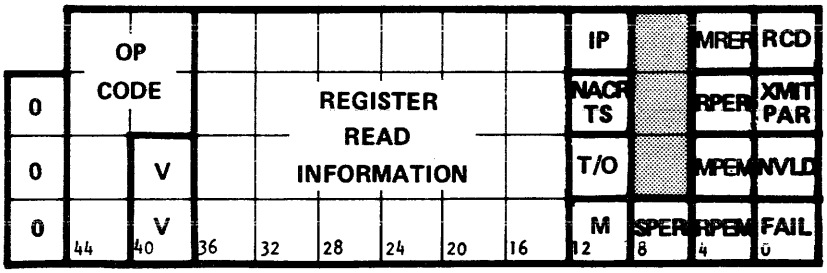


47:6 THE OPERATION CODE FOR THE OPERATION TO BE PERFORMED.
41:2 THE VARIANT FIELD FOR THE OPERATION CODE.
39:24 THE GLOBAL SCAN OPERATION DATA FIELD.
15:1 THE INTERRUPT PENDING BIT.
14:1 THE WACR SET BY TEST AND SET BIT.
13:1 THE TIME-OUT WAITING TO RECEIVE BIT.
12:1 THE MODULE i INVISIBLE BIT (NOT IN MAINTENANCE MODE FOR WRITE ACR).
11:1 TWO MEANINGS:
IF ON, RACR(i) =1 AND WACR(i) =0.
IF OFF, WACR(i) =1.
10:1 THE RECEIVERS PORT LOCK-OUT SWITCH IS ON BIT.
9:1 THE RECEIVERS DEPENDENT BIT IS SET BIT.
8:1 THE RECEIVERS LOCK REGISTER IS SET BIT.
7:1 THE RECEIVER IS SUPERHALTED BIT.
6:1 THE RECEIVER IS HALTED BIT.
5:1 THE RECEIVER RUNNING BIT.
4:1 THE RECEIVER ENGAGED BIT.
3:1 THE RESPONSE RECEIVED BIT.
2:1 THE TRANSMISSION PARITY PROBLEM BIT.
1:1 THE INVALID COMMAND, ILLEGAL PATH, OR LOCK REGISTER ≠1 BIT.
0:1 THE UNSUCCESSFUL COMMUNICATION BIT.

MV2730

Figure 5-54. Global Scan Operation Response Word (No Transmission Errors)

Op CODE Field Value [47:6]	Class	Global System Control Function
010100	3	TEST AND SET i
010101	3	SET MY DSR
010110	3	RESET BY DSR
010111	3	RESET BY RACR
001001	4	READ ACR i
001010	4	READ FWAR
001011	4	ARE YOU THERE (PID)
001100	4	WHO ARE YOU



47:6 THE OPERATION CODE FOR THE OPERATION TO BE PERFORMED.
41:2 THE VARIANT FIELD FOR THE OPERATION CODE.
39:24 THE GLOBAL SCAN OPERATION DATA FIELD.
15:1 THE INTERRUPT PENDING BIT.
14:1 THE WACR SET BY TEST AND SET BIT.
13:1 THE TIME-OUT WAITING TO RECEIVE BIT.
12:1 THE MODULE i INVISIBLE BIT (NOT IN MAINTENANCE MODE FOR WRITE ACR).
11:3 NOT USED.
8:1 THE SENDER PARITY ERROR IN RESPONSE BIT.
7:1 THE MULTIPLE RECEIVER PARITY ERROR IN RESPONSE BIT.
6:1 THE RECEIVER PARITY ERROR IN RESPONSE BIT.
5:1 THE MULTIPLE RECEIVER PARITY ERROR IN MESSAGE BIT.
4:1 THE RECEIVER PARITY ERROR IN MESSAGE BIT.
3:1 THE RESPONSE RECEIVED BIT.
2:1 THE TRANSMISSION PARITY PROBLEM BIT.
1:1 THE INVALID COMMAND, ILLEGAL PATH, OR LOCK REGISTER ≠1 BIT.
0:1 THE UNSUCCESSFUL COMMUNICATION BIT.

MV2731

Figure 5-55. Global Scan Operation Response Word (Transmission Errors)

OP CODE Field Value [47:6]	Class	Global System Control Function
001101	4	YOU ARE
001110	4	READ SINGLE BIT ERROR REG
110000	5	MANUAL HALT
110001	5	MANUAL CLEAR
110010	5	NOT RUNNING
110011	5	SUPER HALTED
110110	5	WRITE ACR i
110111	5	GENERAL CLEAR

Global functional descriptions are not given in this manual. They are specific subjects of the global system documentation and are covered in detail in the B 6800 System Global Memory FETM, Form Number 5010218.

The VV field of the data word is used to specify the direction the GMM is to use (within the global system) in performing the operation specified by the OP CODE. The VV field codes are as follows:

BITS 41	42	Direction of Global Communication Path
0	0	Within
1	1	Across

The SCAN DATA field (bits [39:24]) is used to pass data relevant to the global function specified by the OP CODE field.

The RECEIVER ADDRESS field (bits [15:16]) is used to specify the particular GMM port to which this global system operation is directed. The transmitting port places the address code of the receiver in this field. This address may be either the Port Identification (PID) of the receiver or the logical name of the receiver system or subsystem. After the transmitter port has transmitted a system control message to another processor unit in the global system, it retains control of the system control bus interface, and waits for a response from the receiver.

The receiver accepts the transmitted data from the transmitter, and then proceeds to perform the function indicated by the OP CODE field. Before beginning to do what the OP CODE directed, a parity test of the transmitted data is performed. If a parity error is detected in the transmitted data, the OP CODE is disregarded, and a response word is constructed in the sender's response register (see Figure 5-55). If no parity error is detected in the transmitted data, the receiver executes the instruction contained in the OP CODE field.

When a receiver has completed the instruction contained in the transmitted word, it forms a response word in the sender's response register. This word indicates that the required operation was performed, and gives information or status that is required because of the nature of the performed operation.

While the required operation is being performed by the receiver, the sender retains control of the system control interface bus. Consequently, when the operation is completed, the receiver GMM has access to the sender GMM response register. The sender GMM is responsible for maintaining control over the system control interface bus until the receiver has completed the response. The format of a normal receiver response is shown in Figure 5-54.

After the transmitter has received back the response word, control of the system control interface bus is passed to another processor port for possible control bus transmission. If the next processor needs to use the system control bus, it holds control of the bus until the needs are completed; otherwise, it passes control of the bus to the next processor port. In this manner, control of the system control interface bus is passed from processor port to processor port.

MEMORY STORAGE UNIT PORT INTERFACE

External port interfaces are used to connect the memory control to the units that are remote from the CPU cabinet. The units that are remote from the CPU cabinet and the information that is transmitted on each cable of the interface are as follows:

Unit(s)	Type of Interface	Cables and Signals
External Local Memory Unit	Local Memory 264 wire, six cable.	Six cables are used to interface each of four possible memory storage units to the memory exchange. Each cable contains 44 wires which may be used to pass information, control, and address data between the storage unit and the memory control port. All signal lines of the local memory interface bus are single direction lines, and no cable lines are used to pass data in both directions.

Unit(s)	Type of Interface	Cables and Signals
	Cable Name	Signals on the Cable
	1	This cable is used to pass a 16-bit address to the memory storage unit, and is also used to pass a 3-bit address check value from the storage unit back to the memory control. The other lines on this cable are not used.
	2	This cable is used to pass 12 control signals from the storage unit to the exchange port, or vice versa. The other wires of this cable are not used.
	3	This cable is used to pass 15 write data signals (14:15) and 15 read data signals (14:15) between the storage module and the exchange port. The other wires of this cable are not used.
	4	This cable is the same as cable 3, except that it passes write data bits (29:15) and read data bits (29:15).
	5	This cable is the same as cable 3, except that it passes write data bits (44:15) and read data bits (44:15).
	6	This cable is the same as cable 3, except that it passes write data bits (59:15) and read data bits (59:15).

Unit(s)	Type of Interface	Cables and Signals
Global Memory Module	**Global Memory 120 wires, six cables**	Six cables are used to interface a GMM cabinet to the B 6900 system memory exchange. Each cable contains 20 wires which can be used to pass information, control, and address data between the GMM cabinet and the global memory port. Signal lines of the global memory interface bus are either uni-directional or bi-directional, depending on the individual signal circuit usage.
	Cable Name	Signals on the Cable
	1	This cable is used to pass a 20-bit (GA00 through GA19) address field to the GMM from the B 6900 memory exchange interface. The address field circuits are uni-directional (from the B 6900 to the GMM).
	2	This cable is used to pass the low-order 20-bits of the 60-bit information word (GI00 through GI19) between the B 6900 memory exchange and the GMM. These 20 lines are used bi-directionally for both READ and WRITE type global memory (or system control) operations.
	3	This cable is identical to cable 2, except that it passes information bits GI20 through GI39.

Unit(s)	Type of Interface	Cables and Signals
	Cable Name	Signals on the Cable
	4	This cable passes 12 information bits (GI40 through GI51) in the same way that cables 2 and 3 operate. In addition, this cable is used to pass 10 uni-directional control signals, as follows:

Control Signal Mnemonic	Control Signal Name	From	To
INVA	Invalid Address	GMM	B 6900
GREQ	Global Request	B 6900	GMM

Control Signal Mnemonic	Control Signal Name	From	To
GWRC	Global Write Control (RMW)	B 6900	GMM
GABX	Global Access Begun	GMM	B 6900
GAOX	Global Access Obtained	GMM	B 6900
GPRC (SHARE 1 CABLE LINE)	Global Write Protect Control	B 6900	GMM
GREX (SHARE 1 CABLE LINE)	Global Read Error	GMM	B 6900
GAPL (SHARE 1 CABLE LINE)	Global Address Parity Level	B 6900	GMM
GUEX (SHARE 1 CABLE LINE)	Global Uncorrectable Error	GMM	B 6900
GSCX	Global Scan Control	B 6900	GMM

Cable Name	Signals on the Cable
5	This cable passes 8 check-bit information signals (G152 through G159) in the same way that cables 2 and 3 operate. In addition, this cable is used to pass 5 uni-directional control signals as follows:

Control Signal Mnemonic	Control Signal Name	From	To
GAOR	Global Access Obtained Return	B 6900	GMM
GCWC	Global Clear Write Control	B 6900	GMM
GAEX	Global Address Error	GMM	B 6900
GWEX	Global Write Error	GMM	B 6900
GMMA	Global Memory Module Available GMM	GMM	B 6900

This cable also contains 7 spare unused signal lines.

Unit(s)	Type of Interface	Cables and Signals
	Cable Name	Signals on the Cable
	6	This cable is used to pass 12 uni-directional system control signals as follows:

Control Signal Mnemonic	Control Signal Name	From	To
HALT	Halt	GMM	B 6900
HLTD	Halted	B 6900	GMM
CLER	Clear	GMM	B 6900
CLRD	Cleared	B 6900	GMM
LOAD	Load	GMM	B 6900
STRT	Global Start	GMM	B 6900
SHLT	Super Halted	B 6900	GMM
IDLE	Idle	B 6900	GMM
RUNG	Running	B 6900	GMM
SAVL	System Available	B 6900	GMM
EINT	External Interrupt	GMM	B 6900
AINT	Alarm Interrupt	GMM	B 6900

This cable also contains 8 spare unused signal lines.

Local Memory Port Interface Control Logic

The logical control signals of the port interface (cable 2) are as follows:

Signal Name	Signal Usage
RMW, WCC, PED	Signals RMW (READ/MODIFY/WRITE), WCC (Write Cycle Control), and PED (Parity Error Disable) form a 3-bit code that is used to define the type of operation to be performed by the memory storage unit. The types of operations performed by the storage unit are as follows:

RMW	WCC	PED	Function
0	1	0	Clear WRITE operation
0	0	1	Memory READ restore operation
1	1	0	READ/MODIFY/WRITE

IMC	The Initiate Memory Cycle signal. Two IMC signals are required to perform READ/MODIFY/WRITE memory operations. The memory control generates both IMC signals (one for the READ portion of the operation, followed by another one for the WRITE portion of the operation), and transmits them on the interface IMC wire. The timing of these two IMC signals is a function of the memory control.

Signal Name	Signal Usage
PAR	The Memory Address Parity bit. This signal is sent from the memory control to the memory storage unit to cause the 17-bit address field plus the RMW, WCC, and PED signals to have odd parity. If the number of binary one bits in the address field is even, the PAR signal will be true, thus making an odd number. If the number of binary one bits in the address field is odd, the PAR signal will be false, thus maintaining the odd parity. This signal is only transmitted during the clear WRITE operation. For all other types of memory operations, this signal is forced false.
MPE	The Memory Parity Even signal. This signal is returned from the memory storage unit to the memory control, to indicate whether or not memory address even parity error was detected at the storage unit interface.
WST	The Write Strobe signal. This signal is the write strobe signal for a memory WRITE operation. The memory control generates this signal and transmits it to the memory storage unit which is to perform the WRITE portion of a memory cycle. The system memory control must generate this signal instead of the memory storage unit, because the WRITE portion of a memory cycle is performed after a possible retry of the READ portion is completed.
MSW	The Memory Select Write signal. This signal is used to define whether the read register or the write register is to be used as the source of data for the WRITE portion of a READ/MODIFY/WRITE operation. If the MSW signal is a true level, the write register is the source; otherwise, the read register is the source.
PCS (general clear)	The Memory Storage Unit Clear signal. This signal is generated in the memory control and is used to clear the logic circuits of the memory storage unit.
HAR	The Hold Address for return control signal. This signal is generated in the memory control, and transmitted to the memory storage unit to cause the storage unit to hold the memory address by using its address latch circuits. This signal is required in order to make it possible to single pulse a memory storage unit operation.
MAV. .	The Memory Available control level. This signal is generated in the memory storage unit, and a true level is transmitted to the memory control when the storage unit is powered-up.

Global Memory Port Interface Control Logic

A global system control access requires that a special bit (bit D) on the Z12 bus be true. When bit D of the Z12 memory bus is true during the initiation of an access to the global memory interface, signal GSCX also is true, indicating that a global system control (global scan) operation has been requested.

The control logic signals for a global memory or global system control request are as follows:

Signal Name	Signal Usage
GMMA	Global Memory Module Available. This signal is present at the global memory interface of the CPU cabinet if global memory is available as a resource of the B 6900 system. If this signal is not present, then no global memory is connected to the system, or the global memory is not available for the use of the system.

Signal Name	Signal Usage
GREQ	Global Request. This signal is sent from the B 6900 system to the global memory to indicate that the system requests a global memory operation.
GSCX	Global Scan Control. This signal is sent from the B 6900 system to the global memory to indicate that the request present on the global memory interface is for a scan cycle rather than for a memory cycle.
GAPL	Global Address Parity Level. This signal is an odd parity bit for the 19-bit global address (scan function word) plus the GREQ, GWRC, GPRC, GCWC, and GSCX control signals. This signal is sent from the B 6900 system to the global memory subsystem.
GWRC	Global WRITE Request. This signal is sent from the B 6900 system to the global memory to indicate that a READ/MODIFY/WRITE memory cycle is requested on the word specified by the memory address lines. The information present at the specified address is returned to the B 6900 system, and the information present on the global interface is written into the specified address. The WRITE request may be aborted if this is a protected memory WRITE operation (GPRC is TRUE) and the memory word is protected, or if an address or control error is detected on the global interface bus. If the WRITE request is aborted, the memory accessed information in the address is restored to the same memory address.
GPRC	Global Write Protect Control. This signal is sent to the global memory from the B 6900 system, and requires that the WRITE portion of a READ/MODIFY/WRITE memory operation be aborted if the memory protect bit is true in the data read from memory. The memory word is protected if bit GI48 is TRUE in the READ information. If the WRITE portion is aborted, the READ information is restored to the same memory address, and the WRITE information is not written into memory. The B 6900 must monitor the READ information returned to determine if the WRITE portion of the memory cycle was aborted.
GCWC	Global Clear Write Control. This signal is sent from the B 6900 system to the global memory for both a CLEAR/WRITE memory operation and a global scan operation. If a CLEAR/WRITE memory function is specified, no READ information is returned to the B 6900 system, and the WRITE data is written into the memory address specified. If the memory WRITE function is aborted, the GUEX signal is returned to the B 6900 system. If a scan operation function is specified and GCWC is present, then a SCAN-OUT type function is to be performed. Otherwise, a SCAN-IN function is to be performed.
GABX	Global Access Begun. This signal is returned to the B 6900 system from global memory to indicate that the requested global memory function has been started. When the B 6900 system receives this returned signal, the GREQ signal line is turned off, and the GAPL and GPRC signals are turned off. The GABX signal remains present throughout the remainder of the global memory cycle.
GAOX	Global Access Obtained. This signal is returned to the B 6900 system from global memory to indicate that memory READ data is present on the global memory interface bus. Any error signal associated with the current global memory request (GAEX, GREX, or GUEX) is returned to the B 6900 system at the same time that GAOX is returned.

Signal Name	Signal Usage
GAOR	Global Access Obtained Return. This signal is returned to the global memory to acknowledge the presence of the GAOX signal. This signal, when true, implies that the B 6900 system has captured the memory READ data (or SCAN-IN word) in the logic circuits of the memory controller. When this signal is present at the global memory, the GAOX signal is removed from the global memory interface bus. When the GAOR signal is removed from the global memory interface bus, any error signals present on the bus, plus the GABX signal, will be removed from the global memory interface, thereby indicating the completion of the global request.
GAEX	Global Address Error. This signal is returned to the B 6900 system to indicate that an address parity error was detected on the global memory bus, or that an address error occurred on the module interface (between the global memory control and the global memory storage module).
GREX	Global READ Error. This signal is returned to the B 6900 system to indicate that the information read from the memory module contained an error. The READ information error may be either a single bit error or a multiple bit error (see the GUEX signal description).
GWEX	Global WRITE Error. This signal is returned to the B 6900 system to indicate an error in the WRITE information SCAN-OUT data word. The error present is either a single bit error or a multiple bit error (see the GUEX signal description).
GUEX	Global Uncorrectable Error. This signal is sent to the B 6900 system to indicate an uncorrectable error detected by the global memory. If this signal is TRUE and a WRITE into memory type of operation is in process, the WRITE memory operation will be aborted and the information read from memory during the READ portion of the memory cycle will be restored into the same memory address. GUEX is returned to the B 6900 system to indicate multiple bit errors, and/or memory address errors, and/or memory control signal errors. If GUEX and GREX are present, a multiple bit READ data error is indicated. If GUEX and GWEX are present, a multiple bit WRITE data error is indicated.

Global Memory Port Processor Status and Control Logic

Cable 6 of the global memory port interface is used to pass system status and control information between a B 6900 system and a GMM. The logic signals passed through cable 6 of the global memory interface are as follows:

Signal Mnemonic	Signal Name	Signal Usage
HALT	Halt	This signal is passed from the GMM to the B 6900 system. When TRUE, this causes the B 6900 system processor to HALT at the end of the current operator in process of execution.
HLTD	Halted	This signal is passed from the B 6900 system processor to the GMM. When TRUE, this signal indicates to the GMM that the B 6900 processor is halted.

Signal Mnemonic	Signal Name	Signal Usage
CLER	Clear	This signal is passed from the GMM to the B 6900 system. When TRUE, this signal causes the B 6900 system to be general cleared.
CLRD	Cleared	This signal is passed from the B 6900 system processor to the GMM. When TRUE, this signal indicates to the GMM that the B 6900 processor has raised the internal clear signal line of the B 6900 system.
LOAD	Load	This signal is passed from the GMM to the B 6900 system. When TRUE, the signal causes the B 6900 system to perform a HALT/LOAD sequence from the HALT/LOAD unit. The B 6900 system only accepts this signal input after the B 6900 system is halted.
STRT	Start Global	This signal is passed from the GMM to the B 6900 system. When TRUE, this signal indicates that the GMM has a message in its message buffer for the B 6900 system.
SHLT	Super Halted	This signal is passed from the B 6900 system to the GMM. When TRUE, this signal indicates that the B 6900 processor is in an abnormal state.
IDLE	Idle	This signal is passed from the B 6900 system to the GMM. When TRUE, this signal indicates that the B 6900 system is in an IDLE loop.
RUNG	Running	This signal is passed from the B 6900 system to the GMM. When TRUE, this signal indicates tht the running flip-flop is set in the processor logic.
EINT	External Interrupt	This signal is passed from the GMM to the B 6900 system. When TRUE, this signal causes an external interrupt to be sensed in the B 6900 processor interrupt controller. This signal is only effective when the B 6900 processor is operating in normal state, and has no effect when the processor is operating in control state.
AINT	Alarm Interrupt	This signal is passed from the GMM to the B 6900 system. When TRUE, this signal operates in a manner similar to that of the EINT signal, except that the processor of the B 6900 system is interrupted even if it is operating in control state.
SAVL	System Available	This signal is passed from the B 6900 system to the GMM. When TRUE, this signal indicates that the B 6900 system is present and is powered-up.

MEMORY TESTER LOGIC

The B 6900 has memory test logic designed into the hardware circuits of the CPU cabinet. A separate memory tester with access to local memory is not provided. Therefore, when memory tests are to be performed, their execution preempts any other system operation.

The memory tester logic is designed to be used with memory test routines that are resident in the MDP logic circuits. Memory tests are executed on the B 6900 system through messages on the system operators console (ODT) under control of the MDP Executive routine. Thus, memory testing is only performed by system operators who must direct the system to perform memory tests.

SECTION 6

PROGRAM OPERATORS

GENERAL

The machine language operators are composed of syllables in a program string. The operators are divided into four major classes: primary mode, variant mode, edit mode, and vector mode operators.

SYLLABLE ADDRESSING AND SYLLABLE IDENTIFICATION

A machine language program is a string of syllables which are normally executed sequentially. Each program word in memory contains six 8-bit syllables. The first syllable of a program word is labeled zero and is formed by bits 47 through 40 (see Figure 6-1).

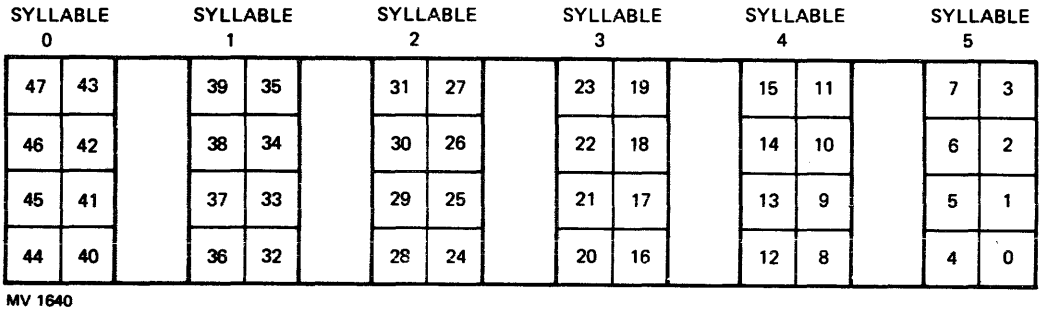


Figure 6-1. Program Word

P AND T REGISTERS

The P register contains the currently active program word. The T registers are the control (instruction) registers. There is one 4-bit T register for each operator family. The T register contains the code for the specific type of operator to be executed by the family, and is usually derived from the four low-order bits of the operator syllable code. The four high-order bits of the operator syllable code are used to select a family strobe. This family strobe is used to define which family is to receive the strobe pulse (execute pulse). Figure 6-2 shows how a program operator code in the P register is decoded to select a family strobe and a T register value. In the example shown in Figure 6-2, a divide operator (OP code 83 hexadecimal) is in the process of being executed, and this operator caused the family A strobe (STRA) to be selected. The family A T register contains a value of three (hexadecimal) which is derived from the four low-order bits of the operator code.

Figure 6-2 also shows an example of how a word of program code is selected to be executed. The addressing mechanism for program code words and the way the controllers of the B 6900 data processor function to provide automatic program code handling operation is also shown in this example.

In the program code handling example shown in Figure 6-2, the Program Base Register (PBR) points at the first word of program code in the current program code segment. The value of the PBR is initially established from the segment descriptor for the current program segment when the procedure is initiated.

The current word of program code in a program segment presently being executed is indicated by the value of the Program Index Register (PIR). The initial value of the PIR for a program segment is established from the PCW word that caused the segment to be executed. The initial value of PIR may also be established from an RCW, if the program segment is executed as the result of an exit or return from another code segment in the same program.

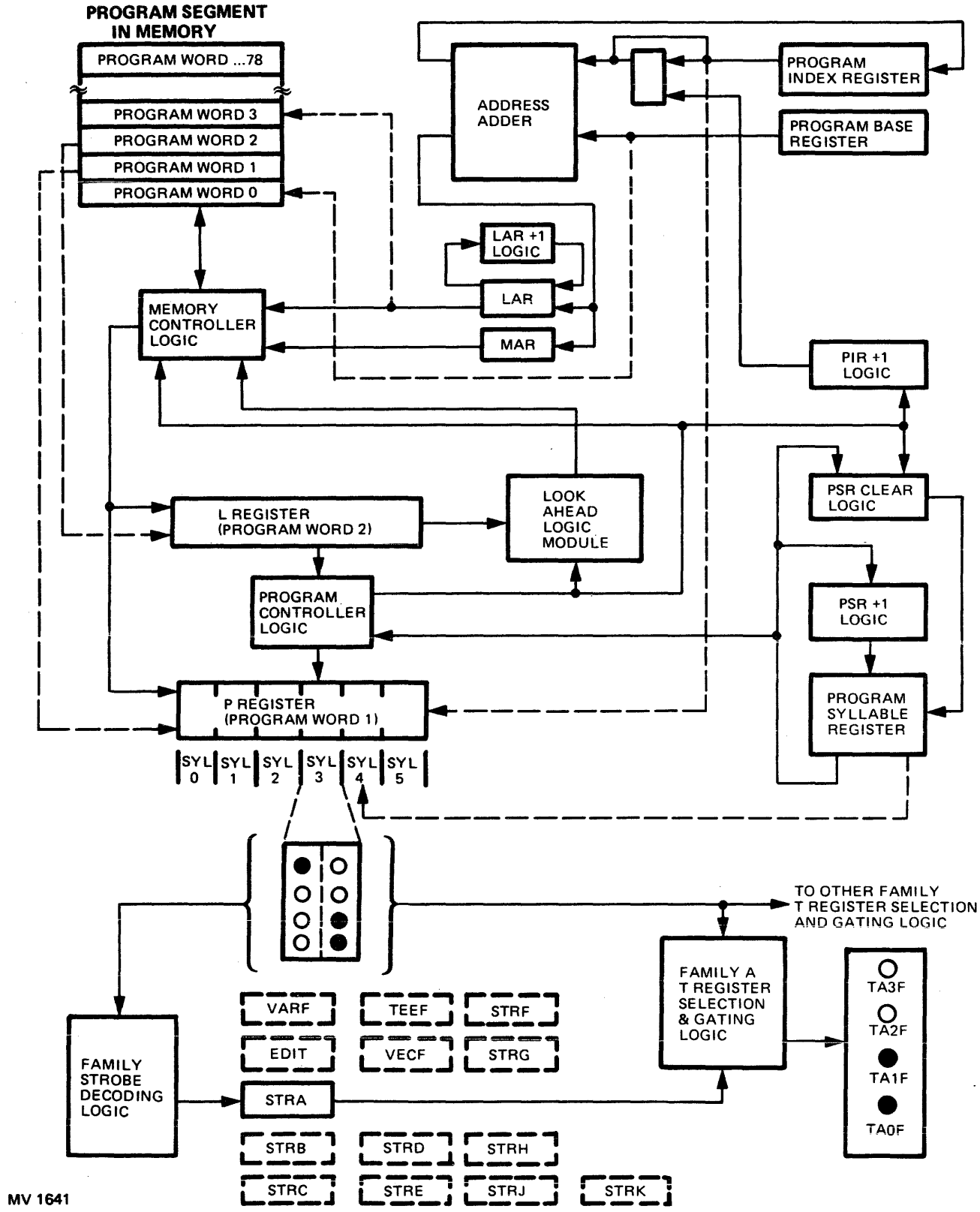


Figure 6-2. Program Word, Syllable Addressing

The first syllable to be executed in a program code segment is derived from the PCW (or alternatively the RCW) that caused entry into the current program segment. In the example shown in Figure 6-2, the Program Syllable Register (PSR) is pointing at syllable four of the P register because the divide operator (in syllable three) is being executed, and the PSR plus one logic has advanced the value of the PSR to point at the next syllable that will be executed.

Program code words in the B 6900 system are normally fetched from system memory by the look ahead logic. The look ahead logic fetches the next word of program code while the current word of program code is being executed, and places it in the L register. When the PSR indicates by its content value that all syllables of program code in the P register have been executed, the program controller causes the next word of program code to be transferred from the L register to the P register. The PSR points at the first syllable in the new program word.

When the next word of program code is transferred from the look ahead logic L register to the P register, the look ahead module causes the next word of program code to be fetched from memory and placed in the emptied L register. The program controller causes the value of the PIR to be incremented by one, as the operators are strobed from the P register. Thus, the PIR always points at the code word the present operator started in. The look ahead logic uses the Look Ahead Address Register (LAR) to address the next word of program code. The LAR has an automatic plus one incrementation feature that causes the LAR to always point at the memory address of the next program word (following the program word that is present in the L register).

The dotted lines in Figure 6-2 show the origin of a word of program code in the P and L registers, and also what word of the program segment is pointed at by an address register. A dotted line is also used to show that the value of the PSR temporarily points at syllable four when syllable three is being executed by the data processor.

OPERATION TYPES

Operations are grouped into three classes: name call, value call, and operators. The two high-order bits (bits 7 and 6) determine whether a syllable begins a value call, name call, or operator (Figure 6-3).

(BITS 7 AND 6) IDENT	SYLLABLE TYPE	NO. OF SYLLABLES	FUNCTION
00	VALUE CALL	2	BRINGS AN OPERAND INTO THE STACK
01	NAME CALL	2	BUILDS AN IRW IN THE STACK
OTHER THAN ABOVE	OTHER OPERATORS	1→7	PERFORMS THE SPECIFIED OPERATION

MV 1642

Figure 6-3. Primary Mode Operator Syllable Decode Table

Name Call

Name call builds an indirect reference word in the stack (see Figure 6-4). Stack adjustment takes place so that the A register is empty. The six low-order bits of the first syllable of this operator are concatenated with the 8-bits of the following syllable to form a 14-bit address couple. The address couple is placed, right-justified, into the A register, with the remainder of the A register filled with zeroes. The TAG field of the A register is set to 001, and the register is marked full.

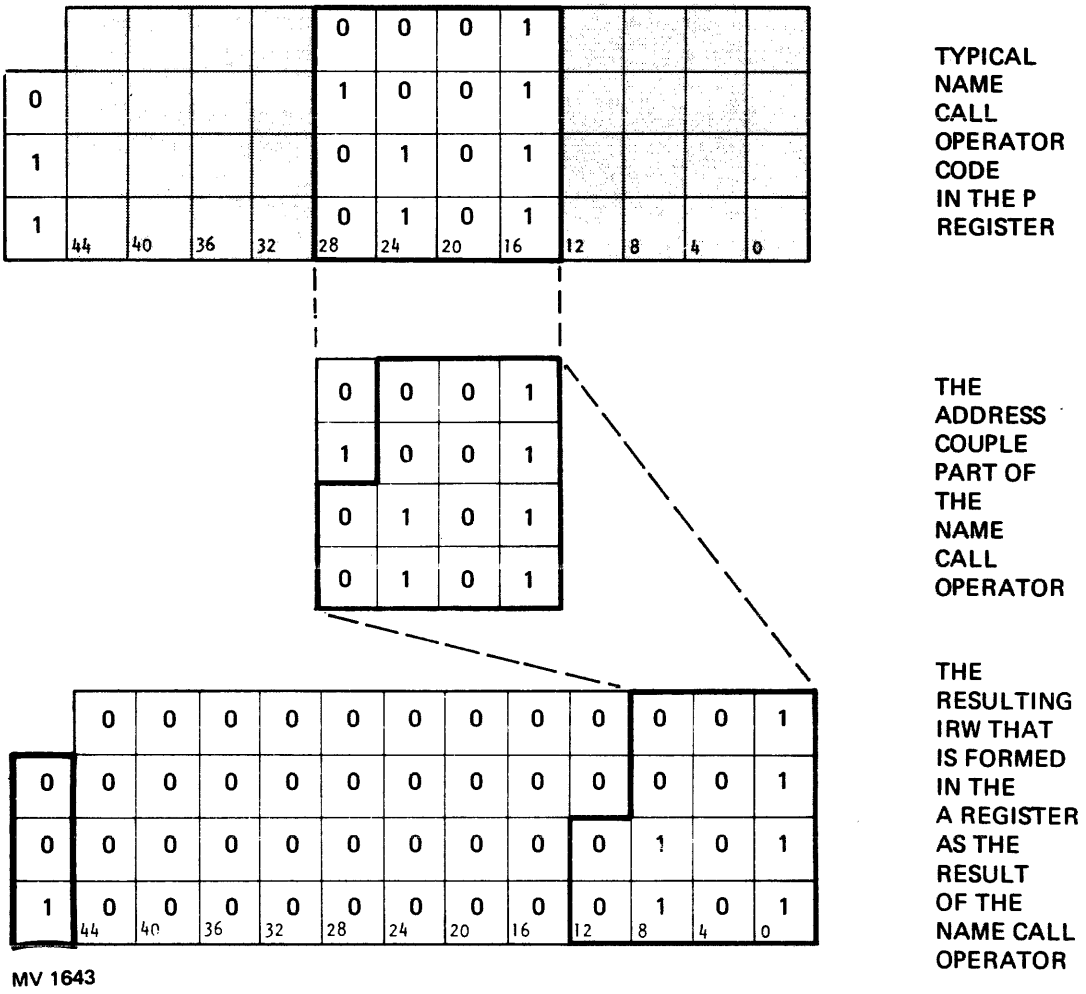


Figure 6-4. Name Call Operator Function

Value Call

Value call loads into the top of the stack the operand referenced by the address couple. The operator is formed in the same manner as the name call operator. If the referenced memory location is an indirect reference word or a data descriptor, additional memory accesses are made until the operand is located. The operand is then placed in the top of stack registers. The operand may be either single- or double-precision, causing either one or two words to be loaded into the top of the stack.

Figure 6-5 is an example of how a value call operator (VALC) is used to cause a word of data located at memory address D2 plus 4 to be fetched and placed in the top of the D3 stack. The current stack is known to begin at the MSCW pointed at by the D3 display IC memory register, because the lexicographical level register contains a value of 3 (LL00, LL01, LL02/, LL03/, LL04/).

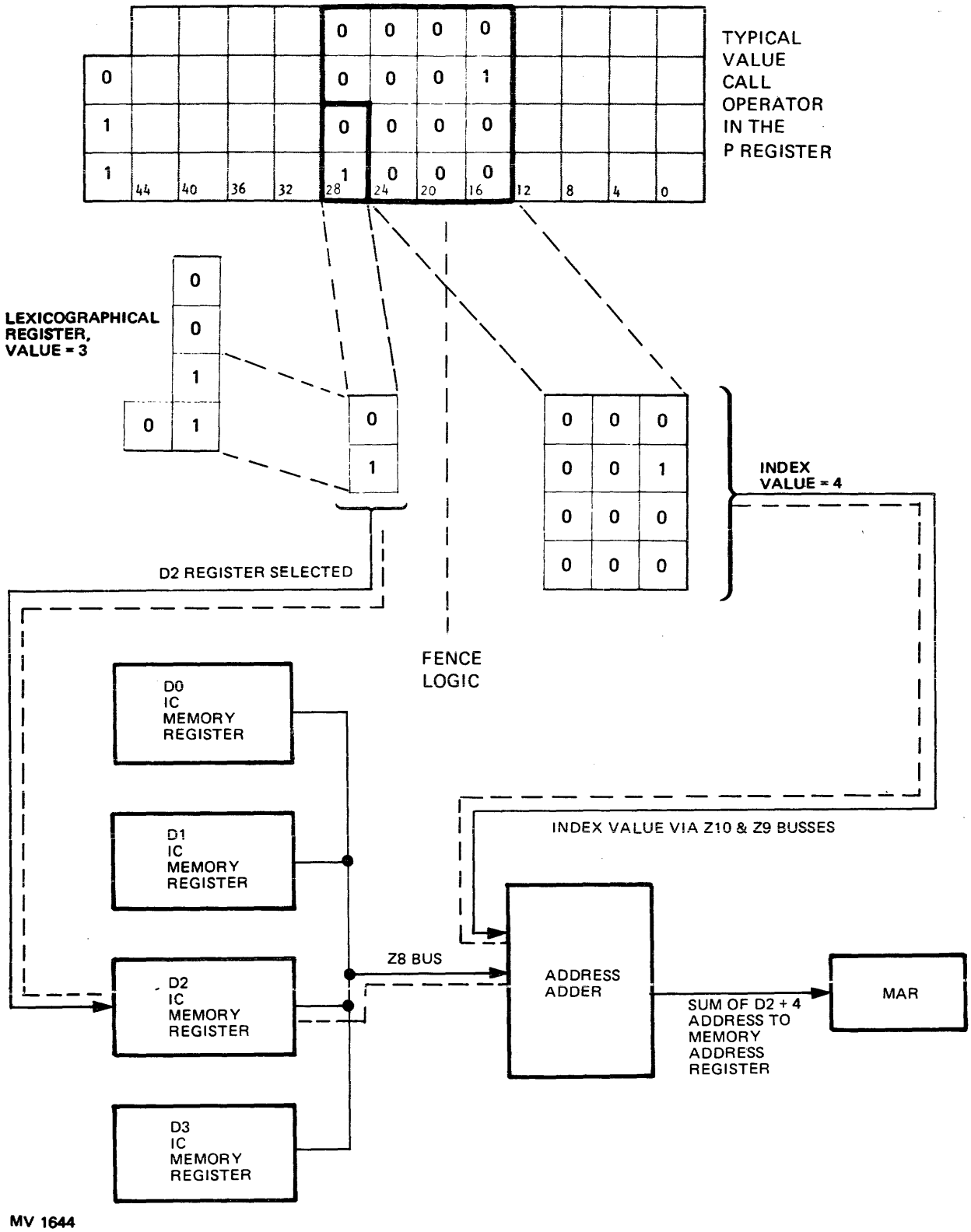


Figure 6-5. Value Call Operator Function

The fence decoding logic defines the number of bits in the address couple that select a display register to provide the base address portion of the value call operation. The fence decoding logic uses the current programming level of the program segment to determine which IC memory display register is selected. The highest order bit of the lexicographical level register that is true in the example is bit LL01, which has a value of two. The fence decoding logic therefore uses the two high-order bits of the address couple to select an IC memory display register as the source of the base address. The bits that are not used by the fence decoding logic to select a display register form the index portion of the value call operation.

Bits 29:5 are used by the fence decoding logic to select a display register. The value of the bits in this field are opposite to the word bit number order; that is, bit 29 of the address couple in the example has a binary value of one, and bit 25 has a binary value of 16. The following equates bits 29:5 to a decimal value and to the display registers which they will select.

Bit Number	Decimal Value	Display Register Selected
29	1	1
28	2	2
27	4	4
26	8	8
25	16	16

Thirty two IC memory display registers may be selected by the fence decoding logic.

In the example in Figure 6-5, it is possible to see how bit 28 is used to select display register two and thus to provide the base portion of the value call address.

The index portion of the address couple is treated in the conventional manner as a binary value. In the example shown in Figure 6-5, bits 16, 17, and 18 have a binary value of 100, which is four decimal.

The absolute memory address placed in the memory address register in the example in Figure 6-5 is the sum of the address from display register two and the index, which has a value of four (that is, D2+4). The word of data in memory at the absolute memory address is fetched and placed in the top of stack register. If the word at D2+4 is an IRW or a data descriptor, then additional fetches from memory will be made. This process continues until an operand or a data word is placed in the top of stack register. Placing an operand or a data word in the top of stack register completes the value call operation.

The value call operator detects an invalid operand error condition if a word with a TAG code of three, four, or six is referenced. If a word with a TAG of seven is referenced by a value call operator, an accidental program entry into the procedure described by the PCW is performed. The final value placed in the stack by a value call operator must have a TAG field of zero or two.

An accidental program entry caused by a value call operator being executed is treated like a subroutine of the procedure that executed the value call operator. The stack of the procedure is marked by an MSCW and an RCW. Then the subroutine referenced by the PCW is executed; it terminates by means of a return operator. The return operator passes a parameter from the subroutine to the procedure that executed the original value call operator. The program flow of the procedure is resumed at the next operator in sequence following the original value call operator.

Operators

Operators vary from one to seven syllables in length. The first syllable of each operator determines the number of additional syllables forming the operator. Upon completion of each operator, the PSR addresses the first syllable beyond all of the syllables comprising the operator.

Operators work on data either as full words (48 data bits plus TAG bits) or as strings of data characters. Word operators work with operands (single or double-precision) in the top of the stack.

String operators are used for transferring, comparing, scanning, and translating strings of digits, characters, or bytes. In addition, a set of micro-operators provides a means of formatting data for input or output.

The string operators use source and destination pointers located in the stack. These pointers are set into the following hardware registers:

1. Source Base Register – (SBR).
2. Source Index Register – (SIR).
3. Source Index Byte Register – (SIB).
4. Source Size Register – (SSZ).
5. Destination Base Register – (DBR).
6. Destination Index Register – (DIR).
7. Destination Index Byte Register – (DIB).
8. Destination Size Register – (DSZ).

In some of the string operators, the source pointer may not be used. In this case, an operand may be in the stack; its characters are circulated as the operand is being used.

String operators have an optional update function; that is, producing updated source and destination pointers and count. At completion of an operation, the source and destination pointers are updated as follows:

1. If the source is an operand, it remains in the stack.
2. If the pointer is a descriptor, the word index fields and byte index fields are updated from SIR/DIR and SIB/DIB. The string size fields are updated from SSZ/DSZ.
3. If the pointer is a data descriptor or a non-indexed string descriptor, it is converted to an indexed string descriptor and updated.

If both the source and destination descriptors have size fields equal to zero, the size registers indicate 8-bit character size. When both a source and destination are required and the size field of one is equal to zero and the other is not, then the size field of the non-zero descriptor is used.

If neither size field is equal to zero and the size fields are not equal and the operator is not translate, the invalid operand interrupt is set and the operator is terminated. The size field is considered equal to zero when the source is an operand.

SECTION 7

PRIMARY MODE OPERATORS

GENERAL

This section defines the functions of the primary operators. In each case, the name of the operator, corresponding mnemonic, and hexadecimal code are shown. Appendix A of this manual lists the operators in alphabetic order, and appendix B lists the operators in numeric order, by mode.

The universal operators are also included in this section.

ARITHMETIC OPERATORS

The arithmetic operators usually require two operands in the top of stack registers. These operands are combined by the arithmetic process specified with the result placed in the top-of-stack. The operands may be either single-precision, double-precision, or intermixed. The specified arithmetic process adapts automatically to the data environment, with the single-precision process invoked if both operands are of the single-precision type, and the double-precision process invoked if either operand is of the double-precision type.

