W I C A T

System 150

(mapped)

Hardware Reference Manual

July 1982

WICAT SYSTEMS INCORPORATED Orem, Utah

187-055-201 A

Copyright Statement

Receipt of this manual must not be construed as any kind of commitment, on the part of WICAT Systems Incorporated, regarding delivery or ownership of items manufactured by WICAT.

This manual is subject to change without notice. A system software subscription entitles you to receive all bulletins and revised editions of manuals pertaining to your WICAT system. Call WICAT Systems's main office for information on this service.

WARNING: The equipment described in this manual generates, uses, and can radiate radio frequency energy, and if not installed in accordance with instructions provided in the hardware documentation for the equipment, may interfere with radio communications. Furthermore, the equipment has been tested and found to comply with the limits for a Class A computing device pursuant to subpart J, Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when the equipment is operated in a commercial environment. Operation of the equipment (described in this manual) in a residential area is likely to cause interference. Where the equipment will be used in a residential area, it is the user's responsibility to ensure that any interference is corrected.

Revision History

First Printing July 1982

The Purpose of This Manual

This document provides the technical information users will need should they want to modify System 150 (mapped) PC boards to interface with peripheral devices that are not supplied by WICAT Systems Inc.

Furthermore, WICAT Systems Inc. assumes no responsibility for difficulties arising from the use of equipment not manufactured by WICAT.

Intended Audience

Field service technicians, OEM representatives, and maintenance personnel on Systems 150 (mapped). Readers must be knowledgable in electronics, and familiar with the basic terminology of computer science.

CHAPTER 1 SYSTEM OVERVIEW

CHAPTER 2 CPU BOARD

CHAPTER 1

SYSTEM OVERVIEW

1.1 INTRODUCTION

This chapter serves as an overview of the WICAT System 150 (mapped) computer system. The information contained herein is for use by field service technicians.

1.2 DEFINITION AND FEATURES

The WICAT System 150 (mapped) is a 68000-based microcomputer system with mainframe capabilities. A single desk-top unit (with detachable keyboard) contains:

- Central Processing Unit (CPU) (Section 1.6.1)
- Memory (Section 1.6.2)
- CRT (terminal screen) (Section 1.6.3)
- Storage and storage backup (Section 1.6.4)
- Several standard and optional peripherals (Section 1.6.5)

1.3 SYSTEM CONFIGURATION

Refer to Figure 1-1.

The PC boards are held in place by the card cage located directly behind the CRT. At the base of the card cage is the motherboard containing six slots into which the boards are inserted.

The standard 150 multiuser configuration leaves two slots empty, but

upon request these slots can be used for two additional memory boards, or one additional memory board and a videodisc controller board.

To the left side of the card cage (with the CRT nearest you) and running perpendicular to the card cage are the graphics terminal boards (connected to each other by a ribbon cable) or a non-graphics terminal board.

To the right side of the CRT are located the Winchester disk and floppy disk drives.

The System 150 has a detachable keyboard connected to the terminal by a coiled cable.

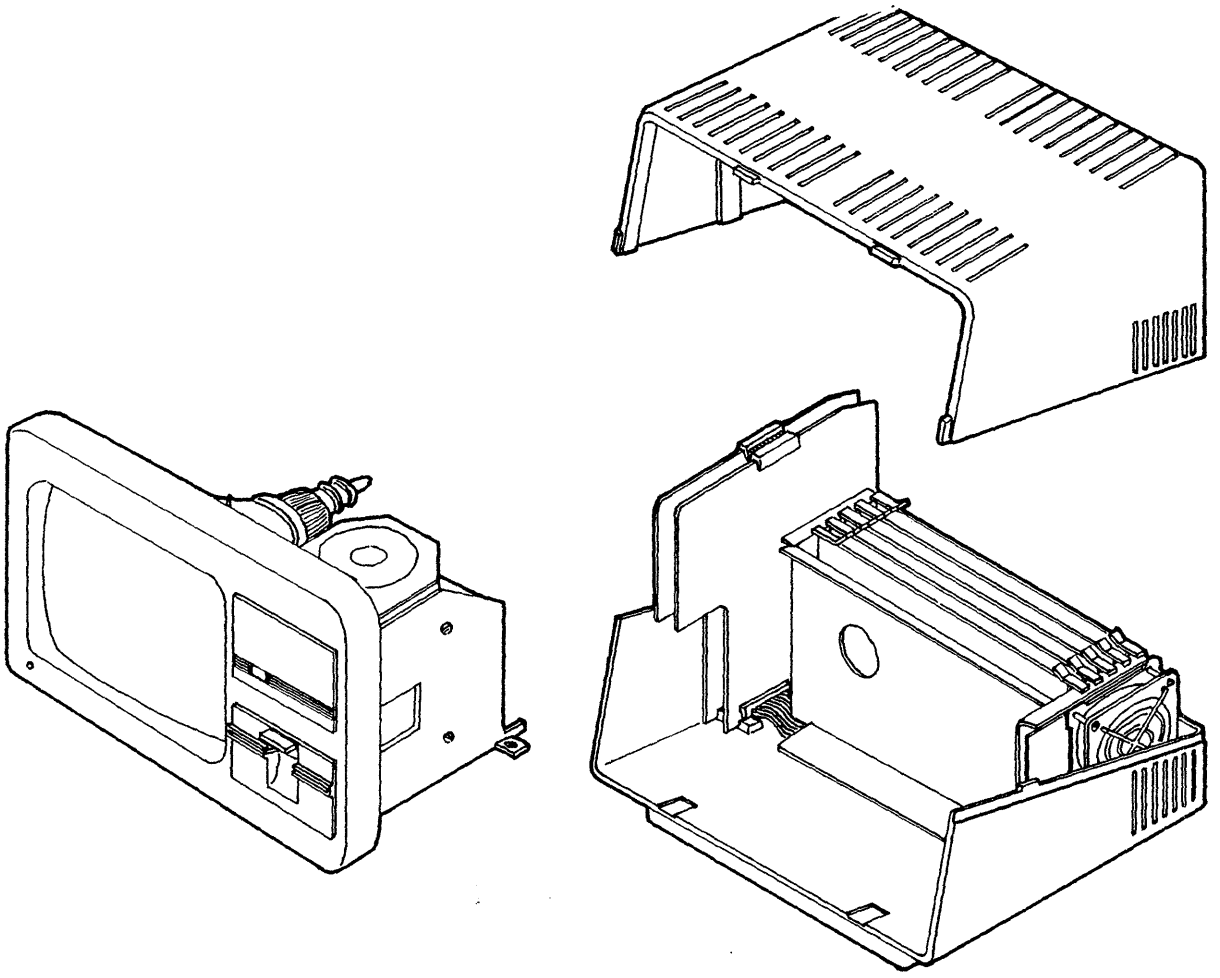


Figure 1-1
System 150 Multiuser (Exploded View)

WARNING

When removing the WD1000/Coupler Board, please exercise extreme caution to avoid damaging the end of the CRT.

1.4 BOARD CONFIGURATION

Refer to Figure 1-2.

The standard CPU chassis contains the following boards. (Indicated in parentheses immediately following the board name are later chapters that give detailed explanations):

- CPU (Chapter 2)
- I/O (Chapter 3)
- Memory Board (Chapter 4)
- WD 1000 and WD1000 Coupler/Floppy Disk Controller (Chapter 5)

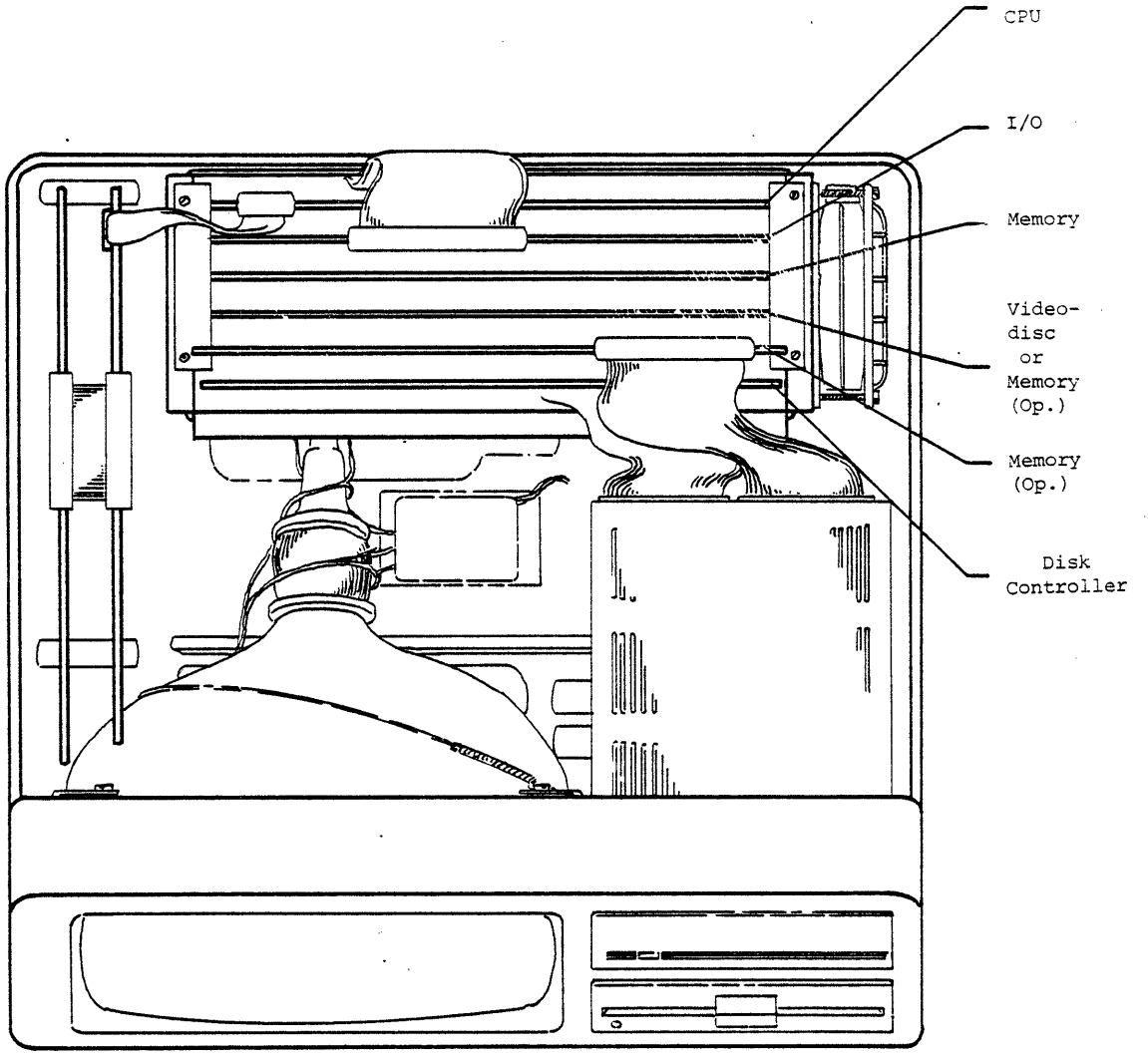


Figure 1-2
Placement of Boards in CPU Chassis

1.5 BOARD INTERACTION

The **bus structure** amounts to a common group of circuit paths over which input and output signals are routed. This structure enables communication between the CPU board and the other PC boards.

1.5.1 Explanation Of Bus Structure

The PC boards and the CPU board are interconnected by the backplane, or motherboard, in the base of the chassis. The CPU board handles bus control.

There are three types of buses: address buses, which consist of all the signals needed to define any of the possible memory or I/O locations in the system; data buses, which handle all communication of instructions and data; and control buses, which are used by the CPU to direct the action of the other elements in the system.

1.5.2 Bus Control

The MC68000 microprocessor allocates CPU time for requesting processes and devices. The basic theory of allocation is simple. First, the CPU receives a request signal for bus use. Second, based on internally defined priorities, the request is either granted immediately or delayed until previous or higher priority requests are processed. Finally, when the request is granted, the requesting bus device returns an acknowledgement signal to the CPU and the cycle continues.

Once a device or process has been granted bus use, it is called the "Bus Master". (WICAT systems currently allow only one bus master at a time.) A device called the "slave" then receives or transmits data from or to the bus master.

1.6 SYSTEM DATA FLOW

As shown in Figure 1-3, information is input to the System 150 through the keyboard. A serial link transfers the data to the terminal CPU, located on one of the boards perpendicular to the card cage (see Figure 1-1). The CPU is connected to the P1 connector on the I/O board by a ribbon connector.

The I/O board now acts as a liaison between the terminal section of

the System 150 and the other system boards. It sends data to the other boards as required through the circuitry of the IEEE 796 bus. Connector P5 connects the additional serial and parallel ports on the I/O board to the connector panel at the back of the chassis.

The I/O board has two other connectors (P4 and P6) that can be connected to peripheral devices. Connector P4 is configured as a full-handshake modem port and connector P6 is used for the optional IEEE 488 (GPIB) bus interface. See Figure 3-1 for connector locations on the I/O board.

The disk controller board controls the Winchester disk drive and the floppy disk drive.

The memory board on all standard 150's handles temporary data storage.

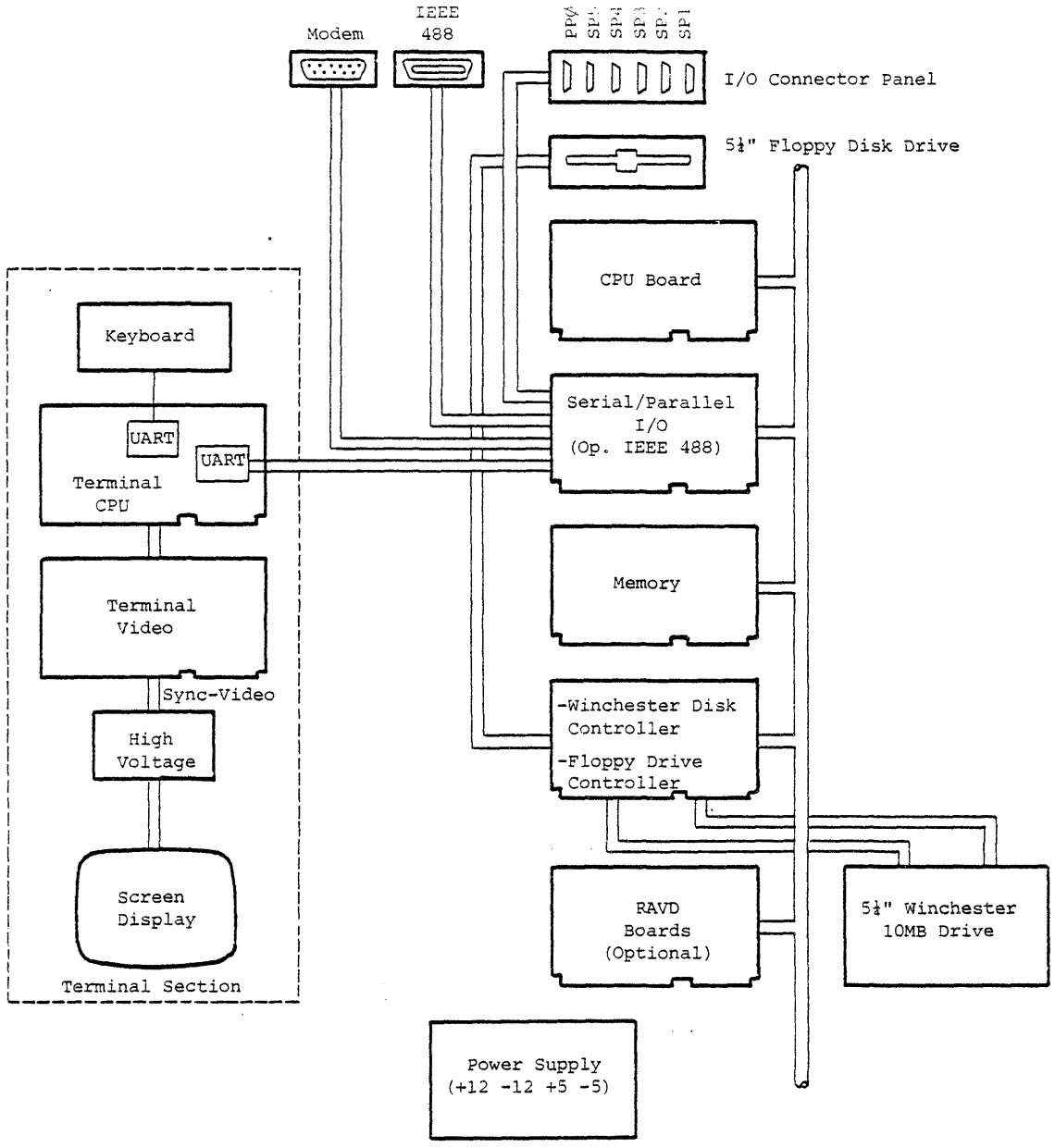


Figure 1-3
System 150 Multiuser Data Flow Chart

1.6.1 Central Processing Unit (CPU)

The processor for the System 150 (mapped) is the Motorola 68000 microprocessor that runs at 8 megahertz (MHz) and executes approximately one million instructions per second. Thirty-two bit internal registers support 32-bit data operations.

See Chapter 2 for more information on the CPU.

1.6.2 Memory

The System 150 (mapped) is equipped with a single memory board containing 256 Kilobytes (Kbytes) of Dynamic Random Access Memory (DRAM) that is expandable to 512 Kbytes by fully populating the board. Two additional memory boards may be inserted into the chassis increasing the available memory space to 1.5 megabytes (Mb).

See Chapter 4 for more information.

1.6.3 CRT

The CRT has a resolution of 300 x 400 pixels, allowing medium-resolution graphics when using the graphics option. Each character is 7 X 9 pixels.

1.6.4 Storage

Mass storage includes a 10 Mb 5-1/4 inch Winchester disk drive and a 5-1/4 inch floppy disk drive for backup and file portability purposes.

1.6.5 Peripherals

The standard and optional peripherals available on the System 150 (mapped) are:

- 5 RS232 serial interfaces (Section 1.6.5.1)
- 16-bit parallel interface (Section 1.6.5.2)
- Battery-backed clock (Section 1.6.5.3)

- Graphics circuitry (optional) (Section 1.6.5.4)
- Videodisc controller (optional) (Section 1.6.5.5)
- IEEE 488 interface (optional) (1.6.5.6)

1.6.5.1 Serial Interfaces -

These interfaces are used as input/output ports for various peripheral devices such as printers, terminals, etc. The interfaces are located on the I/O board (see Chapter 3) and conform to the standard RS232C to ensure asynchronous data transfers.

1.6.5.2 Parallel Interface -

This is a 16-bit parallel port organized as two 8-bit bi-directional ports and set up to act as a standard Centronics interface.

1.6.5.3 Battery-backed Clock -

This clock, located on the I/O board, is a real time calendar clock for the system. It continues to keep time, in the event of a system failure or power-down, with power supplied by an on-board battery.

1.6.5.4 Graphics Circuitry (Optional) -

Two graphics boards located perpendicularly to the CRT allow system graphics to be available as an option.

1.6.5.5 Videodisc Controller (Optional) -

To make the System 150 (mapped) compatible with a videodisc player, a printed circuit (PC) board configured for videodisc control is inserted into the first of the available slots in the chassis (see Figure 1-2).

This option allows the System 150 to interface with a videodisc player's parallel control port.

1.6.5.6 IEEE 488 Interface (Optional) -

Additional components on the I/O board (see Chapter 3 and Figure 3-1) make up this option. The IEEE 488 interface is a standard General Purpose Interface Bus (GPIB), allowing the use of other external peripherals.

CHAPTER 2

CPU BOARD

2.1 INTRODUCTION

This chapter explains the components of the CPU board and focuses on the operations performed by the microprocessor.

2.2 DEFINITION AND FEATURES

The CPU board handles the central processing of the System 150 (mapped) and controls all bus use requests. The key component on the board is the Motorola 68000 microprocessor. The CPU board has the following features:

- Memory map (Section 2.5)
- Error control (Section 2.6)
- On board peripherals and memory (Section 2.7)
- Address decoding (Section 2.8)

2.3 CPU BOARD CONFIGURATION

Refer to Figure 2-1.

The CPU board comprises five main areas of circuitry:

- Multibus Interface (Section 2.2.1)
- On-board ROM (Section 2.2.2)
- Interrupt Circuitry (Section 2.2.3)
- Bus Arbitration Circuitry (Section 2.2.4)
- Memory Mapping Registers (Section 2.2.5)

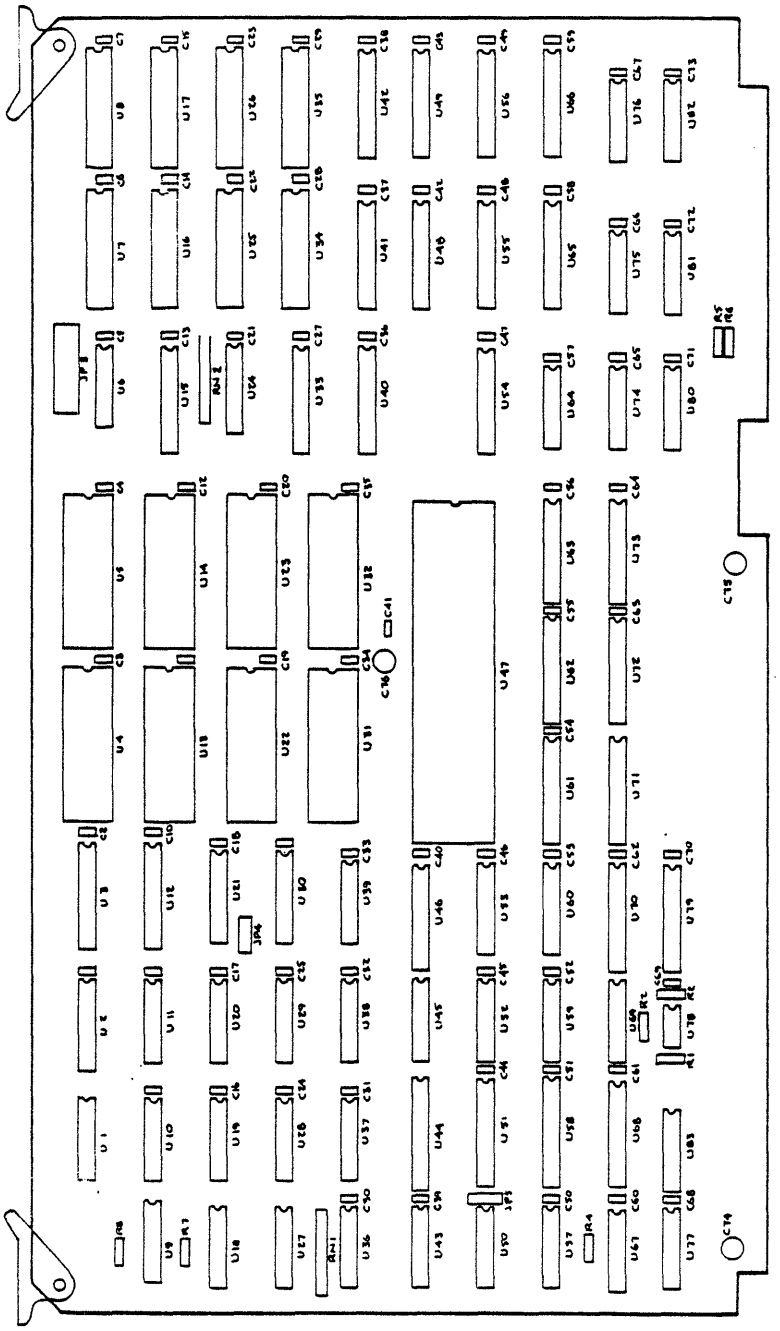


Figure 2-1
CPU Board Circuitry

The circuitry of the CPU board is divided into two principal functions: the CPU bus-associated functions and some system I/O functions. The connectors are the system bus proper. They contain paths and connections for the system data, address, and control.

2.3.1 The MULTIBUS Interface

The MC68000 microprocessor and the INTEL Multibus are incompatible because the Microprocessor defines the lower byte of the data bus as odd and the upper byte of the data bus as even; this data byte definition is reversed on the Multibus. An interface in the form of **byte swap buffer** ensures that odd and even single-byte transfers always take place on the low order data lines. Thus, the CPU board conforms to the INTEL MULTIBUS standard for both eight- and sixteen-bit systems. Word transfers use the full set of sixteen data lines.

CAUTION

When writing a byte to a peripheral device the least significant address bit must be complemented or inverted by the software because of the MC68000 to MULTIBUS incompatibilities.

2.3.2 On Board ROM

The CPU board can support up to 64K bytes of on board Read Only Memory (ROM). ROM address decoding is done through a 74S288 PROM. ROM sizes of 2K X 8, 4K X 8, or 8K X 8 can be accommodated by changing two jumpers and the address decode PROM. The CPU board accepts 2K X 8 EPROMs as standard. The CPU board will accept the other two types of EPROMs if you change the jumpers as specified in Figure 2-2.

EPROM SIZE 4K X 8

Jumper		IN/OUT
JP1	A	OUT
	B	IN
JP2	A	IN
	B	OUT

EPROM SIZE 8K X 8

Jumper		IN/OUT
JP1	A	OUT
	B	IN
JP2	A	OUT
	B	IN

Figure 2-2
Jumper Configuration for Larger EPROMs

2.3.3 Interrupt Circuitry

The CPU board supports seven levels of interrupts, INT0/ through INT6/, with INT0/ having the highest priority. INT0/ is the only nonmaskable interrupt. All interrupts are autovectored to addresses designated by the MC68000. See Section 2.3.5 and the appropriate section of the Motorola's MC68000 16-Bit Microprocessor User's Manual, January 1980.

2.3.4 Bus Arbitration Circuitry

The CPU bus arbitration scheme conforms to the MULTIBUS specification for serial priority bus arbitration. When a MULTIBUS device wants control of the bus it checks its BPRN/ signal. If the BPRN/ signal is low, the Multibus sets its BPRO/ signal high. The high signal disables all lower priority bus requests and sends the MULTIBUS device's request to the CPU. The arbitration sequence is then as follows:

1. The CPU finishes executing the current instruction and sets the BUSY/ line high to tell the requesting device that it can take control of the bus.

2. The requesting device then pulls BUSY/ low and takes control of the bus.

3. When finished with the bus, the requesting device sets its BPRO/ line low, then sets the BUSY/ line high.

4. If no bus requests are pending, the CPU regains control of the bus.

All bus arbitration signals are synchronized with the falling edge of BCLK. Table 2-1 is an address map.