Each double-precision operand occupies two words. The second word of the operand is an extension of the first word of the operand. The mantissa of the first word of the operand contains unit values, and the mantissa of the second word contains a fractional unit value. An implied octal point separates the mantissa of the first word from the mantissa of the second word. When the top-of-stack registers are full, the first word of the first operand is in the A register; the second word of the first operand occupies the X register. The first word of the second operand resides in the B register; the second word of the second operand occupies the Y register. Therefore, double-precision arithmetic processes operate on four words in the stack, instead of two as in single-precision operations. Double-precision arithmetic leaves a two-word result in the top-of-stack.

Add, subtract, and multiply operations which use two integer operands yield an integer result if no overflow occurs. If one or both operands are non-integer, or if the result generates an overflow, the result is non-integer.

When an operator has been entered, the hardware stack-adjust function fills or empties the top-of-stack register as required by the operator. If either register contains an incorrect word, the operator is terminated by an invalid operand interrupt.

ADD (ADD) 80

The operands in the A register and the B register are added algebraically, with the sum left in the B register. At the end of the operation, the A register is marked empty, and the B register is marked full.

If only one of the operands is double-precision, the register (X or Y) associated with the register that contains the single-precision operand is set to all zeroes. The B register is marked as a double-precision operand at completion of the operation.

If the mantissa signs and the exponents are equal, the mantissas are added and the sum placed in the B register. If the sum exceeds 13 single precision (26 double precision) octal digits, the mantissa of the sum is shifted right one octade, rounded, and the exponent is algebraically increased by one. The meaning of exponents and mantissas were given in section 2 of this manual.

If the exponents are equal but the mantissa signs are unequal, the difference of the mantissas plus the appropriate sign is placed in the B register.

If the exponents are unequal, the operands are first aligned. If the alignment causes the smaller operand to be shifted right 14 single precision (27 double precision) octal places, the larger operand is the result.

If the alignment causes the smaller operand to be shifted right, but less than 14 single precision (27 double precision) octal places, the digits of the smaller operand shifted out of the register are saved and used to obtain the rounded result.

If the signs of the operands are equal, the mantissas are added and the sum placed in the B register. If the sum does not exceed 13 single precision (26 double precision) octal digits, the last digit shifted out of the register is used to round the result. If the sum is 14 single precision (27 double precision) octades, the mantissa in B (Y) is rounded to 13 single precision (26 double precision) digits.

If the signs of the operands are unequal, an internal subtraction takes place, with the rounded result placed in the B register.

If the result has an exponent greater than +63 (+32,767), the exponent overflow interrupt is set. If the result has an exponent less than -63 (-32,767), the exponent underflow interrupt is set.

SUBTRACT (SUBT) 81

The operand in the A register is algebraically subtracted from the operand in the B register, with the difference left in the B register. The operation is the same as for the Add operator, except for initial sign comparisons.

MULTIPLY (MULT) 82

The operand in the A register is algebraically multiplied by the operand in the B register. The rounded product is left in the B register.

If the mantissa of either operand is zero, the B register is set to all zeroes.

If both mantissas are non-zero, the product of the mantissa is computed. If the product contains more than 13 single-precision (or 26 double-precision) digits, it is normalized and rounded to 13 single-precision (or 26 double-precision) digits. A mantissa of all sevens is not rounded. Normalization was explained in section 2 of this manual.

If the result has an exponent greater than +63 (+32,767), an exponent overflow interrupt is set. If the result has an exponent less than -63 (-32,767), an exponent underflow interrupt is set.

EXTENDED MULTIPLY (MULX) 8F

The operands in the A and B registers are algebraically multiplied, and a double-precision product is placed in the B and Y registers. The A register is marked empty, and the B register marked full.

The actions outlined for multiply operations also apply to this operator.

If either or both operands are double-precision, then a normal double-precision operation occurs.

DIVIDE (DIVD) 83

The operand in the B register is algebraically divided by the operand in the A register, with the quotient left in the B register. After the operation, the A register is marked empty, and the B register is marked full.

If the mantissa of the B register is zero, the B register is set to all zeroes. If the A register mantissa is equal to zero, the divide by zero interrupt is set. In either case, the operation is terminated.

If the mantissas of both operands are non-zero, they are normalized, and the operand in the B register is divided by the operand in the A register. The quotient is developed to 14 single-precision (or 27 double-precision) digits, rounded to 13 single-precision (or 26 double-precision) digits, and remains in the B register.

If the result has an exponent greater than +63 (32,767), the exponent overflow interrupt is set. If the result has an exponent less than −63 (−32,767), the exponent underflow interrupt is set.

INTEGER DIVIDE (IDIV) 84

The operand in the B register is algebraically divided by the operand in the A register, and the integer part of the quotient is left in the B register. After the operation, the A register is marked empty, and the B register is marked full.

If the mantissa of the B register is zero, the B register is set to all zeroes. If the mantissa of the A register is zero, the divide-by-zero interrupt is set. The operation is terminated in either case.

If the mantissas of both operands are non-zero, they are normalized. If the exponent of the B register is algebraically less than the exponent of the A register after both operands have been normalized, the B register is set to all zeroes. If the exponent of the B register is algebraically equal to or greater than the exponent of the A register, the divide operation proceeds until an integer quotient or a quotient of 13 single-precision (or 26 double-precision) significant digits is calculated.

If an integer quotient is developed, the quotient is left in the B register with a zero exponent for single-precision, and the exponent set to 13 for double-precision. If a non-integer quotient is developed, the integer overflow interrupt is set.

REMAINDER DIVIDE (RDIV) 85

The operand in the B register is algebraically divided by the operand in the A register to develop an integer quotient. The remainder of this division stays in the B register.

If the mantissa of the B register is zero, the B register is set to all zeroes. If the mantissa of the A register is zero, the divide-by-zero interrupt is set. In either case, the operation is terminated.

If both mantissas are non-zero, both operands are normalized. If the exponent of the B register is algebraically less than the exponent of the A register after both operands have been normalized, the operand in the B register is the result. If the exponent of the B register is algebraically equal to or greater than the exponent in the A register, the DIVIDE operation proceeds until an integer quotient is developed; the remainder is then placed in the B register.

If a non-integer quotient is developed, the integer overflow interrupt is set and the operation is terminated.

INTEGERIZE, TRUNCATED (NTIA) 86

The operand in the B register is converted to integer form without rounding, and remains in the B register.

If the operand in the B register cannot be integerized (that is, the exponent is greater than the number of leading zeroes in the operand), the integer overflow interrupt is set and the operation is terminated.

INTEGERIZE, ROUNDED (NTGR) 87

The operand in the B register is converted to integer form. Rounding takes place if the absolute value of the fraction is greater than four. The rounded result is left in the B register.

If the operand in the B register cannot be integerized (that is, the exponent is greater than the number of the leading zeros in the operand), the integer overflow interrupt is set and the operation is terminated.

The operand is rounded, if necessary, by adding one to the mantissa. If a non-integer results from this operation, the integer overflow interrupt is set.

TYPE-TRANSFER OPERATORS

The three type transfer operators are discussed in the following paragraphs.

SET TO SINGLE-PRECISION, TRUNCATED (SNGT) CC

The operand in the top-of-stack register is normalized and set to a single-precision operand; or in the case of a data descriptor, the double-precision bit is set to zero.

If the word in the top-of-stack register is a non-indexed, double-precision data descriptor, the double-precision bit is cleared to zero and the length field multiplied by two.

If the double-precision operand in the top-of-stack register has an exponent greater than +63 after normalization, the exponent overflow interrupt is set. If the exponent is less than -63 after normalization, the exponent underflow interrupt is set, and the operation is terminated.

If the operand in the top-of-stack register is a double-precision operand with an exponent less than +63 or greater than -63, the operand is normalized and the TAG field in the top-of-stack register is set to single-precision.

If the word in the top-of-stack register is neither an operand nor a data descriptor, the invalid operand interrupt is set, and the operation terminated.

If the operand is single-precision, it is normalized and the operation is terminated.

SET TO SINGLE-PRECISION, ROUNDED (SNGL) CD

The operand in the top-of-stack register is changed to a rounded, single-precision operand.

If the double-precision operand in the top-of-stack register has an exponent greater than +63, the exponent overflow interrupt is set. If the exponent is less than -63, the exponent underflow interrupt is set. In either case, the operation is terminated.

If the operand in the top-of-stack register is a double-precision operand with an exponent less than +63 or greater than -63, the operand is normalized, the TAG field in the top-of-stack register is set to single-precision, the operand in the top-of-stack register is rounded from the Y register, and the Y register is set to all zeroes.

If a carry is developed during the rounding operation, the operand is adjusted and the new exponent is checked in the manner discussed in the preceding paragraph.

If the operand is a single-precision operand, it is normalized and no rounding occurs.

SET TO DOUBLE-PRECISION (XTND) CE

The word in the top-of-stack register is set to a double-precision operand, and the Y register is set to all zeroes. If a single-precision data descriptor is present in the top-of-stack register, the double-precision bit is set to one.

If the word in the top-of-stack register is a data descriptor with both the index bit and double-precision bit zero, the double-precision bit is set to one and the length field is divided by two.

If the operand in the top-of-stack register is a double-precision operand, the operation is complete. If it is a single-precision operand, the TAG field in the top-of-stack register is set to double-precision, and the Y is set to all zeroes.

If the word in the top-of-stack register is neither an operand nor a data descriptor, the invalid operand interrupt is set and the operation terminated.

LOGICAL OPERATORS

For LAND, LOR, or LEQV, if only one of the operands is in double-precision form, the other operand is treated as double-precision, with the least significant 13 octades equal to all zeroes.

LOGICAL AND (LAND) 90

Each bit of the B operand result, except for the TAG bits, is set to one where a one appears in the corresponding bit positions in both the A operand and the B operand. The other information bits of the B operand result are set to zero. If the TAGs of the two operands are identical, the TAG in the result is that of the B register. If the TAGs are different, the resultant TAG is double-precision.

LOGICAL OR (LOR) 91

Each bit position of the B operand (except for the TAG bits) is set to one if the corresponding bit position in either the A operand or the B operand is one; otherwise, the bit is set to zero. The TAG bits are set to the value of the second item in the stack except when the A operand is double-precision, in which case the B register TAG is set to double-precision.

LOGICAL NEGATE (LNOT) 92

Each bit in the top word in the stack is complemented except for the TAG bits, which remain unchanged. The result is always stored in the A register.

LOGICAL EQUIVALENCE (LEQV) 93

Each bit of the B operand is set to one, except for the TAG bits, when the corresponding bits of the A operand and the B operand are equal. Each bit of the B operand is set to zero (except for the TAG bits) when the corresponding bits of the A and B operands are not equal. The TAG field is normally set to the value of the second item in the stack except when the A operand is double-precision; in that case, the B-register TAG is set to double-precision.

LOGICAL EQUAL (SAME) 94

All bits, including TAG bits of the A operand and the B operand, are compared. If all bits are equal, a single-precision operand with bit zero set and all other bits reset is stored in the B register. Otherwise, a single-precision operand with all bits reset is stored in the B register. AROF is reset, and BROF is set.

RELATIONAL OPERATORS

The relational operators perform an algebraic comparison on the operands in the A register and the B register. The single-precision result is left in the B register, and the B register is marked full. The result is an operand in integer form with the value one if the relationship has been met, or an operand with all information bits set to zero if the relationship was not met. All relational operations compare the B operand to the A operand.

For all relational operators except equal (EQUL) and not equal (NEQL), the compare flip-flop is set when the relation is equal. For the equal or not equal operators, the compare flip-flop is set when the relationship is greater than equal.

The CMPF flip-flop is used in conjunction with the low order bit of the B register (BR[0:1]) to analyze the result of a relational operation. Table 7-1 shows the states of the CMPF flip-flop and BR[0:1] for various relational operations and possible results of relational operations.

Table 7-1. Relational Operator Indications

Relational	BR[0:1]	CMPF	Comparison Result
EQUAL	0	0	Less than
(8C) (EQUL)	0	1	Greater than
	1	0	Equal
	1	1	Not applicable
GREATER THAN	0	0	Less than
(8A) (GRTR)	0	1	Equal
	1	0	Greater than
	1	1	Not applicable
GREATER THAN	0	0	Less than
OR EQUAL	0	1	Not applicable
(89) (GREQ)	1	0	Greater than
	1	1	Equal
LESS THAN	0	0	Greater than
(88) (LESS)	0	1	Equal
	1	0	Less than
	1	1	Not applicable
LESS THAN	0	0	Greater than
OR EQUAL	0	1	Not applicable
(8B) (LESQ)	1	0	Less than
	1	1	Equal
NOT EQUAL	0	0	Equal
(8D) (NEQL)	0	1	Not applicable
	1	0	Less than
	1	1	Greater than

GREATER THAN (GRTR) 8A

If the B operand is algebraically greater than the A operand, the B register is set to one; otherwise, the B register is set to zero. AROF is reset, and BROF is set.

If the result of the algebraic comparison is "equal", the CMPF flip-flop is set.

GREATER THAN OR EQUAL (GREQ) 89

If the B operand is algebraically greater than or equal to the A operand, the B register is set to one; otherwise, the B register is set to zero.

If the result of the algebraic comparison is "equal", the CMPF flip-flop is set. AROF is reset, and BROF is set.

EQUAL (EQUL) 8C

If the operands in the B and A registers are algebraically equal, the B register is set to one; otherwise, the B register is set to zero.

If the result of the algebraic comparison is "greater", the CMPF flip-flop is set. AROF is reset, and BROF is set.

LESS THAN OR EQUAL (LSEQ) 8B

If the B operand is algebraically less than or equal to the operand in the A register, the B register is set to one; otherwise, the B register is set to zero.

If the result of the algebraic comparison is "equal", the CMPF flip-flop is set. AROF is reset, and BROF is set.

LESS THAN (LESS) 88

If the operand in the B register is algebraically less than the operand in the A register, the B register is set to one; otherwise, the B register is set to zero.

If the result of the algebraic comparison is "equal", the CMPF flip-flop is set. AROF is reset, and BROF is set.

NOT EQUAL (NEQL) 8D

If the operand in the B register is not algebraically equal to the operand in the A register, the B register is set to one; otherwise, the B register is cleared.

If the result of the algebraic comparison is "greater than", the CMPF flip-flop is set. AROF is reset, and BROF is set.

BRANCH OPERATORS

Branch instructions break the normal sequence of serial instruction fetches. Branching may be either relative to the base address of the current program segment or to a location in another program segment. Branch operators can be conditional or unconditional.

BRANCH FALSE (BRFL) AO

If the low-order bit of the A register is zero, the Program Index Register (PIR) and Program Syllable Register (PSR) are set from the next two syllables in the program string. Otherwise, PSR is advanced two syllable positions, and PIR is incremented if necessary.

The two syllables following the actual operator syllable form the new PIR and PSR settings, as follows. The three high-order bits are placed into PSR, and the next 13 low-order bits are placed in the PIR. The Program Register (P) is marked empty to cause an access to the new program word.

BRANCH TRUE (BRTR) A1

If the low-order bit of the A register is one, the PIR and PSR are set from the next two syllables in the program string. Otherwise, PSR is advanced two syllable positions, and PIR is incremented if necessary. The Branch True operator uses the two syllables as previously described for the Branch False operator (BRFL).

BRANCH UNCONDITIONAL (BRUN) A2

The PIR and PSR are set from the next two syllables of the program string. The Branch Unconditional operator uses the two syllables as described for the Branch False operator (BRFL).

DYNAMIC BRANCH FALSE (DBFL) A8

If the low-order bit of the B register is zero and the word in the A register is a Program Control Word (PCW) or an indirect reference to one, a branch is made to the specified syllable of that program segment.

If the low-order bit of the B register is zero and the word in the A register is an operand, PIR and PSR are set from this operand.

If the word in the A register is an operand, it is used in the following manner. The operand is made into an integer. If it is negative or greater than 16,384, the invalid index interrupt is set and the operation is terminated. If bit zero of the operand is zero, PSR is set to zero; otherwise, PSR is set to 011. The next higher-order 20 bits are placed in the PIR. The Program Register is then marked empty to cause access to the new program word.

DYNAMIC BRANCH TRUE (DBTR) A9

If the low-order bit of the B register is one and the word in the A register is a PCW (or an indirect reference to one), a branch is made to the specified syllable of the program segment.

If the low-order bit of the B register is one and the word in the A register is an operand, PIR and PSR are set from this operand.

The operand in the A register is used in this operator in the manner described for the Dynamic Branch False operator (DBFL).

DYNAMIC BRANCH UNCONDITIONAL (DBUN) AA

If the word in the A register is a PCW or an indirect reference to one, a branch is made to the specified syllable of the program segment.

If the word in the A register is an operand, PIR and PSR are set from this operand.

The operand in the A register is used in this operator in the same manner described for the Dynamic Branch False operator (DBFL).

STEP AND BRANCH (STBR) A4

The increment field of the step-index word (SIW) addressed by the contents of the A register is added to its current-value field. If the current-value field is then greater than the final-value field, the PIR and PSR are set from the next two syllables in the program string. Otherwise, the PIR and the PSR are advanced three syllables. The SIW is replaced in memory.

If no SIW is in memory and if an operand is found, it is left in the stack. The A register is set to all zeroes, the PIR and PSR are advanced and the next operator is executed. If no operand is encountered, the invalid operand interrupt is set.

UNIVERSAL OPERATORS

The three universal operators are discussed in the following paragraphs.

NO OPERATION (NOOP) FE

No operation takes place when this operator is encountered. PIR AND PSR are advanced to the next operator. This operator is also valid in the variant and edit modes.

CONDITIONAL HALT (HALT) DF

This operator halts the processor if the CHLT pushbutton on the MDP keyboard is illuminated. If the CHLT pushbutton is extinguished, the operator is treated as a NOOP. This operator is also valid in the variant and edit modes.

INVALID OPERATOR (NVLD) FF

This operator sets the invalid operand interrupt. This operator is also valid in variant and edit modes.

STORE OPERATORS

The store operators use the words in the A register and B register. The operand in the B register is stored in memory at the location addressed by an Indirect Reference Word (IRW) or a data descriptor. If the A register contains an operand, a hardware interchange takes place so that the operand is transferred to the B register.

STORE DESTRUCTIVE (STOD) B8

If the word in the A register is an operand, the A and B operands are interchanged. The data descriptor or IRW in the A register is the address in memory where the operand in the B register (B, Y registers for double-precision) is stored. After the operand is stored, the A register and B register are marked empty and the operation is complete.

If the word addressed by the IRW is a program control word, accidental procedure entry occurs. The spontaneously created Return Control Word (RCW) causes the Store Destructive (STOD) operator to be re-executed upon return from the procedure.

If the word addressed by the data descriptor has the memory protect bit on (bit 48), the memory protect interrupt is set and the operation is terminated.

If the presence bit in the data descriptor is zero, the presence bit interrupt is set. After the information has been made present, the operation is restarted.

STORE NON-DESTRUCTIVE (STON) B9

This operator functions in virtually the same way as the STOD operator. However, at the completion of this operator, the BROF remains set, and the operand is retained in the B register.

OVERWRITE DESTRUCTIVE (OVRD) BA

This operator functions in a manner similar to the STOD operator, except that the OVRD operator overrides memory protection checks. The OVRD operator only writes a single data word into memory. If a double-precision data operand is to be

written into memory, the most significant half is written into memory, and the least significant half of the operand is truncated (not written into memory).

OVERWRITE NON-DESTRUCTIVE (OVRN) BB

This operator functions in the manner similar to the STON operator, except that the OVRN operator overrides memory protection checks. This operator also operates in the same manner as the OVRD operator, with regard to double-precision memory data words.

STACK OPERATORS

The four stack operators are discussed in the following paragraphs.

EXCHANGE (EXCH) B6

The operands in the A register and the B register are exchanged. The A and B registers may contain either operands or control words. The control words are treated as operands by this operator.

DELETE TOP-OF-STACK (DLET) B5

This operator marks the top-of-stack register empty.

DUPLICATE TOP-OF-STACK (DUPL) B7

The operand in the B register is copied into the A register, or the operand in the A register is copied into the B register. At the conclusion of the operation, the register that received the copy is marked full.

PUSH DOWN STACK REGISTERS (PUSH) B4

This operator stores the valid word(s) from the A register and/or B register into the memory portion of the stack. The A and B registers are marked empty.

LITERAL CALL OPERATORS

The five literal call operators are discussed in the following paragraphs.

LIT CALL ZERO (ZERO) B0

This operator sets the A register to all zeroes and marks the register full. The result is a single-precision operand.

LIT CALL ONE (ONE) B1

This operator sets the A register low-order bit (bit 0) to one, leaving all other bits set to zero. The A register is marked full. The result is a single-precision operand.

LIT CALL 8-BITS (LT8) B2

The syllable following the operator is the literal value to be placed in bits 7:8 of the A register. The rest of the A register is set to all zeroes. The A register is marked as full, and the PSR is set to the syllable following the literal.

LIT CALL 16-BITS (LT16) B3

The next two syllables following the operator are a 16-bit literal value placed in bits 15:16 of the A register. The rest of the register is set to all zeroes. The A register is marked full, and PSR is advanced past the 16-bit literal.

LIT CALL 48-BITS (LT48) BE

The next program word is placed in the A register, and the A register TAG is set to all zeroes. The A register is marked full, and the PIR and PSR are advanced to the program syllable following the 48-bit literal value. This operator requires that the 48-bit literal in the program string be word synchronized. If the operator syllable is in any syllable position other than syllable five, the intervening syllables are not executed.

The 48-bit literal word must contain a TAG field value of three (program word); otherwise, an invalid program word interrupt will be sensed when the literal word is present in the P (program) register.

MAKE PROGRAM CONTROL WORD (MPCW) BF

This operator performs a "Lit Call 48-Bits" (LT48) as previously described; however, the TAG is set to a PCW (111), and the stack number register is placed in bits 45:10. The A register is marked full.

INDEX AND LOAD OPERATORS

The four index and load operators are discussed in the following paragraphs.

INDEX (INDX) A6

The Index operator places the integerized value of the B register into the 20-bit length/index field of the descriptor in the A register. The descriptor is marked indexed (bit 45 is set to one), and the copy bit is set (bit 46 is set to one).

If the word in the A register is an operand, the A operand is exchanged with the B operand. If the word in the A register is neither a descriptor nor an IRW pointing to a descriptor, the invalid operand interrupt is set and the operation is terminated. If the indexing value is negative or greater than or equal to the length field of the descriptor, the invalid index interrupt is set and the operation is terminated.

If the descriptor represents an array which is segmented, the index is partitioned into two portions by an approximation algorithm which is determined by the type of data referenced by the descriptor, double-precision word 128, single-precision word 256, four-bit digit-3072, six-bit character-2048, or eight-bit byte-1536. The product of the approximator algorithm is used as an index to the given descriptor to fetch the array-row descriptor. The remainder is used to index the row descriptor.

If the double-precision bit (bit 40) in the descriptor is one, the index value in the B register is doubled. The balance of the operation is as described in the first paragraph of the description of this operator (INDX).

INDEX AND LOAD NAME (NXLN) A5

This operator performs an index operation; after the word in the A register has been indexed, the data descriptor pointed to by this word is brought into the A register. The copy bit (bit 46) of the data descriptor is set to one, and the A register is marked full. If the presence bit (bit 47) is off, the address of the original descriptor is placed in the address field of the stack copy. If the word accessed by the indexed word in the A register is not a data descriptor, the invalid operand interrupt is set and the operation is terminated.

If the data descriptor accessed by the indexed word in the A register has the index bit (bit 45) set to one, the invalid operand interrupt is set and the operation is terminated.

INDEX AND LOAD VALUE (NXLV) AD

This operator performs an index operation. After the word in the A register has been indexed, the operand pointed to by this descriptor is brought to the A register. The A register is marked full.

If the word accessed is other than an operand, the invalid operand interrupt is set and the operator is terminated.

LOAD (LOAD) BD

The Load operator places the word addressed by an IRW or indexed data descriptor in the A register.

If at the start of this operator the A register contains other than a data descriptor or an IRW, the invalid operand interrupt is set and the operation is terminated.

If the word pointed at by the data descriptor is another data descriptor, the latter is marked as a copy (copy bit [bit 46] is set to one), and if the presence bit (bit 47) is off, the address of the original is placed in bits 19:20 of the copy in the stack.

SCALE OPERATORS

Higher-level languages such as COBOL require decimal arithmetic. The Scale Operators provide the means of aligning decimal points prior to the time that the arithmetic operations are performed. In addition, the Scale Right operators provide for binary-to-decimal conversions.

SCALE LEFT (SCLF) CO

This operator uses the second syllable as the scale factor. The operand to be scaled is placed in the B register and integerized. The resulting integer is then multiplied by 10 raised to the power specified by the scale factor.

If scaling of a single-precision operand results in overflow, the single-precision operand is converted to a double-precision integer. A double-precision integer is defined as a double-precision operand with an exponent equal to 13.

If scaling of the operand results in an exponent greater than 13, (double-precision operand), the overflow flip-flop is set to one.

DYNAMIC SCALE LEFT (DSLF) C1

This operator performs virtually the same operation as the Scale Left (SCLF) operator; however, the scale factor is taken from the A register rather than from the program syllable following the operation syllable. The operand in the A register is integerized before scaling takes place.

SCALE RIGHT SAVE (SCRS) C4

This operator uses its second syllable as the scale factor. The operand to be scaled is placed in the B register and is then integerized. The resultant integer is divided by 10 raised to the power specified by the scale factor.

The quotient resulting from the division is left in the A register. The operand in the B register is the remainder which is converted to decimal (4-bit digits) and is left-justified. The A and B registers are both marked full.

If the scale factor is greater than 12, the invalid operand interrupt is set and the operation is terminated.

DYNAMIC SCALE RIGHT SAVE (DSRS) C5

This operator performs virtually the same operation as the Scale Right Save (SCRS) operator; however, the scale factor is obtained from the A register rather than from the program syllable following the operation syllable. The operand in the A register is integerized before being used.

SCALE RIGHT TRUNCATE (SCRT) C2

This operator performs a Scale Right function using its second syllable as the scale factor. The B register is marked as empty at the conclusion of this operator.

DYNAMIC SCALE RIGHT TRUNCATE (DSRT) C3

This operator performs the same operation as the Scale Right Truncate, except that the scale factor is found in the A register and is first integerized by the operator.

SCALE RIGHT FINAL (SCRF) C6

This operator performs a Scale Right operation, except that the quotient in the A register is deleted by marking the A register empty. The sign of the quotient is placed in the external sign flip-flop.

If the quotient was non-zero at the conclusion of the operation, the overflow flip-flop is set.

DYNAMIC SCALE RIGHT FINAL (DSRF) C7

This operator performs a Scale Right Final operation with the scale factor (integerized by the operator before use) found in the A register.

SCALE RIGHT ROUNDED (SCRR) C8

This operator performs a Scale Right operation, and the quotient is rounded by adding one to it if the most-significant digit of the remainder is equal to or greater than five. The remainder is deleted from the stack by marking the B register empty.

DYNAMIC SCALE RIGHT ROUND (DSRR) C9

This operator performs a Scale Right Rounded operation using the scale factor found in the A register.

BIT OPERATORS

The bit operators are concerned with a specified bit in the A register and/or B register.

BIT SET (BSET) 96

This operator sets a bit in the top of stack register. The bit that is set is specified by the program syllable following the operation syllable. If the program syllable defining the bit to be set has a value greater than 47, the invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT SET (DBST) 97

This operator performs a Bit Set Operation upon the bit specified by the operand in the top-of-stack register. This word is integerized before it is used as a bit number.

If the word in the top-of-stack register is not an operand, an invalid operand interrupt is set and the operation is terminated. If, after being integerized, the operand is less than zero or greater than 47, an invalid operand interrupt is set and the operation is terminated.

BIT RESET (BRST) 9E

This operator resets a bit in the top-of-stack register. The bit that is reset is specified by the syllable following the operation syllable. If the program syllable defining the bit to be reset has a value greater than 47, an invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT RESET (DBRS) 9F

This operator performs a Bit Reset operation upon the bit specified by the operand in the top-of-stack register.

If the word in the top-of-the-stack register is not an operand, an invalid operand interrupt is set and the operation is terminated. If, after being integerized, the operand is less than zero or greater than 47, an invalid operand interrupt is set and the operand is terminated.

CHANGE SIGN BIT (CHSN) 8E

The sign bit (bit 46) of the top-of-stack operand is complemented; that is, if it is a one, it is set to zero; if it is a zero the bit is set to one.

TRANSFER OPERATORS

The Transfer Operators transfer any field of bits from one word in the stack to any field of another word in the stack.

FIELD TRANSFER (FLTR) 98

This operator uses the following three syllables to establish the pointers used in the field transfer. This is done in the following manner. The second syllable of the operator is K, the third syllable of the operator is G, and the fourth syllable of the operator is L.

The field in the A register, starting at the bit position addressed by G, is transferred into the B register, starting at the bit position addressed by K. The length of the field in the A and B registers is defined by L. When the specified number of bits have been transferred, the A register is set to empty, the B register is marked full, and the operation is complete.

If the second or third syllables of the operator are found to be greater than 47, or the fourth syllable is greater than 48, the invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD TRANSFER (DFTR) 99

This operator performs a Field Transfer operation, except the B register operand is L. The B register is then reloaded from the stack and this operand is G. The B register is again loaded from the stack, and this operand is K.

If any of the three operands is a non-integer, it is first integerized. Each is checked for a value less than equal to zero or greater than equal to 48, or less than 48, as specified in Field Transfer. If either of these conditions exists in any one of the three operands, an invalid operand interrupt is set and the operation is terminated.

FIELD ISOLATE (ISOL) 9A

This operator isolates a field of the word in the A register, placing it right-justified in the top-of-stack register. The balance of the top-of-stack register is cleared to zeroes. The top-of-stack register is marked full.

This operator uses its second and third syllables as the BIT pointers. The second syllable of the operator addresses the starting bit of the field in the A register. The third syllable of the operator specifies the length of the field to be isolated.

If the value of the second syllable is greater than 47 or the value of the third syllable is greater than 48, as invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD ISOLATE (DISO) 9B

This operator performs a Field Isolate operation, except the first item in the stack specifies the length of the field to be isolated: The second operand in the stack addresses the bit in the word of the third item in the stack that is to be isolated.

If, after being integerized, the value of the first item in the stack is less than zero or greater than 47, an invalid operand interrupt is set and the operation is terminated. If, after being integerized, the value of the second item in the stack is less than zero or greater than 48, an invalid interrupt is set and the operation is terminated.

FIELD INSERT (INSR) 9C

This operator inserts a field from the A register into the B register word. The field in the A register is right-justified, with the length of the field specified by the third syllable of the operator. The second syllable of the operand addresses the starting bit of the field in the B register. At completion the A register is marked empty and the B register is marked full.

If the value of the second syllable of the operator is greater than 47, an invalid operand interrupt is set and the operation is terminated.

If the value of the third syllable of the operator is greater than 48, an invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD INSERT (DINS) 9D

This operator performs a Field Insert operation, except the first item in the stack is used as the insert field data. The second item in the stack is used to specify the length of the field. The third item in the stack is used to address the starting bit in the receiving field in the B register. When the operation is complete, the A register is marked empty and the B register is marked full.

If, after being integerized, the value of the second item in the stack is less than zero or greater than 48, an invalid operand interrupt is set and the operation is terminated. If, after being integerized, the value of the third item in the stack is less than zero or greater than 47, an invalid operand interrupt is set and the operation is terminated.

STRING TRANSFER OPERATORS

String Transfer operators give the system the ability to transfer characters or words from one location in memory to another location in memory. The source and destination pointers are set from string descriptors in the stack.

TRANSFER WORDS, DESTRUCTIVE (TWSD) D3

This operator requires three items in the top-of-stack: an operand, a string descriptor or operand, and a string descriptor. The first operand is integerized and used as the count or repeat field. The second item is either the source data or a descriptor which points at the source string, and the third item is used to address the destination string. The number of words specified by the repeat field is transferred from the source to the destination. At completion of the operation, the A and the B registers are marked empty.

If the memory protect bit is found on during the execution of the Transfer Words operator, the segmented array interrupt is set and the operation is terminated.

TRANSFER WORDS, UPDATE (TWSU) DB

This operator performs the Transfer Words operator, except that at the completion of the transfer of data, the source and destination pointers are updated to point to the location in memory where the transfer ended. The A and B registers are both marked full.

TRANSFER WORDS, OVERWRITE DESTRUCTIVE (TWOD) D4

This operator performs a Transfer Words, Destructive operation, except that it overrides the memory protection checks.

TRANSFER WORDS, OVERWRITE UPDATE (TWOU) DC

This operator performs a Transfer Words, Update operation, except that it overrides the memory protection checks.

TRANSFER WHILE GREATER, DESTRUCTIVE (TGTD) E2

This operator transfers characters from a location in memory pointed to by the source pointer, to a location in memory pointed to by the destination pointer, until the number of characters specified has been transferred or the comparison fails. The TFFF flip-flop is used to indicate the results of the comparison. TFFF is set at the beginning of the operator.

The first item in the stack is used as the delimiter. The second item in the stack, bits 19:20, is the maximum number of characters to be transferred. The third item in the stack is the source data or a source pointer, and the fourth item in the stack is the destination pointer.

The source and destination strings are checked for memory protection. The source character is compared to the delimiter. After each comparison, a decision is made whether the condition has been met. If the condition is met, TFFF remains set to one; if it is not met, it is set to zero. If the result of the comparison is equal, then the CMPF flip-flop is set; otherwise, CMPF is reset.

If the number of characters transferred was equal to the repeat field, the TFFF flip-flop is set to one. The A and B registers are marked empty and the operation is complete.

If the first operand in the stack is not a single-precision operand, an invalid operand interrupt is set and the operation is terminated.

If either the source or destination word has a memory protect bit on (bit 48=1), the segmented array interrupt is set and the operation is terminated.

If the second item in the stack is a descriptor, it is used as the source pointer, and the length field or repeat field is set to 1,048,575. All comparisons are binary (EBCDIC collating sequence).

TRANSFER WHILE GREATER UPDATE (TGTU) EA

This operator performs a Transfer While Greater operation and updates the source pointer and destination pointer to point at the next characters in the source and destination strings. The repeat count is updated to give the number of characters not transferred. If the operation is terminated because the relationship is not met, the source pointer points at the character that failed the comparison. If the result of the comparison is equal, then the CMPF flip-flop is set; otherwise, CMPF is reset.

TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE (TGED) E1

This operator performs a Transfer While operation using the relation greater than or equal to for comparison.

TRANSFER WHILE GREATER OR EQUAL, UPDATE (TGEU) E9

This operator performs a Transfer While Greater or Equal operation. The source pointer, destination pointers, and count are updated at the conclusion of the operation.

TRANSFER WHILE EQUAL, DESTRUCTIVE (TEQD) E4

This operator performs a Transfer While operation with the relation used in the comparison being equal. If the result of the comparison is greater, then the CMPF flip-flop is set; otherwise, CMPF is reset.

TRANSFER WHILE EQUAL, UPDATE (TEQU) EC

This operator performs a Transfer While Equal operation. The source pointer, the destination pointer, and count are updated at the conclusion of the operation. CMPF is set if the result of the comparison is greater; otherwise, CMPF is reset.

TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE (TLED) E3

This operator performs a Transfer While operation, using the less than or equal comparison.

TRANSFER WHILE LESS OR EQUAL, UPDATE (TLEU) EB

This operator performs a Transfer While Less or Equal operation. The source pointer, destination pointer, and count are updated at the conclusion of the operation.

TRANSFER WHILE LESS, DESTRUCTIVE (TLSD) EO

This operator performs a Transfer While operation using the less than comparison. If the result of the comparison is equal, then the CMPF flip-flop is set; otherwise, CMPF is reset.

TRANSFER WHILE LESS, UPDATE (TLSU) E8

This operator performs a Transfer While Less operation. The source pointer, destination pointer, and count are updated at the conclusion of the operation.

TRANSFER WHILE NOT EQUAL, DESTRUCTIVE (TNED) E5

This operator performs a Transfer While operation, using the not equal comparison. CMPF is not used.

TRANSFER WHILE NOT EQUAL, UPDATE (TNEU) ED

This operator performs a Transfer While Not Equal operation. The source pointer, destination pointer, and count are updated at the conclusion of the operation.

TRANSFER UNCONDITIONAL, DESTRUCTIVE (TUND) E6

This operator performs a Transfer Characters until the length is equal to zero. No comparisons are made.

TRANSFER UNCONDITIONAL, UPDATE (TUNU) EE

This operator performs a Transfer Unconditional operation. The source pointer and the destination pointer are updated at the conclusion of the operation.

STRING ISOLATE (SISO) D5

This operator places in the top-of-the-stack, right justified, the number of source characters specified by the repeat field. The first item in the stack is the number of characters in the repeat field. The second item in the stack is either an operand or a descriptor used as the source pointer.

If the number of bits to be transferred is greater than 48, the item is double-precision.

If the number of bits is greater than 96, an invalid operand interrupt is set and the operation is terminated.

If the source data has the memory protect bit (bit 48) set to one, the segmented array interrupt is set and the operation is terminated.

COMPARE OPERATORS

The compare operators perform the specified comparison of two strings of data. The True False Flip-Flop (TFFF) and the Compare Flip-Flop (CMPF) are used to indicate the result of the comparison at the conclusion of the operation. Table 7-2 shows the significance of the state of TFFF and CMPF at the conclusion of a compare type operator.

COMPARE CHARACTERS GREATER, DESTRUCTIVE (CGTD) F2

This operator compares the value of two character strings, one character at a time. The operator compares characters until it encounters a pair which are unequal. If the B string character is greater than the A string character, the TFFF is set; otherwise, it is reset. If the length is depleted and the character strings are equal, the CMPF flip-flop is set. If the characters in the B string are greater than the characters in the A string, the TFFF is set to one. If not, the TFFF is set to zero.

The first item in the stack is an operand which contains the length of the fields being compared. The second item in the stack is an operand or a descriptor pointing at the character string to be compared against. The third item in the stack is a descriptor pointing at the character string to be compared.

If the repeat count is depleted, the TFFF is reset.

If either of the data strings has the memory protect bit on (bit 48=1), the segmented array interrupt is set and the operation is terminated.

All comparisons are by the binary character position in the collating sequence.

COMPARE CHARACTERS GREATER, UPDATE (CGTU) FA

This operator performs a Compare Characters Greater operation. The source pointer and destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE (CGED) F1

This operator performs the Compare Characters operation with the comparison being greater than or equal. If the repeat count ≤ 0, the TFFF is set to one.

Table 7-2. Compare Type Operator Results

Compare	TFFF	CMPF	Comparison Result
=	0	0	Less than equal
	0	1	Greater than equal
	1	0	Equal
	1	1	Not applicable
≠	0	0	Equal
	0	1	Not applicable
	1	0	Less than equal
	1	1	Greater than equal
>	0	0	Less than equal
	0	1	Equal
	1	0	Greater than equal
	1	1	Not applicable
<	0	0	Greater than equal
	0	1	Equal
	1	0	Less than equal
	1	1	Not applicable
⩾	0	0	Less than equal
	0	1	Not applicable
	1	0	Greater than equal
	1	1	Equal
⩽	0	0	Greater than equal
	0	1	Not applicable
	1	0	Less than equal
	1	1	Equal

COMPARE CHARACTERS GREATER OR EQUAL, UPDATE (CGEU) F9

This operator performs a Compare Character Greater or Equal operation. The source pointer and destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS EQUAL, DESTRUCTIVE (CEQD) F4

This operator performs the Compare Characters operation using the equal comparison. If the repeat count ⩽0, then TFFF is set to one.

COMPARE CHARACTERS EQUAL, UPDATE (CEQU) FC

This operator performs a Compare Characters Equal operation. The source pointer and destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE (CLED) F3

This operator performs the Compare Characters operation with the less than or equal comparison. If the repeat count $\leqslant 0$, then TFFF is set to one.

COMPARE CHARACTERS LESS OR EQUAL, UPDATE (CLEU) FB

This operator performs a Compare Characters Less or Equal operation. The source pointer and destination pointers are updated at the conclusion of the operation.

COMPARE CHARACTERS LESS, DESTRUCTIVE (CLSD) FO

This operator performs the Compare Characters operation using the less than comparison. If the repeat count $\leqslant 0$, the TFFF is set to zero.

COMPARE CHARACTERS LESS, UPDATE (CLSU) F8

This operator performs a Compare Characters Less operation. The source pointer and the destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE (CNED) F5

This operator performs the Compare Characters operation using the not equal relation. If the repeat count $\leqslant 0$, then TFFF is set to 0.

COMPARE CHARACTERS NOT EQUAL, UPDATE (CNEU) FD

This operator performs a Compare Characters Not Equal operation. The source pointer and the destination pointer are updated at the conclusion of the operation.

EDIT OPERATORS

The Edit Mode Operators are discussed in the following paragraphs.

TABLE ENTER EDIT, DESTRUCTIVE (TEED) DO

This operator is used to prepare for edit micro-instructions. These edit micro-instructions are contained in memory as a table and not as part of the normal program string. When this operator is entered, program execution is transferred to a table of micro-instructions. The last micro-instruction in this table must be the End Edit operator (see section 9). The table contains Edit Mode operators.

The first item in the stack is a descriptor pointing to the table of exit micro-instructions. The second item in the stack is a single-precision operand or a descriptor pointing at the source string. The third item in the stack is descriptor pointing at the destination.

If the first item in the stack is not a descriptor, the invalid operand interrupt is set and the operation is terminated. If the second item in the stack is a single-precision operand, it is the source string. If the third item in the stack is not a descriptor, the invalid operand interrupt is set and the operation is terminated.

TABLE ENTER EDIT, UPDATE (TEEU) D8

This operator performs a Table Enter Edit operation and updates the source pointer and destination pointer at the completion of the operation.

EXECUTE SINGLE MICRO, DESTRUCTIVE (EXSD) D2

This operator performs the same function as the Table Enter Edit operator, except (a) there is only one micro-operator and it follows this syllable, and (b) the first item in the stack is a single-precision operand that defines the length field.

An end edit operation is performed as an implicit part of the EXSD operator, thus, an explicit END EDIT operator (in program line code) is not required.

EXECUTE SINGLE MICRO, UPDATE (EXSU) DA

This operator performs the same functions as an Execute Single Micro-operator, except that it updates the source pointer and destination pointer at the completion of the edit operator operation.

EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE (EXPU) DD

This operator performs the same functions as an Execute Single Micro-Update operator, except that one pointer is used as both source and destination pointer. The destination pointer is updated at the completion of the operation.

PACK OPERATORS

The two pack operators are discussed in the following paragraphs.

PACK, DESTRUCTIVE (PACD) D1

This operator packs data addressed by the source pointer into the top-of-stack in 4-bit (digit) format. The TFFF is set to one if the source data is negative. A negative number for an 8-bit (byte) format has a zone bit configuration of 1101 in the least significant byte. Data is right-justified as it is placed in the top-of-stack.

The operand in the top-of-stack (TOS) is used as the length field. The second item is the source pointer. The operation then continues until the number of digits specified by the length or repeat field have been packed.

If the length is less than 13, the operand in the top-of-stack is a single-precision operand. If the operand is 13 or greater, the result is a double-precision operand. If the length is not less than 25, an invalid operand interrupt is set and the operation terminated. If initial length is zero, the TOS is filled with zeroes.

If the second item in the stack is an operand, it is the source string and is comprised of 8-bit bytes.

If the source data has the memory protect bit (bit 48) set to one, the segmented array interrupt is set and the operation is terminated.

PACK, UPDATE (PACU) D9

This operator performs a Pack operation, updating the source pointer at the completion of the operation.

INPUT CONVERT OPERATORS

The five input convert operators are discussed in the following paragraphs.

INPUT CONVERT, DESTRUCTIVE (ICVD) CA

This operator converts either 8-bit EBCDIC, or 4-bit digit code to an operand for internal arithmetic operations. The first item in the stack is an operand integerized to form the repeat field. The second item in the stack is a descriptor used as a source pointer.

The input convert operator converts a string of input EBCDIC character data into a numeric operand. The resultant operand may be either single-precision or double-precision. The manner in which the conversion of character data into numeric data is performed is discussed in the following paragraphs.

The four high-order zone bits of the input EBCDIC character are discarded. The remaining four low-order digit bits from the input character form a hexadecimal character, which is placed in the top-of-stack register receiving field.

Each time a source input character is converted, the repeat field is decremented by one. When the repeat field is equal to zero, all input characters have been converted.

If the repeat field value is 13 (decimal) or less, the resultant operand in the TOS register is a single-precision operand. If the repeat field value is between 13 and 24 (decimal), the resultant operand in the TOS register is a double-precision operand. If the repeat field is greater than 24, an invalid operator interrupt is set and the operation is terminated.

The sign of the converted resultant operand is determined from the zone bits of the least significant character in the input character string. For EBCDIC input characters, the sign is positive except when the least significant character zone bits are equal to 1101 binary; then, it is negative. The detected sign bit for the resultant operand is saved in the TFFF flip-flop.

The sign of the converted operand is then set from the TFFF. If the converted operand is a single-precision operand, the TFFF is then set to one. If the converted operand is a double-precision operand, the TFFF is set to zero.

At the completion of the operation, the B register is marked full. The TAG field is set to indicate either a single- or a double-precision operand.

If, after being integerized, the item in the top-of-stack is greater than 23, the invalid operand interrupt is set and the operation is terminated.

INPUT CONVERT, UPDATE (ICVU) CB

This operator performs an Input Convert operation. The source pointer is updated at the completion of the operation.

READ TRUE FALSE FLIP-FLOP (RTFF) DE

This operator places the status of the TFFF into the low-order bit position of the A register. The rest of the A register is set to all zeroes. The A register is marked full at completion of this operation.

SET EXTERNAL SIGN (SXSN) D6

This operator places the mantissa sign of the top word of the stack in the external sign flip-flop. This operand is not deleted from the stack at the end of the operation.

READ AND CLEAR OVERFLOW FLIP-FLOP (ROFF) D7

This operation places the status of the overflow flip-flop in the least-significant bit of the A register, sets the rest of the A register to all zeroes, marks the register full, and sets the overflow flip-flop to zero.

SUBROUTINE OPERATORS

The subroutine operators are discussed in the following paragraphs.

VALUE CALL (VALC) 00 ⇒ 3F

This operator loads the operand addressed by the address couple formed by the concatenation of the six low-order bits of the first syllable and the 8-bits of the following syllable into the A register. The A register is marked full. Figures 7-1 and 7-2 are simplified flow charts of the Value Call operator.

This operator makes multiple memory accesses if the word accessed is either an indexed descriptor, Program Control Word (PCW), or an Indirect Reference Word (IRW).

If the word accessed is an indexed data descriptor, the word addressed by the data descriptor is brought to the top-of-stack. If the double-precision bit (bit 50) in the data descriptor is equal to one, the other half of the double-precision operand is brought to the X register.

If the word accessed is a non-indexed word data descriptor, the word is indexed using the second word in the stack for the index value. The word addressed by the non-indexed data descriptor is brought to the top-of-stack. If the double-precision bit (40) in the data descriptor is equal to one, the other half of the double-processor operand is brought to the X register.

If the word accessed by the data descriptor is another indexed data descriptor, the word addressed by the data descriptor is brought to the top-of-stack, and one of the two preceding paragraphs is repeated.

If a data descriptor does not address an operand, SIRW, word descriptor, or indexed string descriptor, an invalid operand interrupt is set and the operation is terminated.

If the word accessed by the value call is an IRW, the word addressed by the IRW is accessed and evaluated. If the word is an operand, it is placed in the top-of-stack.

If the word accessed by the IRW is another IRW, the operation continues as previously described.

If the word accessed by the IRW is an indexed or non-indexed data descriptor, the operator proceeds as previously described for data descriptors.

If the word accessed by the IRW is a PCW, an accidental entry into the subroutine addressed by the PCW is initiated. A Mark Stack Control Word (MSCW) and a Return Control Word (RCW) are placed in the stack, and an entry is made into the program. Upon completion of the program, a return operator re-enters the flow value call at the label IRW (Figure 7-1).

NAME CALL (NAMC) 40 ⇒ 7F

This operator builds an IRW in the A register. The address couple is formed by concatenating the six low-order bits of the first syllable and the 8-bits of the following syllable. The A register is marked full and the operation is complete.

EXIT OPERATOR (EXIT) A3

This operator returns to a calling procedure from a called procedure resetting all control registers from the RCW and the MSCW. The Exit operator does not return a value to the calling routine. Figure 7-3 shows a simplified flow chart of the Exit operator.

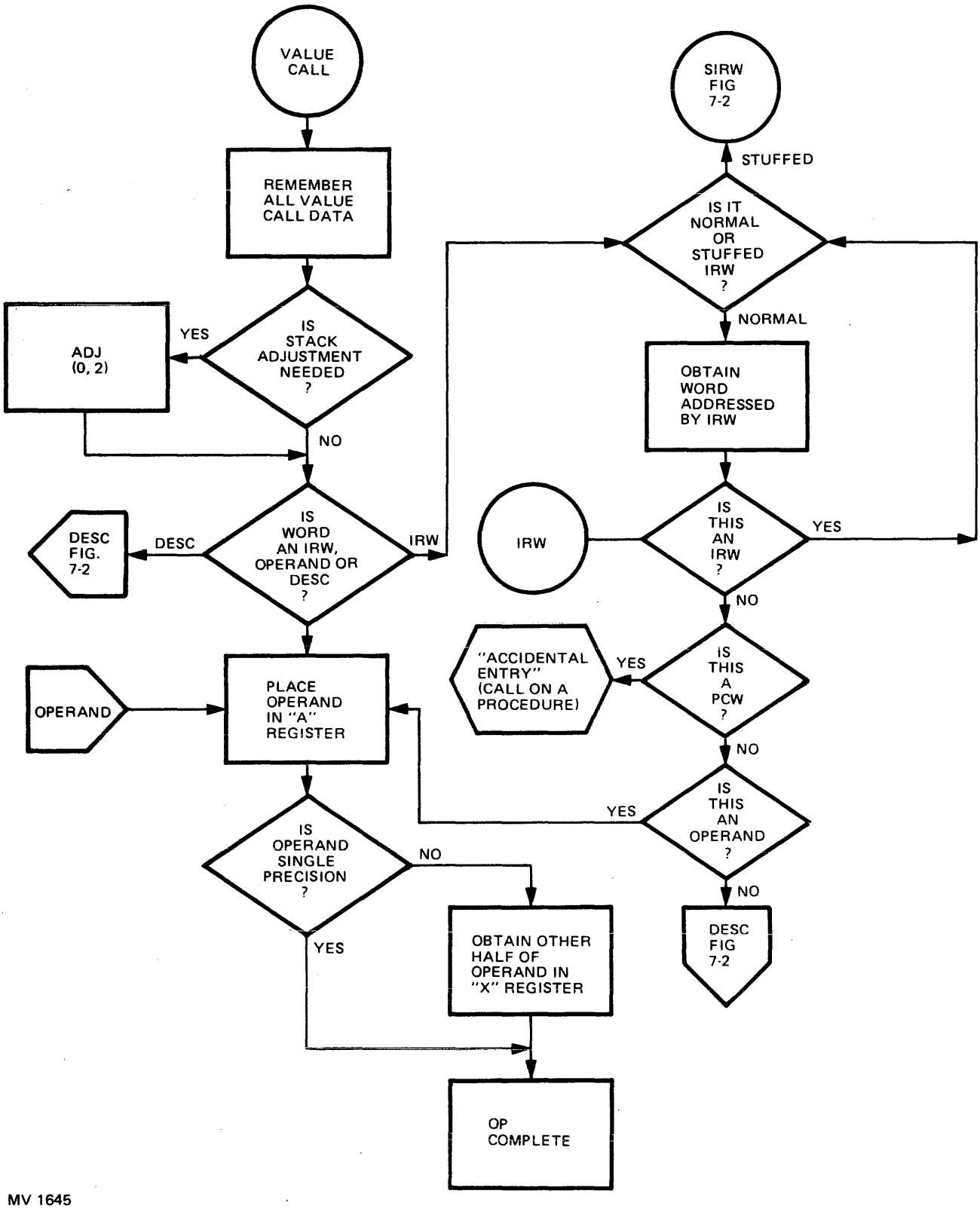


Figure 7-1. Flow of Value Call Operator

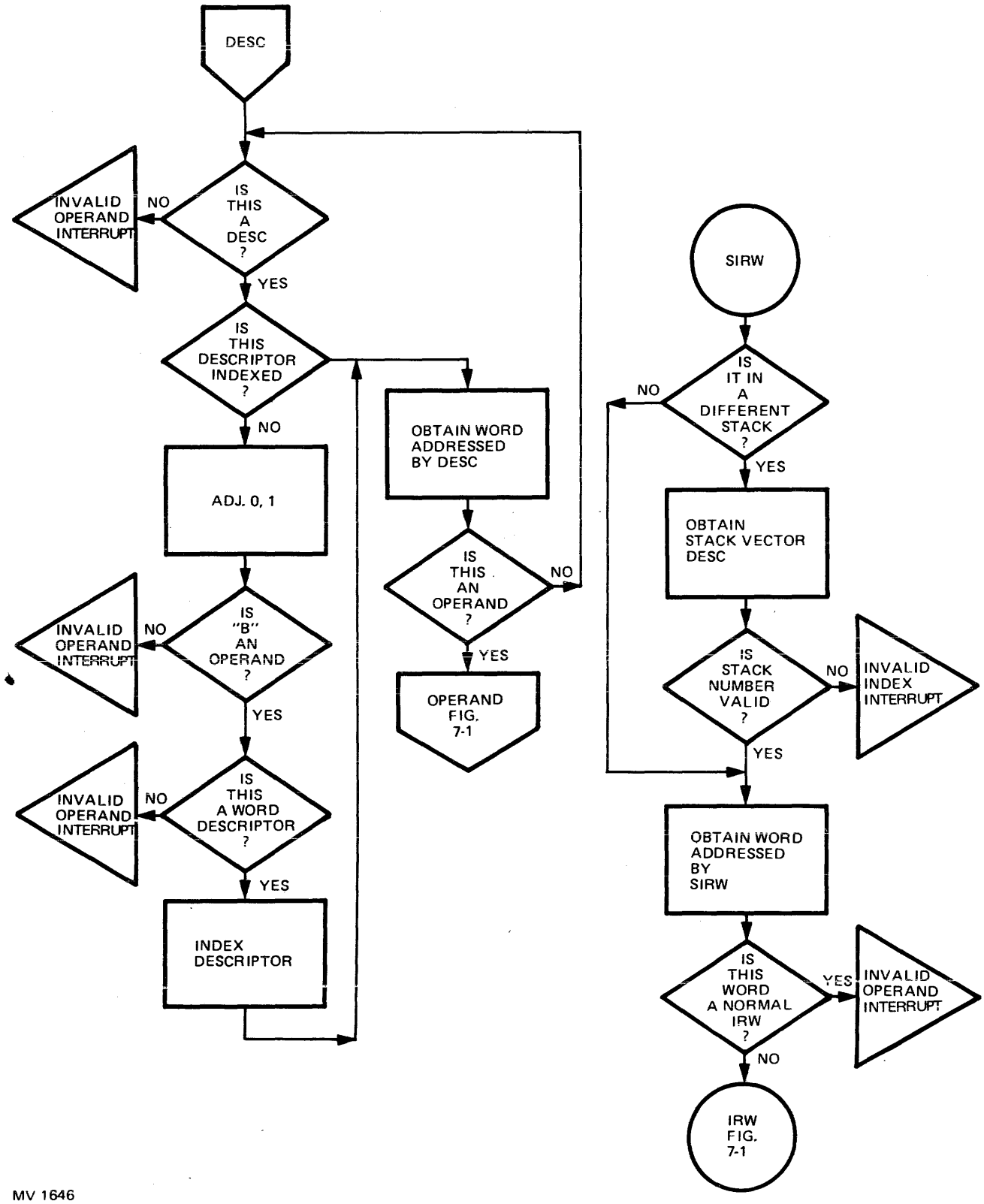


Figure 7-2. Value Call (Descriptor) Operator

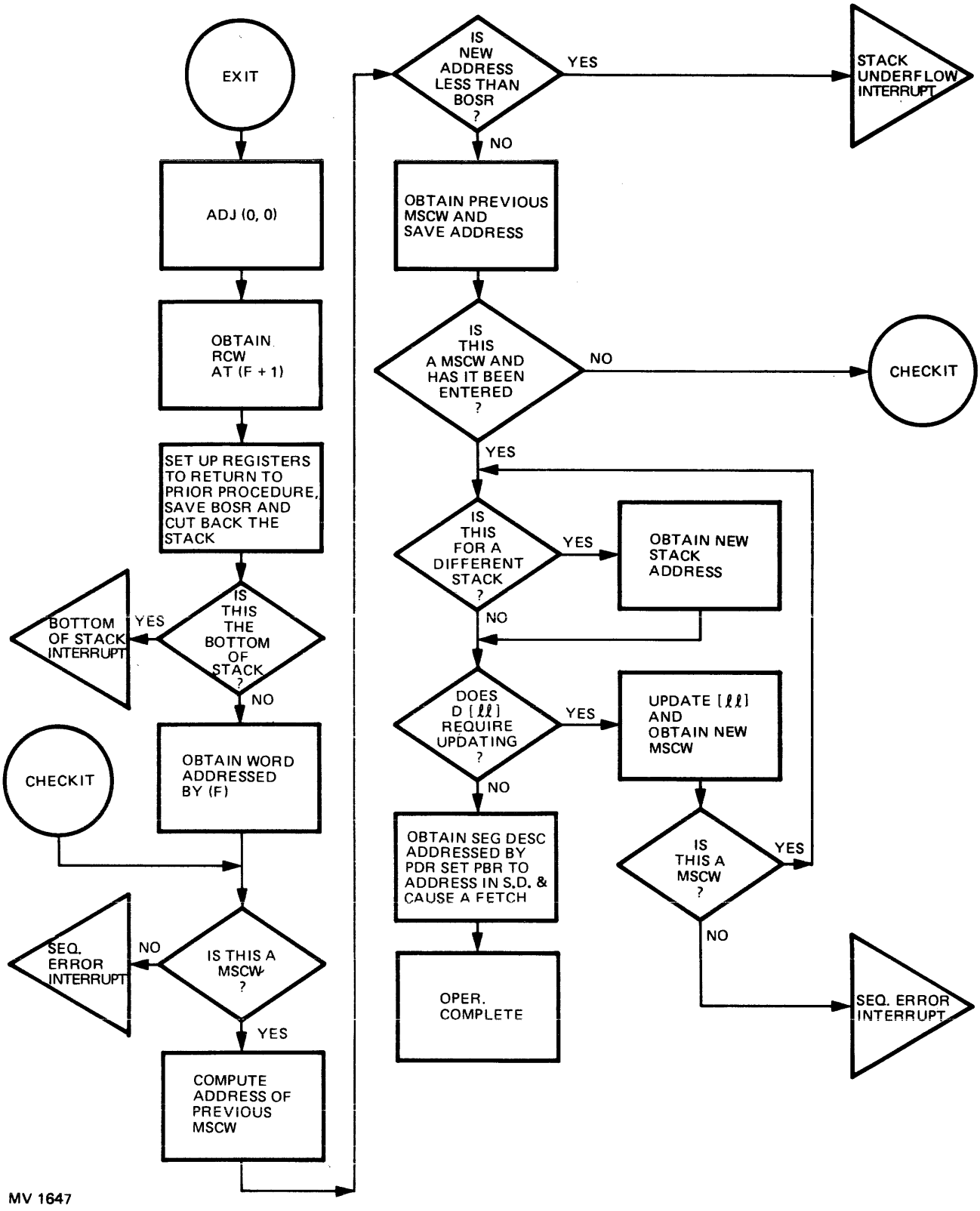


Figure 7-3. Flow of Exit Operator

RETURN OPERATOR (RETN) A7

This operator performs the same functions as an Exit operator, except an operand or name in the B register is returned to the calling procedure. If a name is returned and the V bit (bit 19) in the MSCW is on, the name is evaluated to yield an operand as described in the VALC operator. Figure 7-4 shows a simplified flow chart of the Return operator.

ENTER OPERATOR (ENTR) AB

This operator is used to cause an entry into a procedure from a calling procedure. Entry is to the program segment and syllable addressed by the PCW. Figure 7-5 shows a simplified flow chart of the Enter operator.

The Enter operator accesses the IRW at F + 1, which points to the PCW (or to the PCW directly, without the use of an IRW). The operator then builds a RCW into the stack at F + 1.

EVALUATE (EVAL) AC

This operator loads the A register with an indexed data descriptor or an IRW that addresses A "target," which may be an SIW, an un-indexed data descriptor, a string descriptor, or an operand. The target can be referenced through a chain of accidental entries or IRW. In any case, memory accesses continue to be made until the target is located. The A register is left containing the data descriptor or the IRW which addresses the target. Figure 7-6 is a simplified flow chart of the Evaluate operator.

An indexed data descriptor is left in the A register when the target is referenced by an indexed data descriptor. A stuffed IRW is left in the A register when the target is referenced by IRW(s).

If the A register does not contain a data descriptor or an IRW at the start of this operator, an invalid operand interrupt is set and the operation is terminated.

MARK STACK OPERATOR (MKST) AE

This operator places a Mark Stack Control Word in the B register which contains a pointer to the previous MSCW in the stack. The F register is updated to point at the address of the MSCW.

This operator is used to mark the stack when entry into a procedure is anticipated.

STUFF ENVIRONMENT (STFF) AF

This operator changes a normal IRW to a stuffed IRW so that a quantity may be referenced from a different addressing environment. The displacement field locates the MSCW below the quantity, and the index field locates the quantity relative to the MSCW. Figure 7-7 shows a simplified flow chart of the Stuff Environment operator.

If the word in the A register at the start of the operation is not an IRW, an invalid operand interrupt is set and the operation is terminated.

If, when creating this stuffed IRW, other than an MSCW is accessed, a sequence error interrupt is set and the operation is terminated.

INSERT MARK STACK OPERATOR (IMKS) CF

This operator builds an MSCW and places it below the two top-of-stack quantities.

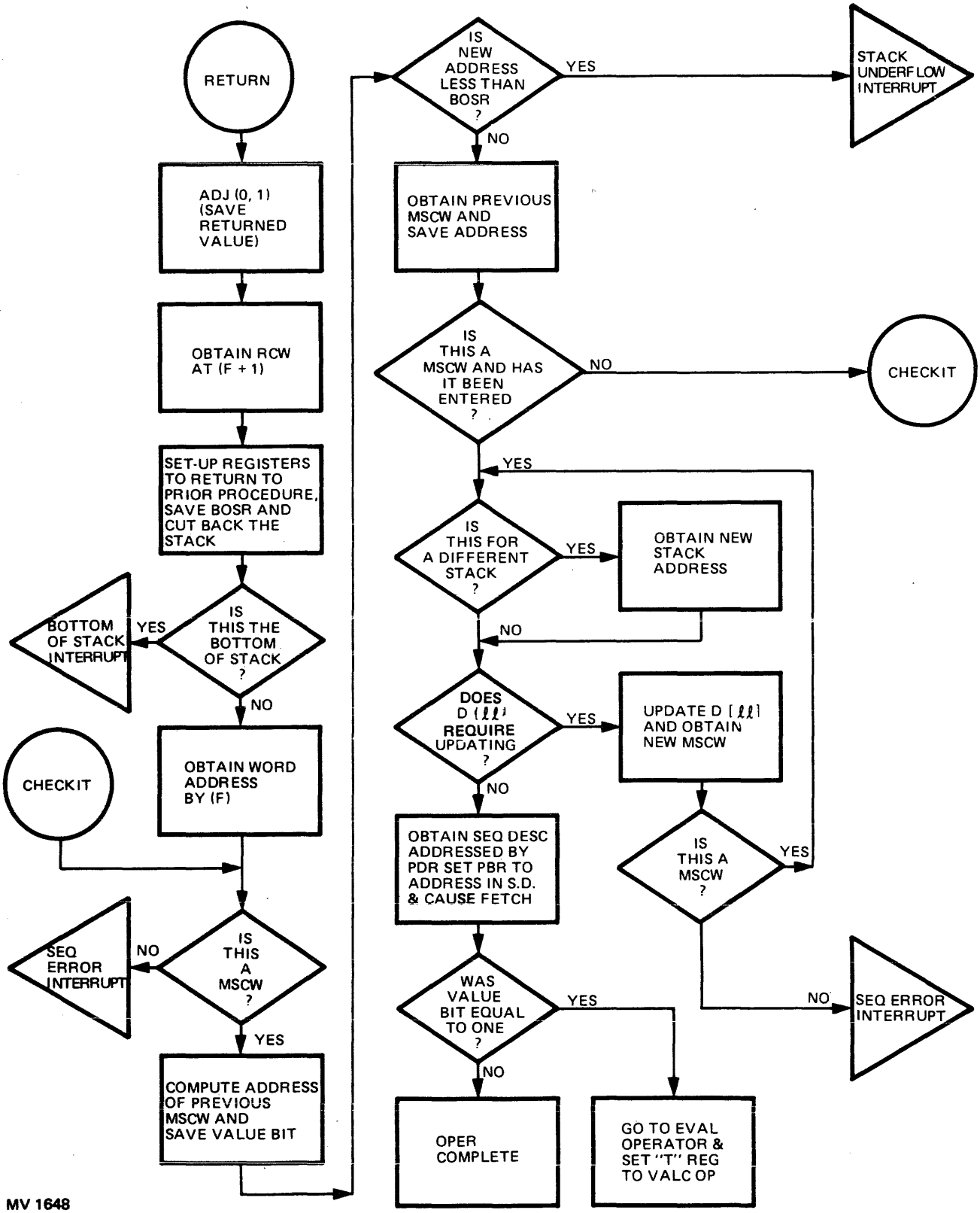


Figure 7-4. Flow of Return Operator

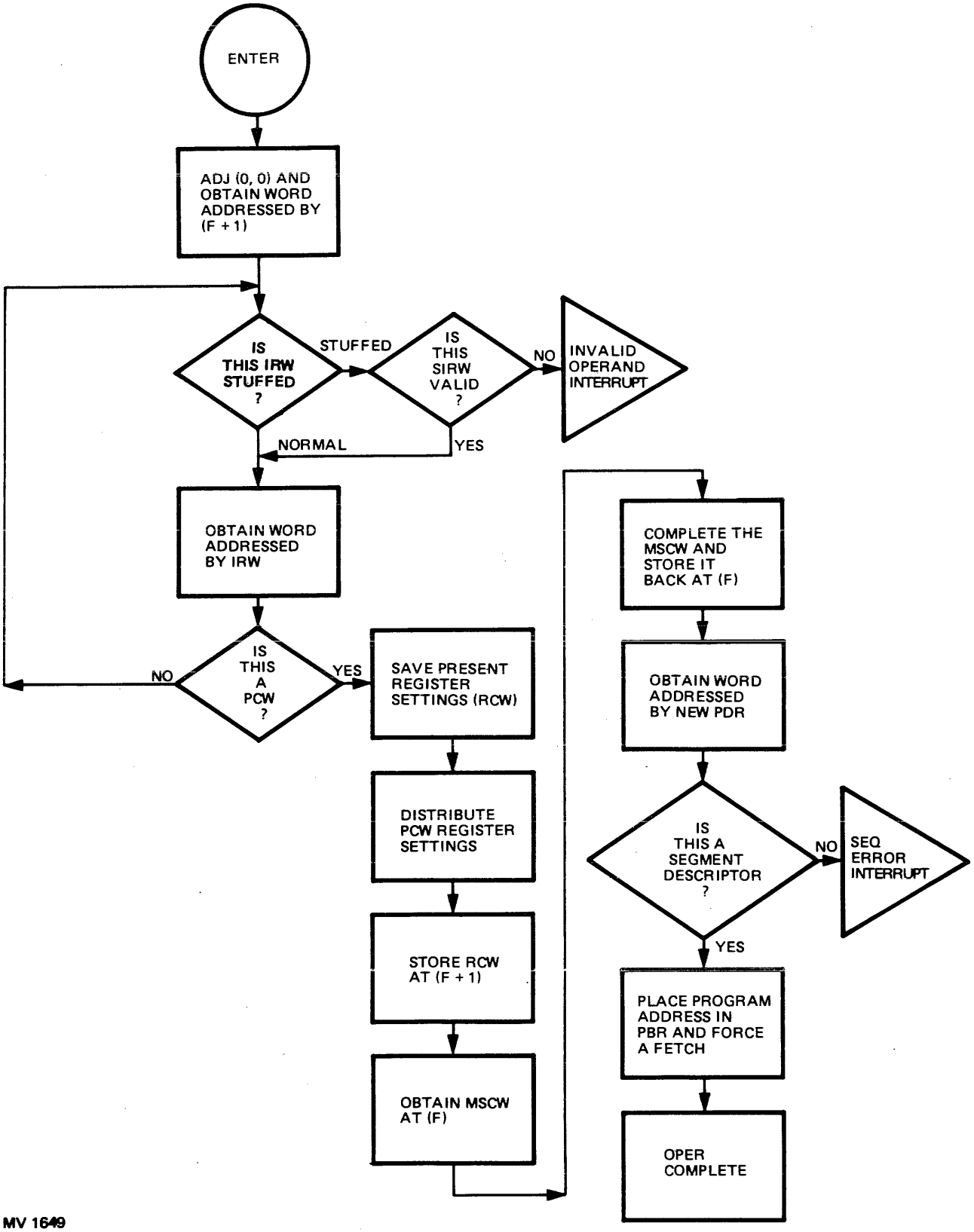


Figure 7-5. Flow of Enter Operator

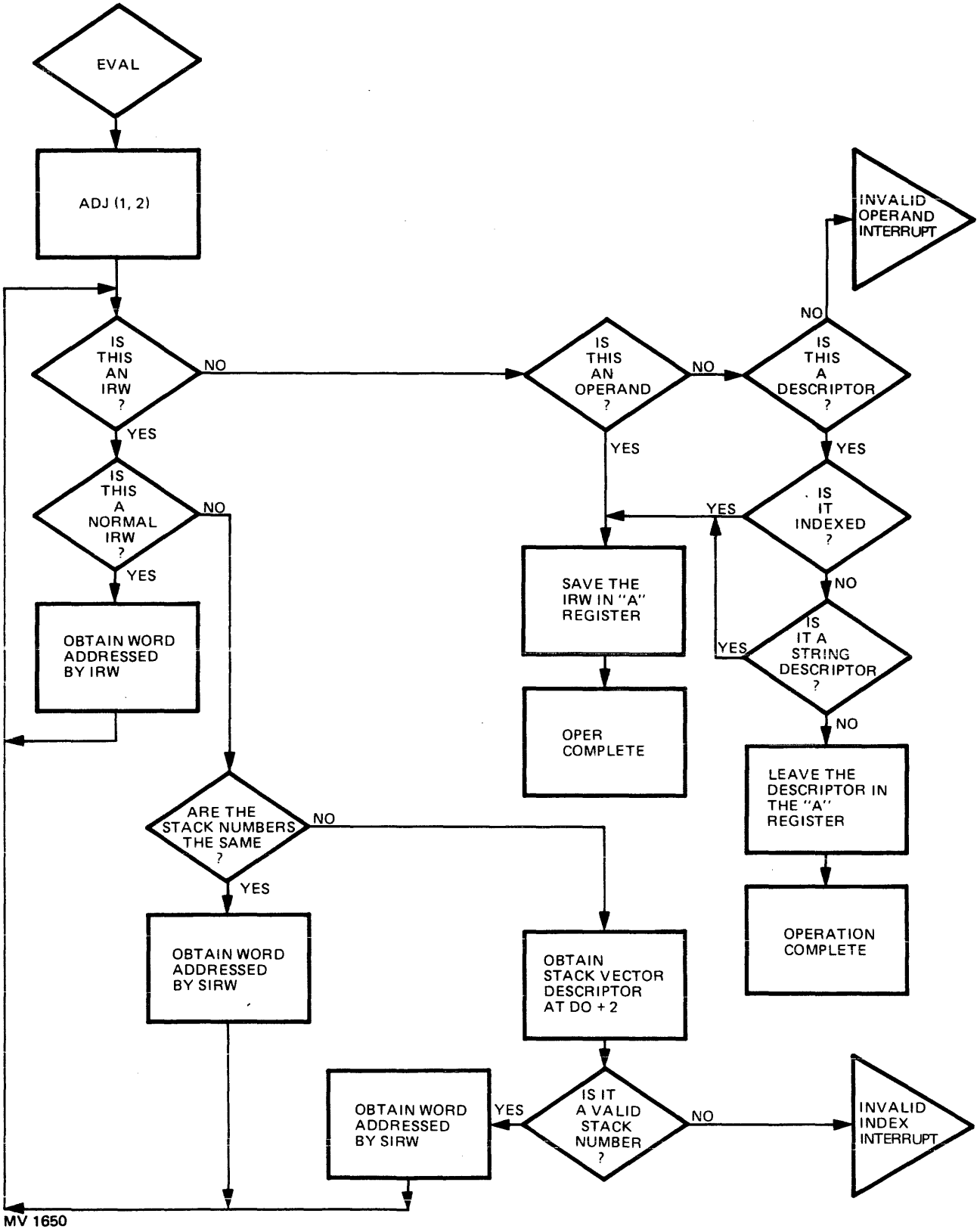


Figure 7-6. Flow of Evaluate Operator

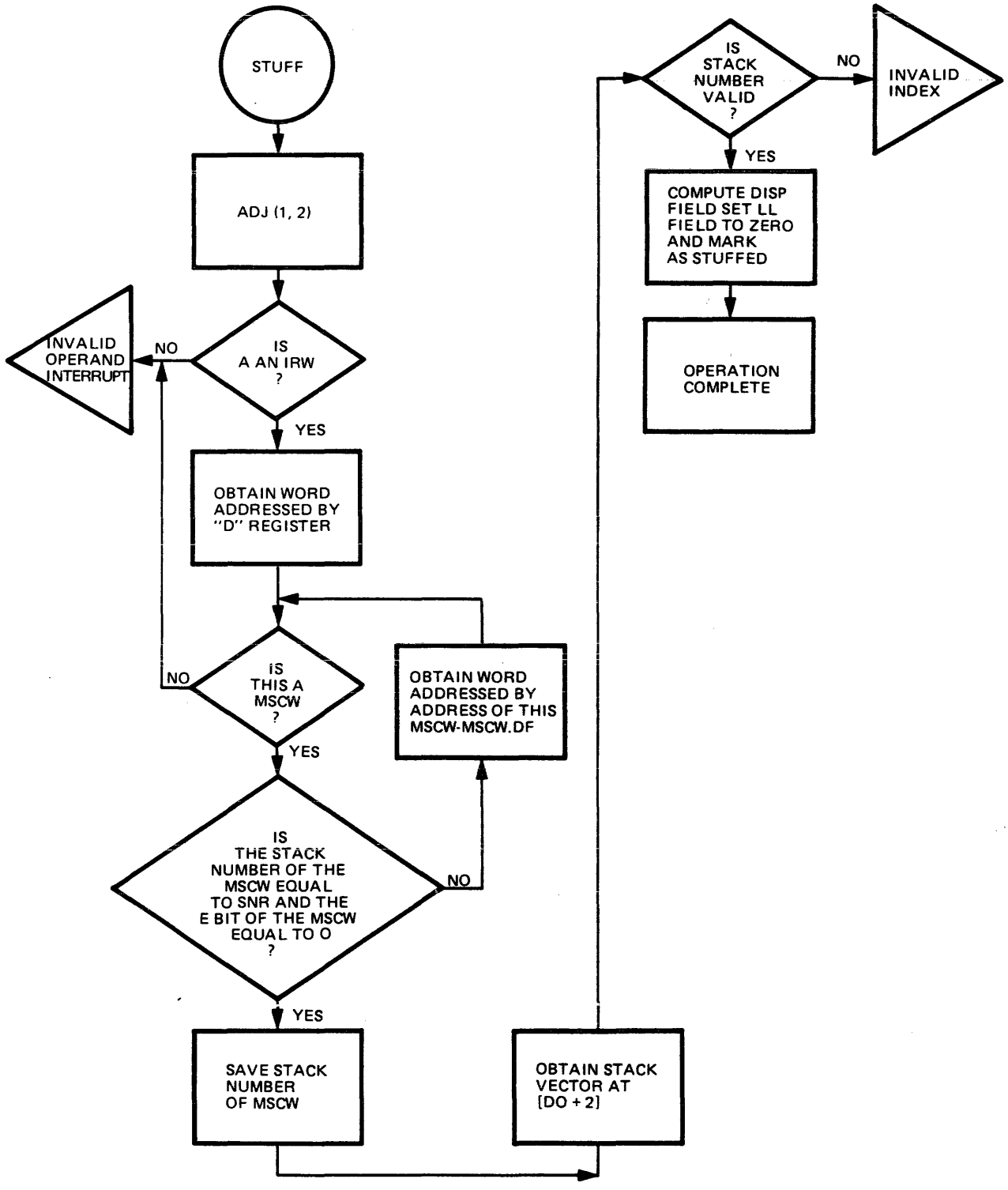


Figure 7-7. Flow of Stuff Environment Operator

ENTER VECTOR MODE OPERATORS

Two different operators are used to cause the B 6900 system to enter into the vector mode of operation. The Vector Mode Enter Single (VMOS) operator is used to enter the vector mode of operation when a single word of program code contains all the vector mode operators to be executed. The Vector Mode Enter Multiple (VMOM) operator is used to enter into the vector mode of operation when the number of vector mode operators to be executed uses more than a single word of program code.

The two methods for entering the vector mode of operation are described in the following paragraphs.

VECTOR MODE ENTER MULTIPLE (VMOM) E7

This operator is used to cause entry into the vector mode of operation in the same way that the VMOS operator performs. The only difference between the operation of the VMOS and the VMOM operators is the number of words of vector mode machine language code that can be used.

If an interrupt occurs while entry into vector mode is in process, the entry process is terminated, and processing resumes with the next normal mode machine language operator in sequence. Since multiple words of vector mode machine language operators are used when the VMOM operator causes entry to vector mode, the first word of normal mode operators may be greatly removed from the VMOM operator code word.

The use of the VMOM operator causes the data processor to retain the address of the next normal mode operator word. This address is required in the event that the entry into vector mode is terminated. The retention of the next normal mode operator word address (in IC memory) is the only difference between the VMOS and VMOM operators.

VECTOR MODE ENTER SINGLE (VMOS) EF

This operator is used to cause entry into the vector mode of operation. Vector mode operations are performed in control state (HHF flip-flop is set). The VMOS operator uses a subset of the table enter edit logic to distribute vector mode parameters in the IC memory address registers of the data processor. The vector mode operator parameters must be on the top of the data processor stack at the beginning of the VMOS operator.

The VMOS operator expects to find three data descriptors and three incrementation parameters present on the top of the data processor stack. The VMOS operator optionally expects that a LENGTH parameter may be present on the top of the data processor stack. If the VMOS operator does not find the three data descriptors on the top of the data processor stack, an invalid operand interrupt is detected, and the VMOS operator releases control to the interrupt controller.

The VMOS operator expects to find that bit 47 (the presence bit) is true in each of the three data descriptors. If any of the three data descriptors do not have the presence bit true, a presence bit interrupt is detected, and the VMOS operator releases control to the interrupt controller.

The order of occurrence of the three data descriptors and the three increment parameters (and optionally, the LENGTH parameter) is as follows:

Parameter	Word Type	Word Usage
Pointer C	Data descriptor	The top word in the data processor stack.
LENGTH	SP operand	When a LENGTH parameter is present, it is the second word in the data processor stack, and its presence is indicated by bit 44 of pointer C being set. If a LENGTH parameter is not present in the stack, a default length value of FFFFF – 1 (HEX) is used.
Pointer A	Data descriptor	If a LENGTH parameter is not present in the data processor stack, pointer A is the second word in the data processor stack. If a LENGTH parameter is present in the stack, then pointer A is the third word in the stack.
Pointer B	Data descriptor	If a LENGTH parameter is not present in the stack, pointer B is the third word in the stack. If a LENGTH parameter is present in the stack, then pointer B is the fourth word in the stack.
Increment C	SP operand	The incrementation value that will be used as the incrementation unit to access data elements of the array pointed at by pointer C.
Increment A	SP operand	The incrementation value used for accessing data elements in the array pointed at by pointer A.
Increment B	SP operand	The incrementation value used for accessing data elements in the array pointed at by pointer B.

If bit 44 (the segmented bit) is true in pointer A or B, an invalid operator interrupt is detected, and the VMOS operator releases control to the interrupt controller.

If pointer A has the read only bit (bit 43) true, a memory protect interrupt is detected, and the VMOS operator releases control to the interrupt controller.

If any of the three types of interrupts described in the preceding paragraphs are detected, the entry into vector mode is terminated, and the program is resumed (in normal state) at the next code word following the vector operator code word. The use of the VMOS operator implies that only one word of vector mode operators is to be used, and the first vector mode operator to be executed is present in syllable zero of the next program code word in sequence. Therefore, the next word of program code (the vector mode code word) is fetched by the program controller and placed in the P register. If an interrupt occurs during the VMOS operator, the interrupt controller fetches another new word of program code (the word following the vector mode code word). Thus, the VMOS operator releases control to the interrupt controller, and the interrupt controller fetches the next word of normal state program code to be executed.

SECTION 8

VARIANT MODE OPERATION AND OPERATORS

ESCAPE TO 16-BIT INSTRUCTION (VARI) 95

The variant mode of operation extends the number of operation codes. These operators are not used as often and require two syllables; the first is the "Escape to 16-Bit Instruction" (VARI) operator. When the VARI operator is encountered, the following syllable is the actual operation and the syllable pointer is positioned beyond the two syllables. The VARI operator is valid only for the syllables covered in this section.

Variant codes EO through EF are detected and cause a programmed operator interrupt. All other unassigned variant codes cause no action and result in a loop timer interrupt.

Variant mode operations are both word- and string-oriented operators.

Appendix A of this manual lists the operators in alphabetic order, and appendix B lists the operators in numeric order by mode.

VARIANT MODE OPERATORS

The variant mode operators are discussed in the following paragraphs.

READ CENTRAL PROCESSOR COUNTER (RCPC) 9540

The RCPC operator returns the current value of the MLIP Processor Timer to the Top-of-Stack register. The 24-bit value returned to the Top-of-Stack represents the time in 2.4 microsecond intervals since the Processor Timer value was last transferred to the Top-of-Stack. The time increment obtained from this circuit is used to provide a method of time-sharing and user-billing by software utility programs. Each time the incrementation of the counter circuit is returned to the Top-of-Stack registers the counter circuit is RESET. Thus, the value returned is either the lapsed time since the timer value was returned, or the lapsed time since a System HALT/LOAD or GENERAL CLEAR function was performed.

RUNNING TIMER INITIALIZE (RUNI) 9541

The RUNI operator causes the MLIP Running Timer circuit to initialize and begin timing. The Running Timer circuit causes the system STATUS RUNI indicator to illuminate while the timer is timing, and to extinguish when the timer circuit times-out (2.041 +/- 0.16 seconds after the timer circuit is initialized). The B 6900 system uses the RUNI indicator logic to show when the CPU is performing a useful function and is not stopped. Under certain privileged-operation conditions the B 6900 system does not exhibit any other visible sign that the CPU is active.

In addition to the RUNI operator code, the CPU micro-module can also initialize the MLIP Running Timer. Thus, certain micro-module operator flows initialize the Running Timer, and under normal system operations the RUNI indicator is never extinguished.

SET TWO SINGLES TO DOUBLE (JOIN) 9542

The operands in the A and B registers are combined to form a double-precision operand that is left in the B and Y registers.

The operand in the A register is placed in the Y register. The A register is marked empty, and the B register TAG field is set to double-precision.

SET DOUBLE TO TWO SINGLES (SPLIT) 9543

The SP(DP) operand in the B register is changed to two single-precision operands which are placed in the A and the B registers; both registers are marked full.

If the operand in the B register is a single-precision operand, the A register is set to all zeroes and the A and B registers are marked full. Both the A and the B register TAG fields are set to single-precision.

If the operand in the B register is a double-precision operand, the Y register operand is placed in the A register and the TAG fields of both the A and B registers are set to single-precision.

IDLE UNTIL INTERRUPT (IDLE) 9544

This operator suspends processor program execution until the program is restarted by an external interrupt. Inhibit Interrupt Flip-Flop (IIFF) is unconditionally reset to allow external interrupts.

SET INTERVAL TIMER (SINT) 9545 (CONTROL STATE OPERATOR)

This operator places the 11 low-order bits of the B register into the interval timer register, and arms the timer. The interval timer decrements each 512 microseconds. The processor is interrupted when the timer reaches zero and is still armed. The interval timer is disarmed when the processor is interrupted by an external interrupt.

The operand used to set the interval timer is integerized before the 11 low-order bits are used. If the operand cannot be integerized, an integer overflow interrupt is set and the operation is terminated.

ENABLE EXTERNAL INTERRUPTS (EEXI) 9546

This operator causes the processor to enter normal state, allowing it to respond to external interrupts. This is accomplished by setting the IIHF flip-flop to zero.

DISABLE EXTERNAL INTERRUPTS (DEXI) 9547

This operator causes the processor to ignore external interrupts. This is accomplished by setting the IIHF to one and entering control state.

WRITE TIME OF DAY (WTOD) 9549

The Write Time Of Day operator causes a right-justified 36-bit value in the Top-of-Stack register to initialize the value of the MLIP Time-of-Day counter circuit. The counter assumes the same value as the Top-of-Stack register, and then proceeds to increment the initial value at a 2.4 microsecond rate. The value of the Time-of-Day counter represents the current time for all B 6900 system operations. The WTOD operator is the method used to SET the system clock to the desired time value.

SCAN OPERATORS

The SCAN-IN functions read information from the Global subsystem to the top-of-stack registers in the data processor. The SCAN-OUT functions write information from the top-of-stack registers in the data processor to the Global memory subsystem.

Parity is checked during transmission of both addresses and information.

SCAN-IN (SCNI) 954A

SCAN-IN uses the A register to specify the type of input required. The input data is placed in the B register. The A register is empty and the B register is full at the completion of the operation. Refer to section 5 for the format of the function and data words for SCAN-IN operations.