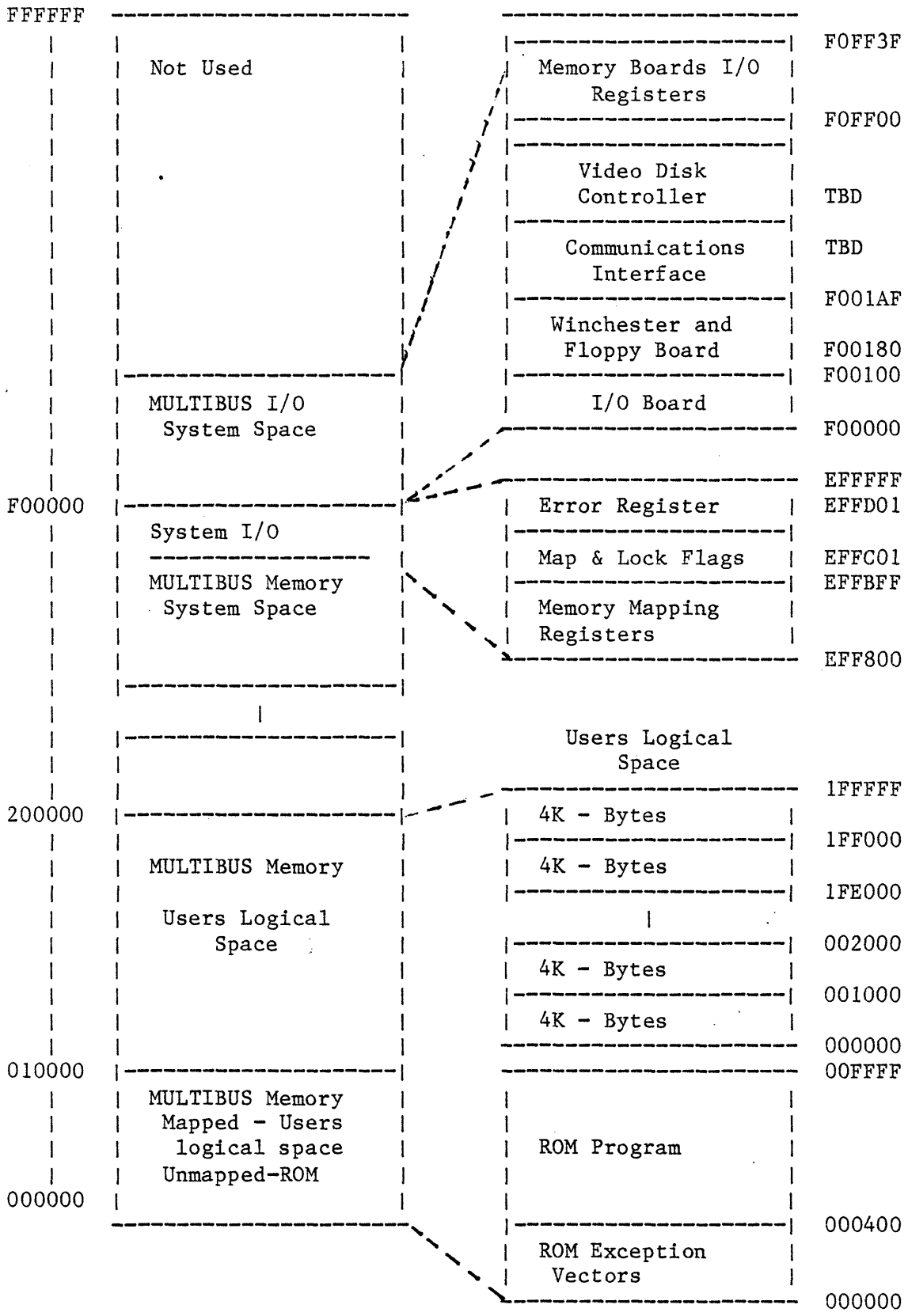


Table 2-1
Memory Address Map

2.3.5 Memory Mapping Registers

1k of high speed static RAM at address EFF800-EFFBFF on the CPU board is available for memory mapping. Address lines A12-A20 are the inputs to the RAMs. The data lines out of the RAMs become the new lines A12-A23. The mapping function is enabled by writing an 80 to EFFC01 and disabled by writing a 00 to EFFC01. This sets and clears a bit addressable latch U2 on the CPU board.

2.4 PROCESSOR CIRCUITRY

The CPU board contains the Motorola MC68000 Microprocessor, which directs control, logic, and arithmetic operations required of the **SYSTEM 150 (mapped)**. The processor circuitry is comprised of the microprocessor, its associated buffers, system clocks; and, bus error, interrupt, and arbitration logic.

Input and output signals of the microprocessor fall into the groups shown in Figure 2-3.

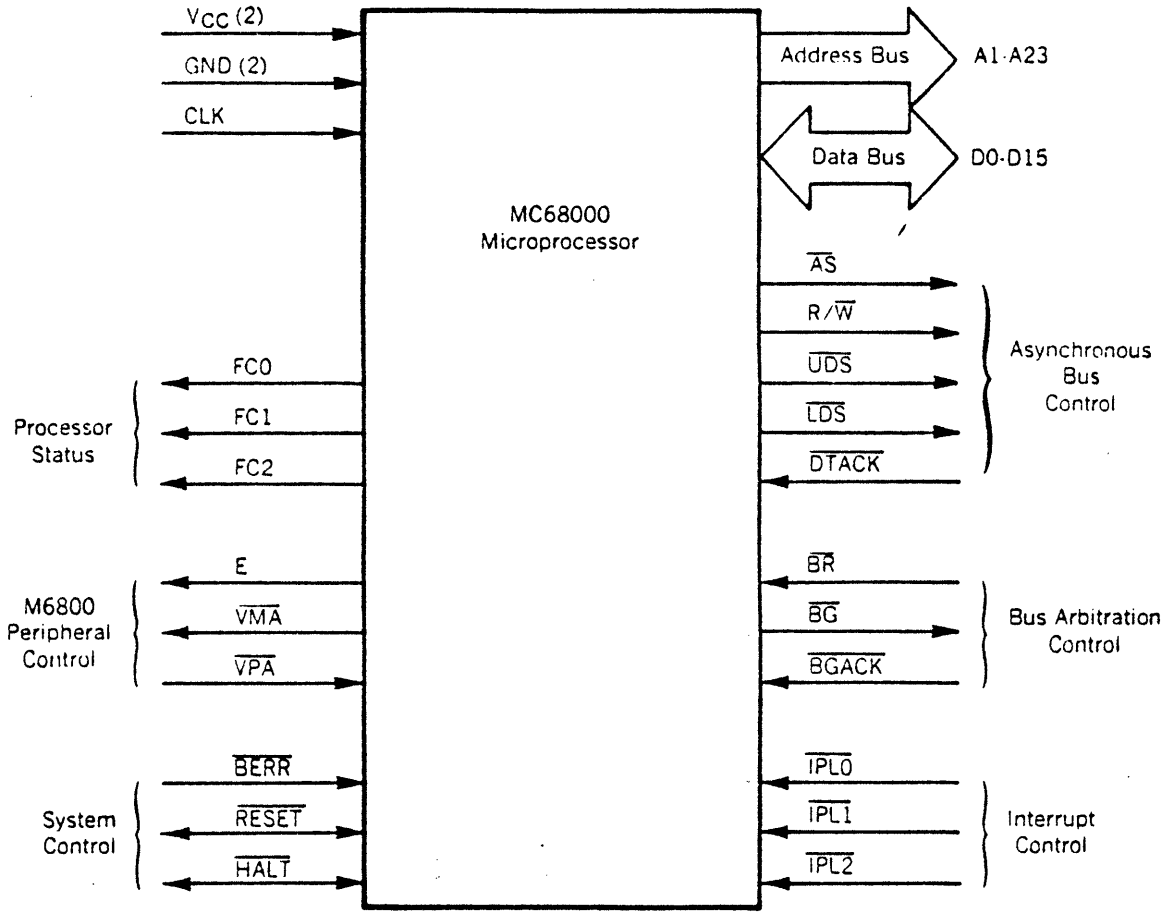


Figure 2-3
Input and Output Signals

2.4.1 Address Bus

The microprocessor uses a 23-bit address bus to select one of eight two-byte megawords. The lower 11 address lines are directly buffered onto the bus connector. Normally, the upper 12 lines are routed directly from the 68000 to the bus connector.

However, the upper 12 lines may be treated as a logical address and the bus transaction may be steered by the mapping registers into a physical location when the processor is in user state, or in supervisor state with memory mapping flag set and the address is less than one megaword (= two megabytes).

The address bus is asserted LOW at the bus connector and is the logical inversion of the address bus at the microprocessor. The address lines from the CPU board are put into a high impedance state when another master controls the bus.

2.4.2 Data Bus

The MC68000 uses a 16-bit data bus to transfer programs or data. The data bus is buffered at the bus connector.

The most significant data bits (D8-15) are the odd bytes, and the least significant bits (D0-D7) are the even bytes. When the processor executes a byte operation, the least significant address bit--A0 is determined from the upper and lower data strobes (see Section 2.3.3). This process determines whether the processor will transfer an even or an odd byte.

Operating on a word or an even address boundary will produce a different effect than operating on a byte at the same address. Executing a word instruction on an odd boundary causes an illegal address trap.

You may connect an eight-bit peripheral to the data bus with the byte access option or the word access option.

1. Byte Access Option

   The data lines can be connected to the least significant eight data bus lines (D0-17) and all accesses to the peripheral may then be byte or word accesses.

2. Word Access Option

   The peripheral data lines can be connected to the most significant eight data bus lines (D8-15) and

all accesses must then be word accesses. A word access will access the odd and the even bytes. A byte instruction may not be used to access the peripheral when connected to the most significant eight data lines.

The data bus is asserted LOW at the bus connector. When another master has control of the bus, the CPU board bus buffers are placed in a high impedance state.

2.4.3 Synchronous Bus Control

NOTE

The following information regarding bus control concerns the 796 bus.

Five control signals listed here coordinate data transfer on a synchronous bus.

1. Memory Read Control (MRDC/)
2. Memory Write Control (MWTC/)
3. I/O Read Control (IORC/)
4. I/O Write Control (IOWC/)
5. Transfer Acknowledge (XACK/)

2.4.3.1 Command Lines (MRDC/, MWTC/, IOWC/, And IORC/) -

These command lines are communication links between the bus masters and bus slaves. There are four command lines for memory and I/O reads and writes. An active command line indicates to the slave that the address lines are carrying a valid address, and that the slave should perform the specified operation.

2.4.3.2 Transfer Acknowledge Line (XACK/) -

This line is the slave's acknowledgement of the master's command. XACK/ indicates to the master that the requested action is complete, and that data has been placed on or accepted from the data lines.

2.4.4 MG68000 Bus Control

Address Strobe (AS L)

AS L is a control signal asserted when the address on the bus is stable and considered valid. It will remain asserted until Data Transfer ACKnowledge (DTACK L) is asserted in response. If the addressed device is not attached or occupied, the level of the Bus ERRor line changes from high to low. The Error Control Circuitry disables AS L before it reaches the bus connector if:

a. an access violation occurs in the current cycle.

b. a parity error occurs in the previous cycle.

Data Transfer ACKnowledge (DTACK)

The device currently being addressed asserts DTACK when that device is ready to terminate the bus transaction. DTACK is negated in response to the negation of Address Strobe.

Upper and Lower Data Strobe (UDS L and LDS L)

The CPU uses UDS L and LDS L to select one or both of the bytes from the word currently being addressed by the address bus.

Read/Write (R/W)

R/W determines the direction of the data transfer. When LOW, data are moving TO memory. When HIGH, data are moving away FROM memory.

2.4.5 Bus Arbitration Control

2.4.5.1 **Single Device Arbitration -**

The MC68000 uses three lines to arbitrate bus use among devices:

1. **Bus Request** (BR L) - an input signal that can be driven by any number of devices wired-OR devices.

2. **Bus Grant** (BG L) - an output signal indicating that the requesting device may use the bus after the current bus transaction finishes.

3. **Bus Grant ACKnowledge** (BGACK L) - when asserted at the end of the bus cycle, BGACK L allows the requesting device to become bus master under three conditions:

   a. Address strobe and DTACK are negated

   b. No other device is using the bus

   c. The bus request is negated

2.4.5.2 **Multi-Device Arbitration -**

Three MSI logic devices on the CPU board automatically prioritize bus requests and grants. On the left (or I/O) connector are eight bus request lines (BR0-7) and eight corresponding bus grant lines (BG0-7). BR7 has the highest priority.

Requests for bus use are prioritized depending on the bus request line used for input. The number of the highest request is latched onto the falling edge of bus grant from the processor. The outputs of this latch then drive the inputs to a three-to-eight line decoder that selects the highest priority bus grant line. Thus, a device simply asserts its bus request line and waits until its bus grant line is asserted. When the device recognizes its bus grant, it waits until the end of the current bus cycle to verify that BGACK is not asserted. After the verification the device asserts BGACK, negates its bus request, and becomes bus master.

If more than one bus request is pending, the arbitration logic selects the bus grant corresponding to the highest priority bus request. After the device with the highest priority has become bus master, the arbitration logic will select the bus grant corresponding to the bus request with the next highest priority.

When the current bus master is through with the bus, the next device immediately takes control. The arbitration logic then selects the next device with the highest priority. Thus, the arbitration logic manipulates the queue according to device priority.

2.4.6 Interrupt Control

The MC68000 supports seven interrupt levels. Level seven is nonmaskable and is the highest priority.

Interrupts are actually a subset of a more general class of operations called exceptions. Interrupts are either autovectored or nonautovectored but the CPU board circuitry simplifies the interrupt process on autovectored interrupts:

a. **autovectored** - The processor generates the interrupt vector number internally, as a function of the interrupt level.

b. **nonautovectored** - The processor reads the interrupt vector number from the interrupting device.

In each case the processor executes an interrupt acknowledge sequence. During this sequence, the function codes indicate an interrupt acknowledge cycle, the address lines A4-23 are set to 1's, and the interrupt level is placed on address lines A1-3. If Valid Peripheral Address, VPA L, (2.3.7.[b]) is asserted during this bus cycle, the interrupt will be interpreted as autovectored, in which case the bus cycle emulates a MC6800 cycle. No DTACK will be expected and the transaction will terminate after approximately 1 microsecond. The interrupting device need not respond because the processor generates its own interrupt vector number based on the level of the interrupt.

If VPA is not asserted during the interrupt acknowledge sequence, the interrupt is considered nonautovectored, and the interrupting device responds with a vector number on data bus lines D0-7 (odd byte). The upper byte is ignored. This transfer takes place just as a normal read operation does. The interrupting device asserts DTACK when the data have been placed on the bus. This number is then multiplied by four to obtain the address of the interrupt vector. **Figure 2-4** is a photographic recording of signal activity during an autovectored interrupt sequence.

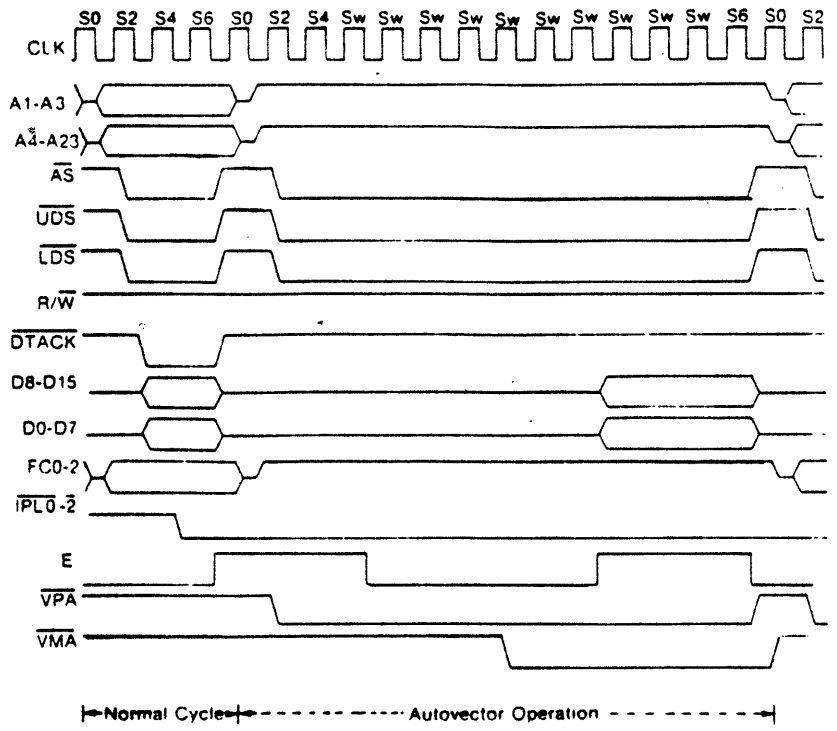


Figure 2-4
Signal Activity During an
Autovectored Interrupt Sequence

After the interrupt vector number is obtained, either by internal generation or by external read, the processor saves the status and return address on the system stack and then uses the vector number to access the interrupt vector (interrupt handler routine address) from memory. The processor then continues execution at the interrupt handler routine.