SCAN-OUT (SCNO) 954B

The SCAN-OUT operation causes the memory control to sense a function code in the top-of-stack register of the data processor. At the conclusion of the SCAN-OUT operator, the top two words of the stack are deleted from the stack.

CONTROL UNIVERSAL INPUT OUTPUT (CUIO) 954C

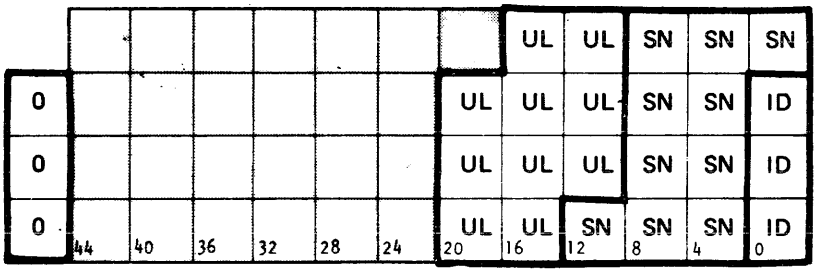
The CUIO operator is executed by the CPU Data Processor to start an MLIP I/O operation sequence. At the beginning of this operator flow a Data Descriptor which points to the first word of an Input Output Command Buffer (IOCB area in system memory) must be present in the CPU Top-of-Stack register.

The CUIO operator causes the IOCB beginning memory address present in the Top-of-Stack register to be strobed into the MLIP R1 register. The MLIP logic generates an Entry Vector to the micro-module (to initiate an MLIP I/O sequence). When the MLIP sequences acknowledge the presence of the IOCB address in the MLIP R1 register, the Data Processor CUIO operation is completed.

READ PROCESSOR IDENTIFICATION (WHOI) 954E

This operator places a word containing the value of the processor ID register in the A register of the data processor.

The format of the word placed in the A register of the data processor is shown in Figure 8-1. At the conclusion of the WHOI operator, the A register is marked full.



50:3 = TAG FIELD
47:25 = NOT USED
22:10 = THE UNIT DESIGN (ERL) LEVEL OF THE CPU.
THIS FIELD IS A BINARY NUMBER WHICH IS DERIVED FROM
A FOREPLANE CONFIGURATION PLUG-ON JUMPER.
ADAPTER OF THE CPU
12:10 = THE SERIAL NUMBER OF THE CPU.
THIS FIELD IS A BINARY NUMBER WHICH IS DERIVED FROM A
FOREPLANE CONFIGURATION PLUG-ON JUMPER ADAPTER
OF THE CPU
2:3 = THE PROCESSOR ID NUMBER OF THE CPU.
THIS FIELD IS A BINARY NUMBER WHICH IS DERIVED FROM A
FOREPLANE CONFIGURATION PLUG-ON JUMPER ADAPTER
OF THE CPU.

MV 1652

Figure 8-1. WHOI Operator Returned Word

OCCURS INDEX (OCRX) 9585

This operator places the following in the B register: a new index value calculated from the Index Control Word (ICW) in the A register, and the operand in the B register (Figure 8-2).

The index word in the B register is integerized. If the index is greater than the maximum integer value (549, 755, 813, 887), the integer overflow interrupt is set and the operation terminated. If either the ICW or the operand has a value of 0, the invalid index interrupt is set and the operation is terminated. If the index value is less than 0 or greater than the SIZE field [31:16] of the ICW, the invalid index interrupt is set and the operation is terminated.

The LENGTH field of the ICW [47:16] is multiplied by the index value [15:16] minus 1, and that value is added to the OFFSET field of the ICW. This result is the new index. The A register is marked empty and the B register is marked full.

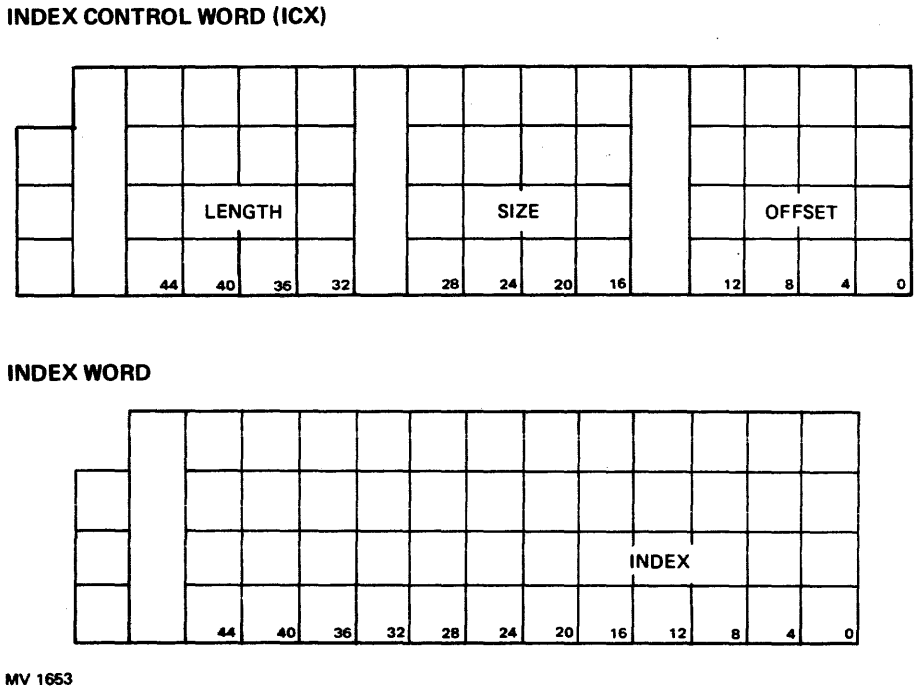


Figure 8-2. Index Control Word (ICW) and Index Word

INTEGERIZE, ROUNDED, DOUBLE-PRECISION (NTGD) 9587

This operator creates (from the operand in the B register) a double-precision, rounded integer in the B register. The B register is marked full. If the word in the B register at the start of this operator is not an operand, the invalid operand interrupt is set and the operation is terminated.

If the operand in the B register is larger than 8 ↑ 26-1 in absolute value, the integer overflow interrupt is set and the operation is terminated.

The B register is marked as a double-precision operand (TAG bits set to 010), and the exponent is set to 13.

LEADING ONE TEST (LOG2) 958B

This operator locates the most significant 1-bit of the word in the B register and places the location of that bit into the B register (bit number + 1). If a 1-bit is not sensed, the B register is set to all zeroes.

The B register is marked full.

NORMALIZE (NORM) 958E

This operator performs normalization of the operand in the top of stack. The normalized operand is left in the B register at the conclusion of the NORM operator, and the B register is marked full. Normalization is defined in Section 2 of this manual.

READ TIME OF DAY (RTOD) 95A7

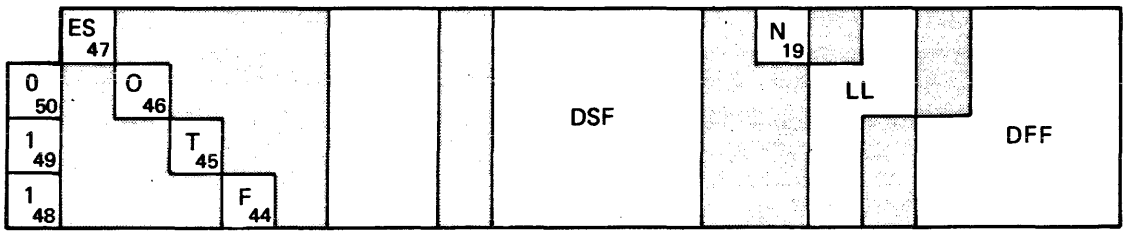
The RTOD operator is used to strobe the current value of the MLIP Time-of-Day register into the CPU Top-of-Stack register, right justified. The current value of the Time-of-Day register is a 36-bit binary value that represents the current count of 2.4 microsecond clock-pulses.

The current count of the Time-of-Day register represents the sum value strobed into the register by a WTOD operator, and a +1 increment for each clock-pulse occurring thereafter. If the B 6900 was GENERAL CLEARED after a WTOD operator was executed, the value in the register represents the number of 2.4 microsecond periods occurring after the GENERAL CLEAR operation. The Time-of-Day counter cycles through a full-count to 0, and continues counting up.

MOVE TO STACK (MVST) 95AF

This operator causes the environment of the processor (or addressing space) to be moved from the current stack to the program stack specified by the operand in the B register.

The operator builds a Top-of-Stack Control Word (TSCW; Figure 8-3) and places it at the base of the current stack as addressed by the base-of-stack register.



ES – EXTERNAL SIGN FLIP FLOP
O – OVERFLOW FLIP FLOP
T – TOGGLE, TRUE-FALSE FLIP FLOP
F – FLOAT FLIP FLOP
DSF – DELTA S-REGISTER FIELD; VALUE OF rS RELATIVE TO BOSR
N – NORMAL-CONTROL STATE FLIP FLOP
LL – ADDRESSING LEVEL
DFF – DELTA F-REGISTER FIELD; VALUE OF rF RELATIVE TO rS

MV 1654

Figure 8-3. Top-of-Stack Control Word (TSCW)

The operand in the B register is integerized and checked against the stack vector for invalid index. The value in the B register is added to the address field of the stack vector descriptor (at D[0]+2) to address the descriptor for the new stack.

The data descriptor for the requested stack is accessed. If the presence bit is "on," the address field is placed into the base-of-stack register. The TSCW is brought up, and the stack is marked "active" by storing the processor ID at the base-of-stack. The TSCW is distributed and the D registers are updated.

If during the integerization the operand in the B register is too large, the integer overflow interrupt is set and the operation is terminated.

If the index value is less than zero or greater than the LENGTH field of the data descriptor for the stack vector array, an invalid index interrupt is set and the operation is terminated.

READ COMPARE FLIP-FLOP (RCMP) 95B3

This operator reads the state of the CMPF flip-flop, and creates a single-precision word in the data processor A register. If the CMPF flip-flop is in the binary one state, the low-order bit (bit zero) of the single-precision word in the A register is set. If the CMPF flip-flop is in the binary zero state, the low-order bit of the A register is reset. The A register is marked full at the conclusion of the operation.

SET TAG FIELD (STAG) 95B4

This operator sets the TAG field (bits 50:3) in the B register to the value of bits 2:3 of the operand in the A register. At the completion of the operation, the A register is marked empty and the B register is left full.

READ TAG FIELD (RTAG) 95B5

This operator replaces the word in the A register with a single-precision operand equal to the TAG field of that word. The TAG bits are placed in bits 2:3. The A register is marked full.

ROTATE STACK UP (RSUP) 95B6

This operator permutes the top three operands of the stack so that the first operand has become the second, the second has become the third, and the third has become the first (see Figure 8-4).

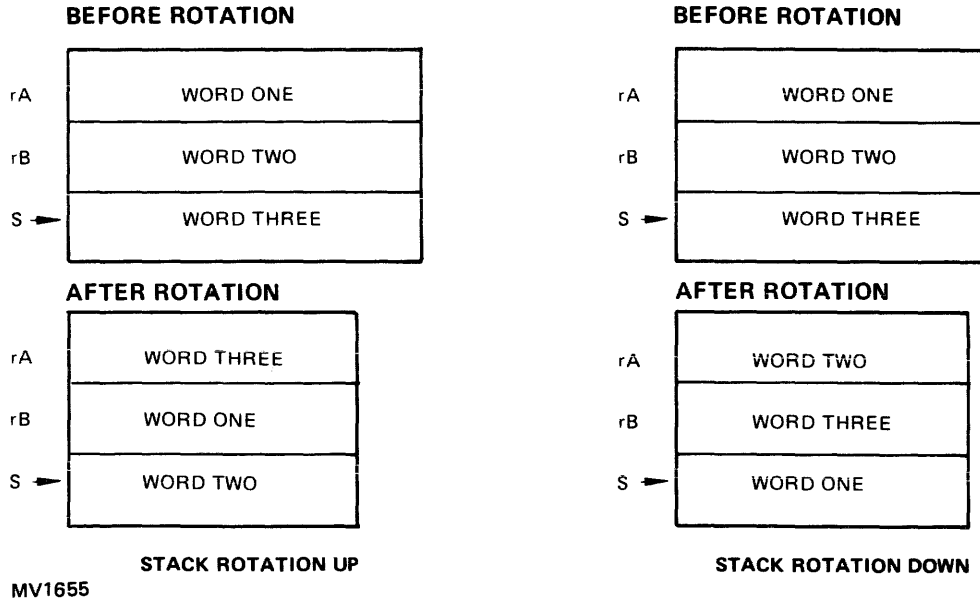


Figure 8-4. Rotate Stack Operations

ROTATE STACK DOWN (RSDN) 95B7

This operator permutes the top three operands of the stack so that the first has become the third, the second has become the first, and the third has become the second (see Figure 8-4).

READ PROCESSOR REGISTER (RPRR) 95B8

This operator reads the contents of one of the eight base registers, eight index registers, or one of the 32 D registers into the A register.

The six low-order bits of the A register selects the processor register to be read.

The decoding of these six bits is as follows:

Bits 5:2	= 10	= Index register
Bits 2:3	= 0,	= PIR
	= 1,	= SIR
	= 2,	= DIR
	= 3,	= TIR, BUF 3
	= 4,	= LOSR
	= 5,	= BOSR
	= 6,	= F
	= 7,	= BUF
Bits 5:2	= 11	= Base register
Bits 2:3	= 0,	= PBR
	= 1,	= IBR
	= 2,	= DBR
	= 3,	= TBR, BUF 2
	= 4,	= S
	= 5,	= SNR
	= 6,	= PDR
	= 7,	= TEMP

If bit 5 is zero, bits 4:5 select the D register equal to the binary value of the bits; that is, bits 4:5 = 00101 select D register 5.

At the completion of this operation, the A register contains the contents of the selected register, and is marked full.

SET PROCESSOR REGISTER (SPRR) 95B9

This operator places the contents of the ADDRESS field of the A register into one of the eight base registers, eight index registers, or 32 D registers selected by the six low-order bits of the word in the B register.

The decoding of the six low-order bits is the same as in the Read Processor Register operator (RPRR) discussed under the previous heading.

The A and B registers are marked empty.

READ WITH LOCK (RDLK) 95BA

This operator performs in a manner similar to the Overwrite operator (see section 7), except the word which was in memory before the overwriting is left in the A register.

COUNT BINARY ONES (CBON) 95BB

This operator counts the number of 1-bit in the single-precision (double-precision) operand in the A register. At the completion of the operation, the total count is left in the A register with the register marked full.

LOAD TRANSPARENT (LODT) 95BC

This operator performs a Load operator (see Section 7) if the word in the A register is a data descriptor or an Indirect Reference Word. If it is neither of these, bits 19:20 of the A register are used as the address to bring an operand to the A register. Copy bit action does not occur.

LINKED LIST LOOKUP (LLLU) 95BD

This operator searches a linked list of words.

The operator starts with an operand in the top-of-stack as the index pointer. The second word in the stack is a non-indexed data descriptor to the array containing the linked list. The third word in the stack is an operand that is the argument.

The base address of the linked list, the length of the list, and the argument value are saved throughout the entire operator process.

The word addressed by the base address plus the index value are read and checked for a value of zero in the address (link) portion of the word (zero denotes the end of the linked list). If the link is non-zero, bits 47:28 are compared to the argument value. If the argument of the linked-list word is less than the argument value, the actions described in this paragraph are repeated, using the link as the new index.

When the value of the argument field of the linked-list word is equal to or greater than the argument value, the operation is complete. The index pointing to the word whose link points to the argument which satisfies the test is left in the A register and is marked full.

If the value of the link portion of the linked-list word is equal to zero, the A register is set to minus one (-1), and marked full as the operation is completed.

If the index value in the linked list word is greater than the length value from the descriptor, an invalid index interrupt is set and the operation is terminated.

When the first word in the stack at the start of this operator is not an operand, an invalid-operand interrupt is set and the operation is terminated.

If the data descriptor has been indexed, the invalid-operand interrupt is set and the operation is terminated.

MASKED SEARCH FOR EQUAL (SRCH) 95BE

At the start of this operator, the word in the A register must be a data descriptor. The operand in the B register is a 51-bit mask. The data descriptor in the A register and the mask in the B register are saved, and the 51-bit argument word is placed into the B register. If the descriptor is indexable (bit 45 equal to zero) one is subtracted from the LENGTH field. If bit 45 is equal to one, the data descriptor is already indexed; therefore, that index is the starting value.

The word addressed by the descriptor is placed in the A register and ANDed with the mask word. The result of this AND function is tested to determine if it is identical to the argument word.

If the comparison is not equal, the INDEX field of the descriptor is decreased by 1 and the operation is repeated. If the INDEX field is equal to 0, the A register is set to a -1 value and marked full. The B register is marked empty.

If an equal comparison is made, the A register contains the index pointing at the last word compared and is marked full. The B register is marked empty.

UNPACK ABSOLUTE, DESTRUCTIVE (UABD) 95D1

This operator unpacks a string of 4-bit digits into 8-bit bytes. At the start of the operator, the word in the A register defines the length of the operand in the B register, that is, the string of digits to be unpacked. The third word in the stack is a string descriptor addressing the destination of the string.

As the specified number of digits are transferred to the destination (most significant bit first), zone fill is as follows:

1. The 8-bit (EBCDIC) format bytes are transferred to the destination string with the four zone-bits to 1111.

2. If the destination size is ZERO, it is set to 8-bit format and handled as in the preceding item (1).

UNPACK ABSOLUTE, UPDATE (UABU) 95D9

This operator performs an Unpack Absolute operation. At the completion of the operation, the destination pointer is updated and left in the stack.

UNPACK SIGNED, DESTRUCTIVE (USND) 95DO

This operator performs an Unpack Absolute operation plus an added function if the External Sign flip-flop is set. Then a zone of 1101 is set in the last byte for 8-bit.

If the destination size is 4-bit, the first digit position of the destination string is set to 1101 provided the External Sign flip-flop is set. If the External Sign flip-flop is 0, the first digit is set to 1100.

UNPACK SIGNED, UPDATE (USNU) 95D8

This operator performs an Unpack Signed operation. At the completion of the operation, the destination pointer is updated.

TRANSFER WHILE TRUE, DESTRUCTIVE (TWTD) 95D3

This operator transfers characters from the source string to the destination string for the number of characters specified by the length operand while the stated relationship is met. If the relationship is not met, the transfer is terminated at that point. The relationship is determined by using the source character to index a table. If the bit indexed is a 1, the relationship is TRUE.

The operator uses the top four words in the stack as follows. The top word addresses the table; the second word is the length of the string to be transferred. The third word in the stack is an operand or a descriptor addressing the source string or a single-precision operand which is the source string; and the fourth word in the stack is a descriptor pointing at the destination string.

The table is indexed as follows to obtain the decision bit. The source character is expanded to 8-bits, if necessary, by appending four leading 0-bits. The three high-order of these eight select a word from the table, thus indexing the table pointer. The remaining five bits of the expanded source character select a bit from this word by their value.

TRANSFER WHILE TRUE, UPDATE (TWTU) 95DB

This operator performs a Transfer While True operation, but updates the source pointer, the destination pointer, and repeat count.

If all the characters specified by the LENGTH field are transferred, the True/False Flip-Flop (TFFF) is set to one (true); otherwise it is set to zero (false).

TRANSFER WHILE FALSE, DESTRUCTIVE (TWFD) 95D2

This operator performs a Transfer While operation and tests for a zero bit in the table.

TRANSFER WHILE FALSE, UPDATE (TWFU) 95DA

This operator performs a Transfer While False operation, but updates the source pointer, the destination pointer, and the repeat count.

If all the characters specified by the LENGTH field are transferred, the True/False Flip-Flop (TFFF) is set to one (true); otherwise it is set to zero (false).

TRANSLATE (TRNS) 95D7

This operator translates the number of characters specified as they are transferred from the source string to the destination string.

The translation uses a table containing the translated characters. The word in the top-of-the stack is a descriptor that addresses the translation table. The second operand in the stack specifies the length of the string. The third word in the stack is a descriptor addressing the source string (or an operand which is the source string), and the fourth word in the stack is a descriptor addressing the destination string. The source and destination are updated at the end of the operation.

The translation occurs as follows. The specified string character is used as an index into the table to locate a character. The located character is transferred to the destination string.

The least significant 32 bits of each table word provide four 8-bit characters. The table sizes are as follows:

1. 4-bit digits provide a 4-word table length.

2. 8-bit bytes provide a 64-word table length.

SCAN WHILE GREATER, DESTRUCTIVE (SGTD) 95F2

This operator scans a string while the characters in the source string are greater than a delimiter character or until the number of characters specified have been scanned.

If all the characters have been scanned at the completion of this operation, TFFF is set to one. If the scan was stopped by the delimiter test before the end of the string, the TFFF is set to zero.

If the delimiter against which the string is compared is equal to the character from the string, then the compare flip-flop (CMPF) is set. If the character in the string is less than the delimiter, then CMPF flip-flop is reset.

At the start of this operator, the delimiter character is right-justified in the top word of the stack. The length of the string to be scanned is the second word of the stack. The source pointer is the third word in the stack.

If the second word in the stack is a descriptor, it is the source pointer, and the length of the character string is set to 1,048,575 (LENGTH field is all ones).

SCAN WHILE GREATER, UPDATE (SGTU) 95FA

This operator performs a Scan While Greater operation and also updates the count and the source pointer. The updated source pointer locates the character that stopped the scan. The number of characters not scanned is placed in the A register, and the register is marked full.

SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE (SGED) 95F1

The operator performs a Scan While operation while the characters in the source string are equal to or greater than the delimiter character. If all the characters have been scanned at the completion of the operation, then the TFFF flip-flop is set.

SCAN WHILE GREATER OR EQUAL, UPDATE (SENU) 95F9

This operator performs a Scan While Greater or Equal operation, but also updates the count and the source pointer.

SCAN WHILE EQUAL, DESCTRUCTIVE (SEQD) 95F4

This operator performs a Scan While operation while the characters in the source string are equal to the delimiter character. If all characters are compared, then the TFFF flip-flop is set.

If the delimiter against which the string is compared is less than the character from the string, then the compare flip-flop (CMPF) is set.

SCAN WHILE EQUAL, UPDATE (SEQU) 95FC

This operator performs a Scan While Equal operation, but also updates the count and the source pointer.

SCAN WHILE LESS OR EQUAL, DESTRUCTIVE (SLED) 95F3

This operator performs a Scan While operation while the characters in the source string are equal to or less than the delimiter character. If all characters are compared, then the TFFF flip-flop is set.

SCAN WHILE LESS OR EQUAL, UPDATE (SLEU) 95FB

This operator performs a Scan While Less or Equal operation, but also updates the count and source pointer.

SCAN WHILE LESS, DESTRUCTIVE (SLSD) 95FO

This operator performs a Scan While operation while the characters in the source string are less than the delimiter character.

SCAN WHILE LESS, UPDATE (SLSU) 95F8

This operator performs a Scan While Less operation, but also updates the count and the source pointer.

If the character from the table (against which the string is compared) is equal to the character from the string, then the compare flip-flop (CMPF) is set.

SCAN WHILE NOT EQUAL, DESTRUCTIVE (SNED) 95F5

This operator performs a Scan While operation while the characters in the source string are not equal to the delimiter character. If all characters are compared, then the TFFF flip-flop is set.

SCAN WHILE NOT EQUAL, UPDATE (SNEU) 95FD

This Operator performs a Scan While not Equal operation, but also updates the count and the source pointer.

SCAN WHILE TRUE, DESTRUCTIVE (SWTD) 95D5

This operator uses each source character as an index into a table to locate a bit in the same fashion as the transfer while True operators. If the bit located contains the value of one, the relationship is true and the scan continues.

The first word in the stack is a descriptor addressing the table. The second and third words in the stack are the same as for all Scan While operators.

SCAN WHILE TRUE, UPDATE (SWTU) 95DD

This operator performs a Scan While True operation, but also updates the count and the source pointer. The number of characters not scanned is placed in the A register.

SCAN WHILE FALSE, DESTRUCTIVE (SWFD) 95D4

This operator performs a Scan While False operation, except the relation is true if the bit found by indexing into the table contains the value of zero.

SCAN WHILE FALSE, UPDATE (SWFU) 95DC

This operator performs a Scan While False operation, but also updates the count and the source pointer.

SECTION 9

EDIT MODE OPERATION AND OPERATORS

GENERAL

The purpose of the edit mode operators is to perform editing functions on strings of data. The editing functions are those which are normally involved in preparing information for output. They include such operators as move, insert, and skip, in the form of micro-operators in either the program string or in a separate table. In the program string, they are single micro-operators, and are entered by use of the execute single micro or single pointer operators (see section 7). If the micro-operators are in a table, the table becomes the program string to be executed. This table is entered by means of the Table Enter Edit operators (see section 7), and is exited through the end edit micro-operator, as defined later in this section.

If the source or destination data has the memory protect bit (bit 48) equal to one, the segmented-array interrupt is set and the current micro-operator is terminated.

Appendix A of this manual lists the operators in alphabetic order, and appendix B lists the operators in numeric order by mode.

EDIT MODE OPERATORS

The edit mode operators are described in the following paragraphs of this section.

MOVE CHARACTERS (MCHR) D7

This micro-operator transfers characters from the source string to the destination string.

If this micro-operator is entered by the table enter edit operator (see Section 7), the number of characters to be transferred is specified by the syllable following the operator syllable.

If this micro-operator is entered by the execute single micro-operator (see Section 7), the number of characters to be transferred is specified by the operand in the top-of-the stack.

MOVE NUMERIC UNCONDITIONAL (MVNU) D6

This micro-operator transfers the four low-order bits of the characters of the source string to the desgination string. If the destination string character size is 8 bits (EBCDIC), the zone bits are set to 1111.

If this micro-operator was entered by use of the table enter edit operator (see Section 7), the number of characters to be transferred is specified by the syllable following the micro-operator syllable.

If this micro-operator is entered by executing the execute single micro-operator (see Section 7), the number of characters to be transferred is specified by the operand in the top-of-the stack.

MOVE WITH INSERT (MINS) DO

This micro-operator performs a move numeric unconditional or an insert operation under the control of the Float flip-flop.

In table edit mode, the second syllable is the repeat value and the third syllable is the character to be inserted under control of the Float flip-flop.

In execute single micro-mode, the repeat field value is the top word-of-stack and the insert character is in the syllable following the micro-operator syllable.

If the Float flip-flop equals zero and the numeric portion of the source characters equals zero, the insert character is moved to the destination string.

If the Float flip-flop is reset and the numeric portion of the source character is not equal to zero, then set the Float flip-flop and perform a Move Numeric Unconditional operation.

The number of characters transferred from the source string to the destination string is defined by the repeat value.

MOVE WITH FLOAT (MFLT) D1

In table edit mode, the second syllable is the repeat value (the number of characters to transfer). The third, fourth, and fifth syllables are the three insert characters. In single-micro mode, the three insert characters are in the second, third, and fourth syllables.

If the Float flip-flop equals zero and the numeric portion of the character in the source string equals zero, the first-insert character is transferred to the destination string.

If the Float flip-flop equals zero and the numeric portion of the character in the source string is not zero, the Float flip-flop is set. If the External Sign flip-flop equals one, the second insert character is transferred to the destination string. If the External Sign flip-flop equals zero, the third insert character is transferred to the destination string. The numeric version of the source character is then transferred.

If the Float flip-flop equals one, the numeric equivalent of the source character is transferred to the destination.

This operation continues for the number of characters defined by the REPEAT field value. This operator can be entered by the Execute Single Micro-operator, with the REPEAT field value in the top word-of-stack.

SKIP FORWARD SOURCE CHARACTERS (SFSC) D2

This micro-operator increments the source pointer registers.

If this micro-operator or any of the following skip micro-operators is entered by the execution of the Execute Single Micro-operator, the number of characters to be skipped is specified by the operand in the top-of-stack. If entry is by the execution of the Table Enter Edit operators, the number of characters to be skipped is specified by the syllable following the micro-operator syllable.

SKIP REVERSE SOURCE CHARACTERS (SRSC) D3

This micro-operator decrements the source pointer registers (also see Skip Forward Source Characters micro-operator, second paragraph).

SKIP FORWARD DESTINATION CHARACTERS (SFDC) DA

This micro-operator increments the destination pointer registers.

SKIP REVERSE DESTINATION CHARACTERS (SRDC) DB

This micro-operator decrements the destination pointer registers.

RESET FLOAT (RSTF) D4

This micro-operator sets the Float flip-flop to zero.

END FLOAT (ENDF) D5

This micro-operator transfers the character in the second syllable of this operator to the destination string if the Float flip-flop contains a zero and the External Sign flip-flop is one.

If the Float flip-flop contains a zero and the External Sign flip-flop also equals zero, then the character in the third syllable of this operator is transferred.

If the Float flip-flop contains one, then it is reset and no characters are transferred.

INSERT UNCONDITIONAL (INSU) DC

This micro-operator places an insert character into the destination string for the number of times specified by the repeat value. When entered by a Table Enter Edit operator, the repeat value is in the syllable following the micro-operator syllable, and the insert character is in the next syllable.

If this micro-operator is entered by an Execute Single Micro-operator, the character to be inserted is in the second syllable, and the repeat value is specified by the operand in the top-of-stack.

INSERT CONDITIONAL (INSC) DD

This micro-operator inserts a string consisting of one or two characters into the destination string. The length of the string is given by the repeat value from the table or the stack.

If the Float flip-flop contains a zero, the first insert character is inserted into the destination string.

If the Float flip-flop contains a one, the second insert character is inserted into the destination string.

The insert characters follow the repeat value syllable in Table Enter Edit operation or the micro-operator syllable in Execute Single Micro-operations.

INSERT DISPLAY SIGN (INSG) D9

This micro-operator places in the destination string the character defined by the syllable following the micro-operator syllable, if the External Sign flip-flop is equal to one.

If the External Sign flip-flop is equal to zero, this operator places in the destination string the character defined by the third syllable of this operator.

INSERT OVERPUNCH (INOP) D8

If the External Sign flip-flop is equal to one, this micro-operator places a sign overpunch in the destination string character of 1101 for EBCDIC.

If the External Sign flip-flop is equal to zero, the operator leaves the destination string character unaltered.

END EDIT (ENDE) DE

This operator terminates a string of Edit micro-operators in Table Enter Edit operation mode.

The microprogram string in the table must end with the End Edit operator.

SECTION 10

VECTOR MODE OPERATORS

GENERAL

The use of Vector Mode provides for an increase in efficiency in the manipulation of arrays. The increase in efficiency is not an automatic feature that applies to all data processor operations. Vector Mode makes it possible for certain software compilers, such as ALGOL or FORTRAN, to specify that Vector Mode rules apply under controlled conditions.

LIMITATIONS OF VECTOR MODE

Vector Mode operations require that the system be operated in control state. This requirement means that a processor performing Vector Mode operations cannot be interrupted to service external interrupts.

Vector Mode operations do not permit segmentation of the arrays. This occurs because presence bit interrupts are disallowed. This limitation requires that the entire extent of the array/arrays must be present in memory while performing vector operations.

Vector Mode operation allows the use of other modes and operators in the B 6900 operator set, subject to the following limitations:

a. String operators and Edit Mode operators are not allowed.

b. No family C operators, except the branching operators (BRTR, BRFL, and so forth) are allowed while operating in Vector Mode.

c. No operator that pseudo-calls a family C operator is allowed while operating in Vector Mode.

d. The LIT 48 and branch operators are not used while performing in Single Program Word Vector Mode (VMES) because of the size of the operator codes, in syllables.

Appendix A lists the operators in alphabetic order, and appendix B lists the operators in numeric order by mode.

HARDWARE FUNCTIONS

The Vector Mode hardware does the following:

a. Utilizes registers to hold the actual addresses of array elements that are referenced.

b. Uses additional registers to contain the increment values used for altering the addresses (indexing) to reference successive array elements.

c. Uses one register to contain a "count" or length that controls the number of iterations.

d. Provides for cycling through one (single-word mode) or more (multiple-word mode) words of code for each iteration.

e. Introduces new operators for use while in Vector Mode to load and store the top-of-stack, and to control iterating and exiting from Vector Mode.

f. Provides two primary mode operators used to enter Vector Mode.

Seven IC memory locations are used as the registers previously mentioned to hold the three absolute addresses, the three corresponding increment values, and the length.

The three addresses are referred to as A, B, and C, respectively.

These registers are loaded automatically from the stack upon execution of either of two Enter Vector Mode operators.

PRIMARY MODE ENTER VECTOR MODE OPERATORS

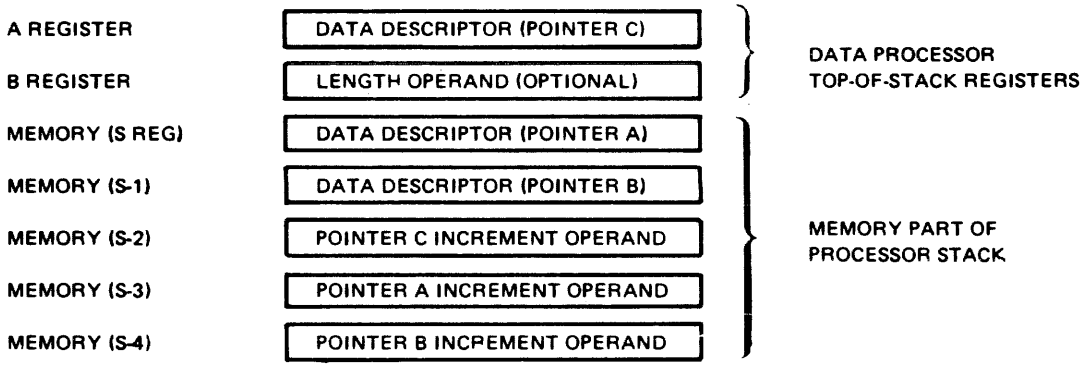
Two primary mode operators are used for Vector operations in the B 6800 Data Processor. These operators are as follows:

Family	Mnemonic	Hexadecimal Code	Operator Description
G	VMOS	EF	Vector Mode Enter Single
G	VMOM	E7	Vector Mode Enter Multiple

ENTER VECTOR MODE OPERATION

An entry into Vector Mode operations occurs when the VMOS (EF) or VMOM (E7) operator is executed from the processor P register. Prior to entering Vector Mode, the processor stack must be properly configured to perform Vector operators (Figure 10-1).

The processor registers and the operating stack must have the following format:



NOTE

If the optional LENGTH operand is not present in the stack, all subsequent required parameters are moved one word closer to the top-of-stack. There are no vacant spaces in the format.

MV4568

Figure 10-1. Vector Mode Stack Configuration

Before entering Vector Mode, the values to be stored in IC memory must be placed in the stack. LENGTH specifies the number of iterations through the code to be executed while in Vector Mode, usually the number of elements in the arrays being manipulated. The presence of a LENGTH value in the stack is indicated by bit 44=1 in Pointer C. Should bit 44=0, a default LENGTH of 2^{20-1} is stored in the LENGTH register. Bit 44 (segmented bit) must be OFF in Pointer A and Pointer B. The software ascertains that bit 44 is ON in Pointer C before using it to indicate the presence of a LENGTH value.

The seven parameters are inserted in IC memory as follows:

Register	Vector Mode Contents of Register
BRS3	Pointer C [19:20] (or Pointer C [39:20] plus [19:20] if I* = 1)
BRS7	LENGTH [19:20] (or 2^{20}-1)
BRS1	Pointer A [19:20] (or Pointer A [39:20] plus [19:20] if I* = 1)
BRS2	Pointer B [19:20] (or Pointer B [39:20] ** plus [19:20] if I* = 1)
IRS3	Pointer C increment [19:20]
IRS1	Pointer A increment [19:20]
IRS2	Pointer B increment [19:20]

*I is the indexed bit, bit 45 in the descriptor.
**Use [35:16] if the size field is not equal to zero.

The Enter Vector Mode operator can be terminated by one of the following interrupts:

Type of Interrupt	Cause of the Interrupt
a. Invalid OP:	Pointer A, B or C not tagged as a data descriptor or Pointer A or B has bit 44=1.
b. Memory Protect:	Pointer A is read only (bit 43=1).
c. Presence Bit:	Pointer A, B or C has bit 47=0.

At the conclusion of the enter Vector Mode flow, the IC memory is configured as follows:

Register Name	Contents of the Register
SIR	The value of the "A" increment
DIR	The value of the "B" increment
TIR	The value of the "C" increment
SBR	The base address of pointer "A"
DBR	The base address of pointer "B"
TBR	The base address of pointer "C"
TEMP	The value of the LENGTH operand

The word in the P register at the end of the Enter Vector Mode flow contains the Vector operators to be executed. The PSR register is equal to zero, and thus specifies that the first Vector Mode operator commences in syllable zero.

If the entry to Vector Mode is the single-word mode entry VMES operator, the single word of code following that entry is held in the P Register (program word fetching is inhibited) and executed a number of times equal to the LENGTH parameter. Each time the word is executed, LENGTH is decremented by one until it becomes zero. Then Vector Mode is exited and normal operation continues with the next word of code in sequence.

VECTOR STACK OPERATORS

Vector Stack operators are a group of twenty-eight operators with a common syllable format (Figure 10-2). Variations of this syllable provide the capabilities of storing or loading the top-of-stack with a single- or double-precision operand and choosing whether or not to increment the pointer.

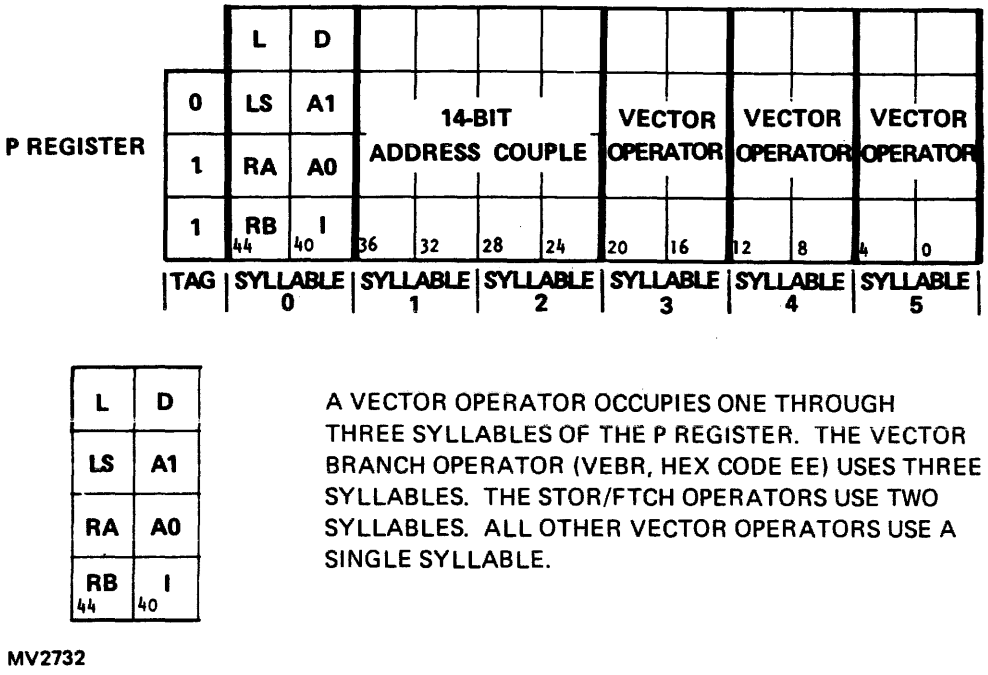


Figure 10-2. Vector Mode Operator Format

The format of the Vector Operator syllable is as follows:

Bit	Description
L	The most significant bit in the Vector operator equals one if a LENGTH factor is passed to the vector stack upon entering Vector Mode; otherwise, L equals zero.
LS	Bit is OFF (0) for a Top-of-Stack Load operator and ON (1) for a Top-of-Stack Store operator.
RA	If a memory protect interrupt is sensed and no LENGTH is passed to the Vector Mode and RA=0, the top-of-stack word is deleted. If RA=1, the top-of-stack word is not deleted.
RB	Same as the RA bit, except that it governs the action taken on the second word of the stack.
D	Double-precision bit. If D=0, load or store single-precision operand (Fam G). If D=1, load or store double-precision operand (Fam H).

Bit	Description
A1, A0	Selects the IC Memory Address register.

A1	A0	
0	0	Load from Pointer A (BRS1)
0	1	Load from Pointer B (BRS2)
1	0	Load from Pointer C (BRS3)

I	When I equals one, the pointer used for the memory address is increased by its corresponding pointer increment following the Load or Store operator. When I equals zero, the pointer increment is inhibited.

VECTOR MODE OPERATOR CODES

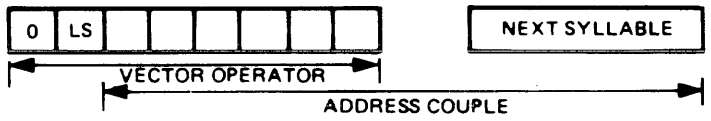
The twenty-seven Vector Mode operators are identified in Figure 10-3.

FAMILY		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
G	E	LDA	LDAI	LDB	LDBI	LDC	LDCI	VMEX		DLA	DLAI	DLB	DLBI	DLC	DLCI	VEBR	
H	F	STA	STAI	STB	STBI	STC	STCI			DSA	DSAI		DSBI	DSC	DSCI	NOOP	NVLD

MV4569

Figure 10-3. Vector Mode Operators

Two other operators are used to load/store the top-of-stack from/to an address couple. They are enabled only when a LENGTH is passed by the Vector Mode entry. Their format is shown in Figure 10-4.



MV4573

Figure 10-4. Load/Store Vector Mode Operators

The address couple is formed from the low-order 6-bits of the Vector operator, and the next operator syllable, which are concatenated to form a 14-bit address couple.

Where: LS=0 then load (FTCH operator), or when
LS=1 then store (STOR operator).

The A register is loaded from (or stored into) the memory location determined by the normal address couple decoding convention (same as Value Call).

The previously listed operator mnemonic codes for Vector Mode are consistent with the mnemonic codes used for the B 6700 System Hardware operator flow charts in the Test and Field Documentation Release Package. Certain software compilers (ALGOL and FORTRAN) have the capability to emit program code mnemonics for Vector Mode operators in a program code

stream when the LIST CODE option is used at compile time. The operator mnemonics emitted by these compilers do not follow the code mnemonics used by the B 6900 System flow charts (see Figure 10-5).

The operator mnemonics emitted by the compilers are subject to review with each revision of the compilers, and may change because of a change in the level of the Master Control Program (MCP) release. The following operator mnemonics are taken from the current release level of the ALGOL and FORTRAN compilers. Subsequent revisions to the compilers may cause these mnemonics to be in error; therefore, care must be taken in using them without recourse to specify MCP release level documentation.

FAMILY		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
G	E	L1AX	L1AI	L1BX	L1BI	L1CX	L1CI	VXIT		1LAX	1LAI	1LBX	1LBI	1LCX	1LCI	VEBR	
H	F	S1AX	S1AI	S1BX	S1BI	S1CX	S1CI			1SAX	1SAI		1SBI	1SCX	1SCI	NOOP	NVLD

MV2733

Figure 10-5. FORTRAN/ALGOL Compiler Vector Mode Operator Mnemonics

VECTOR OPERATORS

The following is a list of Vector Stack operators.