Eighteen wire-wrap pins on the high end of the left (I/O) connecter (location M9) are arranged in six equivalent groups. Each group configures the type of interrupt for levels one through six. Level seven is reserved for the error control circuitry as described in section 2.5. Level six is the leftmost group and level one is the rightmost group of pins, as viewed from the connector edge of the board.

Jumpering the center pin of any group to the pin above it (away from the connector) defines that interrupt level to be autovectored. Jumpering to the pin below it (toward the connector), defines it as nonautovectored.

The example in **Figure 2-2** shows levels six, five, and two defined as autovectored, and levels four, three, and one defined as nonautovectored. The system leaves the factory with all levels defined as autovectored.

If an interrupt level has been defined as autovectored, VPA will be automatically asserted during an interrupt acknowledge bus cycle. If the level is defined as nonautovectored, VPA will not be asserted during interrupt acknowledge.

NOTE

IF VPA IS ASSERTED during the interrupt acknowledge bus cycle, the processor still executes an autovectored interrupt sequence even if the interrupt level is jumpered to be nonautovectored. An interrupting device connected to a nonautovectored interrupt level may still cause an autovectored sequence by asserting VPA during interrupt acknowledge.

2.4.7 System Control

System control comprises three lines, RESET (RESET L), HALT (HALT L), and Bus ERRor (BERR L). RESET and HALT are bidirectional. On power-up and during manual reset conditions, both the RESET and HALT lines are driven as inputs for a minimum of 200 mSec. In no other case is HALT or RESET driven as an input. The processor asserts RESET while executing a reset instruction and asserts HALT when the processor is halted.

A high-to-low transition on the Bus ERRor line tells the processor that a specified time has elapsed without a DTACK response to the assertion of Address Strobe. The DIS CPU implements the Bus ERRor timer with a presettable counter. This counter is clocked at 4 MegaHertz, but is normally held in the clear (all zeroes) condition.

When Address Strobe is asserted, the counter is allowed to begin. After 16 clock cycles, the ripple carry output causes the assertion of Bus ERRor. If, however, DTACK is asserted before the 16 counts can occur, the counter is stopped, and the negation of Address Strobe will again hold the counter in the clear state. The 16 counts with a clock cycle time of 250 nS provides a bus error time out of 4.0 uS. This value is nonadjustable.

Figure 2-5 is a record of a bus error sequence.

When an access or parity error occurs, address strobe is not asserted on the bus. Although the processor expects a DTACK, it receives none. Normally, the bus error timer times out after 4.0 uS and causes the assertion of Bus ERRor.

Because this causes an erroneous bus trap, the circuitry can detect a bus error time out resulting from the blocking of the address strobe. In this case, Bus ERRor is not asserted and the bus error timer overflow latches the error conditions and generates a DTACK so that the processor can finish its bus cycle.

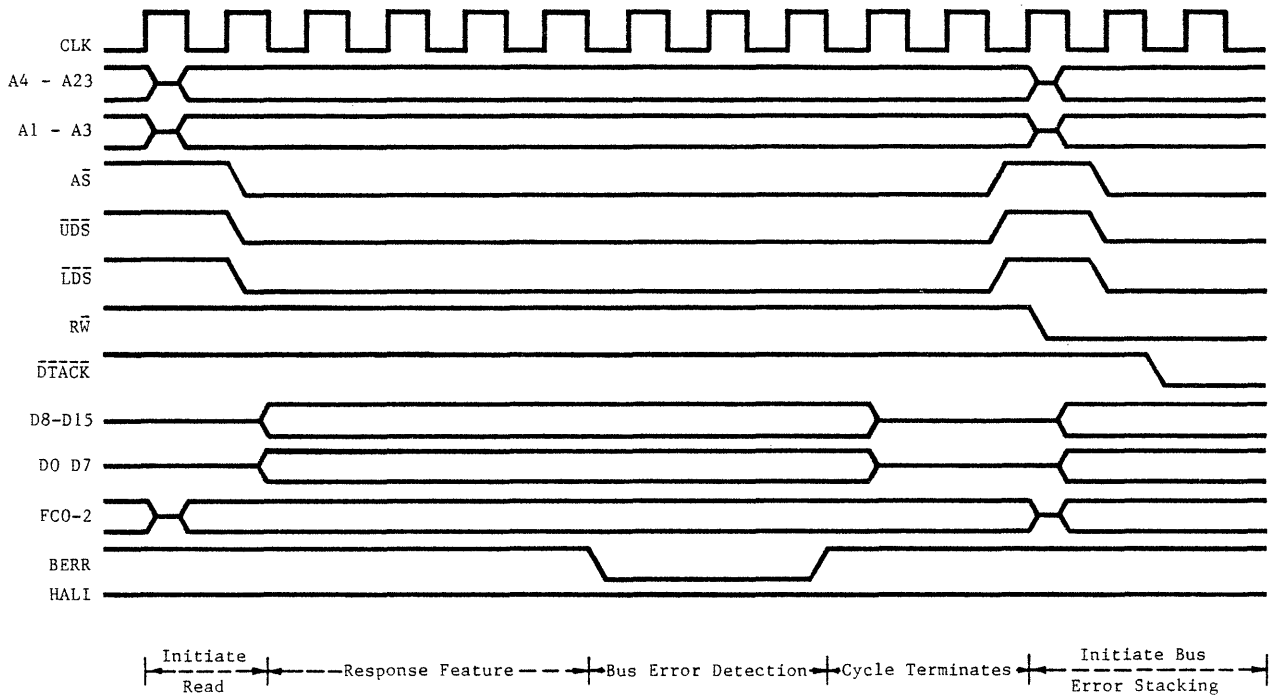


Figure 2-5
Bus Error Sequence

2.4.8 MC68000 Peripheral Control

The MC68000 provides three lines for interfacing MC6800 peripheral devices:

a. Enable (E)- a clock signal that synchronizes transactions between the processor and MC6800 type peripherals. The clock period is ten times the system clock (10 x 125nS = 1.25uS) and has a 60/40 duty cycle (6 clocks low, 4 clocks high).

b. Valid Peripheral Address (VPA L) - asserted by the peripheral device when it recognizes its address on the address bus. VPA L is also used to distinguish between auto and nonautovectored interrupts during an interrupt acknowledge sequence.

c. Valid Memory Address (VMA L) - asserted by the processor in response to the assertion of VPA L during an MC6800 peripheral data transfer.

E and VMA L are buffered onto the bus. VPA L is a shared tri-state input to the CPU board.

2.4.9 Processor Status

The values of the function code lines (FC0-2) determine the status of each MC68000 bus cycle. See **Figure 2-6** for these values.

FC2	FC1	FC0	Cycle Type
0	0	0	(Undefined, Reserved)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Undefined, Reserved)
1	0	0	(Undefined, Reserved)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

Figure 2-6
Function Code Line Values

As seen in the Figure 2-6, whenever FC2 is zero, the processor is in user mode. The memory mapping logic uses FC2 to determine when to map the processor address. The interrupt acknowledge condition is decoded on the CPU board and buffered, along with the values of the function codes, out onto the bus.

2.5 MEMORY MAP

If mapping occurs the original upper three addresss bits (A21-23) are zeroes (users are confined to a one-megaword address space). The next nine address bits (A12-20) will be used to access one of 512 locations of the memory mapping registers. These registers are 16 bits wide, containing 12 bits of new address and three bits of access control information. One bit is not used. Each register location within the memory map represents a 2K word segment of logical space, representing one megaword of memory in all.

When a memory map register is accessed, the 12 bits of new address information replace the original upper 12 address bits. This scheme allows the system to map any of the user's 512 2K word segments into any of the system's 4096 2K word segments. In reality, only the lower 14 megabytes are reserved for routine memory functions. The upper two megabytes are reserved for system and I/O space.

The three bits of access control information interact with the function codes representing the state of the processor and the Memory Mapping Flag to check for access errors as described in chapter four. Figure 2-7 shows the memory mapping register format.

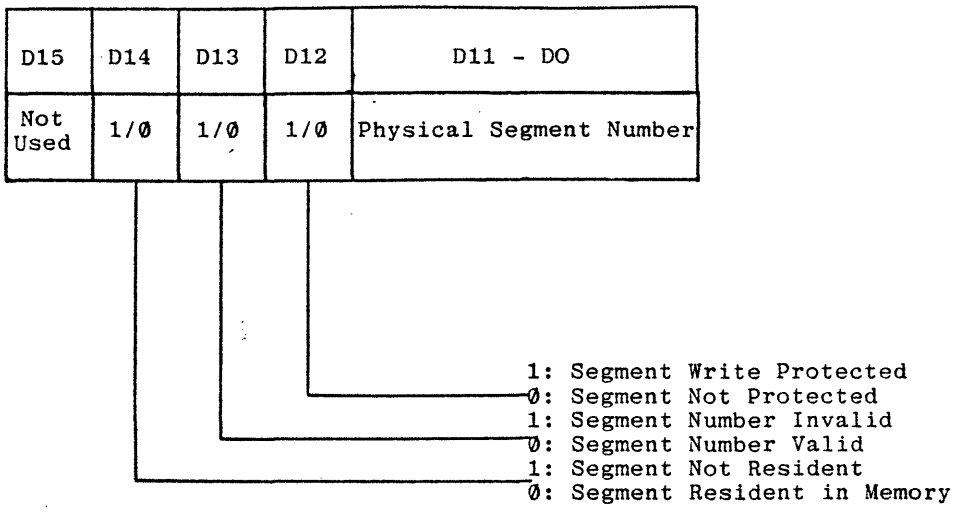


Figure 2-7
Memory Mapping Register Format

The memory map is accessible as read/write memory beginning at location EFF800 (hex).

2.6 ERROR CONTROL

The CPU board error control circuitry monitors the address for **address** violations. When an error occurs, the error type is latched, and a level seven interrupt is generated.

2.6.1 Address Errors

An illegal condition on the address bus causes an address error. Nonmapped addresses are inherently legal except for a word access on a byte boundary, so that an access error can only occur when the map is active. Four address errors are associated with using the memory map:

1. **access violation** - occurs when you try to access outside of user space, defined as 000000 - 1FFFFF (2 MegaBytes).

2. **write violation** - occurs when you try to write to a segment that is write protected.

3. **invalid segment** - occurs when you try to access a nonallocated segment.

4. **nonresident segment** - occurs when you try to access a nonresident segment.

Conditions 2, 3, and 4 above result directly from the access control information stored for each segment in the memory mapping registers.

2.6.2 The Error Register

All error conditions are sampled at the assertion of XACK/ at the end of a bus cycle. There is no danger, however, that an address error will cause erroneous data transfers since the error condition blocked the assertion of address strobe on the bus. In this case the bus error timer will time out asserting XACK/ (thereby latching the error condition).

If an error condition exists, the error register, which holds the values of the error conditions as sampled at the assertion of DTACK, is disabled from further change until it has been read by the CPU. Also, a level 7 autovectored interrupt is generated and held until it

is reset by the reading of the error register. The contents of the error register are defined in the diagram below:

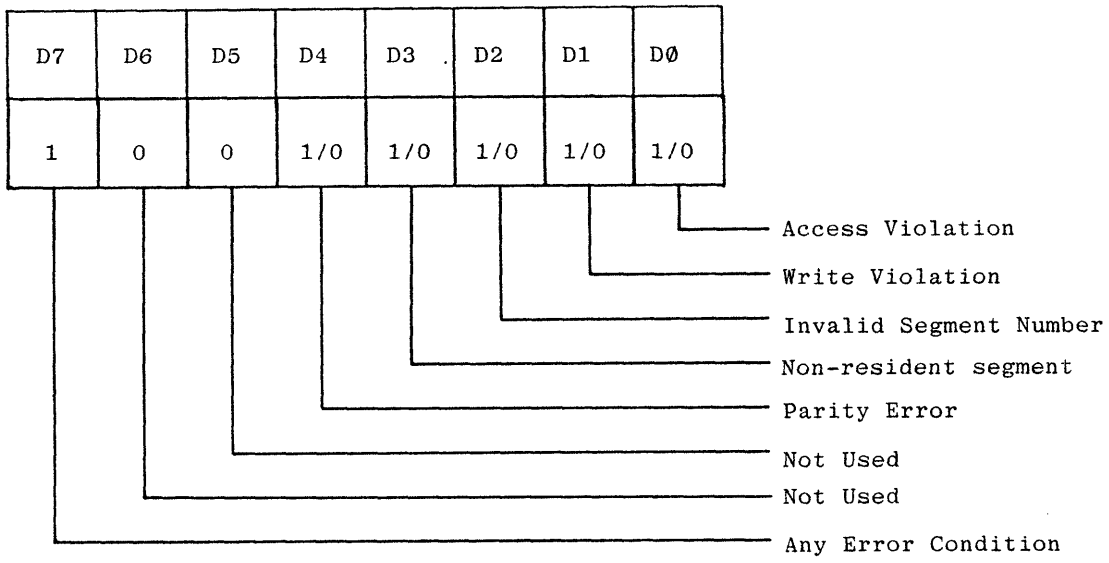


Figure 2-8
Error Register

The error condition flags (bits D0-D4) are active high, i.e., a one indicates that an error occurred. Register bit D7 will be a zero if an error condition is active. Register bits D5 and D6 are always zeroes.

The error conditions are latched at the assertion of XACK/ only if an error is detected at that time. When the error register is read, the level seven interrupt is reset but the contents of the error register remain intact. They will not change until another error is detected and latched.

2.7 ON BOARD PERIPHERALS AND MEMORY

The on board peripheral/memory address and data busses are sourced from the connector on the backplane and thus accessible to any device currently controlling the bus. The single exception to this rule is that another device cannot access the memory mapping registers.

2.7.1 Memory Mapping Flag And The Error Register

Two other devices may be considered on board peripherals: the Memory Mapping Flag and the Error register.