Operator	Hex OP-Code	Description
Load A	E0	The stack is adjusted (0,2) and the single-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack.
Load B	E2	The stack is adjusted (0,2) and the single-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack.
Load C	E4	The stack is adjusted (0,2) and the single-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack.
Load A – Increment	E1	The stack is adjusted (0,2) and the single-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack. Pointer A is increased by its increment (IRS1) following the transfer.
Load B – Increment	E3	The stack is adjusted (0,2) and the single-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack. Pointer B is increased by its increment (IRS2) following the transfer.
Load C – Increment	E5	The stack is adjusted (0,2) and the single-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack. Pointer C is increased by its increment (IRS3) following the transfer.
Store A	F0	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1).

Operator	Hex OP-Code	Description
Store B	F2	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2).
Store C	F4	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3).
Store A – Increment	F1	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1). Pointer A is increased by its increment (IRS1) following the transfer.
Store B – Increment	F3	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2). Pointer B is increased by its increment (IRS2) following the transfer.
Store C – Increment	F5	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3). Pointer C is increased by its increment (IRS3) following the transfer.
Double Load A	E8	The stack is adjusted (0,2) and the double-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack.
Double Load B	EA	The stack is adjusted (0,2) and the double-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack.
Double Load C	EC	The stack is adjusted (0,2) and the double-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack.
Double Load A – Increment	E9	The stack is adjusted (0,2) and the double-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack. Pointer A is increased by its increment (IRS1) following the transfer.
Double Load B – Increment	EB	The stack is adjusted (0,2) and the double-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack. Pointer B is increased by its increment (IRS2) following the transfer.
Double Load C – Increment	ED	The stack is adjusted (0,2) and the double-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack. Pointer C is increased by its increment (IRS3) following the transfer.
Double Store A	F8	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1).
Double Store B	FA	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2).
Double Store C	FC	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3).
Double Store A – Increment	F9	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1). Pointer A is increased by its increment (IRS1) following the transfer.

Operator	Hex OP-Code	Description
Double Store B – Increment	FB	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2). Pointer B is increased by its increment (IRS2) following the transfer.
Double Store C – Increment	FD	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3). Pointer C is increased by its increment (IRS3) following the transfer.
Vector Branch	EE	A three-syllable operator where the two syllables following the operator contain a branch address. If the length count is > 0, the LENGTH count is decremented by one, and the program continues at the next syllable following the address. If the LENGTH is equal to zero, Vector Mode is exited by fetching the program word specified by the branch address.
Vector Mode Exit	E6	Allows the program to exit from Vector Mode to Primary Mode.

VECTOR BRANCH AND VECTOR EXIT OPERATORS

When the entry to Vector Mode is the multiple-word type (VMOM operator), whatever code that follows it is executed under Vector Mode rules. The two Vector Mode operators explained as follows are used only in conjunction with the VMOM operator.

a. Vector Mode Exit operator (VMEX) causes the program to exit from Vector Mode and return to normal mode operations.

b. Vector Branch (VEBR) is a three-syllable operator. The two syllables following the operator code contain the branch address. The Vector Branch operator examines LENGTH. If it is greater than zero, LENGTH is decremented by one, the next two program syllables containing the branch address are skipped, and the program is resumed at the following syllable. If the examined LENGTH is zero, Vector Mode is exited, and normal mode operation commences with the program word located by the branch address.

SECTION 11

INPUT OUTPUT DEVICE OPERATIONS

MLIP GENERAL INFORMATION

Figure 11-1 shows the relationships and operating environment of the Message Level Interface Processor (MLIP) Module in a B 6900 system. The MLIP is a semi-autonomous control device, which is used to create and control interfaces between the software Master Control Program (MCP) and the Universal Input/Output (UIO) device subsystem. Semi-autonomous means that the MLIP must be initiated into operation by the MCP, through execution of a CUIO operator code. Once the MLIP is initialized into operation, the micro-module control logic takes command; and subsequent MLIP operations are determined by the micro-module logic, not by the MCP.

In addition to creating and controlling UIO interfaces, the MLIP also performs other system functions that basically involve the use of timers or time-counting circuits. Some of these timing functions are controlled by inputs to the MLIP from the software operating system, and others are automatic functions of the MLIP logic circuits.

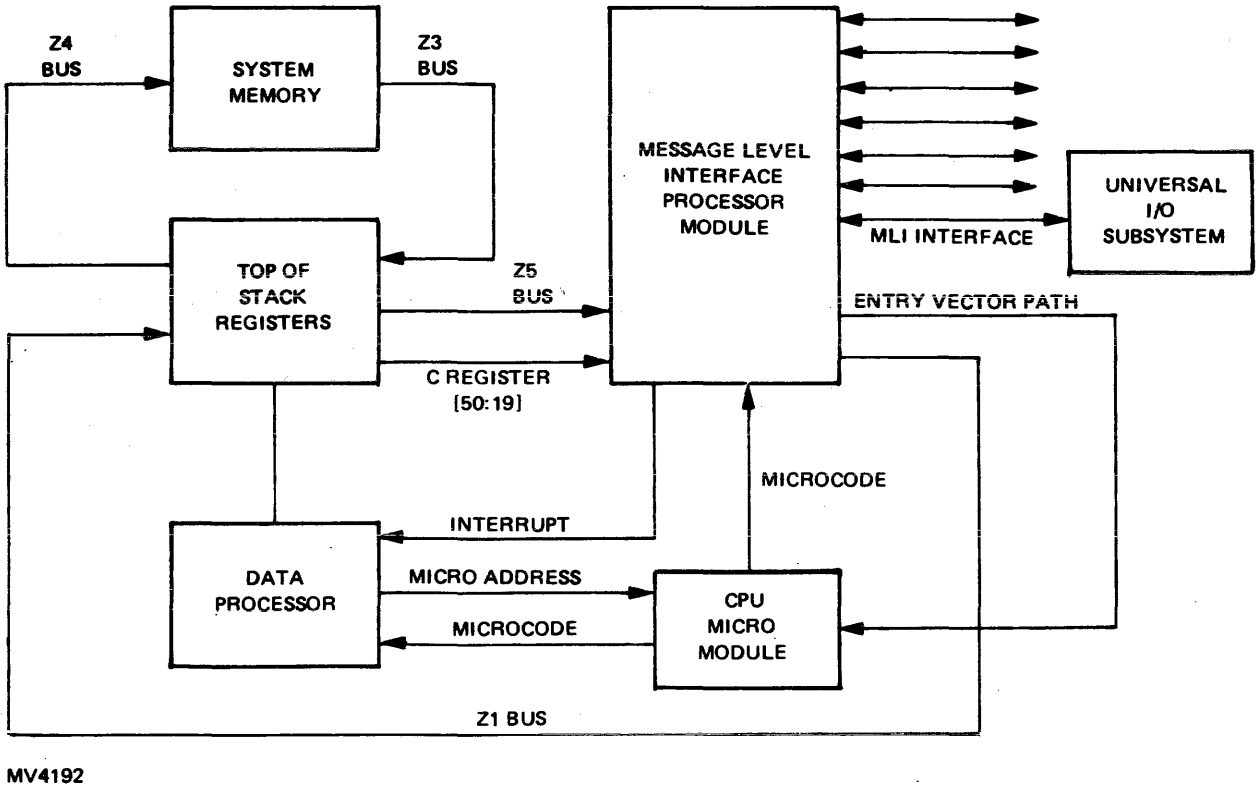


Figure 11-1. B 6900 System MLIP Module Environment

UIO SUBSYSTEM GENERAL INFORMATION

Peripheral I/O devices in a B 6900 system (see Figures 11-2 and 11-3) are controlled by data Link Processor (DLP) adapters. A unique DLP adapter is used for each type of peripheral device connected to a B 6900 system. A DLP adapter contains micro-coded control programs which are unique to the type of I/O device the DLP controls.

DLP adapters are card-package modules which plug into a UIO-Base module backplane of an Input Output Data Communications (IODC) cabinet (see Figure 11-4). The IODC cabinets in a B 6900 system are connected to the MLIP module of the CPU by means of external 25-wire MLI cables. From 1 to 8 MLI cables are connected to the MLIP module, and each MLI cable interfaces up to 8 intraconnected UIO-Base modules to the MLIP module (see Figure 11-5).

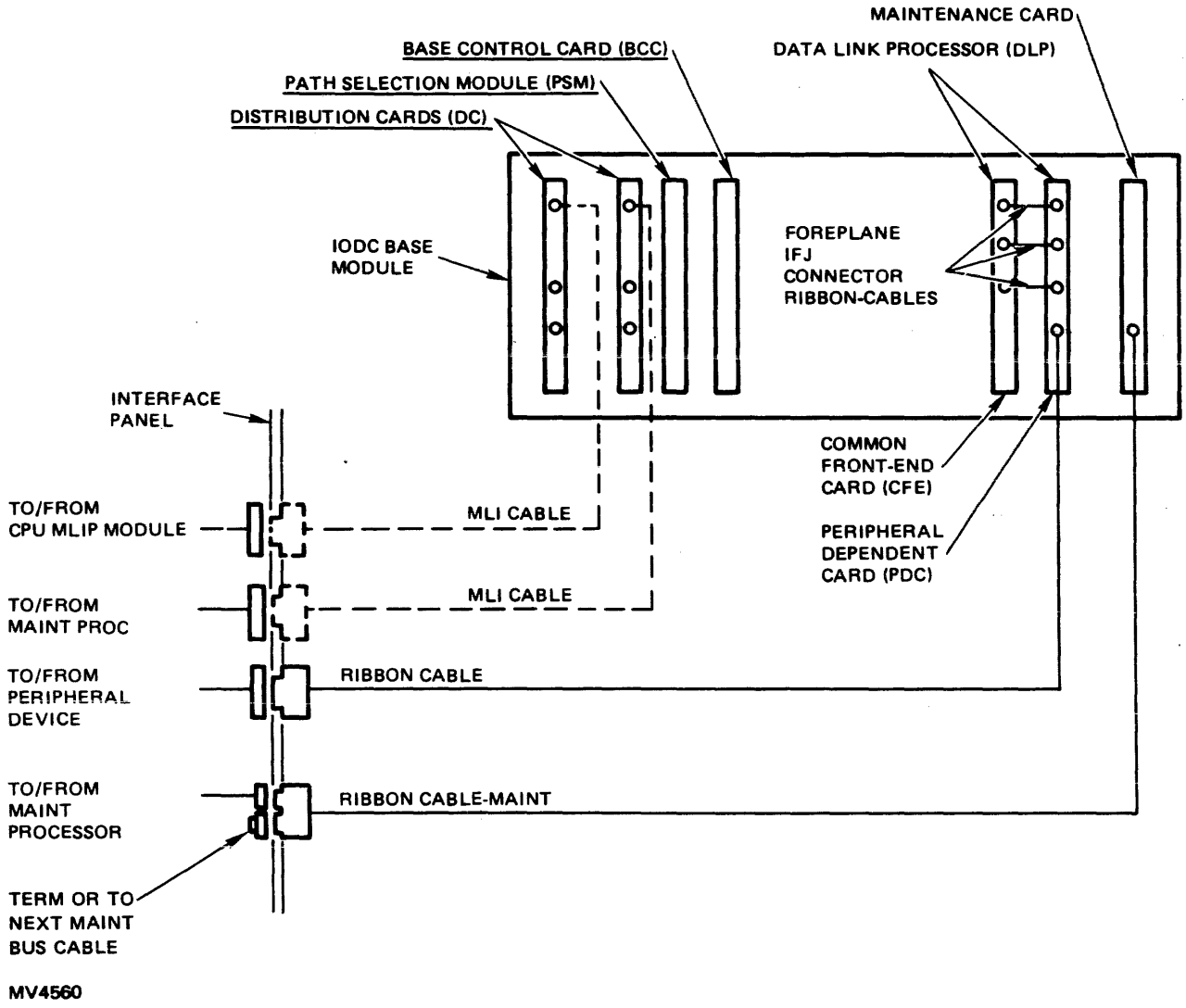


Figure 11-2. IODC Base Module With One DLP

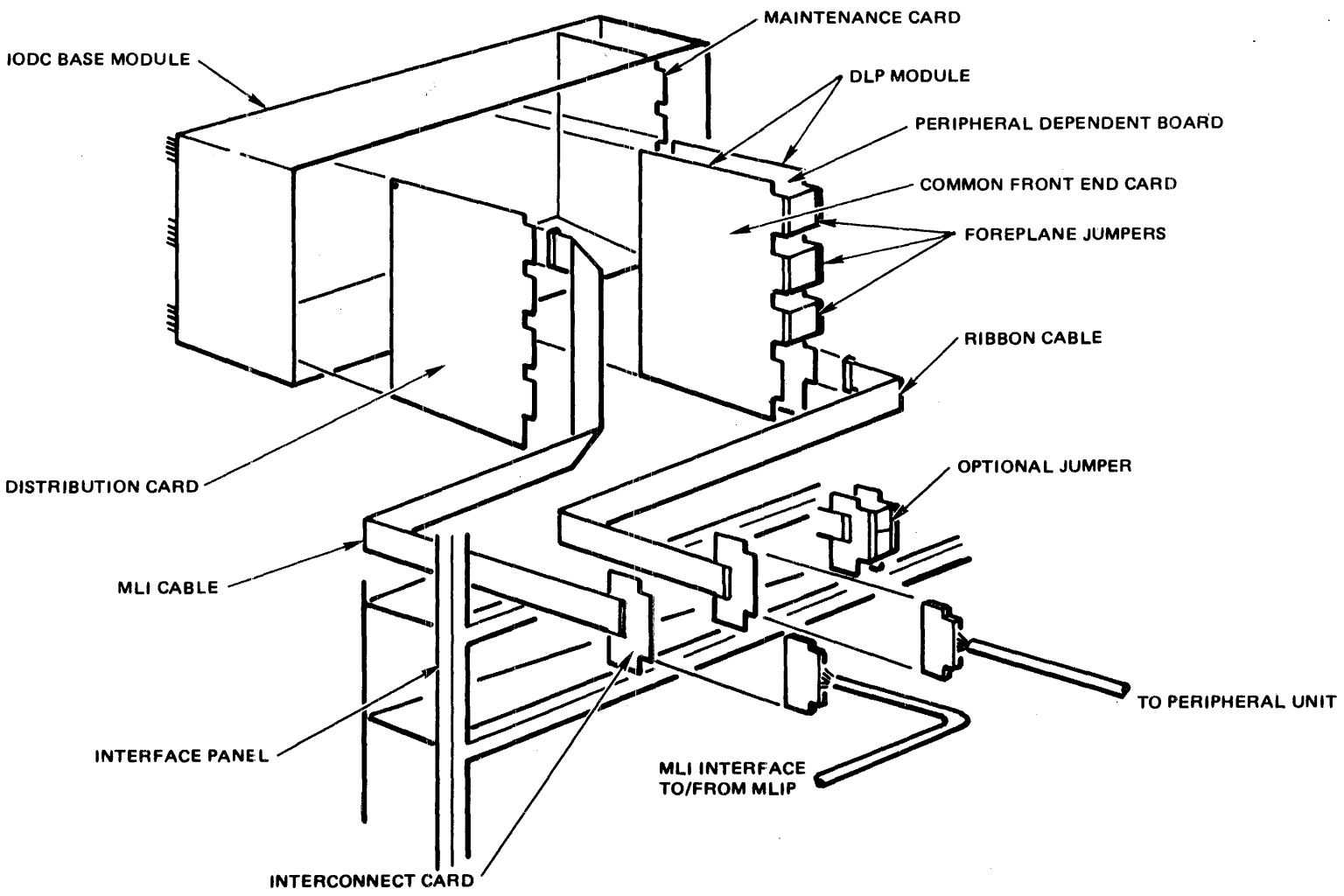


Figure 11-3. B 6900 IODC Base Module Organization

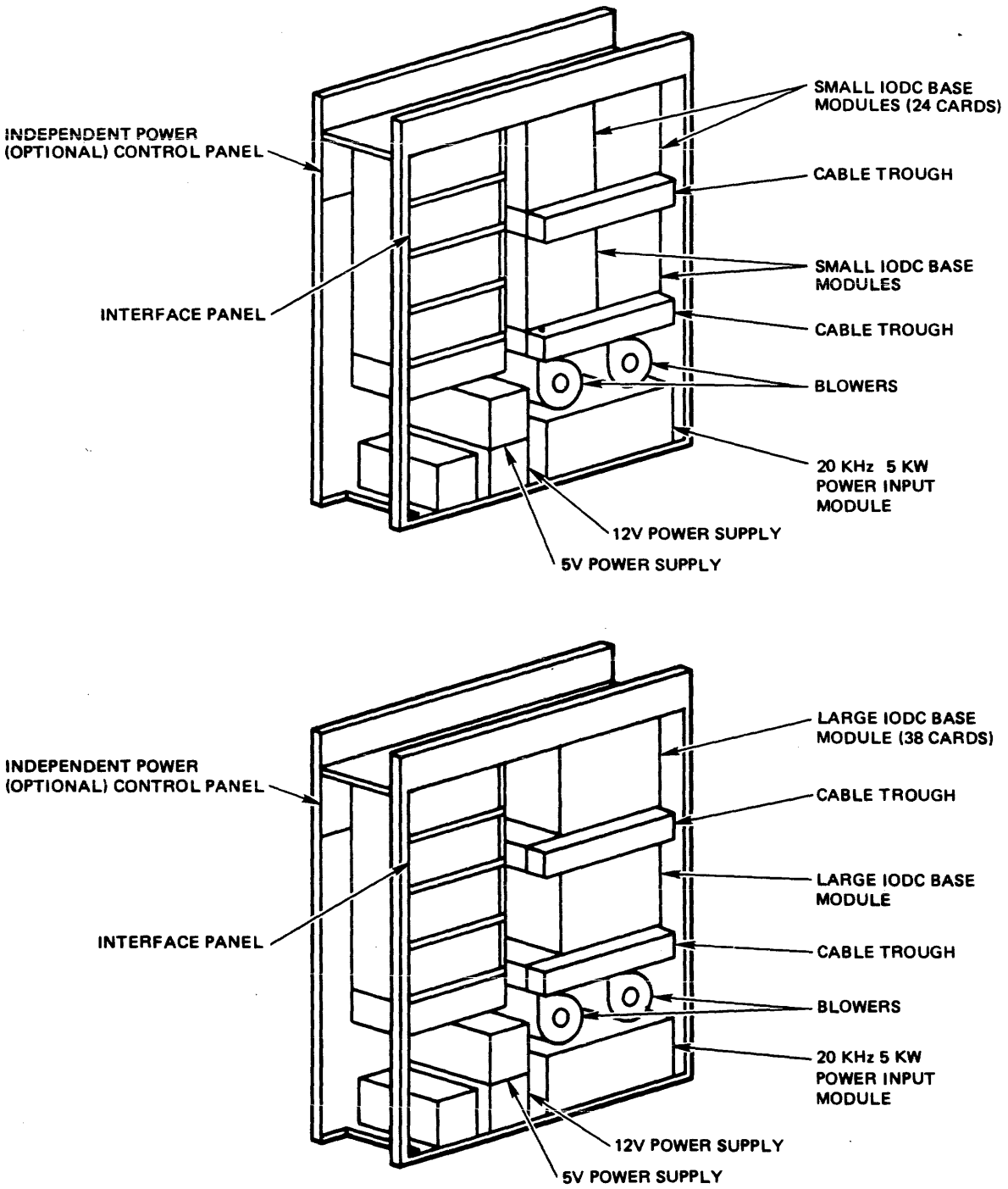


Figure 11-4. B 6900 IODC Base Module Cabinets

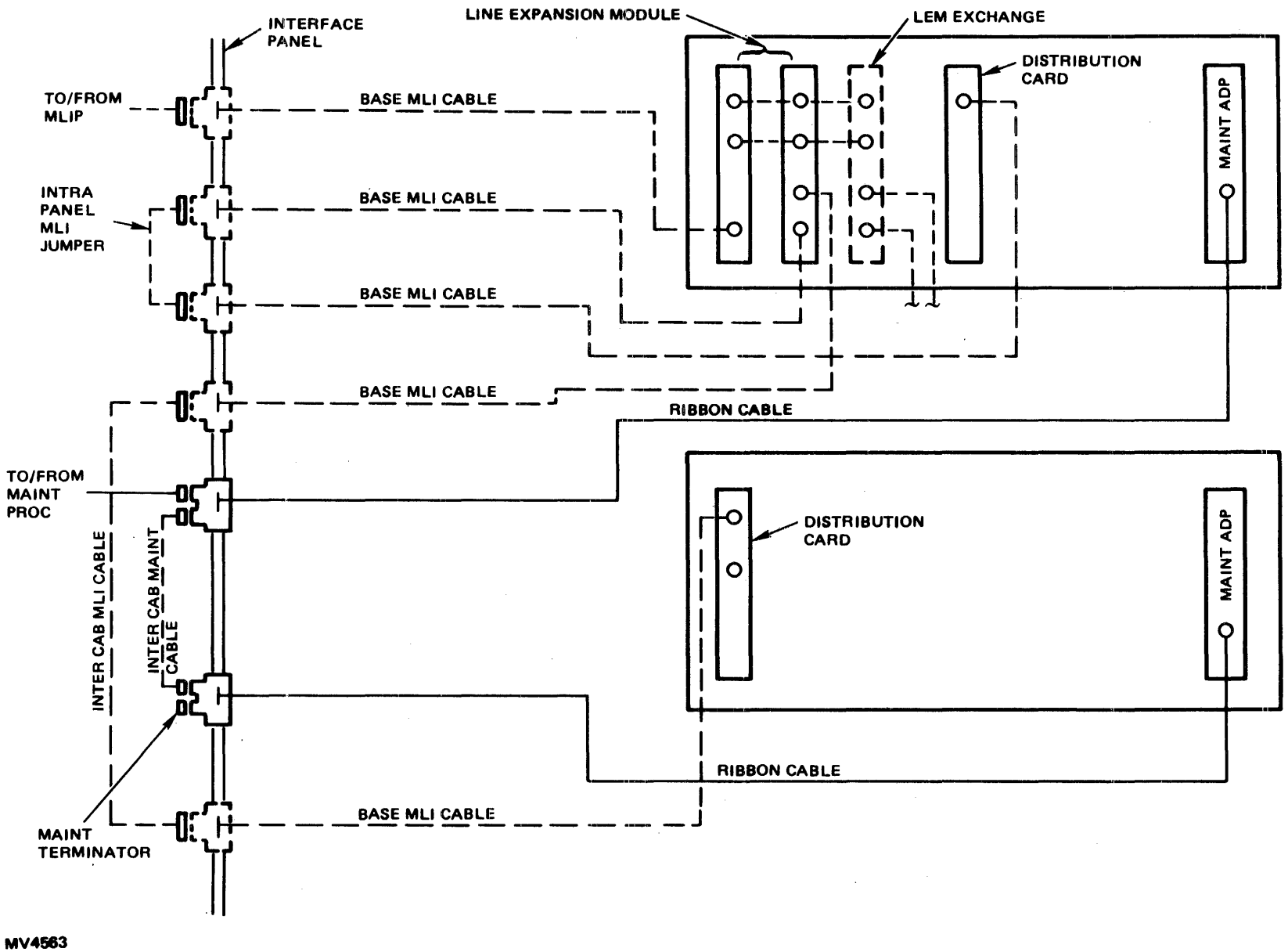


Figure 11-5. Multiple IODC Cable Connections

A separate MLI cable connects the B 6900 Maintenance Processor module to a UIO-Base module. A different external cable interfaces the B 6900 Maintenance Processor to the maintenance logic circuits in the various B 6900 system IODC cabinets.

The 25-line MLI cable contains 17 lines used to transfer a word of data (16-bits of data plus an odd-parity bit) between the MLIP and the UIO subsystem. This cable also contains 4 lines used to send DLP sequence counts and result status to the MLIP module. One line of the MLI interface is a system strobe-signal line used by the MLIP to initiate actions in the UIO subsystem logic. Another line is a strobe signal line used by the UIO subsystem to initiate actions in the MLIP logic. The remaining 2 lines are used for various synchronizing logic levels and signals, during an MLI line communication process.

B 6900 I/O DEVICE OPERATION PROCESSES

I/O peripheral device operations begin when the CPU MLIP module uses one of its MLI Port interface cables to communicate with the UIO subsystem. This interface communication must follow an established MLI interface protocol. The protocol requires that at least 6 MLI data words of UIO control information, in fixed word formats, be passed from the MLIP module to the UIO-Base and DLP logic. The entire process of meeting the requirements of the MLI protocol is commonly referred to as a connection sequence.

The 6 required data-words transferred during a connection sequence are:

1. An MLI Address word that identifies the Base module and DLP addressed.

2. An I/O Descriptor that identifies not only the particular function the I/O peripheral device is to execute, but also the specifications for optional characteristics of the I/O device that apply during the execution.

3. A Longitudinal Parity Word used to verify that the I/O Descriptor word/words are valid MLI connection sequence words.

4. Two Descriptor-Link words that identify the MLI (Host system) making the connection, and contain the memory address of the IOCB that initiated the connection sequence.

5. A Longitudinal Parity Word used to verify that the preceding 3-words are valid connection sequence words.

An MLIP module I/O device initiation process begins when the Data Processor module executes a CUIO operator code, and transfers the starting memory address of an IOCB to the MLIP logic. The MLIP subsequently accesses words of the IOCB, and writes fields from the various IOCB words into the MLIP RAM memory. The MLIP then causes a connection sequence to be performed.

MLIP-To-IODC Connection Sequence Address Word

The first information required by the MLIP to perform a connection sequence is the particular MLI interface port through which the connection sequence is to take place. This information is present in word-4 [19:4] of the MLIP RAM memory, and was originally obtained from word-1 [19:4] of the IOCB. Bits [15:16] of MLIP RAM word-4 (also from IOCB word-1) contain the data required for the first MLI protocol connection sequence.

The data required for an MLI connection are:

1. Base Control Card bit (MLIP RAM word-4, bit-15).
2. Line Expansion Module bit (MLIP RAM word-4, bit-14).
3. Base Module IO bits (MLIP RAM word-4, bits [7:4]).
4. DLP IO (MLIP RAM word-4, bits [3:4]).

The MLIP micro-logic (MAKE.CON sequence) utilizes data from MLIP RAM word-4 to implement a connection between the MLIP and a particular DLP module, over the proper MLI Port and interface cable. The MLIP logic formats and transmits the DLP address to the IODC logic, as shown in Figure 11-6.

	A	A	A	A	B	B	B	B	C	C	C	C	D	D	D	D
P	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
P	BCC	LEM	U	U	U	U	U	U	B	B	B	B	DLP	DLP	DLP	DLP

Where:

P = Odd-Parity bit for other word-bits.
BCC = Base Control Card ID bit.
LEM = Line Expansion Module ID bit.
U = Unused bit-positions in the word.
B = IODC UIO-Base module ID code.
DLP = DLP ID code.

NOTE

The 8 control-signal lines of the MLI interface cable are not shown.

MV4564

Figure 11-6. B 6900 Connection Sequence Address Word Layout

MLIP-To-IODC Connection Sequence I/O Descriptor

The MLIP micro-logic (INIT.IOCB sequence) formats I/O Descriptors from a list of I/O Descriptor data present in system memory. The MLIP accesses the I/O Descriptor list in system memory by means of an address which is contained in the I/O Command Pointer (word 4 of the IOCB). The MLIP logic formats the I/O Descriptor found in system memory into 16-bit data words, and transmits them to the DLP device over the MLI interface. I/O Descriptors may consist of several consecutive 16-bit words on the MLI interface bus.

LPW Word For I/O Descriptor

A Longitudinal Parity Word is formatted and transmitted from the MLIP to the IODC and DLP device. This word represents a block-check of the data transmitted in the I/O Descriptor. The format of an LPW word is defined later in this section.

MLIP-To-IODC Connection Sequence Descriptor Link Words

The MLIP micro-logic (INIT.IOCB sequence) utilizes data from the MLIP RAM memory, word-7 and word-A, to create 2 Descriptor Link words. The 2 words are transmitted to the IODC logic during the MLI interface connection process, and are used subsequently by the IODC to reconnect to the MLIP during other sequences of the IO device operation process.

The format of the 2 Descriptor Link data words on the MLI interface are shown in Figure 11-7.

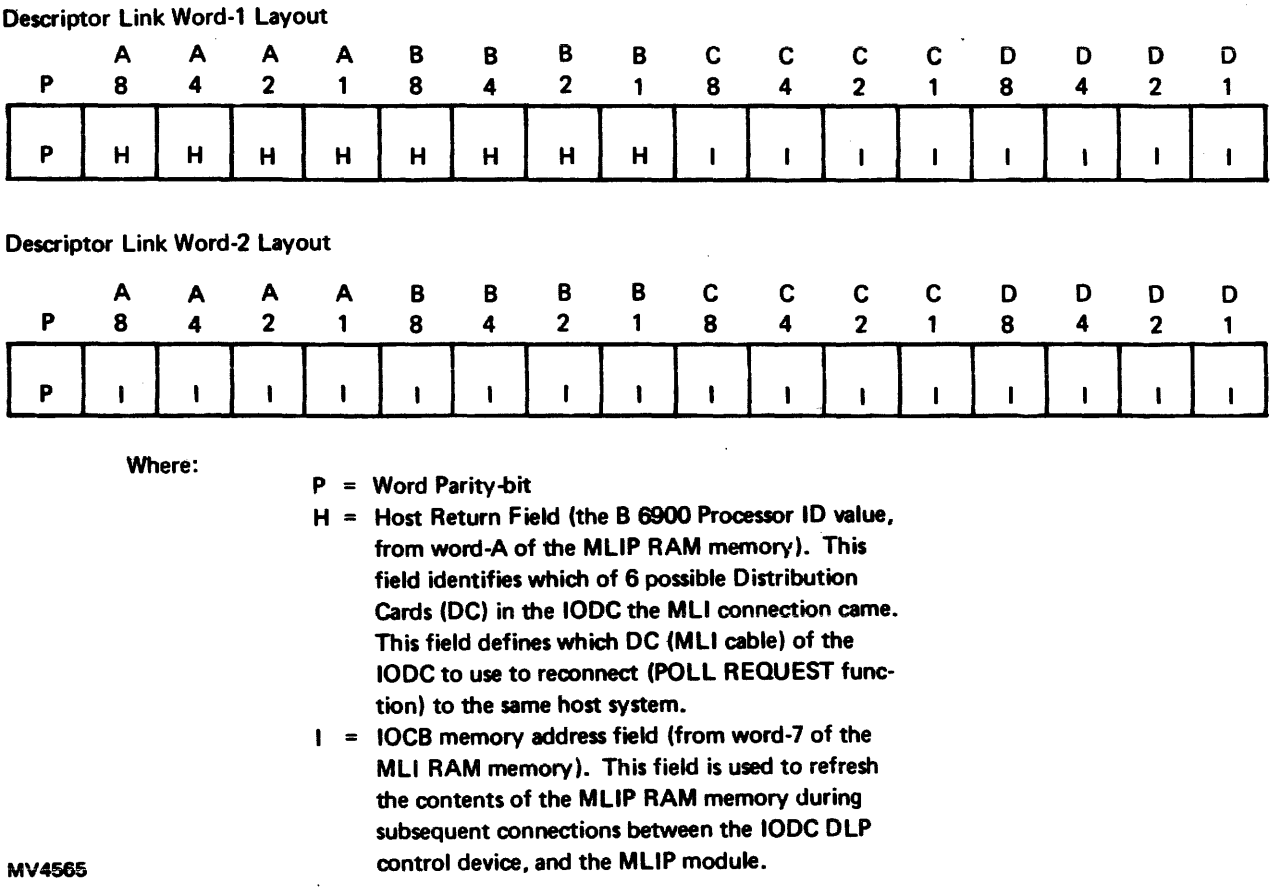


Figure 11-7. B 6900 Connection Sequence Descriptor Link Word Layouts

MLIP-To-IODC Connection Sequence LPW Word

The MLIP micro-logic generates and formats a Longitudinal Parity Word (LPW) for all serial data transmitted over an MLI interface. The LPW word is automatically transmitted to the IODC after the last serial dataword has been transmitted.

An LPW-word represents the combined longitudinal parity (Block-check value) for the datawords transmitted over the MLI interface during the present connection sequence. An LPW accounts over the MLI cable for only longitudinal parity of words transmitted in a single transmission burst, in a single direction. If line-turnarounds are performed during a connection sequence, an LPW-word is generated and transmitted each time a new burst of date is transmitted in either direction over the MLI interface cable.

The IODC logic contains circuits that generate and format an LPW-word as serial data-words are received over the MLI interface cable. When the LPW-word from the MLIP is received by the IODC logic, it is compared to the LPW-word generated by the IODC logic from preceding MLI serial datawords. If the 2 LPW-words do not compare equal, an MLI interface data parity-error is detected by the IODC logic.

Both the MLIP logic and the IODC logic generate and transmit LPW-words as part of their normal data transmission operations over an MLI interface. Both also generate an LPW-word for serial data received, and compare that LPW-word against the LPW-word received over the MLI interface cable. Thus, either the MLIP or the IODC can detect an MLI interface data parity-error condition.

IODC-To-MLIP Connection Sequence

After a POLL TEST connection sequence has been completed by the MLIP module, the MLI connection protocol provides for a disconnect to occur. A disconnect is necessary because the DLP device to which the MLIP is connected needs to execute one or more of its internal micro-code procedures at this point.

Whatever the cause of an MLI disconnect, once a connection has been completed the DLP can initiate the next connection sequence, because the IODC logic now contains Descriptor Link data. The Descriptor Link data identifies the proper DC (MLI cable) to be used for the second or subsequent connection sequence.

IODC-To-MLIP Connection Sequence Global Priority Word

During a POLL REQUEST operation connection sequence, the requesting DLP device must return a Global Priority word of data to the MLIP logic. This word is required because an IODC can contain 1-to-8 DLP devices, and the MLIP must know which DLP is initiating the connection sequence. The Global Priority word has the format shown in Figure 11-8.

	A	A	A	A	B	B	B	B	C	C	C	C	D	D	D	D
P	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
P	U	U	U	U	U	B	B	B	B	DLP	DLP	DLP	DLP	GP	GP	GP

Where:

P = Word Parity Bit
U = Unused, a field filled with zeroes
B = IODC Base module identity code
DLP = DLP identity code (relative location of the DLP device within the IODC Base module
GP = The Global Priority of the DLP that initiated the POLL REQUEST operation

MV4566

Figure 11-8. B 6900 IODC POLL REQUEST Global Priority Word Layout

IODC-To-MLIP POLL REQUEST Priority Resolution In The IODC

The Global Priority field contains the Global Priority of the DLP that initiated the POLL REQUEST. It is possible that more than one DLP device in an IODC Base module will require a POLL REQUEST operation at the same time. If such a condition occurs, the IODC module logic will resolve POLL REQUEST sequence conflicts between its DLP devices on the basis of their configured Global Priority. The DLP with the highest Global Priority performs a POLL REQUEST operation first, and other DLPs must wait until they have the highest Global Priority in their Base before they can initiate a POLL REQUEST operation.

IODC-To-MLIP POLL REQUEST Global Priority Resolution In The MLIP

The MLIP Priority Sequencer Logic samples the Global Priority values from POLL REQUESTs, over all 8 possible MLI interface ports, and in port number order. If two or more MLI interface ports contain simultaneious POLL REQUESTs, the Priority Sequencer logic selects the MLI port that has the highest Global Priority to initiate the next POLL REQUEST. If two or more MLI port POLL REQUESTs share the highest Global Priority request present, then the Priority Sequencer logic selects the MLI with the highest port number as the one for which the next POLL REQUEST operation is performed. Other MLI ports that are trying to initiate a POLL REQUEST operation must wait until they have the highest current priority before they are granted a POLL REQUEST sequence.

IOCB ORGANIZATION AND WORD LAYOUTS

The I/O device specifications placed in memory by the MCP are in a fixed-word format called an Input/Output Control Block (IOCB). An IOCB occupies 15 consecutive memory addresses to which other word addresses may be appended for software control purposes. Figure 11-9 shows the layout and identification of the first 15 words in an IOCB.

The first 15 words of an IOCB are defined in the following subsections. The order of definition in an IOCB is the same as the order of the words in system memory.

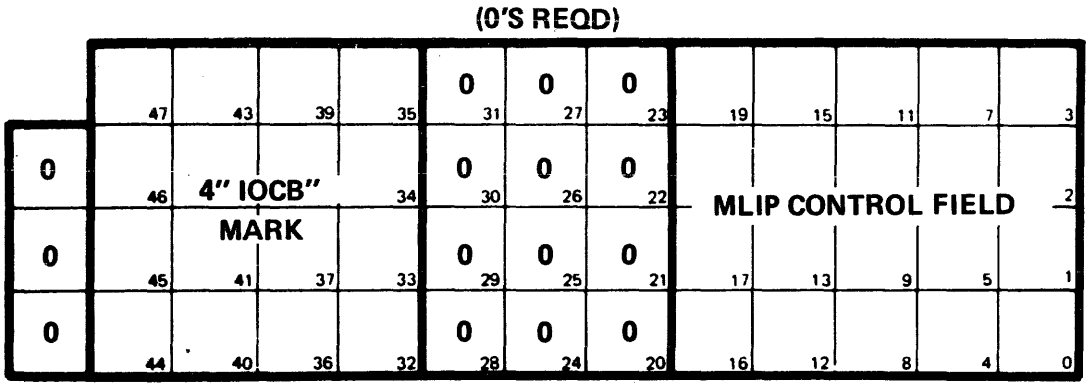
Word Number	Mnemonic	Word Contents
0	[CW]	Control Word
0	[LCPAW]	DLP Address Word
2	[CQHP]	Command Queue Header Pointer
3	[SELFP]	IOCB Self Pointer
4	[LCPCP]	DLP I/O Command Pointer
5	[LCPRP]	DLP I/O Result Pointer
6	[LCPCRL]	DLP Command/Result Lengths
7	[RM]	Result Mask
8	[RQHP]	Result Queue Head Pointer
9	[NL]	Next IOCB Link
A	[CDP]	MLIP Current Data Area Pointer
B	[CL]	MLIP Current I/O Length
C	[HRSLT]	MLIP State and Result
D	[STIME]	I/O Start Time
E	[FTIME]	I/O Finish Time
F ↓ ↓		F 10 11 12 IOCW (67/68 IOCW) 13 14 15 16 17

MV4193

Figure 11-9. IOCB Word Format and Layout

IOCB Control Word

Figure 11-10 shows an IOCB (word ZERO) Control Word. The IOCB Control Word (CW) is a formatted operand containing the 16-bit "IOCB" mark in bits [47:16], and the 20-bit MLIP control-field in bits [19:20]. The control-field defines the type of operation the MLIP is to perform.



NOTE

THE 4"IOCB" MARK IS DEFINED AS FOLLOWS:

	47 0	0	1	1	
	0	0	1	0	
	0	0	0	1	
	1	0	0	1 32	
VALUE =	I	O	C	B	(HEXADECIMAL)

MV4194

Figure 11-10. IOCB Control Word Layout

MLIP CONTROL-FIELD BIT DEFINITIONS

The meaning of each bit in the MLIP control-field is described below:

Bit 0 = Queue-at-Head. If this bit is SET during the execution of a CUIO operator, the IOCB (or the chain of IOCB's) will be inserted at the front of the Command Queue. The IOCB is inserted at the end of the Command Queue if bit 0 is RESET.

Bit 1 = MLIP/DLP-Command. If this bit is SET, the IOCB contains a command to be interpreted and executed by the logic of the MLIP (not by a UIO-DLP). If this bit is RESET, the MLIP directs the command to a UIO-DLP and utilizes information from other words in the IOCB to determine which UIO-DLP is to be addressed, and what kind of operation it has to perform.

Bit 2 = Attention. If this bit is SET, the MLIP will construct an MLIP Result Word which has both the Attention-bit and the Exception-bit SET at the conclusion of the I/O operation.

Bit 3 = Cause I/O Finish Interrupt. If this bit is SET, the MLIP unconditionally causes an I/O finish CPU interrupt (External Interrupt-I/O Finish) at the completion of the I/O operation. If this bit is RESET, the MLIP only causes the I/O Finish interrupt to occur when an I/O error-condition such as a I/O data-parity error is detected.

Bit 4 = Memory Override/Memory Protect. If this bit is SET, the MLIP ignores the tag of memory words during memory operations (Override). If this bit is RESET, the MLIP terminates data transfer to memory if it READS (or attempts to WRITE into) a memory address that has an odd-numbered TAG-FIELD value (Memory Protect).

Bit 5 = Input. If this bit is RESET and DLP goes to Read-Status (STC=4) the MLIP flags a DLP Status-Error and disallows I/O input-data transfers. This bit must be SET for all input operations (data transfers from the peripheral device to system memory). See definition of [6:2], below.

Bit 6 = Output. If this bit is RESET and DLP goes to Write-Status (STC=8) and this bit is RESET, the MLI flags a DLP Status-Error. See definition of [6:2], below.

If bits 5 and 6 are both SET, an Echo Command is performed by the DLP. If bits 5 and 6 are both RESET, any attempt to transfer data as input or output will cause a DLP Status-Error to be flagged and no data will be transferred.

Bit 7 = Output Zeros. If this bit is SET and the Output-bit also is SET, the MLIP sends to the DLP bytes of binary zeroes for the specified record-length. This type of data transfer has one valid , function to perform a magnetic-tape erase.

[10:3] = Tag Control. The value of this field defines how tags are handled during data transfer operations.

[10:3] = 001: Transfer Single Byte Tag. Tags are treated as one additional byte of data. During output the 3-bits of TAG are placed in the 3 least significant bit positions of the additional byte, and the most significant bit is RESET. During input the 3 least significant bits of the additional byte are transferred to the TAG-FIELD of the memory word, and the other bits of the additional byte are ignored.

[10:3] = 010: Transfer Double Byte Tag. Tags are treated as two additional bytes of data. During output the 3-bits of the word TAG are placed in the 3 least significant bits of the most significant byte, and the remaining 13 bits of the byte are RESET to ZERO. During input, the 3 least significant bits of the most significant byte are placed in the TAG-FIELD of the memory word, and the other 13 bits of the byte are ignored.

[10:3] = 100: Force Tags to Single (0). Memory word TAG-FIELDS are not treated as part of data, and are not transferred. During input, the TAG-FIELD of each memory word is unconditionally RESET. During output, memory TAG-FIELDs are skipped.

[10:3] = 110: Force Tags to Double (2). This value is valid only during input. It performs the same as Force Tags to Single (above) except the TAG of each word is SET to double (2). *skip 1
[10:3] = 111 :

Force Tag to Code (3). This value is valid only during input. It performs the same as Force Tags to Single (above) except the TAG of each word is SET to Code (3).

Bit 11: Word/Character Oriented Transfer. If this bit is SET, the amount of data to be transferred is counted in words, which must be accessed by means of a word Data-Descriptor. If this bit is RESET, the data to be transferred is counted in characters, which must be accessed by means of a string Data-Descriptor.

Bit 12: Memory Direction. If this bit is SET, the MLIP transfers data into memory in a reverse direction. If this bit is RESET, data is transferred into memory in a forward direction. If this bit and the Output-bit are both SET, an Invalid MLIP Control Field error is returned and no data is transferred.