If the processor is in supervisor state and the processor address bus is carrying an address below 2 MegaBytes, then the value of the Memory Mapping Flag will determine whether the address will be mapped. The address will be mapped if the value of the flag is a one. It will not be mapped if it is a zero. The Memory Mapping Flag is set by writing 80 to location EFFC01. This is an even byte address. It may be reset by writing 00 to the same location. The value of the flag may not be read and is automatically reset to zero when the processor begins an interrupt acknowledge bus cycle, and when the system is reset.

The Error Register is a read-only byte location in memory. Its contents represent the error conditions present when the last address or parity error occurred. An error generates a level seven interrupt. Reading the Error Register (address = EFFD01) clears the interrupt.

2.7.2 On Board Memory

Sockets for 8 UV-EPROMs, either 2Kx8, 4Kx8, or 8Kx8 allow for 16K, 32K, or 64K bytes of ROM. The CPU board supports the memory board and any other part that conforms to these standards. In addition, the board also supports either the memory board standards by selecting the appropriate jump options. The memory type select jumpers are located between C7 and C8 on the CPU board. It consists of six pins arranged in three rows of two columns and two jump connections. (See Figure 2-9 for the various jumper configurations.)

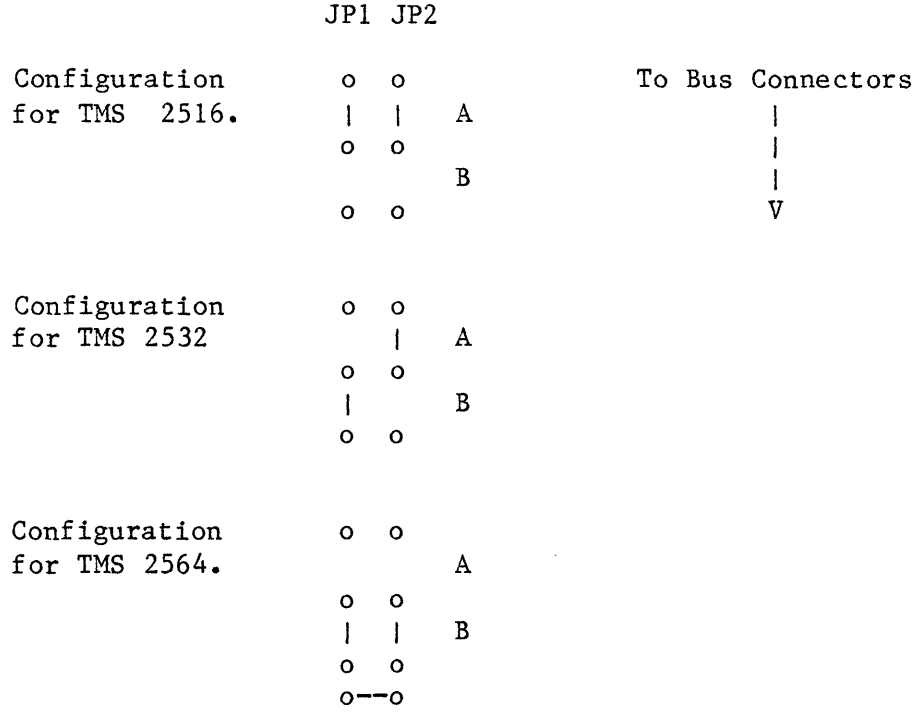


Figure 2-9
Jumper Configurations

The ROM is configured to reside in the lower 16K (or 32K or 64K) bytes of system memory. Figure 2-10 shows the board placement.

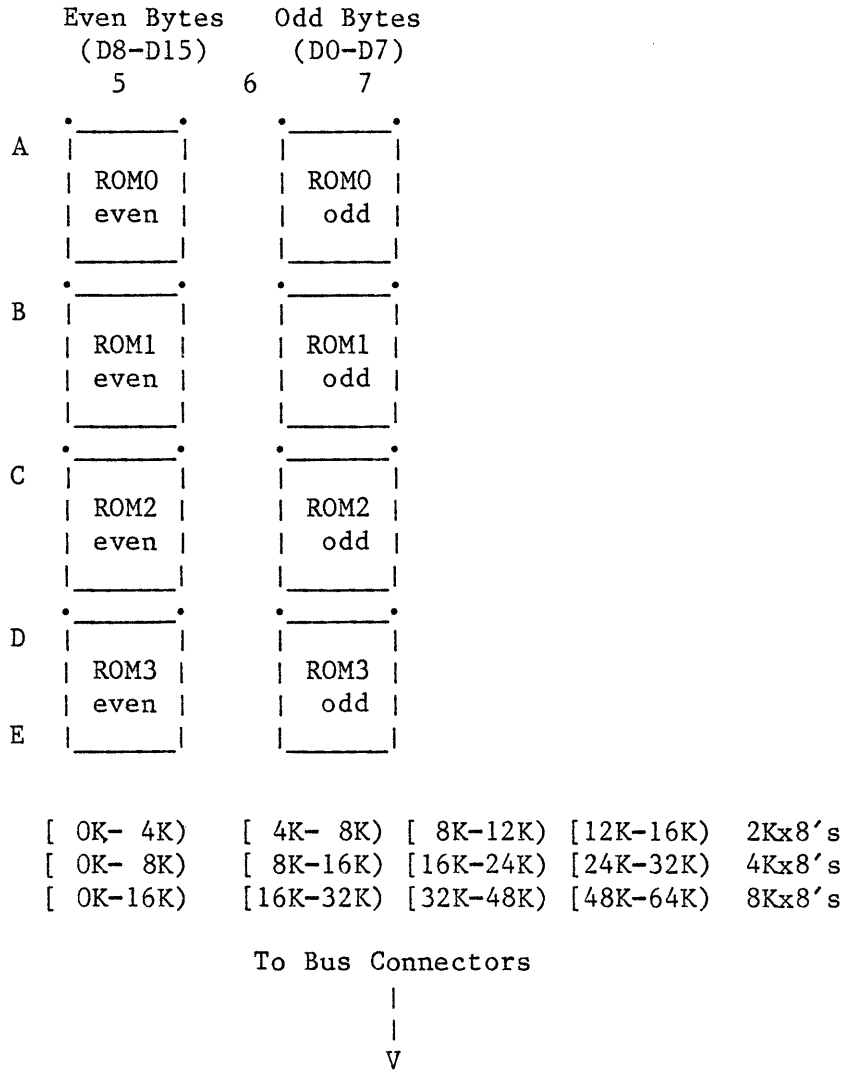


Figure 2-10
ROM Configuration

2.8 ADDRESS DECODE

The address decode logic decodes the value of the address bus and drives select lines for the on board peripherals and memory as well as general purpose I/O device and buffer select lines.

2.8.1 On Board Device Selection

The on board device selects are generated as the outputs of a 74154, 4 to 16 line decoder. The decoder is enabled when the address E00Bxx (x = don't care) appears on the address bus. The next to the least significant hex digit determines which on board device is being selected, while the least significant hex digit is reserved for register selection within the device itself. This scheme allows for a maximum of 16 on board peripherals. Currently only seven of these select lines are used. DTACK is asserted whenever E00Bxx appears on the address bus.

The memory mapping registers are selected whenever E00Cxx - E00Fxx appears on the address bus, providing for 1K bytes (512 registers) of memory map. DTACK is asserted whenever the map is selected as an I/O device.

The on board ROM is selected whenever 00xxxx appears on the address bus. This allows for a maximum of 64K bytes of ROM; 32K bytes are now implementable. DTACK is asserted whenever this area of memory is addressed.

2.8.2 CPU Board Address Decoding

The following summarizes the CPU board address decoding:

On Board Peripherals/Memory -

Read Only Memory:	000000 - 003FFF	(2Kx8's) (standard)
	000000 - 007FFF	(4Kx8's)
	000000 - 00FFFF	(8kx8's)
Clock/Timer:	TBD	
Memory Mapping Flag:	EFFC01	
Error Register:	EFFD01	
Memory Mapping Registers:	EFF800 - EFFBFF	

Off Board Memory -

RAM/ROM: 020000 - EFF7FF

I/O and Off Board Space -

F00000 - F0FFFF

2.8.3 EPROM Configuration Firmware

One 74S288 PROM on the CPU board must be programmed according to the EPROM configuration for your System 150. Tabulated below is the firmware programming for the three EPROM configurations possible with the CPU board. Standard configuration is 2K x 8 EPROMS. Use the nonstandard PROM programs (Tables 2-3 and 2-4) only after the EPROMS on the CPU board have been updated to the size indicated above the tables. All address and data information is in hexadecimal.

Address	Data	Address	Data
0	0E	10	0F
1	0D	11	0F
2	0B	12	0F
3	07	13	0F
4	0F	14	0F
5	0F	15	0F
6	0F	16	0F
7	0F	17	0F
8	0F	18	0F
9	0F	19	0F
A	0F	1A	0F
B	0F	1B	0F
C	0F	1C	0F
D	0F	1D	0F
E	0F	1E	0F
F	0F	1F	0F

Table 2-2
2K X 8 EPROMS (standard)

4K X 8 EPROMS

Address	Data	Address	Data
0	0E	10	0F
1	0E	11	0F
2	0D	12	0F
3	0D	13	0F
4	0B	14	0F
5	07	15	0F
6	07	16	0F
7	0F	17	0F
8	0F	18	0F
9	0F	19	0F
A	0F	1A	0F
B	0F	1B	0F
C	0F	1C	0F
D	0F	1D	0F
E	0F	1E	0F
F	0F	1F	0F

Table 2-3
4K X 8 EPROMS

Address	Data	Address	Data
0	0E	10	0F
1	0E	11	0F
2	0E	12	0F
3	0E	13	0F
4	0D	14	0F
5	0D	15	0F
6	0D	16	0F
7	0D	17	0F
8	0B	18	0F
9	0B	19	0F
A	0B	1A	0F
B	0B	1B	0F
C	07	1C	0F
D	07	1D	0F
E	07	1E	0F
F	07	1F	0F

Table 2-4
8K X 8 EPROMS

2.9 PAL EQUATIONS

Uncommitted logic space in the chips is configured as per the following equations:

PAL10L8 PAL DESIGN SPECIFICTION
P/N 318-021-001
CONTROL SIGNAL GENERATOR - POSTITON G2, S150 MMU CPU BOARD

SYSIO PA8 PA9 PA10 /PA0 /MWTC /MRDC /BHEN /CBUSY GND
SEL UHALF ERRSEL FLGSEL IOXACK LBYTE SWBYTE HBYTE LHALF VCC

```
LHALF   = /PA0
          +BHEN

UHALF   = PA0
          +BHEN

LBYTE   = SEL*BHEN
          +SEL*/PA0

SWBYTE  = SEL*/BHEN*PA0
```

CPU BOARD
PAL EQUATIONS

```
HBYTE   = SEL*BHEN

IOXACK  = SYSIO*MRDC*CBUSY
          +SYSIO*MWTC*CBUSY

ERRSEL  = SYSIO*MRDC*PA10*/PA9*PA8*/PA0

FLGSEL  = SYSIO*MWTC*CBUSY*PA10*/PA9*/PA8*/PA0
```

DESCRIPTION:

This chip generates the byte control lines for the onboard I/O which includes the ROM, the flag register, the error register, and the mape registers.

PAL12H6 PAL DESIGN SPECIFICATION
P/N 318-022-001
ERROR AND VAS GENERATOR - POSTITON B4, S150 MMU CPU BOARD

```
NRS NVS SWP /AS RW FC2 /MAPPED MAPFLG PIN9 GND
PIN11 PIN12 PIN13 AV WV SNV SNR VADD USP VCC


VADD = MAPPED*/NRS*/NVS*RW*AS
       +MAPPED*/NRS*/NVS*/SWP*AS
       +FC2*/MAPFLG*AS
       +FC2*AS*/USP

AV   = /FC2*/USP

SNV  = NVS*MAPPED

WV   = SWP*/RW*MAPPED

SNR  = NRS*MAPPED
```

DESCRIPTION:

This PAL generates the Valid Address Strobe and the error flags.

CHAPTER 3

I/O BOARD

3.1 INTRODUCTION

This chapter deals with the physical and logical aspects of the I/O board. Some of the items discussed are: I/O board configuration, the serial interface, the parallel port, etc.

3.2 DEFINITION AND FEATURES

The I/O board interfaces with the other boards through the Multibus in the motherboard or backplane (see Figure 1-2 System 150 (mapped) System Flow). The I/O board allows the system to communicate with the outside world. The board has a serial communications ports called a UART (Universal Asynchronous Receiver Transmitter) to "talk" to the internal display screen and "listen" to the detachable keyboard for instructions from the user. The additional serial ports on the board allow the same two-way communication with other terminals, printers, modems, etc., through the connections on the back panel.

This board also has a general purpose 16-bit parallel port to communicate with parallel I/O peripheral devices and an optional specialty interface called an IEEE 488 bus interface. (see Figure 3-1 below).

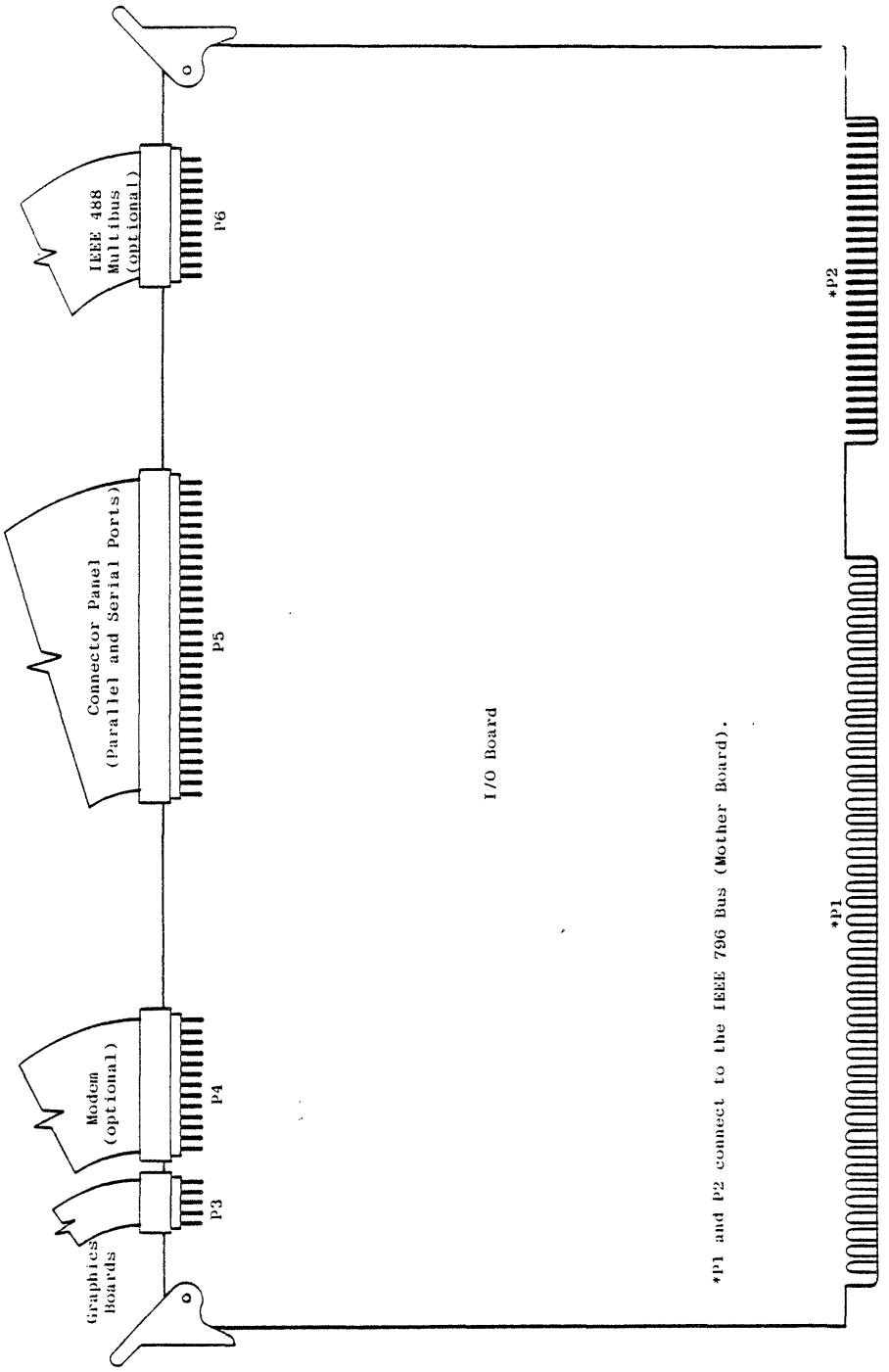


Figure 3-1
Multiuser I/O Board (Pin Connections)

3.3 I/O BOARD CONFIGURATION

The I/O board comprises seven main areas of circuitry (see Figure 3-2 below):

- the serial interface (Section 3.2.1)
- the real-time clock (Section 3.2.2)
- the interval timers (Section 3.2.3)
- the general purpose parallel port (Section 3.2.4)
- the parallel port direction and LED register (Section 3.2.5)
- the select/configuration switches (Section 3.2.6)
- the IEEE-488 bus interface and DMA controller (optional) (Section 3.2.7)

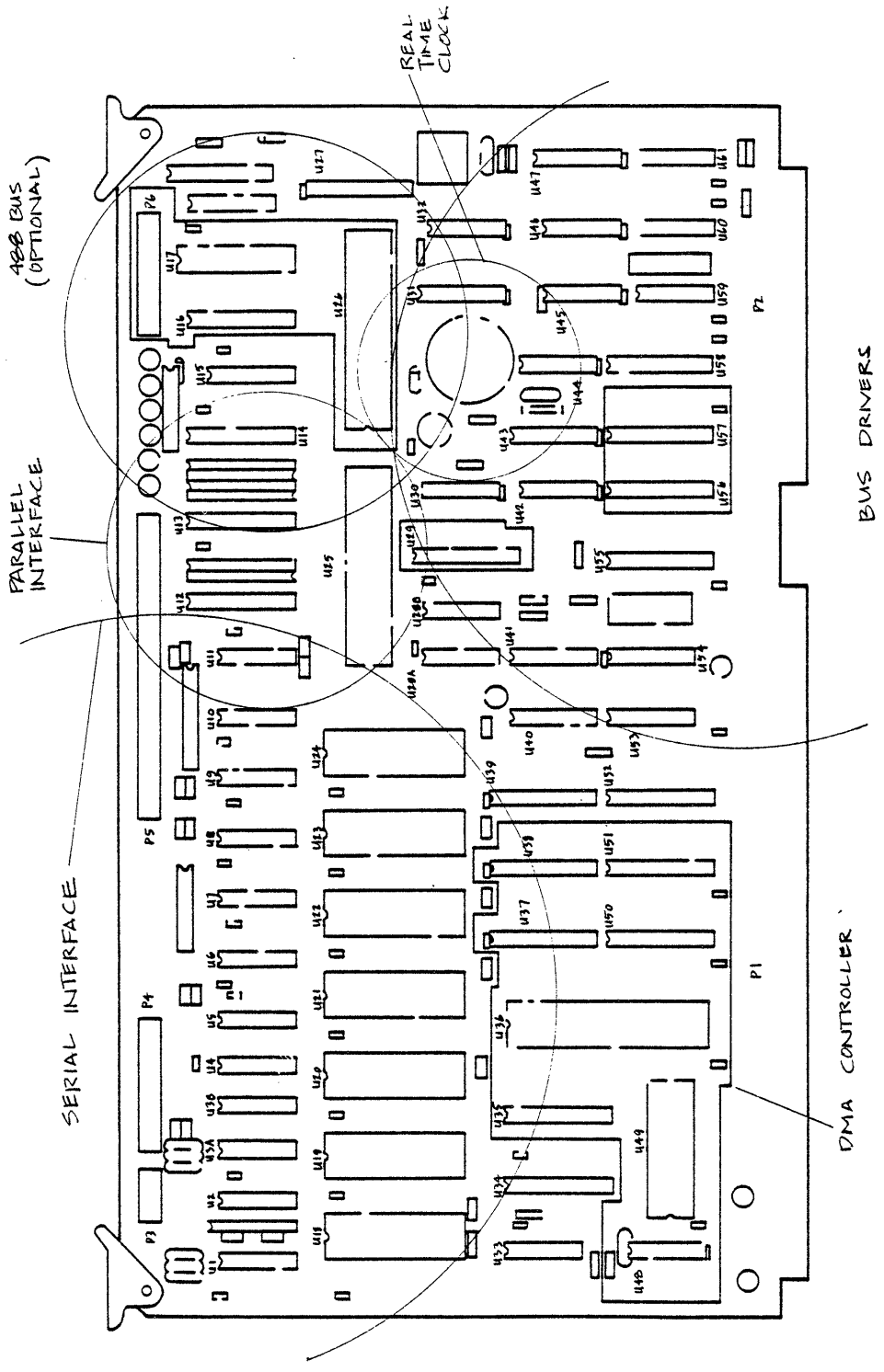


Figure 3-2
Multiuser I/O Board Circuitry

3.3.1 The Serial Interface

Seven RS232C serial interfaces with full handshaking can be implemented on the I/O board. In the standard board configuration the handshaking is dormant, but by changing simple jumper combinations, any or all of the handshaking lines can be made functional (see Appendix A Pin Locations). The baud rate is generated in the UARTs and is software selectable from 110 baud to 19.2K baud. Many of the handshaking lines can also be programmed to perform various functions depending on the application (see Appendix A Pin Locations).

All seven UARTs (0-6) are selected at the even address locations on the Memory Map (see Figure 3-3 Memory Map) between F00000 and F00036 with the first UART using the first four locations, the second using the next four, and so on.

All UARTs use interrupt level 5 (INT5/) on the Multibus. The UARTs are tied to the lower byte of the data bus, and address lines BADR1-BADR2 are used for internal register selection. UARTs 1-5 communicate externally through the I/O connector board on the back of the chassis. UART 0 connects to the internal terminal (transparent to the user).

UART number six is configured to be used with a modem and for this reason is brought off the board on its own connector, P4. This connector does not connect to the I/O port panel on the back of the system, and, for proper operation, it requires full handshaking, including a data carrier detect (DCD) input signal (located at P4-8).

F000XX

00-06	Serial Port 0
08-0E	Serial Port 1
10-16	Serial Port 2
18-1E	Serial Port 3
20-26	Serial Port 4
28-2E	Serial Port 5
30-36	Serial Port 6
40-5E	Parallel Port & Interval Timers
60-7E	Calender Clock
D0	LEDs/Parallel Port Direction
D2	Select/Configuration Switches

Figure 3-3
System 150 I/O Memory Map

3.3.2 The Real-Time Clock

The real-time clock (RTC) is a calendar clock that can be set and read from the microprocessor. The RTC can be set to give data on the following:

- tenths of seconds
- seconds
- tens of seconds
- minutes
- tens of minutes
- hours
- tens of hours
- days
- tens of days
- day of week
- months
- tens of months
- automatic leap-year

A battery backup circuit provides power to the RTC for 1-2 years, thus maintaining time and date even when the power is off for extended periods. The RTC is selected at addresses F00060-F0007E, and address lines BADR1 - BADR4 are used for internal register selection (see Figure 3-3 I/O Memory Map). The RTC data bus is four bits wide and is tied to the four least significant bits of the low byte of the data bus.

3.3.3 The Interval Timers

Two interval timers are included on the I/O board. Each timer is software-programmable to operate in several different modes, and can interrupt the microprocessor when software-specified conditions occur. The timers are contained in the SY6522 Timer/PIA IC and are tied to the lower byte of the data bus. Address lines BADR1-BADR4 select the internal register. The SY6522 IC is selected at address F00040-F0005E, and uses interrupt level 6 (INT6/) on the Multibus.

3.3.4 The Parallel Port

The I/O board has a 16-bit, general purpose, bidirectional parallel port, with four handshaking lines. The parallel port is actually two bidirectional eight-bit ports, port A and port B, supplied by the SY6522 Timer/PIA IC. Two handshaking lines are supplied with each eight-bit port, and each line can be programmed to operate in a variety of ways. Both eight-bit ports are buffered with bidirectional buffers that can be programmed as either inputs or outputs.

To configure the ports as either inputs or outputs, the correct data must be written to the SY6522 IC and also to a bit-addressable latch that controls the port buffers. The addressable latch is described in detail in the parallel port direction and LED register description that follows (see Section 3.2.5).

On power up both eight-bit ports are configured as inputs. The SY6522 IC is selected at address F00040-F0005E, and uses interrupt level 6 (INT6/) on the Multibus. The SY6522 is tied to the lower byte of the data bus, and address lines BADR1 - BADR4 are used for internal register selection.

3.3.5 The Parallel Port Direction And LED Register

The parallel port buffers and the six LEDs are controlled by an addressable latch. The latch is selected at address location F000D0. The function of each output is described on the following page.

7	6	5	4	3	2	1	Ø
LED6	LED5	LED4	LED3	LED2	LED1	BDIR	ADIR

- BDIR - Direction control for port B. A "1" at this output causes the buffer on port B to become an output, while a zero (0) causes the buffer to become an input.

- ADIR - Direction control for port A. A "1" at this output causes the buffer on port A to become an output, while a zero (0) causes the buffer to become an input.

- LED1-LED6 - On/off control for the SIX LEDs. A low at one of these outputs causes the corresponding LED to be turned on, while a "1" causes the corresponding LED to be turned off.

- All outputs are cleared to zeroes on power up and on reset.

Figure 3-4
Explanation of Addressable Latch Operation

An addressable latch is a write-only latch on which only one bit is written at a time. This is accomplished by using three bits of the input data as an address, to select which of the latch outputs is to be written to. Another bit of the input data is used as the data to be written to the addressed output. The input data byte is organized as shown below:

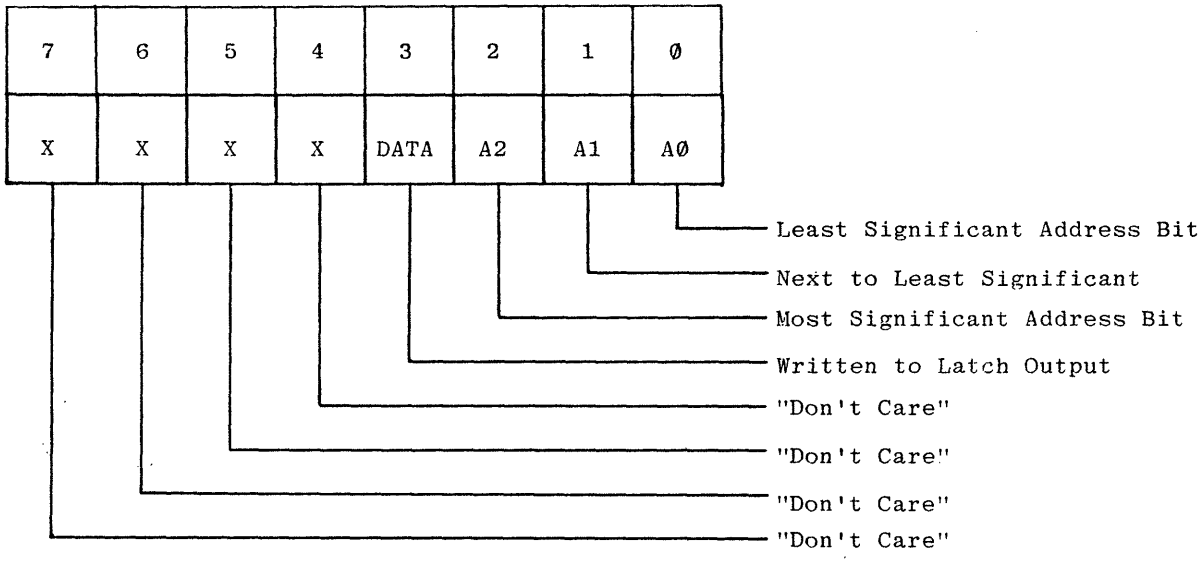


Figure 3-5
Input Data Byte Organization

3.3.6 The Select/Configuration Switches

The eight DIP switches may be read by the system. Switches one through five are used for the address select of the IEEE-488 bus interface (see Section 3.2.7). Switches six through eight are used as test/configuration switches. The switches are selected at address F000D2 and are read only.

3.3.7 IEEE 488 Multibus

This set of components is optional. When implemented it acts as a General Purpose Interface Bus (GPIB), connecting other optional peripherals to the I/O board.

CHAPTER 4

MEMORY MODULE

4.1 INTRODUCTION

This chapter contains a description of the memory board signals, the electrical characteristics, and the board configuration, including the location and the use of address switches.