Bit 13: Continue Count at End of Length. When this bit is SET and the Input-bit also is SET, the MLIP does not terminate the transfer of data between the DLP and MLIP when the LENGTH count reaches ZERO. Instead, the DLP continues transferring data to the MLIP; but the MLIP does not store the additional data in memory. The I/O length continues to be counted and the value used to determine the actual record length. If this bit and the Output-bit are both SET, an Invalid MLIP Control Field error is returned and no data is transferred.

Bit 14: Ignore Count Error. If this bit is SET, the MLIP will SET the Count-Error bit when the LENGTH Count at the end of the I/O operation is equal to zero. If this bit is RESET the MLIP will not SET the Count-Error bit because of LENGTH being equal to zero at the end of the I/O operation.

Bit 15: Dont Count. If this bit is SET when this IOCB is initiated/completed, the MLIP logic will not increment or decrement the ACTIVE-count field in the Command Queue CW word. This bit is used during CANCEL type I/O command operations. The purpose of an I/O CNACEL operation is to terminate a currently executing I/O operation. An I/O operation that is CANCELed decrements the ACTIVE-count field in its Command Queue CW word, when its IOCB completes.

Bit 16: Ignore Suspend All Queues. If this bit is SET, the MLIP will not suspend the Command Queue IOCBs when adding an IOCB to the Result Queue, regardless of the setting of the Suspend-All-Queues flag.

17: Immediate. If this bit is SET and the MLIP is pointing at the first IOCB in a queue, the MLIP attempts to initiate the IOCB regardless of the ACTIVE-COUNT and LIMIT field values in the Command Queue CW word. Initiating an IOCB which has its Immediate-bit SET does not increment the ACTIVE-COUNT value of the Command Queue CW word. If the UIO-DLP is BUSY, then the MLIP attempts to add this IOCB to a Horizontal Queue.

[19:2] : This field is not used and must contain ZEROES.

VALID CONTROL-FIELD BIT CONFIGURATIONS

Figure 11-11 identifies valid Control-Field bit configurations in an IOCB (CW) word. A valid CW Control-Field bit configuration is used to cause a particular type of I/O Command to be initiated by a UIO-DLP. The CW control-field is also used to cause the MLIP to perform a function that is internal to the MLIP, and does not involve the use of an MLI interface or a UIO-DLP device. A UIO-DLP (Data Link Processor) is the final control for a B 6900 system peripheral Input/Output device. Control-Field bit configurations originate in an IOCB in system memory and are passed through the MLIP module and an MLI interface cable to a UIO-DLP that is located in a B 6900 system IODC module cabinet.

BIT NO.	17 IMMEDIATE	16 IGNORE SUSPEND ALL QUEUES	15 DONT COUNT	14 IGNORE COUNT ERROR	13 CONTINUE COUNT	12 DIRECTION	11 WORD ORIENTED	10 TAG CONTROL	9	8 OUTPUT ZEROS	7 OUTPUT	6 INPUT	5 MEMORY OVERRIDE	4 CAUSE I/O FINISH	3 ATTENTION	2 MLIP/DLP	1 QUEUE AT HEAD	0
MLIP OP	X	X	X	0	0	0	X	1	0	0	0	0	0	0	X	X	1	X
TEST	X	X	X	0	0	0	X	1	0	0	0	0	0	0	X	X	0	X
INPUT	X	X	0	X	X	0	1	0	0	1	0	0	1	1	X	X	0	X
	X	X	0	X	X	0	1	0	1	0	0	0	1	1	X	X	0	X
	X	X	0	X	X	X	X	1	0	0	0	0	1	X	X	X	0	X
	X	X	0	X	X	X	X	1	1	0	0	0	1	X	X	X	0	X
	X	X	0	X	X	X	X	1	1	1	0	0	1	X	X	X	0	X
	X																	
OUTPUT	X	X	0	X	0	0	1	0	0	1	0	1	0	1	X	X	0	X
	X	X	0	X	0	0	1	0	1	0	0	1	0	1	X	X	0	X
	X	X	0	X	0	0	X	1	0	0	0	1	0	X	X	X	0	X
	X	X	0	X	0	0	X	1	0	0	1	1	0	X	X	X	0	X
	X																	
ECHO	X	X	0	X	0	0	X	1	0	0	0	1	1	X	X	X	0	X

NOTE

1 = THE BIT MUST BE SET TO A BINARY 1.

0 = THE BIT MUST BE RESET TO A BINARY 0.

X = THE BIT MAY BE EITHER SET OR RESET.

MV4195

Figure 11-11. Valid Commands in CW Control-Field

The MLIP OP operation specified in Figure 11-11 identifies a function of the MLIP logic that does not use an MLI interface or address a UIO-DLP. The MLIP OP functions performed by an MLIP are specified in detail within the paragraphs entitled MLIP Commands. The difference between an MLIP OP function and a UIO-DLP function is the code contained in the Command specification, which is pointed at by DLPCP (word 4) of the IOCB. Bit-1 of the CW Control-Field is the key value used to select an MLIP operation or a UIO-DLP operation.

IOCB DLP Address Word

Figure 11-12 shows an IOCB (word one) DLP Address word. The DLP Address word (DLPAW) is a formatted operand which contains the address environment of a UIO-DLP. The MLIP utilizes the DLP address data to connect to the DLP specified over an MLI interface.

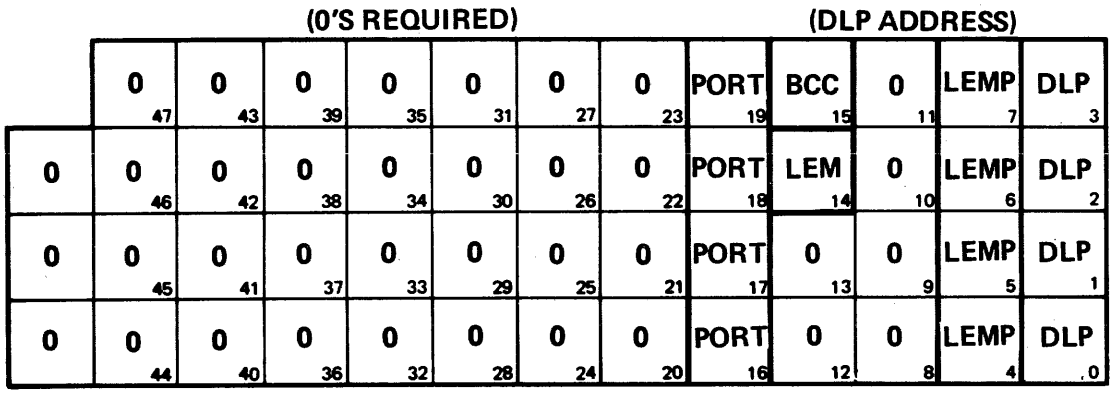


Figure 11-12. IOCB DLP Address Word Layout

DLP ADDRESS WORD FIELD AND BIT DEFINITIONS

The definition of the fields in an IOCB DLP Address are as follows:

Field	Bits	Meaning or Usage
Port	[19:4]	Selects the MLI port (0-7, HEX) to be used by the MLIP for this command.
BCC	[15:1]	If SET, the command is directed to the Base Control Card (BCC) of the IODC.
LEM	[14:1]	If SET, the command is directed to the Line Expansion Module (LEM) of the IODC selected by LEMP [07:4].
LEMP	[07:4]	Selects the IODC module (0-7,HEX), if LEM, [14:1], is SET.
DLP	[03:4]	If BCC and LEM are RESET, this field selects the DLP location (0-7,HEX) in the IODC module addressed by [19:4].

Command Queue Header Pointer Word

Figure 11-13 shows an IOCB (word 2) Command Queue Header Pointer. A Command Queue Header Pointer (CQHP) is an unsegmented, unindexed word Data Descriptor that points at the Command Queue Header. The MCP initializes this word, and its values are not changed during the operation of the I/O device.

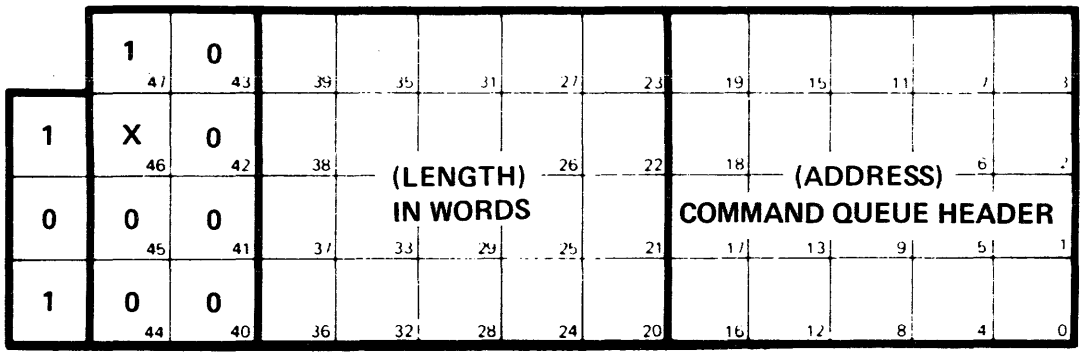


Figure 11-13. IOCB Command Queue Header Pointer Word Layout

IOCB Self Pointer Word

Figure 11-14 shows an IOCB (word 3) IOCB Self Pointer word layout. A Self Pointer (SELFP) is a present, unsegmented, unindexed word Data Descriptor. The Self Pointer points at the first word (word zero) of the IOCB in which the Self Pointer is located. Self Pointers are used to link the IOCB (of which they are part) into a Command Queue or into a Result Queue in system memory. The use of Self Pointers allows an IOCB to remain in a fixed memory address, and to be associated with other IOCBs by means of a series of Next Link Pointers to all IOCBs that are linked together. Linking an IOCB into a queue involves copying the Self Pointer into the Next IOCB Link (word 9) of the previous IOCB in the queue. The value fields of a Self Pointer are never changed, because the address of the IOCB in memory remains constant throughout an I/O operation.

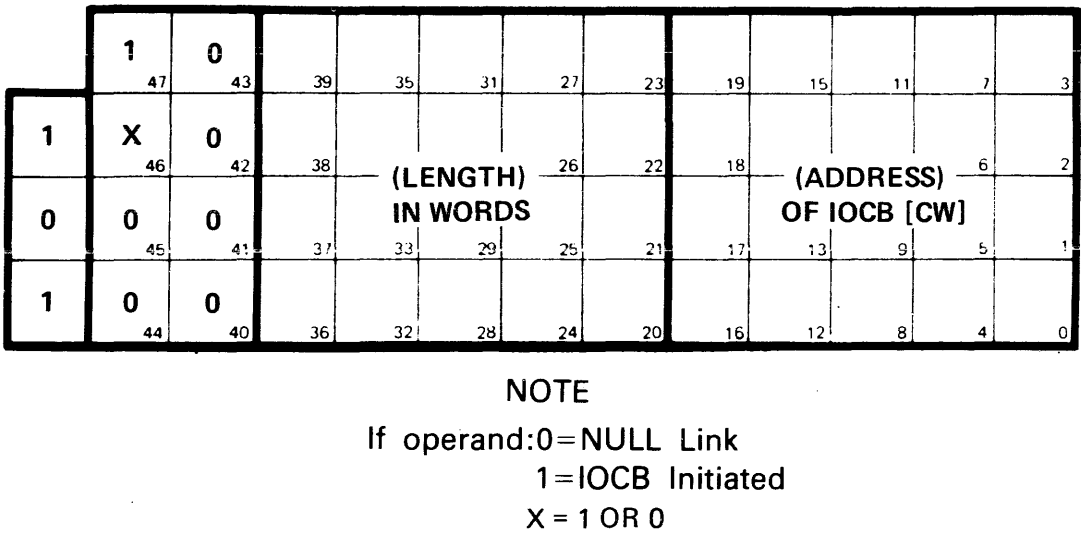


Figure 11-14. IOCB Self Pointer Word Layout

IOCB DLP Command Pointer Word

Figure 11-15 shows an IOCB (word 4) DLP Command Pointer word. A DLP Command Pointer (DLPCP) is an unsegmented word Data Descriptor that is present. The Descriptor points at the actual DLP Command Descriptor address in system memory. The values of the DLP Command Pointer are established by the MCP before the I/O operation is initiated, and they remain unchanged during the entire I/O operation sequence. Refer to the description of the IOCB Control Word (IOCB word ZERO) defined in the previous paragraph.

NOTE

The layout of I/O Command Descriptors for various types of DLP devices are given in the B 6900 Pocket Reference, Form No. 5011497.

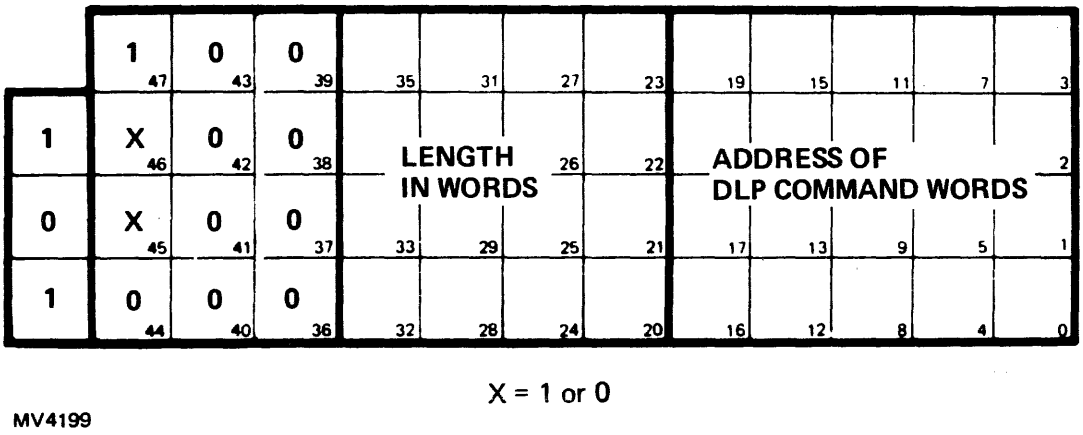


Figure 11-15. IOCB DLP Command Pointer Word Layout

IOCB DLP Result Pointer Word

Figure 11-16 shows an IOCB (word 5) DLP Result Pointer word. The DLP Result Pointer (DLPRP) word is an unsegmented word Data Descriptor which is present and which points to the address in memory where the MLIP is to store the DLP Result Descriptor at the conclusion of the I/O device operation. This pointer is initiated by the MCP before the start of the I/O device operation, and remains unchanged during the operation sequence.

NOTE

The layout of I/O Result Descriptors for various types of DLP devices are given in the B 6900 Pocket Reference, Form Number 5011497.

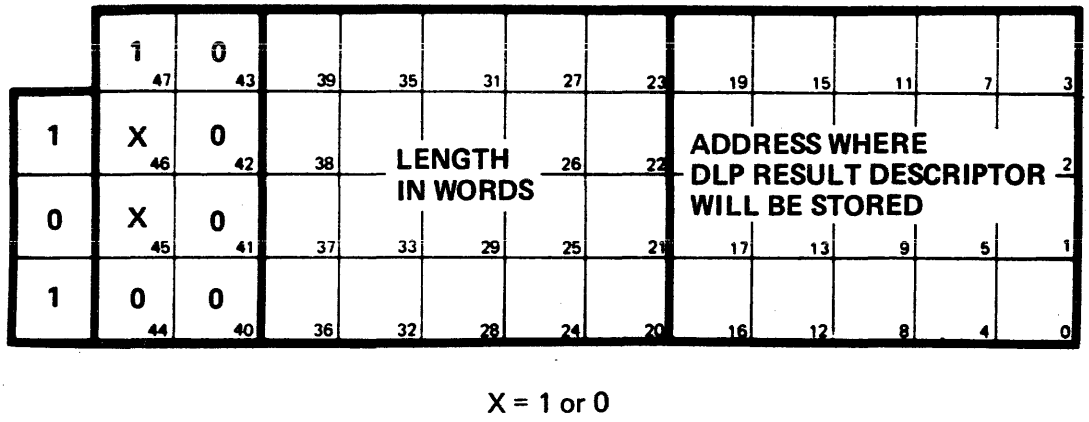


Figure 11-16. IOCB DLP Result Pointer Word Layout

IOCB DLP Command/Result Length Word

Figure 11-17 shows an IOCB (word 6) DLP Command/Result Length word. The DLP Command/Result Length word (DLPCRL) is a formatted operand that contains the length of the DLP Command and I/O result status, in 16-bit bytes. Both length values in the DLP Command/Result Length word must be an even number of 16-bit byte increments. The values of this word are used to determine the maximum number of bytes to be transferred between the MLIP and the IODC module, over the MLI interface. If the maximum number of bytes for transfer is exceeded, an Unexpected DLP Status error is reported to the Interrupt Controller. This word is initialized by the MCP before the I/O operation is started, and remains unchanged during the execution of the I/O operation.

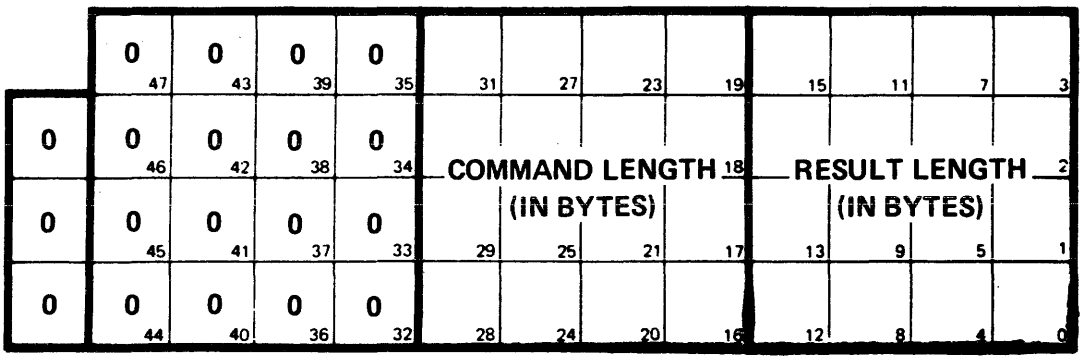


Figure 11-17. IOCB DLP Command/Result Length Word Layout

IOCB Result Mask Word

Figure 11-18 shows an IOCB (word 7) Result Mask word. The Result Mask word (RM) is a formatted operand that is used to limit the conditions of I/O device operation that can cause an exception or error to be sensed by the Interrupt Controller Logic. Bits [47:48] of the Mask word are ANDed with corresponding bits from the MLIP State And Result word (IOCB word C). An interrupt is sensed if corresponding bits in the Mask and the MLIP State And Result word are both SET. An interrupt is not sensed if either or both corresponding bits are RESET. When the MCP generates the contents of the IOCB, it determines which bits are SET in the Result Mask word and which of the exceptions present in the MLIP Result And Status word can cause an interrupt to be sensed.

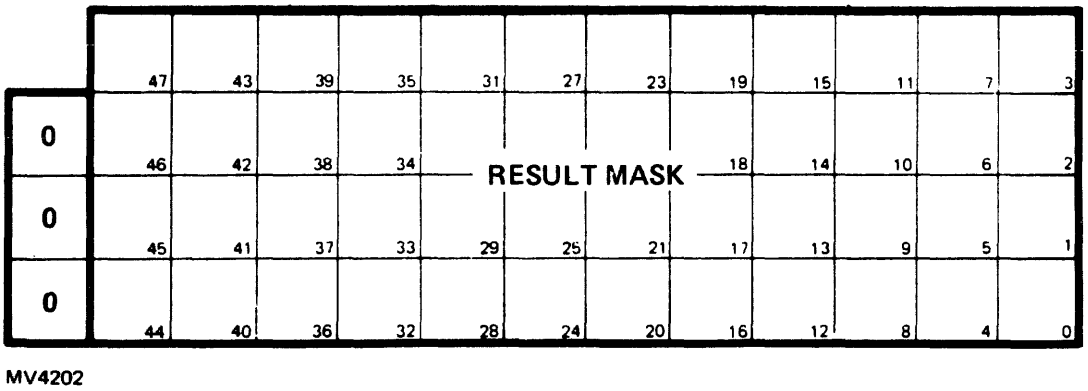


Figure 11-18. IOCB Result Mask Word Layout

IOCB Result Queue Head Pointer Word

Figure 11-19 shows the IOCB (word 8) Result Queue Head Pointer. The Result Queue Head Pointer (RQHP) is an unsegmented, indexed word data descriptor which is present and which points at a Result Queue Head word in the Result Queue array of system memory. When an I/O operation sequence is completed, the IOCB for that I/O operation is de-linked from the Command Queue and linked into a Result Queue. The IOCB is always linked at the tail of the Result Queue. The Result Queue Head Pointer points at the head of the Result Queue, and from this reference address the IOCB is linked into the Result Queue at the tail of the queue.

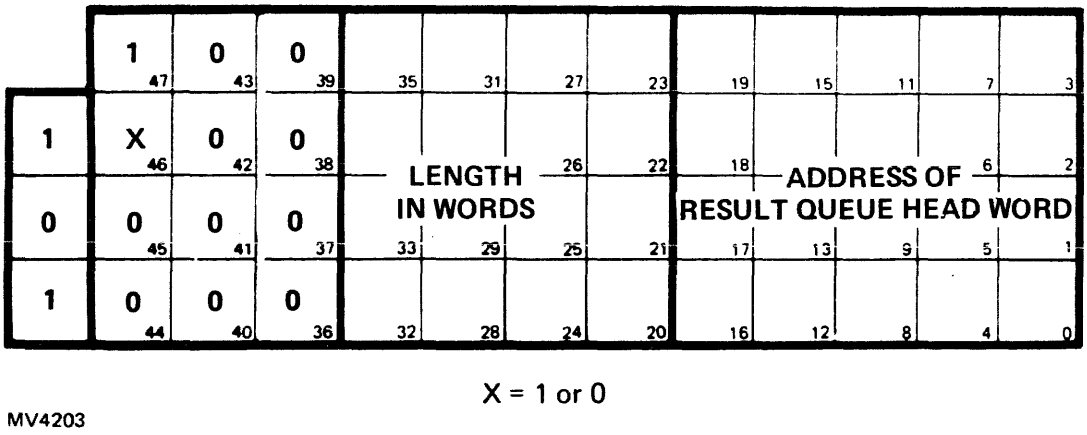


Figure 11-19. IOCB Result Queue Head Pointer Word Layout

IOCB Next IOCB Link Word

Figure 11-20 shows the Next IOCB Link Word (word 9). The Next Link word (NL) is a present, unsegmented, unindexed word Data Descriptor that points at the first word of the next IOCB in squence (following the IOCB of which this Next Link word is a part) in the queue. The MCP initializes the Next Link word before the I/O operation sequence is started.

The MLIP changes the value of a Next IOCB Link word during various parts of the I/O device sequences. When a connection is made over the MLI, to initialize the operation of the UIO-DLP, the MLIP replaces the Next Link word with an integer operand of one (1). This operand is used to show that the I/O operation is in process.

If an IOCB is the last IOCB in the queue of IOCBs, the MLIP replaces the Next IOCB Link word with an integer operand value of zero (0). There is no next IOCB to be linked in, thus the zero operand shows that this is the last IOCB in the queue. If a subsequent IOCB is linked into the queue at the tail, it becomes the last IOCB. The Self Pointer of this subsequent IOCB is written in the Next Link word of the previous IOCB, overwriting the integer operand of zero, thus extending the IOCB linkage to include the new tail IOCB. The Next Link word of the new tail IOCB is replaced by integer value zero, to mark it as the last IOCB in the queue.

If an IOCB is ENQUEUEd at the head of the queue, the MLIP replaces the value of the Next IOCB Link word with the IOCB Self Pointer (word 3) of the original queue head IOCB. Consequently, the newly enqueued IOCB is at the head of the queue, and its Next IOCB Link word points at the IOCB originally at the head of the queue. The MLIP must also adjust the address of the Head IOCB Link word in the Command Queue (word 1) if a new IOCB is ENQUEUEd at the head of the queue.

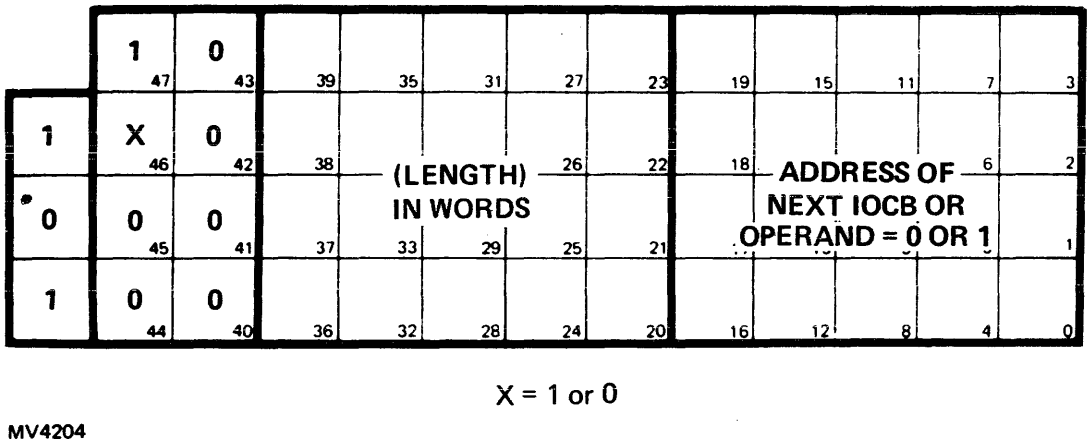


Figure 11-20. IOCB Next IOCB Link Word Layout

IOCB Current Data Area Pointer Word

Figure 11-21 shows the Current Data Area Pointer (word 10). The Current Data Area Pointer (CDP) is a present, unsegmented, string or word Data Descriptor which points to where data transfer begins or resumes. If CDP is a string Data Descriptor, then the SIZE FIELD must indicate 8-bit character format (the only data format used in the B 6900 system). The type of Data Descriptor used for CDP must agree with the word/character oriented transfer-bit (bit-11 of word zero) field of the IOCB CW word. The CDP word of the IOCB is initialized by the MCP before the CUIO operator is executed, and is updated by the MLIP as each block of data is transferred between the UIO-DLP and system memory. For reverse data transfer operations, the CDP must be set to indicate the last word or character of the data, plus 1.

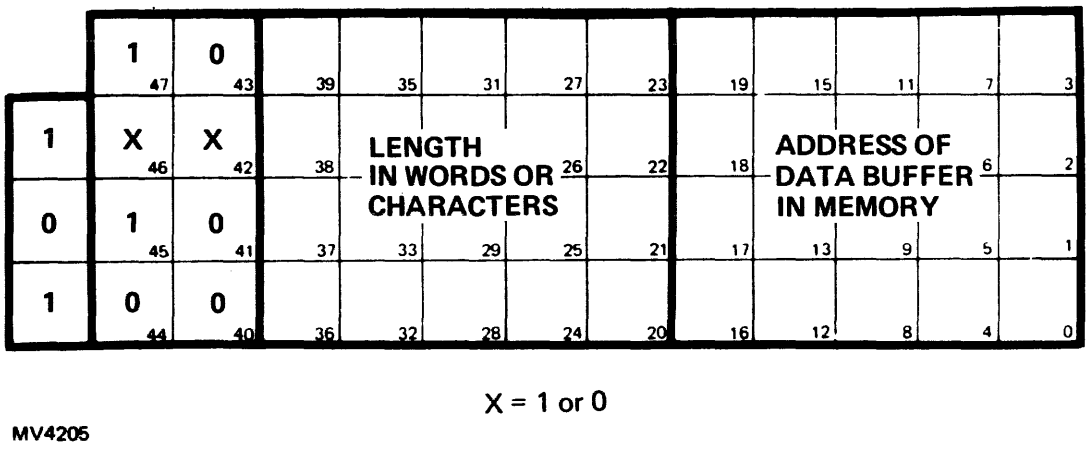


Figure 11-21. IOCB MLIP Current Data Area Pointer Word Layout

IOCB MLIP Current I/O Length Word

Figure 11-22 shows the MLIP Current I/O Length word (word 11). The MLIP Current I/O Length word (CL) is a formatted operand that contains an integer length count value in bits [19:20]. The state of bit-46 (Sign Of Mantissa) is also significant in the CL word. The value of the length count represents the amount of I/O data yet to be transferred during this I/O operation.

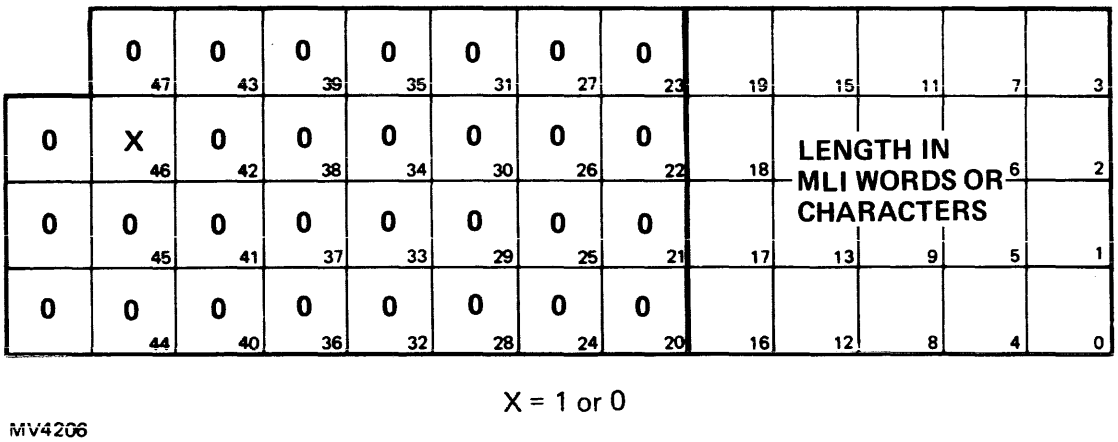


Figure 11-22. IOCB MLIP Current I/O Length Word Layout

If this I/O operation is a word-oriented data transfer, then the integer value in the CL word represents the number of 16-bit bytes of data yet to be transferred over the MLI interface. If this I/O operation is a character-oriented data transfer, then the integer value in the CL word represents the number of 8-bit bytes yet to be transferred over the MLI interface.

The integer value in the CL word can be a negative value (bit-46 = 1) if the Continue-Count-At-End-Of-Length bit in the CW word is SET (see the definition of bit-13 for word ZERO of the IOCB).

IOCB MLIP State and Result Word

Figure 11-23 shows the MLIP State And Result Word (word 11). The MLIP State And Result word (HRSLT) is a formatted operand that contains the micro-module state and MLIP status report for an I/O operation. The MCP initializes the HRSLT word in the IOCB to all-zeroes. The MLIP logic of the micro-module causes the fields of the HRSLT word to be initialized at the beginning of an I/O operation and updates the fields of the HRSLT word each time the IOCB is accessed. At the conclusion of an I/O operation, the values of the fields in the HRSLT word represent the accumulated status information from all of the hardware circuits and modules used during the entire I/O operation.

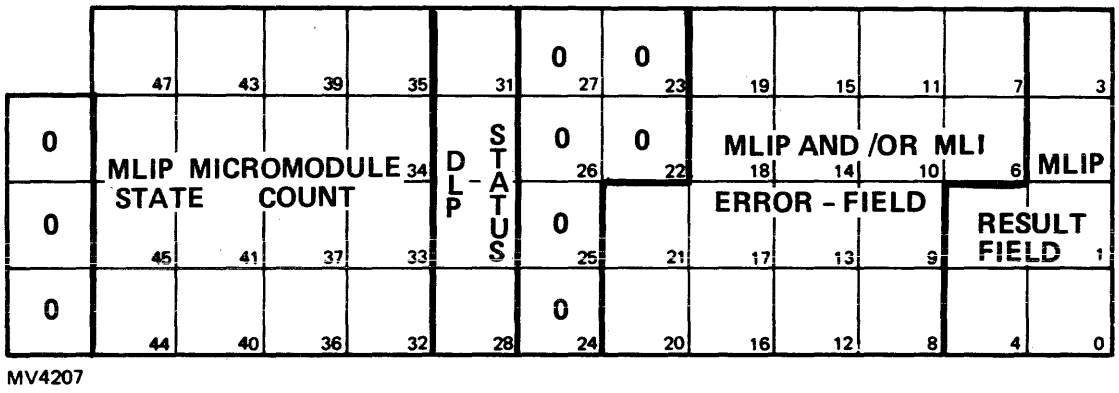


Figure 11-23. IOCB MLIP State and Result Word Layout

STATE AND RESULT WORD BIT AND FIELD DEFINITIONS

Following are the field/bit definitions for the IOCB State and Result word:

Bits [47:16]: MLIP State. This field contains the current micro-module address value. The MLIP must remember the micro-module address in case an error occurs in the micro-module while it is performing an MLIP sequence. The micro-module address is reported in the P-3 Interrupt Parameter for all interrupts originating in the MLIP module.

The micro-module logic for the MLIP updates the State Count field in the State And Result word of the IOCB each time the IOCB is accessed during an I/O operation.

Bits [31:32]: MLIP Result. This field contains result status from the DLP, MLIP/MLI, and MLIP operations. The following is a list of all bits in this field and their usage:

Bits [31:4]: DLP Status Field. If bit-16 is SET, this field contains the DLP status returned at the time of the error.

Bits [27:6]: This field must be zero.

Bits [21:16]: MLIP/MLI Error Field. The meaning or usage of each bit in this field is described below.

Bit 21: Invalid MLIP Command. If this bit is SET, it indicates a wrong combination of bits SET in the MLIP Command field of the IOCB Control Word.

Bit 20: MLI Time-out. If this bit is SET, it means the MLIP timed-out during an operation using the MLI interface. Either a DLP strobe was not returned to the MLIP within 8-milliseconds, or the UIO-DLP was busy for more than 2-seconds.

Bit 19: MLIP hung the DLP. If this bit is SET, it means the MLIP attempted to hang the DLP in response to a DLP-error.

Bit 18: DLP Busy. If this bit is SET, it means the MLIP, while attempting to connect to this DLP, has found the DLP busy and cannot place the IOCB in a horizontal queue.

Bit 17: Non-present DLP. If this bit is SET, it means the MLIP attempted to connect to a DLP that is not present.

Bit 16: Unexpected DLP Status. If this bit is SET, it means the DLP presented the MLIP with a status other than the one expected. In this condition the MLIP disconnects and leaves the DLP hung.

Bit 15: MLI LPW Error. If this bit is SET, it indicates the MLIP has detected incorrect longitudinal parity on the MLI interface.

Bit 14: MLI Vertical Parity Error. If this bit is SET, it indicates a parity-error detected on a word being transferred over the MLI interface.

Bit 13: Invalid MLIP Control Field. If this bit is SET, it indicates a wrong combination of bits in the MLIP Control field; that is, TAG-TRANSFER with character-oriented I/O.

Bit 12: Improper IOCB Word. If this bit is SET, it indicates an error in the format of an IOCB word.

Bits [11:4]: IOCB Index. The bits on in this field point to the incorrect word in the IOCB, when Bit-12 is SET. For example, Bits [11:4] = 9 indicates the next IOCB Link word (word number nine) containing an error.

Bit 7: Count Error. If this bit is SET, it means the Input or Output bit in the MLIP Control field was SET and the Current Length bit was not equal to ZERO at the end of the I/O. This bit will not be SET if the Ignore Count Error bit is SET in the MLIP Control field.

Bit 6: Memory Protect. If this bit is SET, it indicates an attempt to transfer a word to the DLP with an odd TAG, or to Overwrite a word with an odd TAG on input when the Memory Override bit is RESET in the MLIP Control field.

Bits [5:6]: MLIP Result Field. This field contains the overall results of an MLIP operation. The definition of each bit follows.

Bit 5: Completed After Queue Suspended. If this bit is SET, it means the I/O finished while the Command Queue was marked as suspended.

Bit 4: MLIP/Hardware Error. If this bit is SET, it means the MLIP detected an error and the parameters and error information will be reported through the Error IOCB.

Bit 3: MLIP/MLI Error. If this bit is SET, it indicates a bit SET in the MLIP/MLI Error field [21:16].

Bit 2: DLP error. If this bit is SET, it means a bit in the first 48 bits of the DLP result descriptor is on after ANDing the DLP result descriptor with the Result Mask.

Bit 1: Attention. If this bit is SET, it means the Software Attention was SET in the MLIP Control field.

Bit 0: Exception. If this bit is SET, it means that another bit in the MLIP Result field [5:5] is SET.

IOCB I/O Start Time Word

Figure 11-24 shows the I/O Start Time word (word 13). The I/O Start Time word (STIME) is a formatted integer value word that contains a Time-Of-Day value. The MLIP logic causes the Time-Of-Day to be SET in this word when the I/O device is initiated into operation. The Time-Of-Day value indicates the 24-hour clock time at the beginning of the I/O operation, in 2.4 microseconds increments.

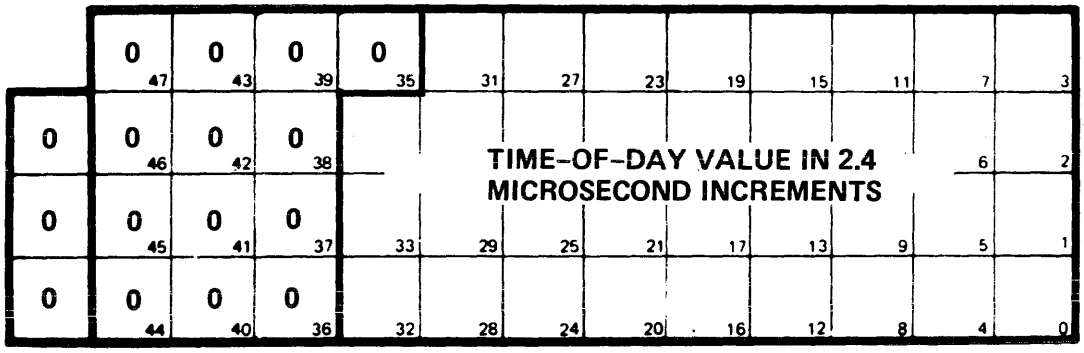


Figure 11-24. IOCB I/O Start Time Word Layout

IOCB I/O Finish Time Word

Figure 11-25 shows the I/O Finish Time word (word 14). The I/O Finish Time word (FTIME) is similar to I/O Start Time word (STIME), except that where I/O Start clocks the initialization of an I/O operation, I/O Finish Time clocks the termination of an I/O operation. The Time-Of-Day value for FTIME has the same accuracy as that for STIME.

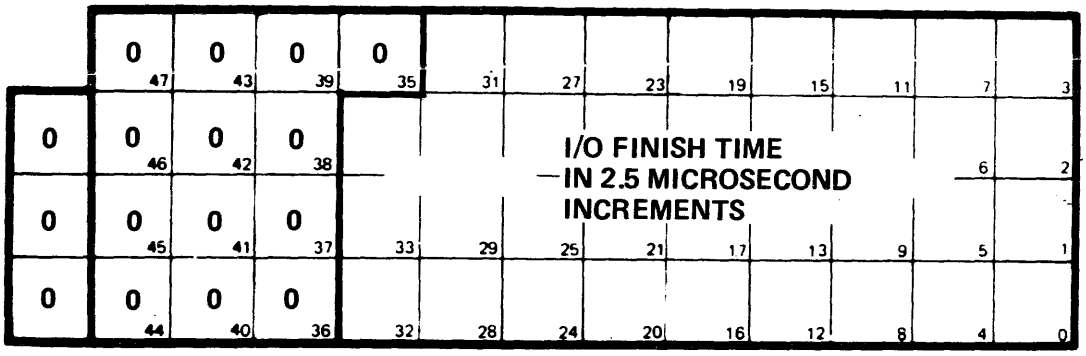


Figure 11-25. IOCB Finish Time Word Layout

COMMAND QUEUE ORGANIZATION AND WORD LAYOUTS

A Command Queue is an organization of IOCBs that are scheduled to be executed, but have not yet been initiated. The location of a Command Queue in system memory is an MCP software control function that is specified in CQHP (word 2) of the IOCB when control of the IOCB is passed to the MLIP by execution of the CUIO operator.

Manipulation of the contents of a Command Queue is a function of the logic for the MLIP. While an IOCB is ENQUEUEd in a Command Queue, the B 6900 system software does not touch the IOCB. When the I/O operation specified by an IOCB is terminated, the MLIP links the IOCB into the Result Queue specified in RQHP (word 8) of the IOCB. Manipulation of the contents of a Result Queue is a function of the system software. While an I/O operation is in process, the IOCB for the operation is not part of a queue in memory; it is an independent area that is referenced by an address in the UIO-DLP.

Figure 11-26 shows the word organization of a Command Queue. Each of the words that are present in the Command Queue are defined in subsequent paragraphs.

Word	Mnemonic	Word Contents
0	[CW]	Control Word
1	[HEAD]	HEAD IOCB LINK
2	[TAIL]	TAIL IOCB LINK
3	[HQHP]	HORIZONTAL QUEUE HEAD POINTER
4	[HQL]	HORIZONTAL QUEUE LINK

Figure 11-26. Command Queue Word Format and Layout

Command Queue Control Word

The Command Queue Control Word (CW) is a formatted operand that is used to identify the first word in a Command Queue. The CW contains data used by the MLIP logic to monitor and control its activity with the IOCBs that are linked into this particular Command Queue. Figure 11-27 shows the Command Queue Control Word (word zero).

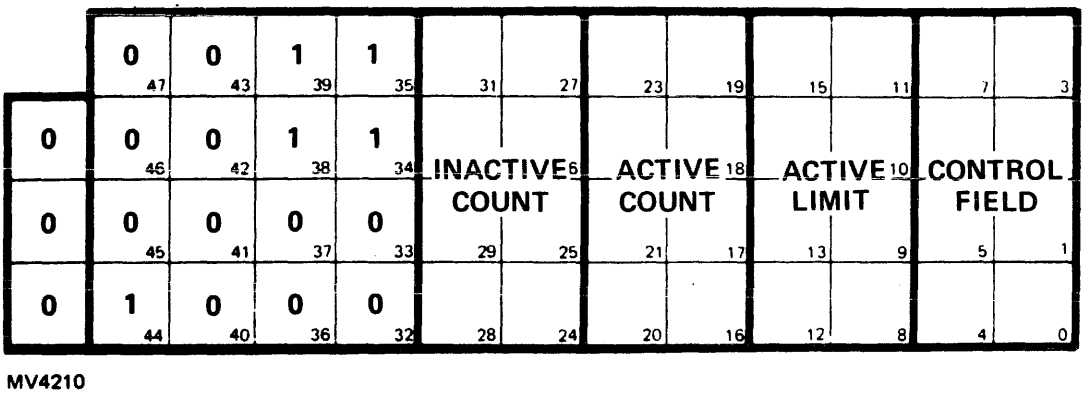


Figure 11-27. Command Queue Control Word Layout

The B 6900 system software control program (MCP) initializes the monitor and control values of the Control Word, when it establishes the Command Queue in system memory. The logic of the MLIP normally updates the values of the Control Word when an IOCB is linked into the queue, and when an IOCB is delinked from the queue (see exceptions specified by bits-15/16 of IOCB word zero, CW).

COMMAND QUEUE CONTROL WORD BIT DEFINITIONS

The meaning of the fields in a Command Queue Control Word are as follows:

Bits [47:16]: Command Queue Header Mark. This field contains 4"10CC", and is used by the MLIP to identify this word as the first word of a Command Queue (see the Mark explanation NOTE for word zero [CW] of the IOCB word layout, in Figure 11-10).

Bits [31:8]: Inactive Count. This field contains the number of IOCB's that are currently linked into this Command Queue, but have not yet been activated by the MLIP.

Bits [24:8]: Active Count. This field contains the number of IOCB's that are linked into this Command Queue, and are currently being processed by the MLIP.

Bits [16:8]: Active Limit. This field contains a non-zero value (initialized by the MCP, and not changed by the MLIP) which is the maximum number of IOCB's that can be normally active in this Command Queue at any given time. The Immediate-bit of an IOCB (word zero) CW is the basis for an exception to the normal Command Queue Limiting procedures.

Bits [7:8]: Control Field Bits

Bits [7:5]: These bits must be ZERO

Bit 2: Horizontal Queue Present. If this bit is Set, it means this Command Queue can be linked as a Horizontal Queue. This bit is used in a manner consistent with the definition of a Horizontal Queue Head Pointer (word 3, HQHP, below).

Bit 1: Waiting. If this bit is SET, it means this Command Queue has been dynamically linked into a Horizontal Queue.

Bit 0: Suspended. If this bit is SET, the MLIP only initiates an IOCB linked into this Command Queue if its Immediate-bit is SET. The Suspended-bit is SET by the MLIP logic when an I/O in this Command Queue finishes with an error, or when the Global MLIP Suspend-All-Queues flag is SET and the Ignore-Suspend-All-Queues flag of an IOCB in this Command Queue is RESET.

Command Queue Head IOCB Link Word

Figure 11-28 shows the Command Queue Head IOCB Link word (word 1). The Command Queue Head Link word (HEAD) is initialized by the MCP when the Command Queue is formed, as a formatted operand with all bits equal to ZERO.

When an IOCB is linked into this Command Queue, the logic of the MLIP causes the initial formatted operand (left by the MCP) to be replaced by a present, unsegmented, unindexed word Data Descriptor pointing to the first IOCB in the Command Queue. The Head Link word always points at the first IOCB in the Command Queue; therefore, if a subsequent IOCB is ENQUEUEd at the head of this Command Queue, the Head Link word is replaced by a new Head IOCB Link word which points at the new first IOCB in the Command Queue.

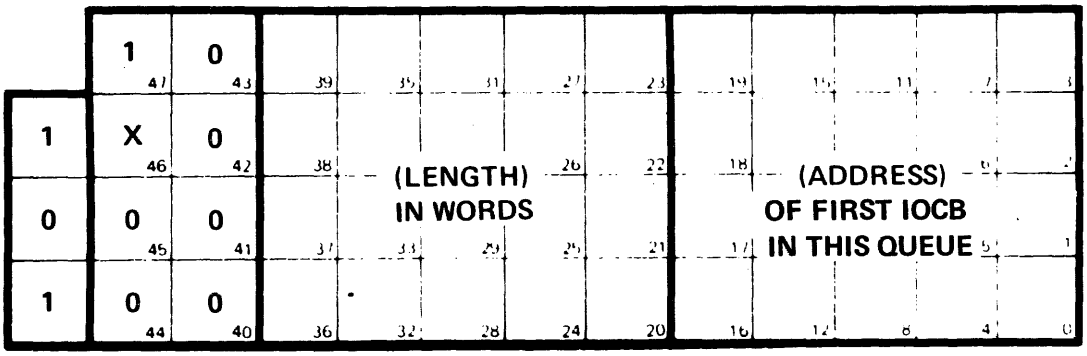


Figure 11-28. Command Queue Head IOCB Link Word Layout

Command Queue Tail IOCB Link Word

Figure 11-29 shows the Command Queue Tail IOCB Link word (word 2). The Tail IOCB Link word (TAIL) is initialized by the MCP as a formatted operand with all bits SET to zeroes. When an IOCB is ENQUEUEd in this Command Queue, the logic of the MLIP replaces the formatted operand word (left by the MCP) with a present, unsegmented, unindexed word Data Descriptor that points at the last IOCB in the Command Queue. If a subsequent IOCB is ENQUEUEd at the tail of the Command Queue, the MLIP replaces the Tail IOCB Link word with a new link word that points at the last IOCB, newly ENQUEUEd in the Command Queue.

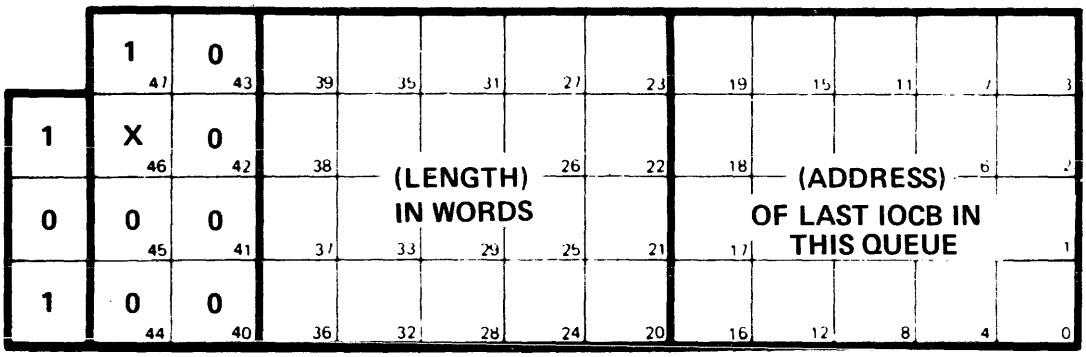


Figure 11-29. Command Queue Tail IOCB Link Word Layout

Command Queue Horizontal Queue Head Pointer Word

Figure 11-30 shows the Command Queue Horizontal Queue Head Pointer (word 3) word. The Horizontal Queue Head Pointer (HQHP) word is initialized by the MCP as either a formatted operand (with all bits zeroes), or as a present, unsegmented, indexed word Data Descriptor.

If the Horizontal Queue Head Pointer word is a formatted operand then this Command Queue cannot be horizontally queued. A Horizontal Queue Head Pointer word is never changed after initialization. This status is established by the MCP and is never changed, regardless of any capability of the Command Queue to be horizontally queued.

If the Horizontal Queue Head Pointer is a word Data Descriptor, it points to a Horizontal Queue Head in the Horizontal Queue Array. This status marks this Command Queue with the capability of being horizontally queued.

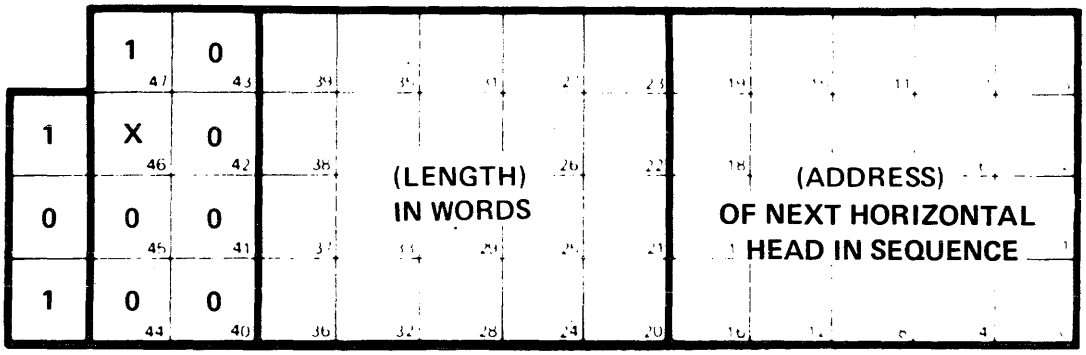


Figure 11-30. Command Queue Horizontal Queue Head Pointer Word Layout

Command Queue Horizontal Queue Link Word

Figure 11-31 shows the Command Queue Horizontal Queue Link word (word 4). The Horizontal Queue Link word (HQL) is initialized by the MCP, as a formatted operand (with all bits equal to zeroes). The MLIP subsequently replaces the formatted operand with a present, unsegmented, unindexed word Data Descriptor, to dynamically link Command Queues into the Horizontal Queue. The MLIP updates the Horizontal Queue Link word, so that it always points to the first Head word in the horizontal queue.

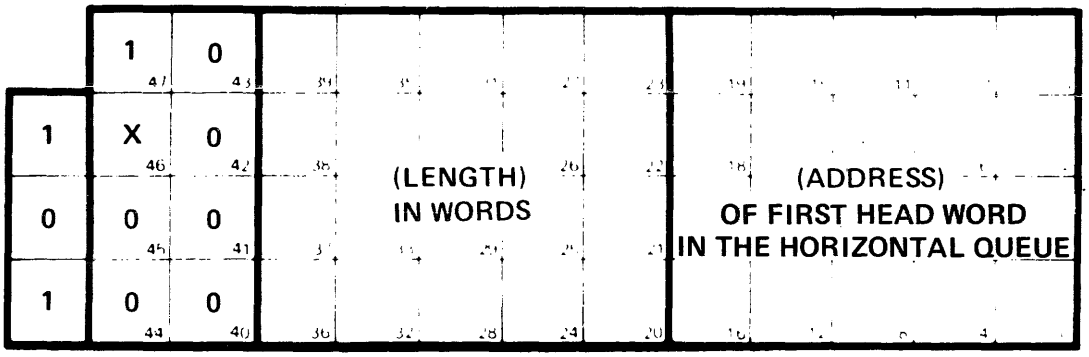


Figure 11-31. Command Queue Horizontal Queue Link Word Layout

HORIZONTAL QUEUE ORGANIZATION AND WORD LAYOUTS

Figure 11-32 shows the organization of a Horizontal Queue array. A Horizontal Queue array is a dynamic control linkage used to associate Command Queues for a common UIO-DLP device. Dynamic control means that the number of Command Queues associated in a Horizontal Queue array depends on the number of I/O operations in process, or waiting to be processed by the common UIO-DLP.

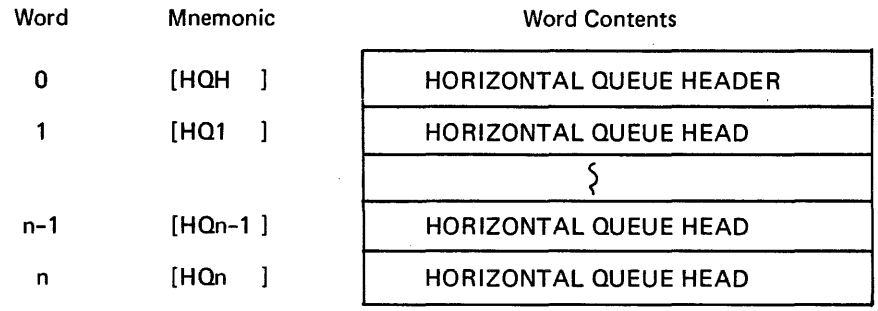


Figure 11-32. Horizontal Queue Array Word Format and Layout

The B 6900 system MCP determines whether or not a Command Queue can be horizontally queued (see HQHP, word 3 of Command Queue). If the MLIP ENQUEUEs an IOCB into a Command Queue that contains an HQHP, and if the UIO-DLP is BUSY when the MLIP subsequently attempts to initiate the I/O operation specified by the IOCB, then the MLIP logic links the Command Queue into the Horizontal Queue for the UIO-DLP. Command Queues are linked into a Horizontal Queue on a First-In-First-Out basis. That is, the oldest Command Queue in a Horizontal Queue is linked by the first Horizontal Queue Head word in the array, and the latest Command Queue is linked by the last Head word in the array.

When the MLIP links a Command Queue into a Horizontal Queue, it also completes the Command Queue Horizontal Link word (HQL, word 4 of the Command Queue). An HQL word always points to the first Horizontal Queue Head word in the Horizontal Queue array, and thus associates all Command Queues in the Horizontal Queue. The Command Queue Horizontal Link word preserves the First-In-First-Out principal for Horizontal Queuing, because the reference address into the Horizontal Queue always points at the oldest Command Queue Head word in the Horizontal Queue array.

When a Command Queue linked in a Horizontal Queue is completed, the logic of the MLIP dynamically deletes its Horizontal Queue Head word from the Horizontal Queue array and moves subsequent Head words in the array up in priority, to fill the space created by the deletion.

Horizontal Queue Array Header Word

Figure 11-33 shows the Horizontal Queue Header word (word ZERO). The Horizontal Queue Header word (HQH) is a formatted operand that marks the beginning of a Horizontal Queue array in system memory. The MCP places the Header word in memory, and the MLIP never accesses this word. The fields of the HQH word are used only for software purposes.

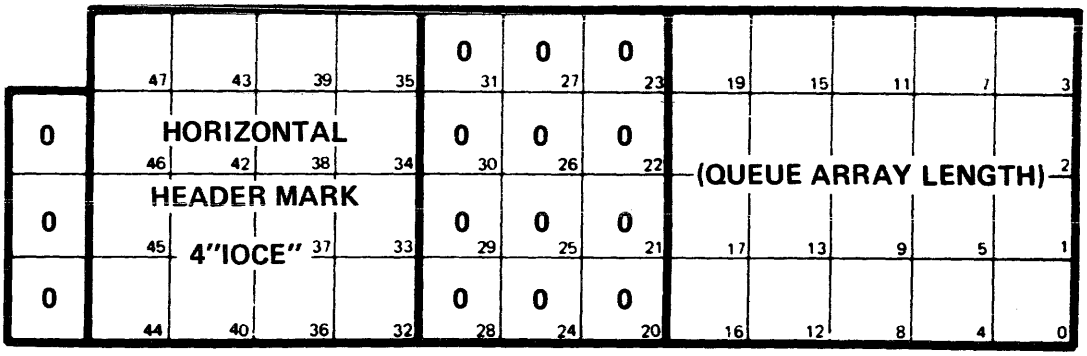


Figure 11-33. Horizontal Queue Array Header Word Layout

HORIZONTAL QUEUE HEADER WORD FIELD AND BIT DEFINITION

The fields of the Horizontal Queue Header word are as follow:

Bits [47:16]: Horizontal Queue Header Mark. This field contains 4"10CE" and is set up by the MCP to identify this word as a Horizontal Queue Header Word.

Bits [19:20]: Queue Length. This field contains the length of the horizontal queue in words.

Horizontal Queue Head Word

Figure 11-34 shows the Horizontal Queue Head word (words 1 through n). The Horizontal Queue Head word (HQn) is initialized by the MCP as a formatted operand with all bits zeroed. The MLIP logic replaces the HQn word with a present, unsegmented, unindexed word Data Descriptor that points at the memory address of the Command Queue Header (word zero), when a Command Queue is linked into the Horizontal Queue array.

When a horizontally queued Command Queue is completed, the MLIP deletes the HQn word from the Horizontal Queue array.

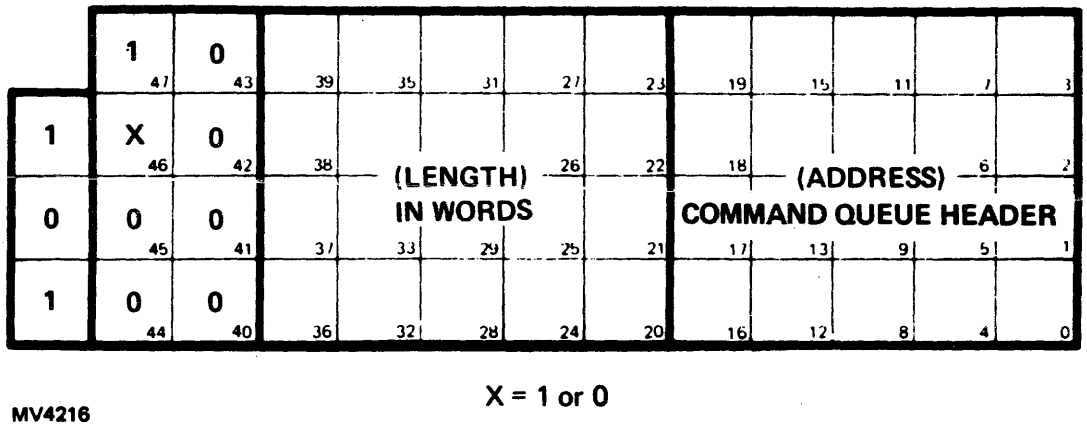


Figure 11-34. Horizontal Queue Array Horizontal Queue Head Word Layout

MLIP COMMANDS

The MCP generates IOCBs for MLIP path control logic commands. These commands are not sent to the UIO subsystem; instead, they are executed by internal logic circuits of the MLIP. Basically, MLIP commands establish values of control logic parameters that are used by the micro-module while performing other MLIP sequences. By exercising control over the path selection criteria of the MLIP, the MCP controls the overall operation of the I/O device subsystem.

The format of an IOCB for an MLIP Command is similar to that for an I/O device operation, except that bit-1 of the IOCB Control Word (the MLIP/DLP-Command bit) is SET. For an I/O Command, bit-1 is RESET. When the logic of the MLIP INITIATE sequence references the DLP Command Descriptor (through the reference provided in word-4 of the IOCB) it finds an MLIP Command Word instead of an I/O device Command Descriptor. Figure 11-35 shows the layout of an MLIP Command Word.

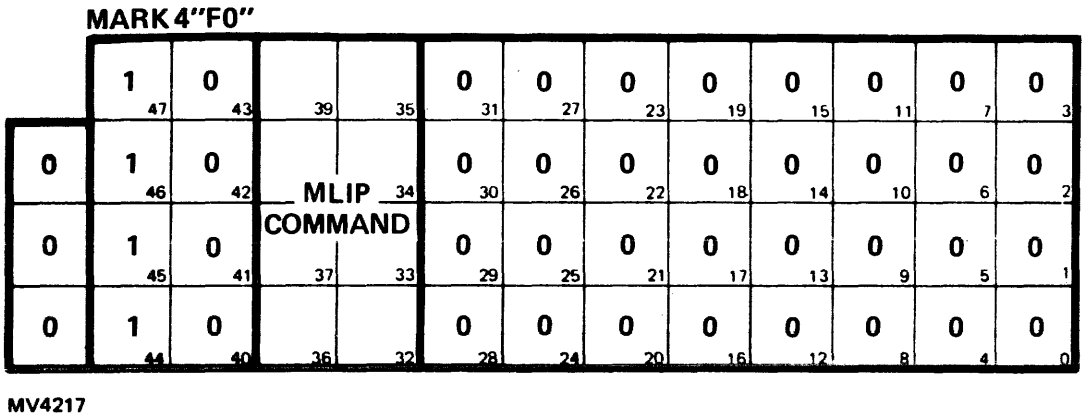


Figure 11-35. MLIP Command Word Layout

The fields in an MLIP Command Word are as follows:

Bits [47:8] MLIP Command Word Flag Field. This field must contain the value 4"F0", to indicate that this is indeed an MLIP Command Word. If the MLIP/DLP-Command bit of the IOCB Control Word is SET, the INITIATE sequence logic expects to find the Flag-field equal to 4"F0", to mark this word as an MLIP Command Word. If this field is not equal to 4"F0", then an ALARM interrupt is generated by the Interrupt Controller, and the MLIP Command is aborted. If this field is equal to 4"F0", then the INITIATE sequence causes the MLIP Command (indicated by the value of [39:8]) to be executed.

Bits [39:8] MLIP Command Descriptor Code

4"01" Wait For Error (Error-IOCB). The IOCB is identified by this code as an Error-IOCB, and the MLIP logic writes its absolute memory address in the MLIP RAM memory. The MLIP uses this address whenever its logic requires an Error-IOCB. (To report an error that is not otherwise reported in another IOCB, refer to MLIP Error-IOCB paragraphs.)

4"02" Clear DLP. The MLIP Selectively Clears the UIO-DLP device specified in the DLP Address Word (word-1 of the IOCB). After the DLP has been Selectively Cleared, the IOCB is linked into the Result Queue specified in word-8 of the IOCB.

4"03" General Clear. The MLIP initiates a Master Clear of all MLI-Ports. The IOCB is then linked into the Result Queue specified in word-8 of the IOCB.

4"04" SET Suspend-All-Queues flag. The MLIP causes the flag-bit to be SET, and then links the IOCB into the Result Queue specified in word-8 of the IOCB. The MCP executes this MLIP Command just before executing a MEMORY-DUMP procedure, to prevent MLIP data in memory from being changed while the dump procedure is in process.

4"05" RESET Suspend-All-Queues Flag. The MLIP causes the flag-bit to be RESET, and then links the IOCB into the Result Queue that is specified in word-8 of the IOCB. The MCP executes this MLIP Command just after a MEMORY-DUMP procedure. The execution of this MLIP Command allows the MLIP to resume I/O Commands that were interrupted while a dump procedure was being performed.

4"06" Read DLP-Status. The MLIP logic causes a CONNECT-sequence to be performed to the DLP specified in the DLP Address Word (word-1) of the IOCB. The Result-status that is returned by the UIO-DLP is written into the MLIP State And Result Word (word-12 of the IOCB), and then the IOCB is linked into the Result Queue specified in word-8 of the IOCB.

4"07" Activate Queue. The MLIP RESETS the Suspended-bit in the Command Queue Header Word referenced by word-2 of the IOCB. If appropriate the MLIP also initiates the first IOCB in the Command Queue that is activated. The IOCB that caused the Command Queue to be activated is then linked into the Result Queue specified by word-8 of the IOCB.

4"08" Return Queue. The MLIP accesses the Command Queue referenced by word-2 of the IOCB. The Head IOCB Link (word-1 of the Command Queue) and the Tail IOCB Link (word-2 of the Command Queue) are RESET to all zeroes. The original Head IOCB Link word value is placed in the first word referenced by the DLP I/O Result Pointer (word-5 of the IOCB). The IOCB that caused the Command Queue to be returned is then linked into the Result Queue specified by word-8 of the IOCB. A returned Command Queue is not delinked from a possible Horizontal Queue into which it may be linked. The Inactive-Count field of the returned Command Queue Control Word (word ZERO) is RESET to zero.

4"09" Read MLIP Status. The MLIP formats a word of MLIP Status as shown in Figure 11-36. The formatted MLIP Status word is written into system memory. The memory address of the status word is referenced by the DLP I/O Result Pointer (IOCB, word-5). The IOCB is then linked into the Result Queue specified by word-8 of the IOCB.

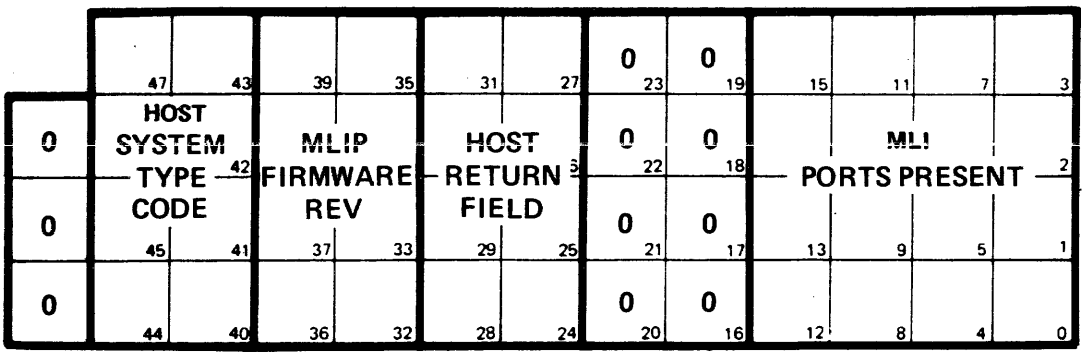


Figure 11-36. MLIP Status Word Layout

RESULT QUEUE ORGANIZATION AND WORD LAYOUTS

Figure 11-37 shows the organization of a Result Queue structure. A Result Queue is the final structure into which the MLIP links a Command Queue, after the I/O operation specified by the Command Queue has terminated. Linking a Command Queue into a Result Queue returns control of the I/O operation to the B 6900 system software, and deletes all records of the I/O operation from the logic of the MLIP module.

Word	Mnemonic	Word Meaning
0	[RQH]	RESULT QUEUE HEADER
1	[RQ1]	RESULT QUEUE HEAD
		?
n-1	[RQn-1]	RESULT QUEUE HEAD
n	[RQn]	RESULT QUEUE HEAD

Figure 11-37. Result Queue Word Format and Layout

Result Queue Header Word

Figure 11-38 shows the Result Queue Header Word (word ZERO). The Result Queue Header Word (RQH) is a formatted operand, used to mark the beginning of a Result Queue in system memory. The MCP initializes the RQH Word, and the MLIP never accesses this word.

The fields of the Result Queue Header Word are as follow:

Bits [47:16]: Result Queue Header Mark. This field contains 4"10CF" and is set up by the MCP identify this word as a Result Queue Header Word.

Bits [19:20]: Queue Length. This field contains the length of the result queue.

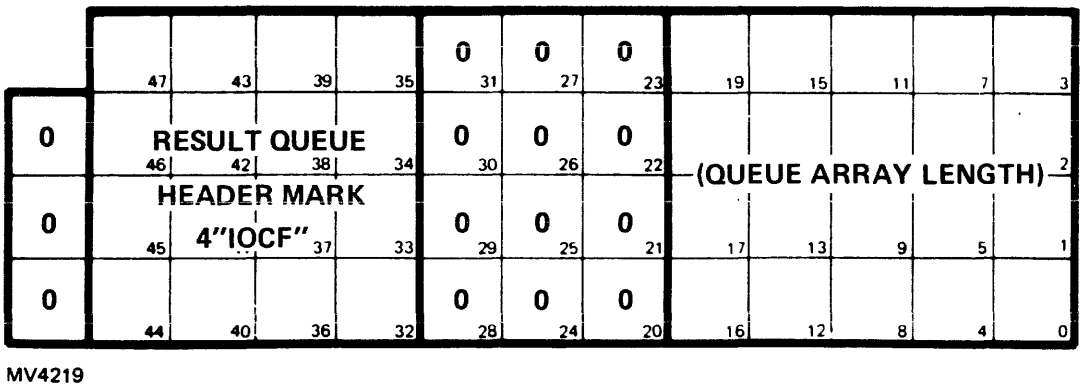


Figure 11-38. Result Queue Header Word Layout

Result Queue Head Word

Figure 11-39 shows the Result Queue Head Word (1 through n). This word (RQn) is initialized in system memory by the MCP as a formatted operand with all bits equal to zeroes. The MLIP delinks a Command Queue at the termination of the I/O operation, and links the Command Queue into the Result Queue. The procedure the MLIP used to link the Command Queue into the Result Queue is to replace the formatted operand in the Result Queue with a Result Queue Head pointer that points at the Command Queue Header word.

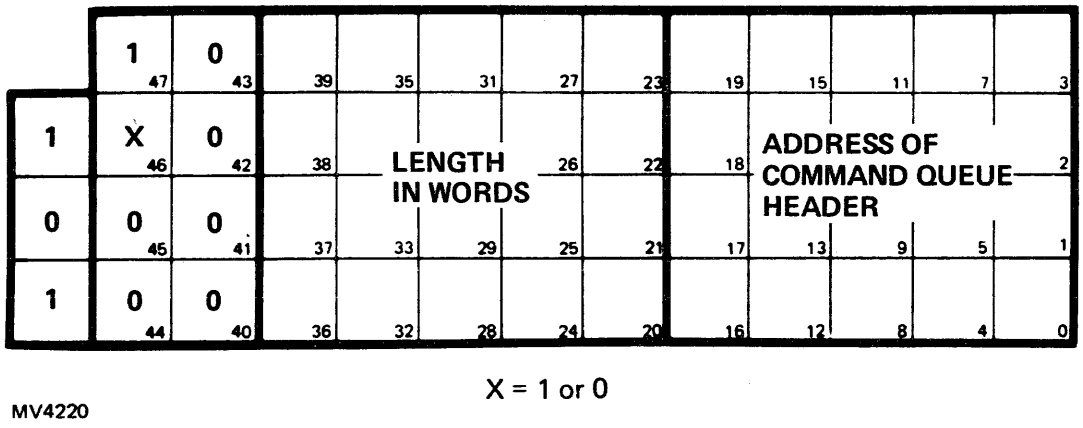


Figure 11-39. Result Queue Head Word Layout

ERROR-IOCB WORD FORMATS AND STRUCTURES

There are three categories of I/O operation errors that can be detected by the logic of the MLIP and reported to the B 6900 software operating system. Two of these error categories have been defined and described previously. They are (1) hardware errors that affect a single DLP (reported to the software by means of a Result Descriptor), and (2) logical errors that affect the MLIP (reported to the software by means of the IOCB State And Result Word). The third category consists of errors that may affect the entire I/O subsystem of the B 6900 and, therefore, have no proper place to be reported in the formats of Result Descriptors and/or IOCB State And Result Words.

An Error-IOCB is a mechanism for reporting errors of the third category. Such errors consist of, but are not limited to, Invalid Descriptor Link Words and Invalid Queue Words. Third-category type errors are detected by the logic of the MLIP micro-module sequences, and cause automatic micro-code error-handling subroutines to be executed by the micro-module.

Figure 11-40 shows the word layout of an Error-IOCB in system memory. An Error-IOCB must be initialized by the system software before a third-category error-condition can be reported. When an Error-IOCB is present, the absolute memory address of the first word in the IOCB is stored in the Data Storage section (word 1) of the MLIP RAM memory.

Word	Mnemonic	Word Meaning
0	[ ]	SYSTEM TYPE, FIRMWARE REV, ERROR-CODE, DLP
1	[ ]	ERROR-TYPE (ALARM/HARDWARE INTERRUPT)
2	[ ]	MEM. ADDR., GLOBAL PAR., PROM-CARD, MICRO-ADDR.
3	[ ]	IC MEM. ADDR. VALUES FOR IRS7 (BUF), BRS7 (TEMP)
4	[ ]	TOP-OF-STACK Z REGISTER VALUE
5	[ ]	UNDEFINED
6	[ ]	TOP-OF-STACK C REGISTER VALUE
7	[ ]	UNDEFINED
8	[ ]	WORD IN MICRO-STACK AT MICRO-STACK POINTER ADDR.
9	[ ]	WORD IN MICRO-STACK AT MS POINTER +1
10	[ ]	WORD IN MICRO-STACK AT MS POINTER +2
11	[ ]	WORD IN MICRO-STACK AT MS POINTER +3
12	[ ]	UNDEFINED
13	[ ]	WORD 0 OF MLIP RAM (MLIP.STAT.REG)
14	[ ]	WORD 1 OF MLIP RAM (ERR.IOCB.ADR)
15	[ ]	WORD 2 OF MLIP RAM (ERR.RSLT.ADR)
16	[ ]	WORD 3 OF MLIP RAM (WOLP.ADR)
17	[ ]	WORD 4 OF MLIP RAM (COLP.ADR)
18	[ ]	WORD 5 OF MLIP RAM (HQH.ADR)
19	[ ]	WORD 6 OF MLIP RAM (CQ.ADR)
20	[ ]	WORD 7 OF MLIP RAM (IOCB.ADR)
21	[ ]	WORD 8 OF MLIP RAM (CQ.OW.1)
22	[ ]	WORD 9 OF MLIP RAM (CQ.CW.2)
23	[ ]	WORD A OF MLIP RAM (HST.RET.FLD)
24	[ ]	WORD B OF MLIP RAM (BR.STAT.REG)
25	[ ]	WORD C OF MLIP RAM (NOT USED)
26	[ ]	WORD D OF MLIP RAM (NOT USED)
27	[ ]	WORD E OF MLIP RAM (NOT USED)
28	[ ]	WORD F OF MLIP RAM (NOT USED)
29	[ ]	CONTENTS OF MLIP REGISTER 3
30	[ ]	UNDEFINED
31	[ ]	UNDEFINED

Figure 11-40. Error-IOCB Organization And Layout

Error-IOCB Word Zero Layout

Figure 11-41 shows the bit-field layout of the first word in an Error-IOCB. This word defines a B 6900 system MLIP, the current Firmware Revision of the MLIP micro-module logic, the type of error being reported in the Error-IOCB, and the UIO-DLP Address of the peripheral device operating when the error occurred.

The usage or meaning of the bit-fields in word ZERO of an Error-IOCB is as follows:

[47:8] System Type. This field always contains the value 4"01", which identifies the system type as a B 6900 system.

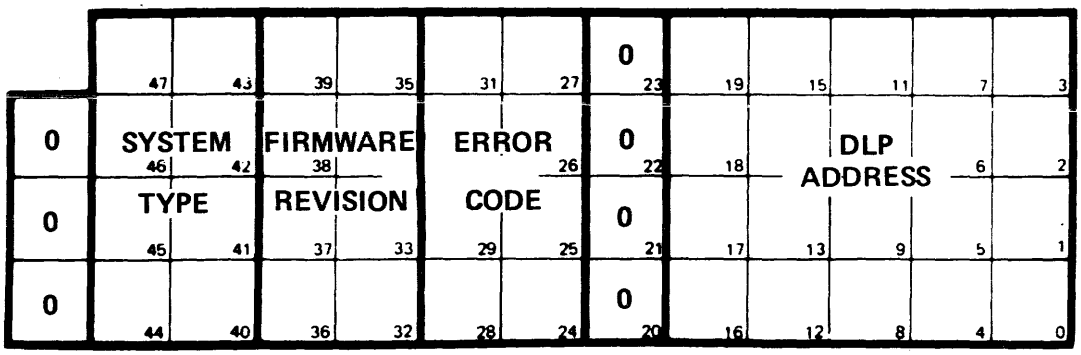
[39:8] Firmware Revision. This field contains a literal value from the micro-module, which identifies the current level of the CPU Micro-code.

[31:8] Error Code. This field contains a code that identifies the kind of error that is being reported in this Error-IOCB:

- 4"01" An Invalid Descriptor Link was detected.
- 4"02" An Invalid Global Parameter was detected.
- 4"03" A Descriptor Link LPW error was detected.
- 4"04" A DLP Address mismatch error was detected.
- 4"05" A Host Return Field mismatch error was detected.
- 4"10" A Queueing error was detected.
- 4"20" A Memory/Hardware error was detected.
- 4"40" An Error-IOCB was discontinued.

[23:4] This field must contain all zeroes.

[19:20] DLP Address. This field contains the address-vector through the MLI interface Ports to the peripheral device UIO-DLP that was operating when the error was detected.



MV4221

Figure 11-41. Error-IOCB Word Zero Layout

Error-IOCB Word One Layout

Figure 11-42 shows the layout of an Error-IOCB, word-1. Word-1 of an Error-IOCB contains a field that specifies the type of interrupt that caused the Error-IOCB to be completed. There are two conditions that can cause an Error-IOCB to be completed. The first is a hardware failure in the MLIP logic, a PROM card module error, and the second is a Link-Word or Pointer-error that causes an addressing failure within the I/O device control parameters of system memory.

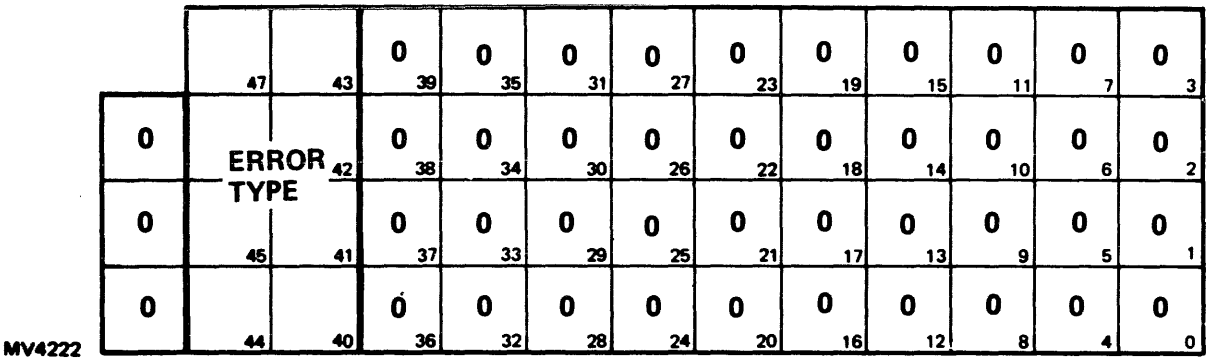


Figure 11-42. Error-IOCB Word One Layout

Error-IOCB Word Two Layout

Figure 11-43 shows the layout of word-2 in an Error-IOCB. This word contains key information that is useful in reconstructing the MLIP conditions at the time in which the Error occurred. The data contained in this word is as follows:

[47:20] Last Memory Address. This field contains the absolute memory address value of the last memory access request originated by the logic of the MLIP.

[27:1] This bit contains a zero.

[26:3] Global Parameter. This field contains the Global Parameter value for the last POLL REQUEST sequence performed by the logic of the MLIP.

[23:4] PROM Card Error Code. This field contains a code-value that identifies the MLIP plug-in PROM card-module that detected a parity-error condition.

[19:8] This field contains zeroes.

[11:12] Micro-module Address. This field contains the micro-module address of the MLIP sequence in which the Error-IOCB was completed.

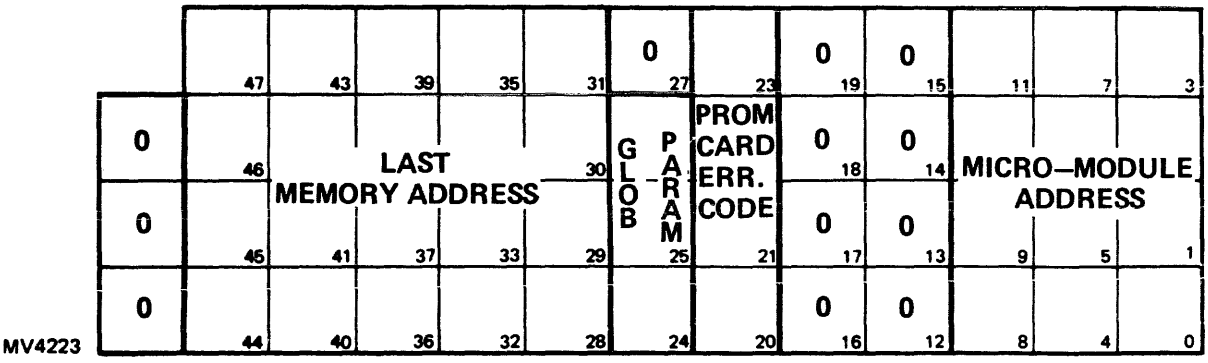


Figure 11-43. Error-IOCB Word-Two Layout

Error-IOCB Word Three Layout

Figure 11-44 shows the layout of word-3 in an Error-IOCB. This word contains the contents of 2 IC Memory Registers, which are used by the logic of the MLIP to process memory request addresses and to perform the mathematics for incrementing/decrementing I/O device data length counts. Figure 11-44 shows the layout of this Error-IOCB word.

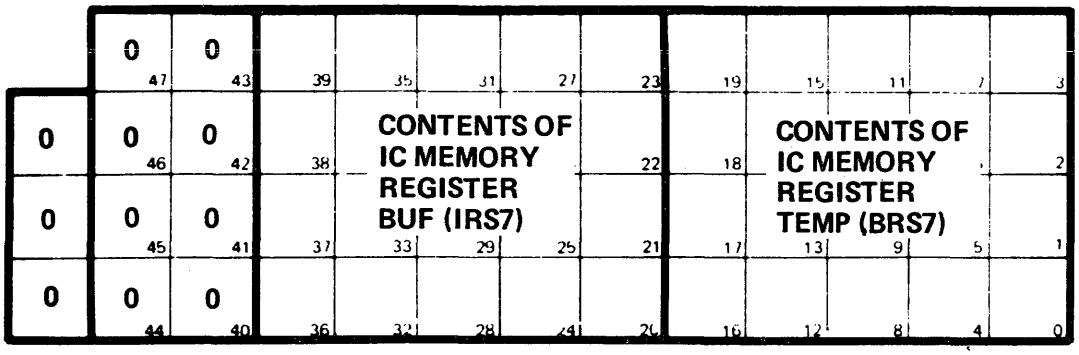


Figure 11-44. Error-IOCB Word Three Layout

Error-IOCB Word Four Layout

Figure 11-45 shows the layout of word-4 in an Error-IOCB. This word of the Error-IOCB contains the contents of the Top-of-Stack Z Register at the time when the Error-IOCB was completed. The logic of the MLIP utilizes the Z Register to process data between the MLIP and system memory.

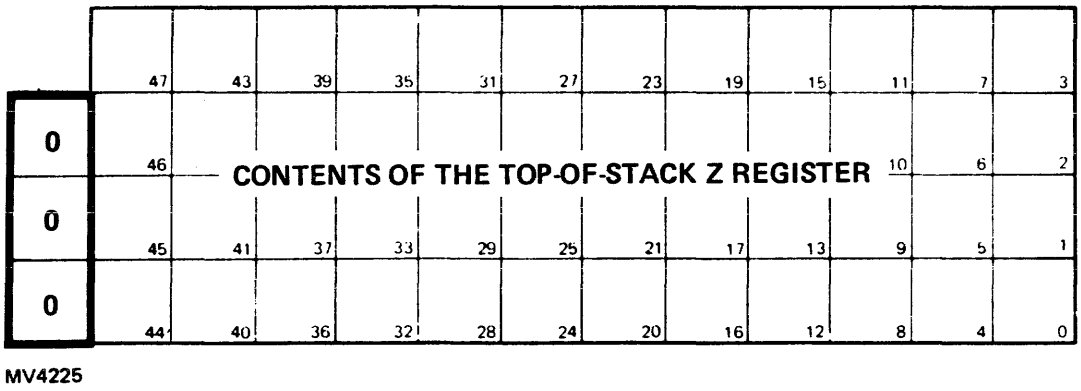


Figure 11-45. Error-IOCB Word Four Layout

Error-IOCB Word Six Layout

Figure 11-46 shows the layout of word-6 of an Error-IOCB. This word of the Error-IOCB contains the contents of the Top-of-Stack C Register when the Error-IOCB was completed. The logic of the MLIP utilizes the C Register to process data between the MLIP and system memory.

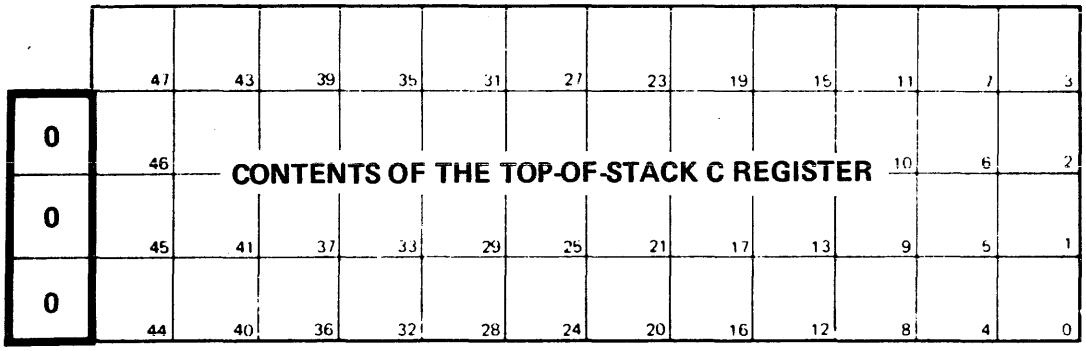


Figure 11-46. Error-IOCB Word Six Layout

Error-IOCB Word-8 Through Word-11 Layout

Figure 11-47 shows the layout of word-8 through word-11 of an Error-IOCB. These Error-IOCB words, containing the 4 most recent words written in the MLIP RAM Micro-stack section represent data used by the micro-module to manage the MLIP micro-code sequence subroutines. They are valuable for error analysis because they indicate the MLIP sequences that preceded the error IOCB condition. Word-8 of the Error-IOCB contains the most-recent micro-code word that was written into the Micro-stack section of the MLIP RAM. Word-9 of the Error-IOCB contains the next-most-recent word written into the Micro-stack section; word-10 holds the second-from-most-recent word in the Micro-stack section; and word-11 contains the third-from-most/ word in the Micro-stack section.