4.2 DEFINITION AND FEATURES

The memory board is a Dynamic RAM Memory Module with Error Checking and Correction (ECC). The module is organized as 256K words by 16 bits (K = 1024) and uses the 64K DRAM. Features of the memory module include:

- Compatibility with both the Intel Multibus protocol and the IEEE 796 bus specifications
- Additional six bits of check bit data support error detection and correction (EDAC)
- A control status register and an error status register
- Decoding:
  - 24 lines of address capability
    - A 20-bit address field with a 4K byte granularity
    - Four additional address lines to select any of 16 1-megabyte pages
    - Data access in either word or byte mode

The following diagram shows the configuration of the memory module and identifies some of the circuitry areas.

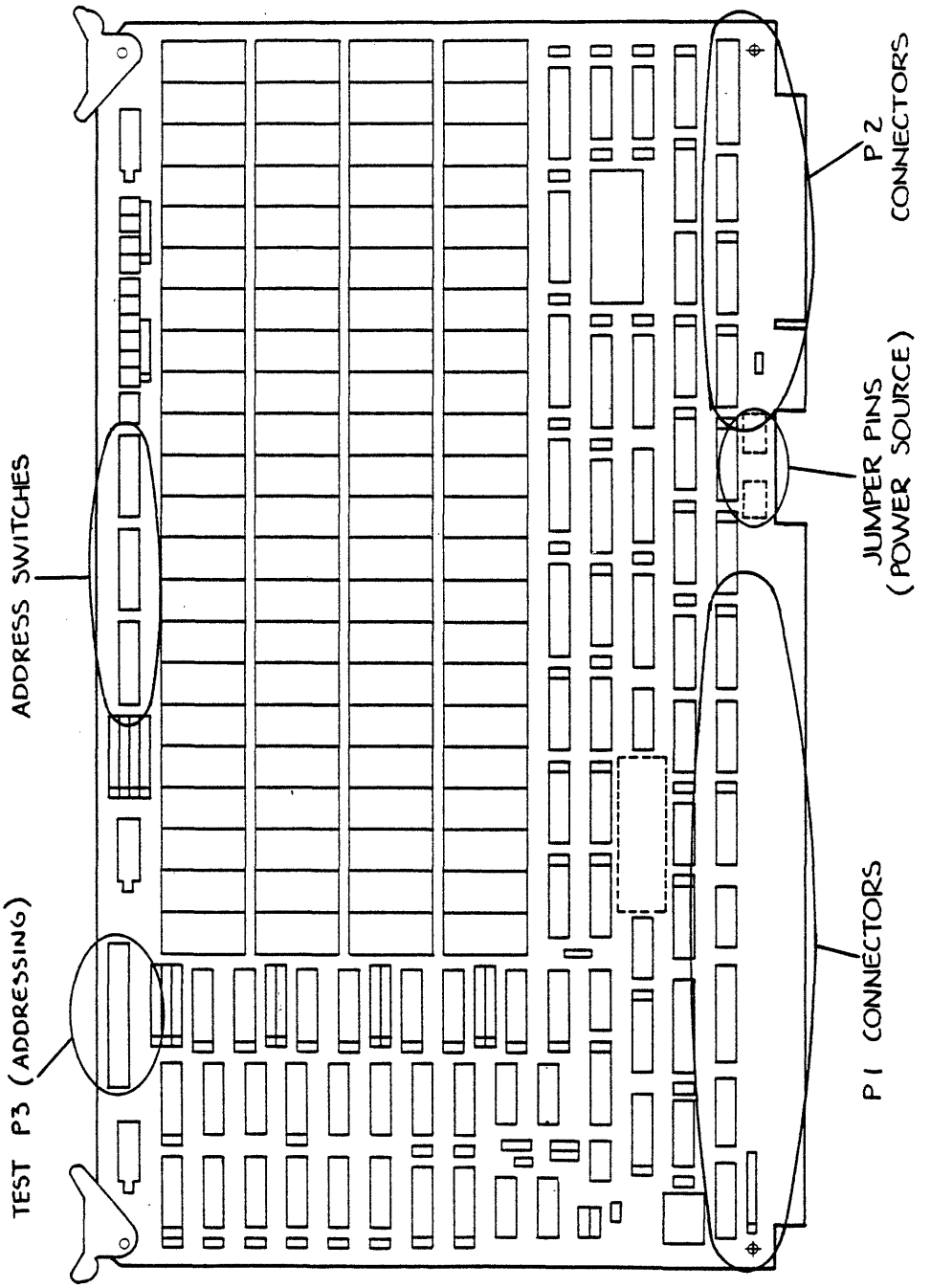


Figure 4-1
256K Memory Module

4.3 MEMORY MODULE SIGNALS

Signals common to the Multibus system are defined in the Intel Multibus Specification Manual 9800683 or the IEEE 796 bus specification. Signals peculiar to the memory module are described in Section 4.3.1.

4.3.1 Advanced Acknowledge (AACK/)

AACK/ warns the bus master of valid read data, thus avoiding unnecessary wait states. AACK/ is identical to Transfer Acknowledge (XACK/) except that it occurs earlier in the cycle during read operations. AACK/ is brought out to the bus on connector P2.

4.4 ELECTRICAL CHARACTERISTICS

4.4.1 EDAC

EDAC corrects single bit errors and detects double bit/gross errors. When enabled, EDAC completes all operations having a READ error by attempting to write corrected data back to memory.

4.4.2 Status Registers (CSR And ESR)

The memory module has a control status register (CSR) and an error status register (ESR). From the card edge with the card in place, you can see the error status displayed on the LEDs.

CSR and ESR are accessed through an I/O port base address. CSR is selected with ADR0 = 1 (an electrical high); ESR with ADR0 = 0. The I/O port base address is designated with eleven onboard switches. Eight switches are compared with ADR4/ through ADR8/, and the three more significant start address switches are compared with ADR1/ through ADR3/. Selecting addresses this way permits the eight more significant bits of the I/O port address to be identical to other memory boards in the system and to mirror the I/O port address selection made based on position of the memory board within the address space.

See Section 4.8.5 for the switch settings.

4.4.2.1 Control Status Register -

CSR controls and stores information on errors and power failure. You can write to the CSR or read from it, depending on whether you are executing an I/O Read Cycle (IORC) or an I/O Write Cycle (IOWC).

4.4.2.1.1 CSR Read Format -

Figure 4-2 shows the format for reading data from the CSR:

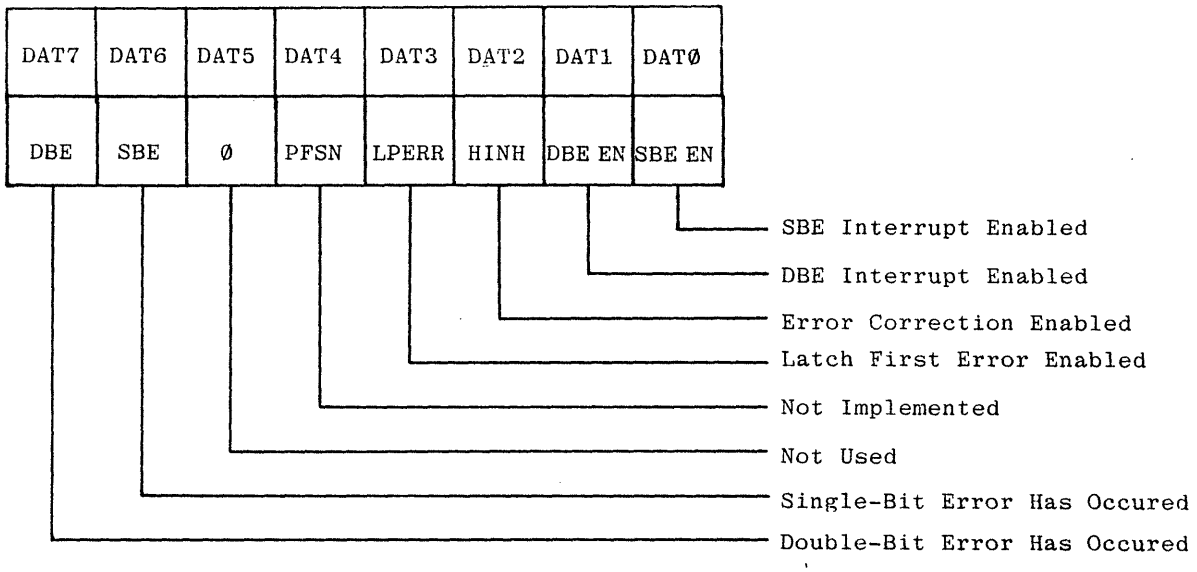


Figure 4-2
CSR Read Format

Figure 4-3 shows the format for writing data into the CSR:

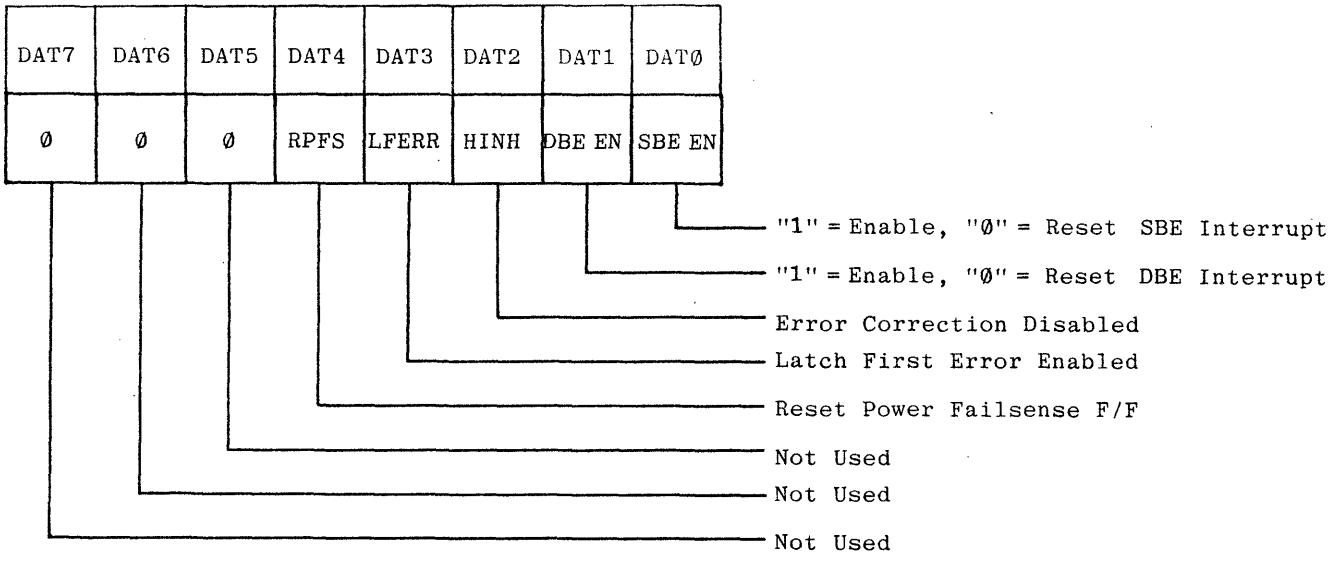


Figure 4-3
CSR Write Format

4.4.2.2 CSR Flag Control Bits - The system supports six CSR Flag Control Bits, described below.

Double Bit Error Flag (DBE)

DBE indicates that two bits in the same word failed or that a gross error has been detected. DBE is set when a double bit error occurs, then reset when DBE EN control bit is taken to zero. Setting DBE EN back to a logic one enables detection of the next double bit error.

Single Bit Error Flag (SBE)

SBE, a read only signal, indicates that a single bit error has been detected. SBE is set when the error is detected, then reset when SBE EN control bit is taken to logic zero. Setting SBE EN control bit back to a logic one enables the detection of the next single bit error.

Latch First Error (LFERR)

LFERR allows you to select whether ERR data are updated each time a SBE/DBE is detected, or only once on the first

error detected. Writing a logic one to the LFERR control bit will enable LFERR so that the next error can be stored. When you set LFERR to zero, the last error will always update the ESR. LFERR can be read or written.

Error Correction Disabled (HINH)

When equal to logic one, HINH disables error detection, error correction, and the write function to the checkbits. Use HINH only for testing. HINH can be read or written, and the board cannot be initialized when this bit is set.

Enable DBE Interrupt (DBE EN)

When DBE EN equals logic one, interrupts on DBEs are possible. Taking DBE EN control bit to a zero resets any current DBE flag bit. Leaving DBE EN at a zero disables DBE interrupts. DBE EN can be read or written.

Enable SBE Interrupt (SBE EN)

When SBE EN equals logic one, interrupts on SBEs are possible. Taking SBE EN control bit to a zero resets any current SBE. Leaving SBE EN at zero disables SBE interrupts. SBE EN can be read or written.

4.4.2.3 Error Status Register -

When a single bit error occurs and the conditions for FE/LE have been satisfied, ESR stores the error information. You can only read data from the ESR. Clear the ESR by

- writing to it,
- pressing RESET (S1), or
- making INIT/ active.

Figure 4-4 shows the format for reading data from the ESR:

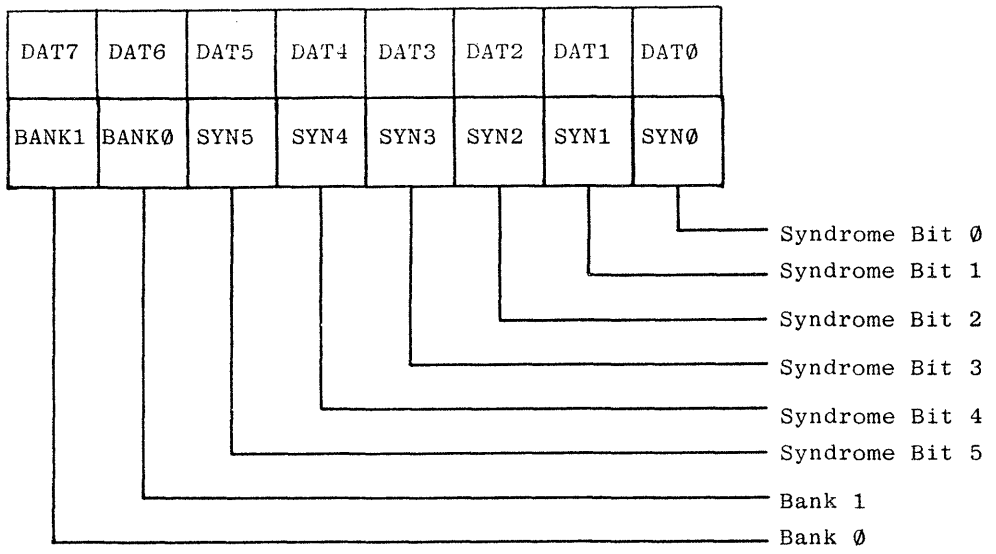


Figure 4-4
ESR Read Format

4.4.2.4 ESR Signal Definitions -

Bank:

Bank 1 and Bank 0 define the physical row of DRAMS (BANK) in which an SBE occurs. Table 4-1 shows the relationship between bank address and the reference designator for memory devices within that row.