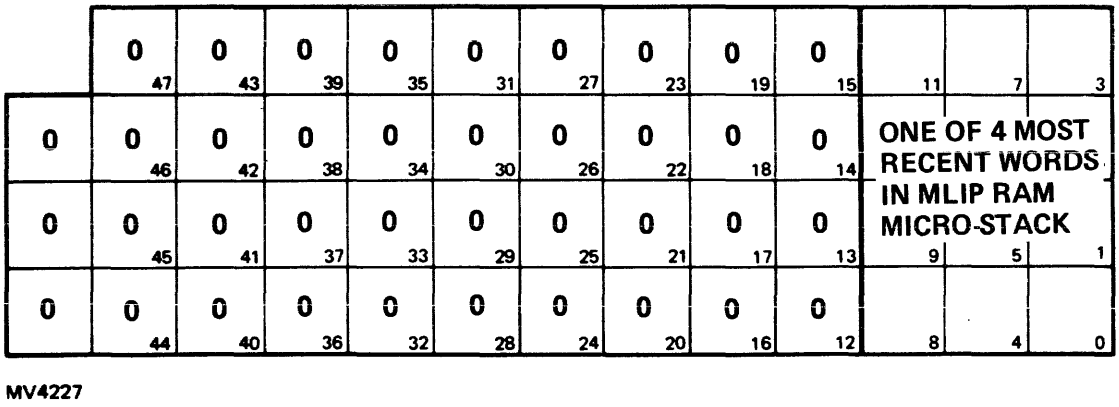


Figure 11-47. Error-IOCB Word-8 Through Word-11 Layout

Error-IOCB Word-13 Through Word-28 Layout

Figure 11-48 shows the layout of word-13 through word-28 of an Error-IOCB. These words contain the first 16-words from the MLIP RAM, which are the Data-Register section. These words contain the specificaitons from the IOCB, for the I/O device operation in process when the Error-IOCB was completed.

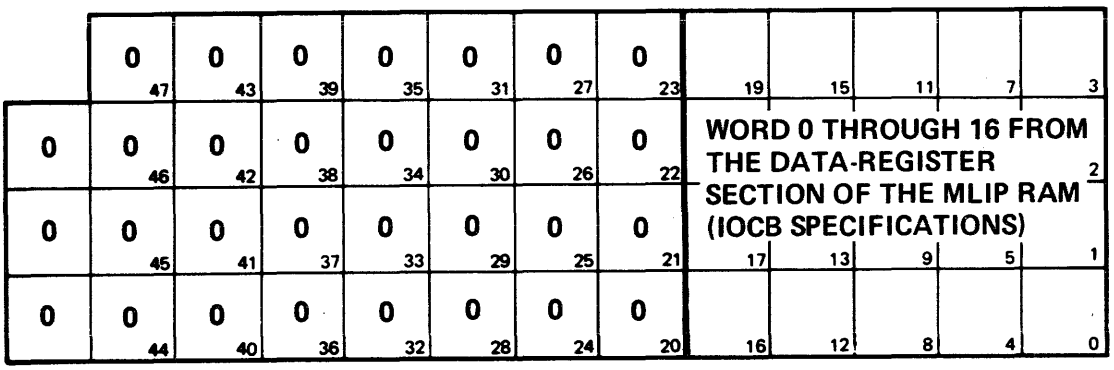


Figure 11-48. Error-IOCB Word-13 Through Word-28 Layout

Error-IOCB Word-29 Layout

Figure 11-49 shows the layout of word-29 in an Error-IOCB. This word contains the value of MLIP hardware Register-3. Register-3 of the MLIP contains the initial and remaining I/O operation LENGTH count. Each time 16-bits of peripheral data are transferred between the MLIP and system memory, the value of Register-3 is incremented/ decremented so that it contains the instantaneous value of remaining 16-bit bytes to be transferred. The value of word-29 in the Error-IOCB is therefore the number of 16-bit bytes of data not yet transferred when the MLIP completed the Error-IOCB.

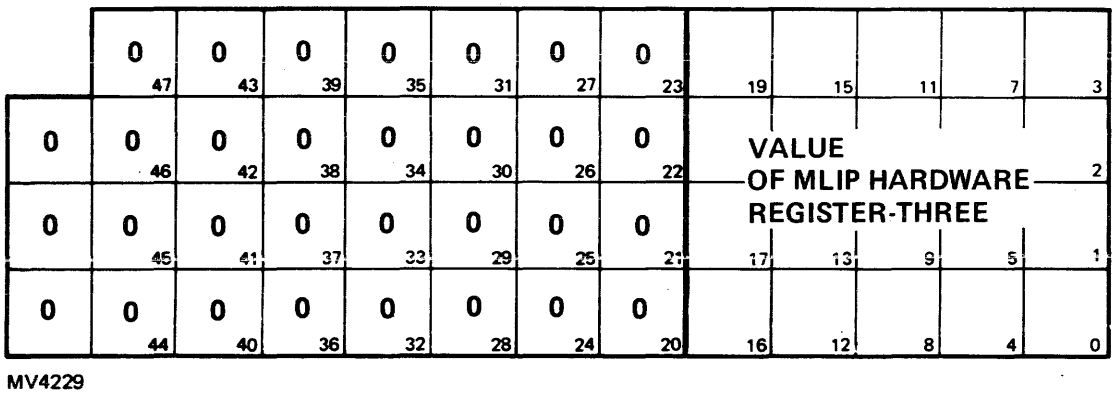


Figure 11-49. Error-IOCB Word Layout

GLOSSARY OF MLIP/UIO OPERATING TERMS

The following are some miscellaneous terms and mnemonics useful in understanding MLIP/IODC concepts:

MLIP	Message Level Interface Processor – protion of CPU logic which controls operations between the Data Processor and the IODC and its associated DLP's
IODC	Input Output Data Communication – subsystem utilized for I/O and Datacomm operations, common to the MLI interface specifications.
IOCB	Input Output Control Block – a contiguous area of memory containing the necessary information for the performance of an I/O or MLIP operation.
CUIO	Communicate with Universal I/O – a variant mode operator (954C) which starts an operation to the MLIP or IODC using a data descriptor found in the top of the stack pointing to the first word of the IOCB.
IOCB MARK	A value of 4"10CB" found in [47:16] of the first word in an IOCB used by the logic to verify this is actually the first word of an IOCB.
ERROR IOCB	An IOCB set aside by the MCP to be used by the MLIP to terminate an I/O operation when normal error termination is not possible.
MLI	Message Level Interface – a 25 line bidirectional interface between the MLIP and the IODC containing data and control information.
MLIP/CPU INTERFACE	Connection between CPU and MLIP, primarily Z1 bus, Z5 bus, C register, and micro-module address lines.
MLIP/UIO INTERFACE	Connection between IODC and MLIP called MLI.
DLP	Data Link Processor – a specialized micro-processor used to transfer information to and from a peripheral device.
POLL TEST	Process of MLIP connecting to IODC.
POLL REQUEST	Process of IODC reconnecting to MLIP following operation initiated by MLIP.
GLOBAL PRIORITY WORD	A word returned to MLIP during POLL REQUEST indicating priority of each DLP requesting connection to the MLIP.
COMMAND QUEUE	A linking together of IOCB's in the order in which they will be performed.
RESULT QUEUE	A linking together of IOCB's as the I/O operation is completed.
COMMAND QUEUE HEADER	A structure used to maintain the current state of a command queue.
RESULT QUEUE HEADER	A structure used to maintain the current state of the completed I/O operations.

APPENDIX A

OPERATORS, ALPHABETICAL LIST

Name	Mnemonic	Hexa-Decimal Code
ADD	ADD	80
BIT RESET	BRST	9E
BIT SET	BSET	96
BRANCH FALSE	BRFL	A0
BRANCH TRUE	BRTR	A1
BRANCH UNCONDITIONAL	BURN	A2
CHANGE SIGN BIT	CHSN	8E
COMPARE CHARACTERS EQUAL DESTRUCTIVE	CEQD	F4
COMPARE CHARACTERS EQUAL, UPDATE	CEQU	FC
COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE	CGED	F1
COMPARE CHARACTERS GREATER OR EQUAL, UPDATE	CGEU	F9
COMPARE CHARACTERS GREATER, DESTRUCTIVE	CGTD	F2
COMPARE CHARACTERS GREATER, UPDATE	CGTU	FA
COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE	CLED	F3
COMPARE CHARACTERS LESS OR EQUAL, UPDATE	CLEU	FB
COMPARE CHARACTERS LESS, DESTRUCTIVE	CLSD	F0
COMPARE CHARACTERS LESS, UPDATE	CLSU	F8
COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE	CNED	F5
CONTROL UNIVERSAL INPUT OUTPUT	CUIO	95C4
COMPARE CHARACTERS NOT EQUAL, UPDATE	CNEU	FD
CONDITIONAL HALT (all modes)	HALT	DF
COUNT BINARY ONES	CBON	95BB
DELETE TOP-OF-STACK	DLET	B5
DISABLE EXTERNAL INTERRUPT	DEXI	9547
DIVIDE	DIVD	83
DOUBLE LOAD A	DLA	E0
DOUBLE LOAD A INCREMENT	DLAI	E9
DOUBLE LOAD B	DLB	E2
DOUBLE LOAD B INCREMENT	DLBI	EB
DOUBLE LOAD C	DLC	E4
DOUBLE LOAD C INCREMENT	DLCI	ED
DOUBLE STORE A	DSA	F8
DOUBLE STORE A INCREMENT	DSAI	F9
DOUBLE STORE B	DSB	FA
DOUBLE STORE B INCREMENT	DSBI	FB
DOUBLE STORE C	DSC	FC
DOUBLE STORE C INCREMENT	DSCI	FD
DUPLICATE TOP-OF-STACK	DUPL	B7
DYNAMIC BIT RESET	DBRS	9F
DYNAMIC BIT SET	DBST	97
DYNAMIC BRANCH FALSE	DBFL	A8
DYNAMIC BRANCH TRUE	DBTR	A9

Name	Mnemonic	Hexa-Decimal Code
DYNAMIC BRANCH UNCONDITIONAL	DBUN	AA
DYNAMIC FIELD INSERT	DINS	9D
DYNAMIC FIELD ISOLATE	DISO	9B
DYNAMIC FIELD TRANSFER	DFTR	99
DYNAMIC SCALE LEFT	DSLF	C1
DYNAMIC SCALE RIGHT FINAL	DSRF	C7
DYNAMIC SCALE RIGHT ROUND	DSRR	C9
DYNAMIC SCALE RIGHT SAVE	DSRS	C5
DYNAMIC SCALE RIGHT TRUNCATE	DSRT	C3
ENABLE EXTERNAL INTERRUPTS	EEXI	9546
END EDIT (edit mode)	ENDE	DE
END FLOAT (edit mode)	ENDF	D5
ENTER	ENTR	AB
EQUAL	EQUL	8C
ESCAPE TO 16-BIT INSTRUCTION	VARI	95
EVALUATE	EVAL	AC
EXCHANGE	EXCH	B6
EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	EXPU	DD
EXECUTE SINGLE MICRO, DESTRUCTIVE	EXSD	D2
EXECUTE SINGLE MICRO, UPDATE	EXSU	DA
EXIT	EXIT	A3
EXTENDED MULTIPLY	MULX	8F
FIELD INSERT	INSR	9C
FIELD ISOLATE	ISOL	9A
FIELD TRANSFER	FLTR	98
GREATER THAN	GRTR	8A
GREATER THAN OR EQUAL	GREQ	89
IDLE UNTIL INTERRUPT	IDLE	9544
INDEX	INDX	A6
INDEX AND LOAD NAME	NXLN	A5
INDEX AND LOAD VALUE	NXLV	AD
INPUT CONVERT, DESTRUCTIVE	ICVD	CA
INPUT CONVERT UPDATE	ICVU	CB
INSERT CONDITIONAL (edit mode)	INSC	DD
INSERT DISPLAY SIGN (edit mode)	INSG	D9
INSERT MARK STACK	IMKS	CF
INSERT OVERPUNCH (edit mode)	INOP	D8
INSERT UNCONDITIONAL (edit mode)	INSU	DC
INTEGER DIVIDE	IDIV	84
INTEGERIZE, ROUNDED	NTGR	87
INTEGERIZE, TRUNCATED	NTIA	86
INTEGERIZE, ROUNDED DOUBLE-PRECISION	NTGD	9587
INVALID OPERATOR (all modes)	NVLD	FF
LEADING ONE TEST	LOG2	958B
LINKED LIST LOOKUP	LLLU	95BD
LESS THAN	LESS	88
LESS THAN OR EQUAL	LSEQ	8B
LIT CALL ONE	ONE	B1

Name	Mnemonic	Hexa-Decimal Code
LIT CALL ZERO	ZERO	B0
LIT CALL 8-BITS	LT8	B2
LIT CALL 16-BITS	LT16	B3
LIT CALL 48-BITS	LT48	BE
LOAD	LOAD	BD
LOAD A	LDA	E0
LOAD A INCREMENT	LDAI	E1
LOAD B	LDB	E2
LOAD B INCREMENT	LDBI	E3
LOAD C	LDC	E4
LOAD C INCREMENT	LDCI	E5
LOAD TRANSPARENT	LODT	95BC
LOGICAL AND	LAND	90
LOGICAL EQUAL	SAME	94
LOGICAL EQUIVALENCE	LEQV	93
LOGICAL NEGATE	LNOT	92
LOGICAL OR	LOR	91
MAKE PROGRAM CONTROL WORD	MPCW	BF
MARK STACK	MKST	AE
MASKED SEARCH FOR EQUAL	SRCH	95BE
MOVE CHARACTERS (edit mode)	MCHR	D7
MOVE NUMERIC UNCONDITIONAL (edit mode)	MVNU	D6
MOVE TO STACK	MVST	95AF
MOVE WITH FLOAT (edit mode)	MFLT	D1
MOVE WITH INSERT (edit mode)	MINS	D0
MULTIPLY	MULT	82
NAME CALL	NAMC	40⇒7F
NO OPERATION (all modes)	NOOP	FE
NORMALIZE	NORM	958E
NOT EQUAL	NEQL	8D
OCCURS INDEX	OCRX	9585
OVERWRITE DESTRUCTIVE	OVRD	BA
OVERWRITE NON-DESTRUCTIVE	OVRN	BB
PACK DESTRUCTIVE	PACD	D1
PACK UPDATE	PACU	D9
PUSH DOWN STACK REGISTERS	PUSH	B4
READ AND CLEAR OVERFLOW FLIP-FLOP	ROFF	D7
READ CENTRAL PROCESSOR COUNTER	RCPC	9540
READ COMPARE FLIP-FLOP	RCMP	95B3
READ PROCESSOR IDENTIFICATION	WHOI	954E
READ PROCESSOR REGISTER	RPRR	95B8
READ TAG FIELD	RTAG	95B5
READ TIME OF DAY	RTOD	95A7
READ TRUE/FALSE FLIP-FLOP	RTFF	DE
READ WITH LOCK	RDLK	95BA
REMAINDER DIVIDE	RDIV	85
RESET FLOAT (edit mode)	RSTF	D4
RETURN	RETN	A7
ROTATE STACK DOWN	RSDN	95B7

Name	Mnemonic	Hexa-Decimal Code
ROTATE STACK UP	RSUP	95B6
RUNNING INDICATOR	RUNI	9541
SCALE LEFT	SCLF	C0
SCALE RIGHT FINAL	SCRF	C6
SCALE RIGHT ROUNDED	SCRR	C8
SCALE RIGHT SAVE	SCRS	C4
SCALE RIGHT TRUNCATE	SCRT	C2
SCAN-IN	SCNI	954A
SCAN-OUT	SCNO	954B
SCAN WHILE EQUAL, DESTRUCTIVE	SEQD	95F4
SCAN WHILE EQUAL, UPDATE	SEQU	95FC
SCAN WHILE FALSE, DESTRUCTIVE	SWFD	95D4
SCAN WHILE FALSE, UPDATE	SWFU	95DC
SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	SGED	95F1
SCAN WHILE GREATER OR EQUAL, UPDATE	SGEU	95F9
SCAN WHILE GREATER, DESTRUCTIVE	SGTD	95F2
SCAN WHILE GREATER, UPDATE	SGTU	95FA
SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	SLED	95F3
SCAN WHILE LESS OR EQUAL, UPDATE	SLEU	95FB
SCAN WHILE LESS, DESTRUCTIVE	SLSD	95F0
SCAN WHILE LESS, UPDATE	SLSU	95F8
SCAN WHILE NOT EQUAL, DESTRUCTIVE	SNED	95F5
SCAN WHILE NOT EQUAL, UPDATE	SNEU	95FD
SCAN WHILE TRUE, DESTRUCTIVE	SWTD	95D5
SCAN WHILE TRUE, UPDATE	SWTU	95DD
SET DOUBLE TO TWO SINGLES	SPLT	9543
SET EXTERNAL SIGN	SXSN	D6
SET INTERVAL TIMER	SINT	9545
SET PROCESSOR REGISTER	SPRR	95B9
SET TAG FIELD	STAG	95B4
SET TO DOUBLE-PRECISION	XTND	CE
SET TO SINGLE-PRECISION, ROUNDED	SNGL	CD
SET TO SINGLE-PRECISION, TRUNCATED	SNGT	CC
SET TWO SINGLES TO DOUBLE	JOIN	9542
SKIP FORWARD DESTINATION CHARACTERS (edit mode)	SFDC	DA
SKIP FORWARD SOURCE CHARACTERS (edit mode)	SFSC	D2
SKIP REVERSE DESTINATION CHARACTERS (edit mode)	SRDC	DB
SKIP REVERSE SOURCE CHARACTERS (edit mode)	SRSC	D3
STEP AND BRANCH	STBR	A4
STORE A	STA	F0
STORE A INCREMENT	STAI	F1
STORE B	STB	F2
STORE B INCREMENT	STBI	F3
STORE C	STC	F4
STORE C INCREMENT	STCI	F5
STORE DESTRUCTIVE	STOD	B8
STORE NON-DESTRUCTIVE	STON	B9

Name	Mnemonic	Hexa-Decimal Code
STRING ISOLATE	SISO	D5
STUFF ENVIRONMENT	STFF	AF
SUBTRACT	SUBT	81
TABLE ENTER EDIT, DESTRUCTIVE	TEED	D0
TABLE ENTER EDIT, UPDATE	TEEU	D8
TRANSFER UNCONDITIONAL, DESTRUCTIVE	TUND	E6
TRANSFER UNCONDITIONAL, UPDATE	TUNU	EE
TRANSFER WHILE EQUAL, DESTRUCTIVE	TEQD	E4
TRANSFER WHILE EQUAL, UPDATE	TEQU	EC
TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	TGED	E1
TRANSFER WHILE GREATER OR EQUAL, UPDATE	TGEU	E9
TRANSFER WHILE GREATER, DESTRUCTIVE	TGTD	E2
TRANSFER WHILE GREATER, UPDATE	TGTU	EA
TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	TLED	E3
TRANSFER WHILE FALSE, DESTRUCTIVE	TWFD	95D2
TRANSFER WHILE FALSE, UPDATE	TWFU	95DA
TRANSFER WHILE TRUE, DESTRUCTIVE	TWTD	95D3
TRANSFER WHILE TRUE, UPDATE	TWTU	95DB
TRANSFER WHILE LESS OR EQUAL, UPDATE	TLEU	EB
TRANSFER WHILE LESS, DESTRUCTIVE	TLSD	E0
TRANSFER WHILE LESS, UPDATE	TLSU	E8
TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	TNED	E5
TRANSFER WHILE NOT EQUAL, UPDATE	TNEU	ED
TRANSFER WORDS OVERWRITE DESTRUCTIVE	TWOD	D4
TRANSFER WORDS OVERWRITE UPDATE	TWOU	DC
TRANSFER WORDS, DESTRUCTIVE	TWSD	D3
TRANSFER WORDS, UPDATE	TWSU	DB
TRANSLATE	TRNS	95D7
UNPACK ABSOLUTE, DESTRUCTIVE	UABD	95D1
UNPACK ABSOLUTE, UPDATE	UABU	95D9
UNPACK SIGNED, DESTRUCTIVE	USND	95D0
UNPACK SIGNED, UPDATE	USNU	95D8
VALUE CALL	VALC	00 ⇒ 3F
VECTOR BRANCH	VEBR	EE
VECTOR MODE ENTER MULTIPLE	VMEM	EF
VECTOR MODE ENTER SINGLE	VMES	E7
VECTOR MODE EXIT	VMEX	E6
WRITE TIME OF DAY	WTOD	9549

APPENDIX B

OPERATORS, NUMERICAL LIST

Hexa-Decimal Code	Name	Mnemonic
PRIMARY MODE		
00⇒3F	VALUE CALL	VALC
40⇒7F	NAME CALL	NAMC
80	ADD	ADD
81	SUBTRACT	SUBT
82	MULTIPLY	MULT
83	DIVIDE	DIVD
84	INTEGER DIVIDE	IDIV
85	REMAINDER DIVIDE	RDIV
86	INTEGERIZE, TRUNCATED	NTIA
87	INTEGERIZE, ROUNDED	NTGR
88	LESS THAN	LESS
89	GREATER THAN OR EQUAL	GREQ
8A	GREATER THAN	GRTR
8B	LESS THAN OR EQUAL	LSEQ
8C	EQUAL	EQUL
8D	NOT EQUAL	NEQL
8E	CHANGE SIGN BIT	CHSN
8F	EXTENDED MULTIPLY	MULX
90	LOGICAL AND	LAND
91	LOGICAL OR	LOR
92	LOGICAL NEGATE	LNOT
93	LOGICAL EQUIVALENCE	LEQV
94	LOGICAL EQUAL	SAME
95	ESCAPE TO 16-BIT INSTRUCTION	VARI
96	BIT SET	BSET
97	DYNAMIC BIT SET	DBST
98	FIELD TRANSFER	FLTR
99	DYNAMIC FIELD TRANSFER	DFTR
9A	FIELD ISOLATE	ISOL
9B	DYNAMIC FIELD ISOLATE	DISO
9C	FIELD INSERT	INSR
9D	DYNAMIC FIELD INSERT	DINS
9E	BIT RESET	BRST
9F	DYNAMIC BIT RESET	DBRS
A0	BRANCH FALSE	BRFL
A1	BRANCH TRUE	BRTR
A2	BRANCH UNCONDITIONAL	BRUN
A3	EXIT	EXIT
A4	STEP AND BRANCH	STBR
A5	INDEX AND LOAD NAME	NXLN
A6	INDEX	INDX
A7	RETURN	RETN
A8	DYNAMIC BRANCH FALSE	DBFL

Hexa-Decimal Code	Name	Mnemonic
A9	DYNAMIC BRANCH TRUE	DBTR
AA	DYNAMIC BRANCH UNCONDITIONAL	DBUN
AB	ENTER	ENTR
AC	EVALUATE DESCRIPTOR	EVAL
AD	INDEX AND LOAD VALUE	NXLV
AE	MARK STACK	MKST
AF	STUFF ENVIRONMENT	STFF
B0	LIT CALL ZERO	ZERO
B1	LIT CALL ONE	ONE
B2	LIT CALL 8-BITS	LT8
B3	LIT CALL 16-BITS	LT16
B4	PUSH DOWN STACK REGISTERS	PUSH
B5	DELETE TOP-OF-STACK	DLET
B6	EXCHANGE	EXCH
B7	DUPLICATE TOP-OF-STACK	DUPL
B8	STORE DESTRUCTIVE	STOD
B9	STORE NON-DESTRUCTIVE	STON
BA	OVERWRITE DESTRUCTIVE	OVRD
BB	OVERWRITE NON-DESTRUCTIVE	OVRN
BD	LOAD	LOAD
BE	LIT CALL 48-BITS	LT48
BF	MAKE PROGRAM CONTROL WORD	MPCW
C0	SCALE LEFT	SCLF
C1	DYNAMIC SCALE LEFT	DSLF
C2	SCALE RIGHT TRUNCATE	SCRT
C3	DYNAMIC SCALE RIGHT RUNCATE	DSRT
C4	SCALE RIGHT SAVE	SCRS
C5	DYNAMIC SCALE RIGHT SAVE	DSRS
C6	SCALE RIGHT FINAL	SCRF
C7	DYNAMIC SCALE RIGHT FINAL	DSRF
C8	SCALE RIGHT ROUNDED	SCRR
C9	DYNAMIC SCALE RIGHT ROUND	DSRR
CA	INPUT CONVERT, DESTRUCTIVE	ICVD
CB	INPUT CONVERT, UPDATE	ICVU
CC	SET TO SINGLE-PRECISION, TRUNCATED	SNGT
CD	SET TO SINGLE-PRECISION, ROUNDED	SNGL
CE	SET TO DOUBLE-PRECISION	XTND
CF	INSERT MARK STACK	IMKS
D0	TABLE ENTER EDIT, DESTRUCTIVE	TEED
D1	PACK DESTRUCTIVE	PACD
D2	EXECUTE SINGLE MICRO, DESTRUCTIVE	EXSD
D3	TRANSFER WORDS, DESTRUCTIVE	TWSD
D4	TRANSFER WORDS OVERWRITE DESTRUCTIVE	TWOD
D5	STRING ISOLATE	SISO
D6	SET EXTERNAL SIGN	SXSN
D7	READ AND CLEAR OVERFLOW FLIP-FLOP	ROFF
D8	TABLE ENTER EDIT, UPDATE	TEEU
D9	PACK UPDATE	PACU

Hexa-Decimal Code	Name	Mnemonic
DA	EXECUTE SINGLE MICRO, UPDATE	EXSU
DB	TRANSFER WORDS, UPDATE	TWSU
DC	TRANSFER WORDS OVERWRITE UPDATE	TWOU
DD	EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	EXPU
DE	READ TRUE/FALSE FLIP-FLOP	TRFF
DF	CONDITIONAL HALT	HALT
E0	TRANSFER WHILE LESS, DESTRUCTIVE	TLSD
E1	TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	TGED
E2	TRANSFER WHILE GREATER, DESTRUCTIVE	TGTD
E3	TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	TLED
E4	TRANSFER WHILE EQUAL, DESTRUCTIVE	TEQD
E5	TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	TNED
E6	TRANSFER UNCONDITIONAL, DESTRUCTIVE	TUND
E7	VECTOR MODE ENTER SINGLE	VMES
E8	TRANSFER WHILE LESS, UPDATE	TLSU
E9	TRANSFER WHILE GREATER OR EQUAL, UPDATE	TGEU
EA	TRANSFER WHILE GREATER, UPDATE	TGTU
EB	TRANSFER WHILE LESS OR EQUAL, UPDATE	TLEU
EC	TRANSFER WHILE EQUAL, UPDATE	TEQU
ED	TRANSFER WHILE NOT EQUAL, UPDATE	TNEU
EE	TRANSFER UNCONDITIONAL, UPDATE	TUNU
EF	VECTOR MODE ENTER MULTIPLE	VMEM
F0	COMPARE CHARACTERS LESS, DESTRUCTIVE	CLSD
F1	COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE	CGED
F2	COMPARE CHARACTERS GREATER, DESTRUCTIVE	CGTD
F3	COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE	CLED
F4	COMPARE CHARACTERS EQUAL, DESTRUCTIVE	CEQD
F5	COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE	CNED
F8	COMPARE CHARACTERS LESS, UPDATE	CLSU
F9	COMPARE CHARACTERS GREATER OR EQUAL, UPDATE	CGEU
FA	COMPARE CHARACTERS GREATER, UPDATE	CGTU
FB	COMPARE CHARACTERS LESS OR EQUAL, UPDATE	CLEU
FC	COMPARE CHARACTERS EQUAL, UPDATE	CEQU
FD	COMPARE CHARACTERS NOT EQUAL, UPDATE	CNEU
FE	NO OPERATION	NOOP
FF	INVALID OPERATOR	NVLD

VARIANT MODE

9540	READ CENTRAL PROCESSOR COUNTER	RCPC
9541	RUNNING TIMER	RUNI
9542	SET TWO SINGLES TO DOUBLE	JOIN
9543	SET DOUBLE TO TWO SINGLES	SPLT
9544	IDLE UNTIL INTERRUPT	IDLE

Hexa-Decimal Code	Name	Mnemonic
9545	SET INTERVAL TIMER	SINT
9546	ENABLE EXTERNAL INTERRUPTS	EEXI
9547	DISABLE EXTERNAL INTERRUPTS	DEXI
9549	WRITE TIME OF DAY	WTOD
954A	SCAN-IN	SCNI
954B	SCAN-OUT	SCNO
954C	CONTROL UNIVERSAL INPUT OUTPUT	CUIO
954E	READ PROCESSOR IDENTIFICATION	WHOI
9585	OCCURS INDEX	OCRX
9587	INTEGERIZE, ROUNDED, DOUBLE-PRECISION	NTGD
958B	LEADING ONE TEST	LOG2
958E	NORMALIZE	NORM
95A7	READ TIME OF DAY	RTOD
95AF	MOVE TO STACK	MVST
95B3	READ COMPARE FLIP-FLOP	RCMP
95B4	SET TAG FIELD	STAG
95B5	READ TAG FIELD	RTAG
95B6	ROTATE STACK UP	RSUP
95B7	ROTATE STACK DOWN	RSDN
95B8	READ PROCESSOR REGISTER	RPRR
95B9	SET PROCESSOR REGISTER	SPRR
95BA	READ WITH LOCK	RDLK
95BB	COUNT BINARY ONES	CBON
95BC	LOAD TRANSPARENT	LODT
95BD	LINKED LIST LOOKUP	LLLU
95BE	MASKED SEARCH FOR EQUAL	SRCH
95D0	UNPACK SIGNED, DESTRUCTIVE	USND
95D1	UNPACK ABSOLUTE, DESTRUCTIVE	UABD
95D2	TRANSFER WHILE FALSE, DESTRUCTIVE	TWFD
95D3	TRANSFER WHILE TRUE, DESTRUCTIVE	TWTD
95D4	SCAN WHILE FALSE, DESTRUCTIVE	SWFD
95D5	SCAN WHILE TRUE, DESTRUCTIVE	SWTD
95D7	TRANSLATE	TRNS
95D8	UNPACK SIGNED, UPDATE	USNU
95D9	UNPACK ABSOLUTE, UPDATE	UABU
95DA	TRANSFER WHILE FALSE, UPDATE	TWFU
95DB	TRANSFER WHILE TRUE, UPDATE	TWTU
95DC	SCAN WHILE FALSE, UPDATE	SWFU
95DD	SCAN WHILE TRUE, UPDATE	SWTU
95DF	CONDITIONAL HALT	HALT
95F0	SCAN WHILE LESS, DESTRUCTIVE	SLSD
95F1	SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	SGED
95F2	SCAN WHILE GREATER, DESTRUCTIVE	SGTD
95F3	SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	SLED
95F4	SCAN WHILE EQUAL, DESTRUCTIVE	SEQD
95F5	SCAN WHILE NOT EQUAL, DESTRUCTIVE	SNED
95F8	SCAN WHILE LESS, UPDATE	SLSU
95F9	SCAN WHILE GREATER OR EQUAL, UPDATE	SGEU

Hexa-Decimal Code	Name	Mnemonic
95FA	SCAN WHILE GREATER, UPDATE	SGTU
95FB	SCAN WHILE LESS OR EQUAL, UPDATE	SLEU
95FC	SCAN WHILE EQUAL, UPDATE	SEQU
95FD	SCAN WHILE NOT EQUAL, UPDATE	SNEU
95FE	NO OPERATION	NOOP
95FF	INVALID	NVLD
EDIT MODE		
D0	MOVE WITH INSERT	MINS
D1	MOVE WITH FLOAT	MFLT
D2	SKIP FORWARD SOURCE CHARACTERS	SFSC
D3	SKIP REVERSE SOURCE CHARACTERS	SRSC
D4	RESET FLOAT	RSTF
D5	END FLOAT	ENDF
D6	MOVE NUMERIC UNCONDITIONAL	MVNU
D7	MOVE CHARACTERS	MCHR
D8	INSERT OVERPUNCH	INOP
D9	INSERT DISPLAY SIGN	INSG
DA	SKIP FORWARD DESTINATION CHARACTERS	SFDC
DB	SKIP REVERSE DESTINATION CHARACTERS	SRDC
DC	INSERT UNCONDITIONAL	INSU
DD	INSERT CONDITIONAL	INSC
DE	END EDIT	ENDE
DF	CONDITIONAL HALT	HALT
FE	NO OPERATION	NOOP
FF	INVALID	NVLD
VECTOR MODE		
E0	LOAD A	LDA
E1	LOAD A INCREMENT	LDAI
E2	LOAD B	LDB
E3	LOAD B INCREMENT	LDBI
E4	LOAD C	LDC
E5	LOAD C INCREMENT	LDCI
E6	VECTOR MODE EXIT	VMEX
E8	DOUBLE LOAD A	DLA
E9	DOUBLE LOAD A INCREMENT	DLAI
EA	DOUBLE LOAD B	DLB
EB	DOUBLE LOAD B INCREMENT	DLBI
EC	DOUBLE LOAD C	DLC
ED	DOUBLE LOAD C INCREMENT	DLCI
EE	VECTOR BRANCH	VEBR
F0	STORE A	STA
F1	STORE A INCREMENT	STAI
F2	STORE B	STB
F3	STORE B INCREMENT	STBI
F4	STORE C	STC

Hexa-Decimal Code	Name	Mnemonic
F5	STORE C INCREMENT	STCI
F8	DOUBLE STORE A	DSA
F9	DOUBLE STORE A INCREMENT	DSAI
FA	DOUBLE STORE B	DSB
FB	DOUBLE STORE B INCREMENT	DSBI
FC	DOUBLE STORE C	DSC
FD	DOUBLE STORE C INCREMENT	DSCI

APPENDIX C

DATA REPRESENTATION

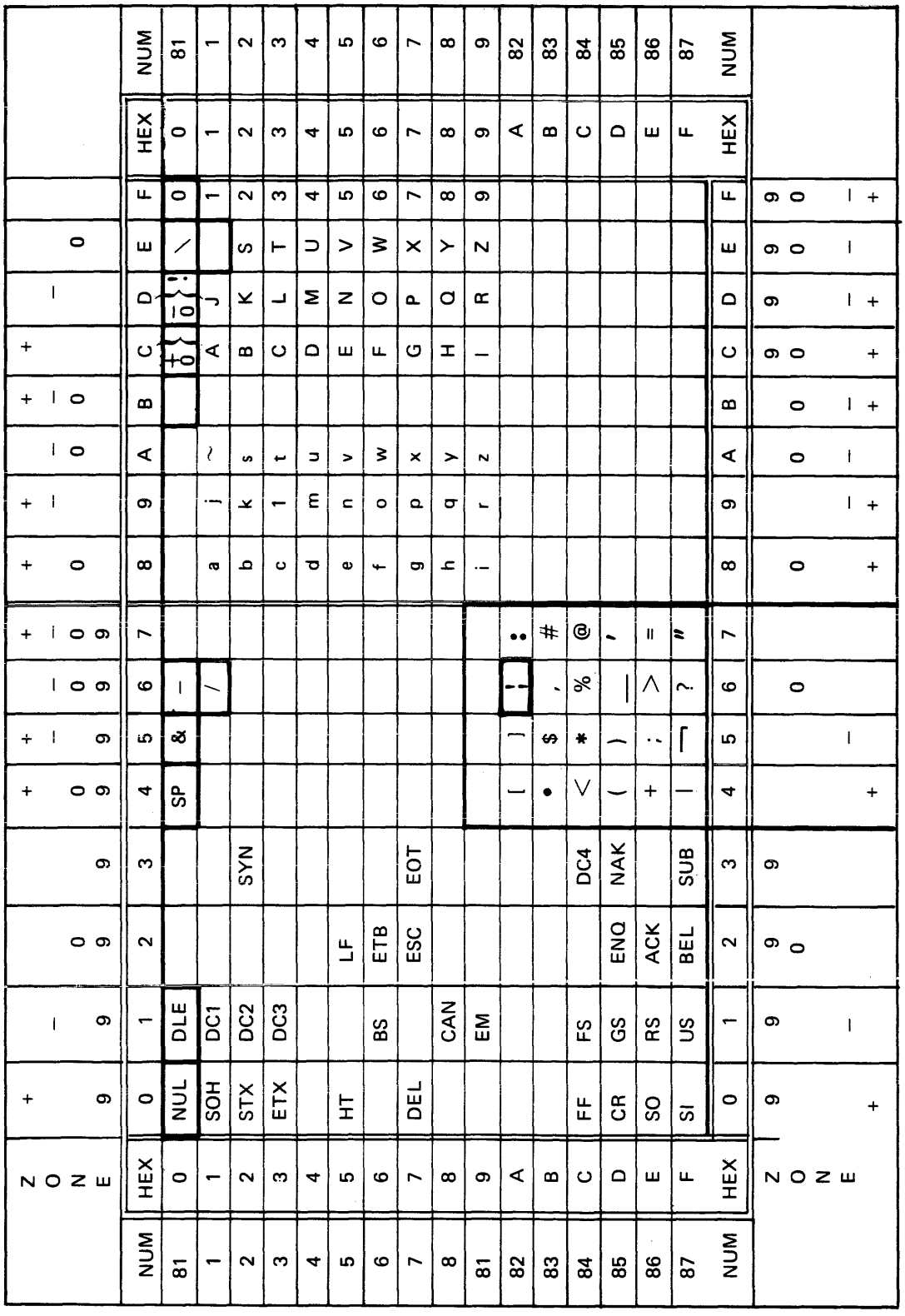
EBCDIC Graphic	Decimal Value	EBCDIC Internal	Hex. Graphic	EBCDIC Card Code	Octal
BLANK	64	0100 0000	40	No Punches	60
[	74	0100 1010	4A	12 8 2	33
.	75	0100 1011	4B	12 8 3	32
<	76	0100 1100	4C	12 8 4	36
(	77	0100 1101	4D	12 8 5	35
+	78	0100 1110	4E	12 8 6	
|	79	0100 1111	4F	12 8 7	37
&	80	0101 0000	50	12	34
]	90	0101 1010	5A	11 8 2	76
$	91	0101 1011	5B	11 8 3	52
*	92	0101 1100	5C	11 8 4	53
)	93	0101 1101	5D	11 8 5	55
;	94	0101 1110	5E	11 8 6	56
	95	0101 1111	5F	11 8 7	57
-	96	0110 0000	60	11	54
/	97	0110 0001	61	0 1	61
,	107	0110 1011	6B	0 8 3	72
%	108	0110 1100	6C	0 8 4	73
_	109	0110 1101	6D	0 8 5	74
>	110	0110 1110	6E	0 8 6	16
?	111	0110 1111	6F	0 8 7	14
:	122	0111 1010	7A	8 2	15
#	123	0111 1011	7B	8 3	12
@	124	0111 1100	7C	8 4	13
'	125	0111 1101	7D	8 5	17
=	126	0111 1110	7E	8 6	75
"	127	0111 1111	7F	8 7	77
(+)PZ	192	1100 0000	C0	12 0	20
A	193	1100 0001	C1	12 1	21
B	194	1100 0010	C2	12 2	22
C	195	1100 0011	C3	12 3	23
D	196	1100 0100	C4	12 4	24
E	197	1100 0101	C5	12 5	25
F	198	1100 0110	C6	12 6	26
G	199	1100 0111	C7	12 7	27
H	200	1100 1000	C8	12 8	30
I	201	1100 1001	C9	12 9	31
(!)MZ	208	1101 0000	D0	11 0	40
J	209	1101 0001	D1	11 1	41

*All other codes

EBCDIC Graphic	Decimal Value	EBCDIC Internal	Hex. Graphic	EBCDIC Card Code	Octal
K	210	1101 0010	D2	11 2	42
L	211	1101 0011	D3	11 3	43
M	212	1101 0100	D4	11 4	44
N	213	1101 0101	D5	11 5	45
O	214	1101 0110	D6	11 6	46
P	215	1101 0111	D7	11 7	47
Q	216	1101 1000	D8	11 8	50
R	217	1101 1001	D9	11 9	51
¢	224	1110 0000	E0	0 8 2	
S	226	1110 0010	E2	0 2	62
T	227	1110 0011	E3	0 3	63
U	228	1110 0100	E4	0 4	64
V	229	1110 0101	E5	0 5	65
W	230	1110 0110	E6	0 6	66
X	231	1110 0111	E7	0 7	67
Y	232	1110 1000	E8	0 8	70
Z	233	1110 1001	E9	0 9	71
0	240	1111 0000	F0	0	00
1	241	1111 0001	F1	1	01
2	242	1111 0010	F2	2	02
3	243	1111 0011	F3	3	03
4	244	1111 0100	F4	4	04
5	245	1111 0101	F5	5	05
6	246	1111 0110	F6	6	06
7	247	1111 0111	F7	7	07
8	248	1111 1000	F8	8	10
9	249	1111 1001	F9	9	11

APPENDIX D

B 6900 EBCDIC/HEX CARD CODE



Use of the B 6900 EBCDIC/HEX Card Code Chart.

a. Locate the desired EBCDIC graphic code within the table.

b. The two-part hexadecimal code is read as follows:

1. The first part is found in the vertical column above or below the desired EBCDIC code.

2. The second part is found in the horizontal row either to the right or left of the desired EBCDIC code.

(a) Examples:

SYN = 32
F = C6

c. The two-part card code is found in the same manner as HEX (2) except the zone and numeric bits are read from the very outer portion of the table.

1. Examples:

SYN = 9 2
F = + 6

2. The card code exceptions to the above procedures are enclosed in heavy lines on the chart and are defined below:

(1) 00 = + 0981 (NUL)
(2) 10 = + -981 (DLE)
(3) 20 = -0981
(4) 30 = + -0981
(5) 40 = BLANK
(6) 50 = + (&)
(7) 60 = - (-)
(8) 70 = + -0
(9) CO = + 0 ({) ($\overset{+}{\text{o}}$)
(10) DO = - 0 (}) ($\bar{\text{o}}$)
(11) EO = 0 82 (\)
(12) FO = 0 (0)
(13) 61 = 0 1 (/)
(14) E1 = -09 1
(15) 6A = + - (¦)

INDEX

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

Documentation Evaluation Form

Title: B 6900 System Reference Manual

Form No: 5010986

Date: July 1981

Burroughs Corporation is interested in receiving your comments and suggestions regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

From:

Name

Title

Company

Address

Phone Number Date

Remove form and mail to:

Burroughs Corporation
Documentation Dept., TIO - West
P.O. Box 4040
El Monte, CA 91734
U.S.A.

Printed in U.S.America July 1981 Form 5010986