Bank 1	Bank 0	Memory Reference Designators		
0	0	u000	through	u021
0	1	u100	through	u121
1	0	u200	through	u221
1	1	u300	through	u321

Table 4-1
Bank Signals Reference Table

Syndrome bit 5 through bit 0 (SYN5 - SYN0)

Bits 5 through 0 define the hamming code generated by the SN74LS630 EDAC device when a SBE occurs. Table C shows the relationship between the syndrome code and bit location within a given bank.

Syndrome Code (Bit 5 4 3 2 1 0)	Data Bit	Check Bit	Bit Location X=0,1,2,3 (Bank)
1 1 0 1 0 0	0		X00
1 1 0 0 1 0	1		X01
1 1 0 0 0 1	2		X02
1 0 1 1 0 0	3		X03
1 0 1 0 1 0	4		X04
1 0 1 0 0 1	5		X05
1 0 0 1 0 1	6		X06
1 0 0 0 1 1	7		X07
0 1 1 1 0 0	8		X08
0 1 1 0 1 0	9		X09
0 1 0 1 1 0	10		X10
0 1 0 1 0 1	11		X11
0 1 0 0 1 1	12		X12
0 0 1 1 1 0	13		X13
0 0 1 1 0 1	14		X14
0 0 1 0 1 1	15		X15
1 1 1 1 1 0		0	X16
1 1 1 1 0 1		1	X17
1 1 1 0 1 1		2	X18
1 1 0 1 1 1		3	X19
1 0 1 1 1 1		4	X20
0 1 1 1 1 1		5	X21
0 0 0 0 1 1	Gross Error Condition		
1 1 1 1 0 0	Gross Error Condition		

Note: 1 = lamp on

Table 4-2
Syndrome Code

The syndrome codes for DBEs are mutually exclusive of any SBE codes. If you clear the ESR by writing to it, then by reading the ESR and comparing it to zero you can check (poll) the board for errors.

4.4.2.5 Error Status LEDs -

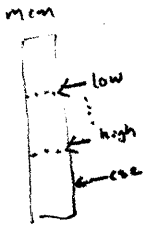
The ten light emitting diodes (LEDs) near the address switches display the error status: the left-most light indicates a DBE; the next light indicates a SBE. Interpretation of the next eight lights is the same as the ESR (see Section 4.4.2.2). Pressing reset (S1) or writing to the ESR clears the ten LEDs. An active input on the INIT/line clears the LEDs and the ESR.

4.4.3 Memory Features

The memory features described in this section fall into one of six categories.

- Addressing (Section 4.4.3.1)
- EDAC (Section 4.4.3.2)
- Interrupt options (Section 4.4.3.3)
- Power fail reset (Section 4.4.3.4)
- Power source (Section 4.4.3.5)
- Advanced Acknowledge (Section 4.4.3.6)

4.4.3.1 Addressing -



Three sets of address switches (Figure 4-1) decode the lower and upper end of memory and select the CSR/ESR base address. Memory address range extends to 1024KB with 4KB granularity. The CSE/ESR address range extends to 4KB. See Section 4.8 for examples of switch settings.

4.4.3.1.1 Starting Address Selection -

Bus address bits ADRC/ through AD13/MSB (Most Significant Bit) are compared with switches S2-8 through S2-1 respectively. When the bus address bits are greater than or equal to the selected switch settings, the conditions for the lower end of the address range are satisfied. To select the memory board however, conditions for the ending address must also be satisfied.

4.4.3.1.2 Ending Address Selection -

Bus address bits ADRC/ through AD13/ (MSB) are compared with switches S3-8 through S3-1 respectively. When the bus address bits are less than or equal to the selected switch settings, the conditions for the upper end of the address range are satisfied. Read about the lower end address requirements in 4.8.2. The ending address switch specifies the last 4KB block to be addressed; i.e., if the ending address switches were set to 43 hexadecimal, the last byte to be addressed would be 43FFF.

4.4.3.1.3 CSR/ESR Address Selection -

Bus address bits ADR4/ through ADRB/ and ADR1/ through ADR3/ are compared with switches S4-8 through S4-1 and switches S2-3 through S2-1 respectively. Including S2-3 through S2-1 in the selection of the CSR/ESR address permits all memory boards to have the same I/O base address with selection being determined by the starting address for the memory. A closed switch represents a logical zero. Read section 4.8 for examples of switch settings.

4.4.3.1.4 CSR/ESR Address Range -

A three-jumper pin arrangement permits the removal of ADR8/ through ADRB/ from the CSR-ESR address selection. Connecting wirewrap post B5 to C5 yields an address range of 4096 bytes. Connecting B5 to A5 limits the range to 256 bytes. The smaller range uses only eight bits and is therefore normally enabled only when an eight-bit microprocessor is used for I/O communications.

4.4.3.2 Error Detection And Correction (EDAC) -

EDAC Enabled

EDAC is enabled by an active INIT/ during power up. When the CSR bit HINH is low (see 4.4.2.1), the EDAC device, SN74LS630, is enabled. EDAC generates checkwords, syndrome bits, and error flags (DBE and SBE), and corrects data words. Corrected data from the EDAC are stored in a data latch so that during a read operation the corrected data:

- are available on the bus,
- can replace error data (write back on error), and
- are made available for byte write operations

EDAC Disabled

When the CSR bit HINH is high, the EDAC device is held in an input mode, and the error flags (DBE and SBE) are held reset. Read data are not corrected, and the write operation to the checkbits is inhibited. Thus,

- write data generate no checkbits, and
- writing a word to a location with HINH active

modifies the 16 data bits but leaves the six check bits unchanged from the last write operation to where HINH was inactive.

4.4.3.3 Interrupt Options -

Hardware Programming

Interrupt request lines INT0/ through INT7/ generate nonbus vectored interrupts to the bus master. These eight lines can be connected to any combination of three signals: SBE L, DBE L, and PFIN L. Normal configuration is SBE L to 0, DBE L to INT0/.

Interrupts are wired so that a SBE or DBE causes INT7/ to be asserted. No interrupts are jumpered at shipment time. Figure 4-1 shows the jumper pin locations.

Software Programming

With SBE EN and DBE EN bits in the CSR, you can enable or disable SBE interrupt and DBE interrupt respectively. To clear the current interrupt, disable the interrupt, then reenable so that the interrupt will operate on future errors.

4.4.3.4 Power Source -

The memory module has a three-section power plane and a solid ground plane. All logical devices are on one or a combination of these planes.

4.4.3.5 RAM Configuration -

The memory module can be configured with 64K DRAMS and 64K DRAMs, i.e., 64 DRAMs with 32K of usable memory. Address programming of the 64K DRAM partials depends on which part of the DRAM is usable; four combination are possible. When using partials, the module comprises only one DRAM type.

The programming information shown in Figure 4-7 depends on the type of DRAM used. Altering this information is impossible unless you replace all of the DRAMs on the memory module.

4.5 TESTABILITY

Three test features ensure the optimum effectiveness of computer-based incircuit and functional test systems.

4.5.1 Testing EDAC Logic

Thorough testing of EDAC logic requires access to the six checkbits generated/checked by the EDAC device (SN74LS630). The data from these memory locations are unavailable through the bus I/O. Via connector P4 the EDAC test can access these checkbits. Table 4-3 shows the connector configuration.

PIN	MNEMONIC	DESCRIPTION	I/O	PIN	MNEMONIC	DESCRIPTION	I/O
G1	GND	Ground	-	H1	SYNLCK H	Syndrome Latch Clock	I
G2	GND	Ground	-	H2	LMWTC L	Latched Write Command	I
G3	REFMUX H	Refresh CYC/$\overline{\text{CPU CYC}}$	O	H3	RAMDSBL L	RAM Disable	I
G4	d0H	Start of Cycle	O	H4	CBWTEN L	Checkbit Write Enable	I
G5	LATC80 H	Latched Checkbit 0	O	H5	DI16 H	Buffered Check-	I/O
G6	LATC81 H	Data 0-5 1	O	H6	DI17 H	bit Data	I/O
G7	LATC82 H	(Test Only) 2	O	H7	DI18 H		I/O
G8	LATC83 H	3	O	H8	DI19 H		I/O
G9	LATC84 H	4	O	H9	DI20 H		I/O
G10	LATC85 H	5	O	H10	DI21 H		I/O

Table 4-3
P4 Connector Pin Assignments

4.5.2 Address Testing

A connector P3 ensures that all combinations of starting and ending addresses are properly decoded. Each of the 24 ungrounded pins on P3 is in parallel with one of the address select switches. Figure 4-4 shows the connector configuration.

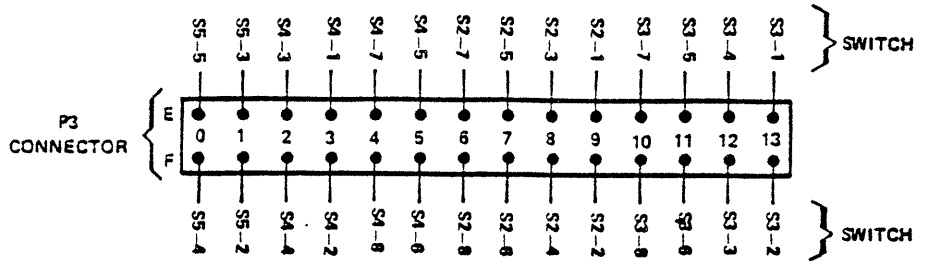


Figure 4-4
Addressing Connections

4.5.3 Testing Arbitration

To test arbitration, plot the skew between refresh request and CPU request versus the start delay time as shown in Figure 4-4. Figure 4-5 shows the connector configuration for arbitration testing.

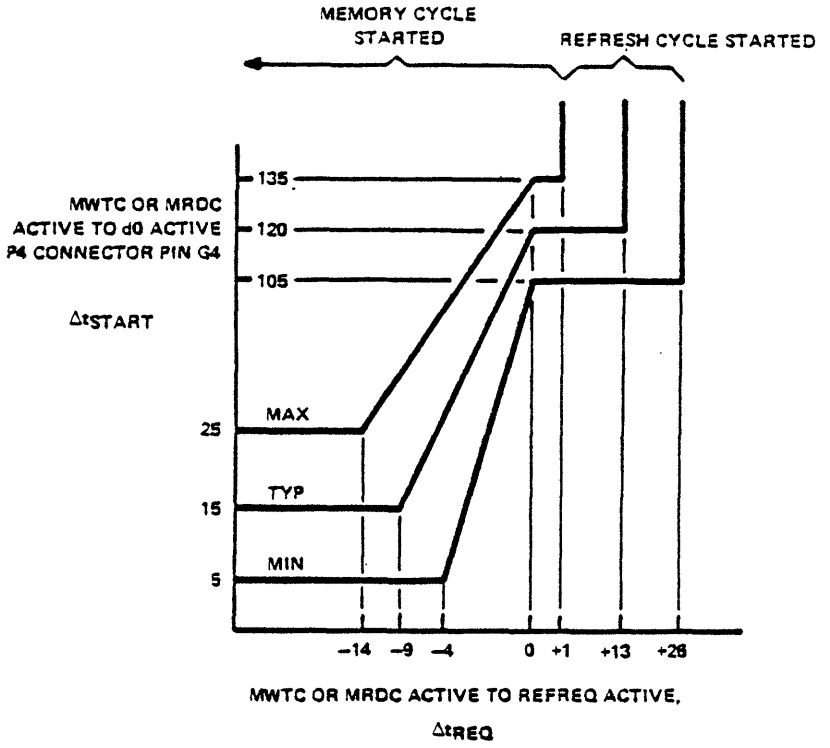


Figure 4-5
Arbitration Testing Plot

NOTE

These test features may change as it becomes necessary to modify the test programs.

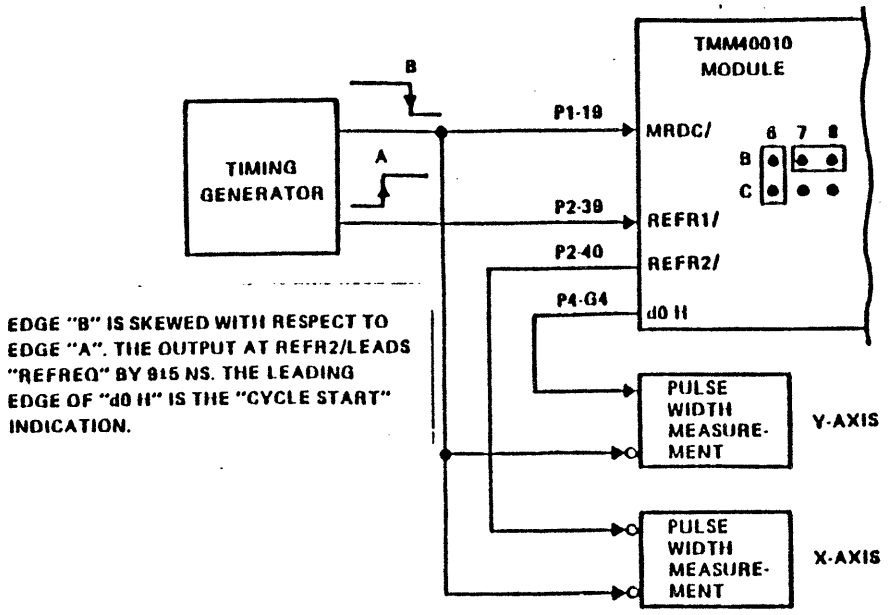


Figure 4-6
Arbitration Testing Connections

4.6 MODES OF OPERATION

Input/output (I/O) and memory are the two basic modes of operation. I/O operations are between the CSR and ESR. Memory operations involve data transfers between memory. Figures 4-7 A, B, and C present the major decisions and events that take place during each memory or I/O cycle.

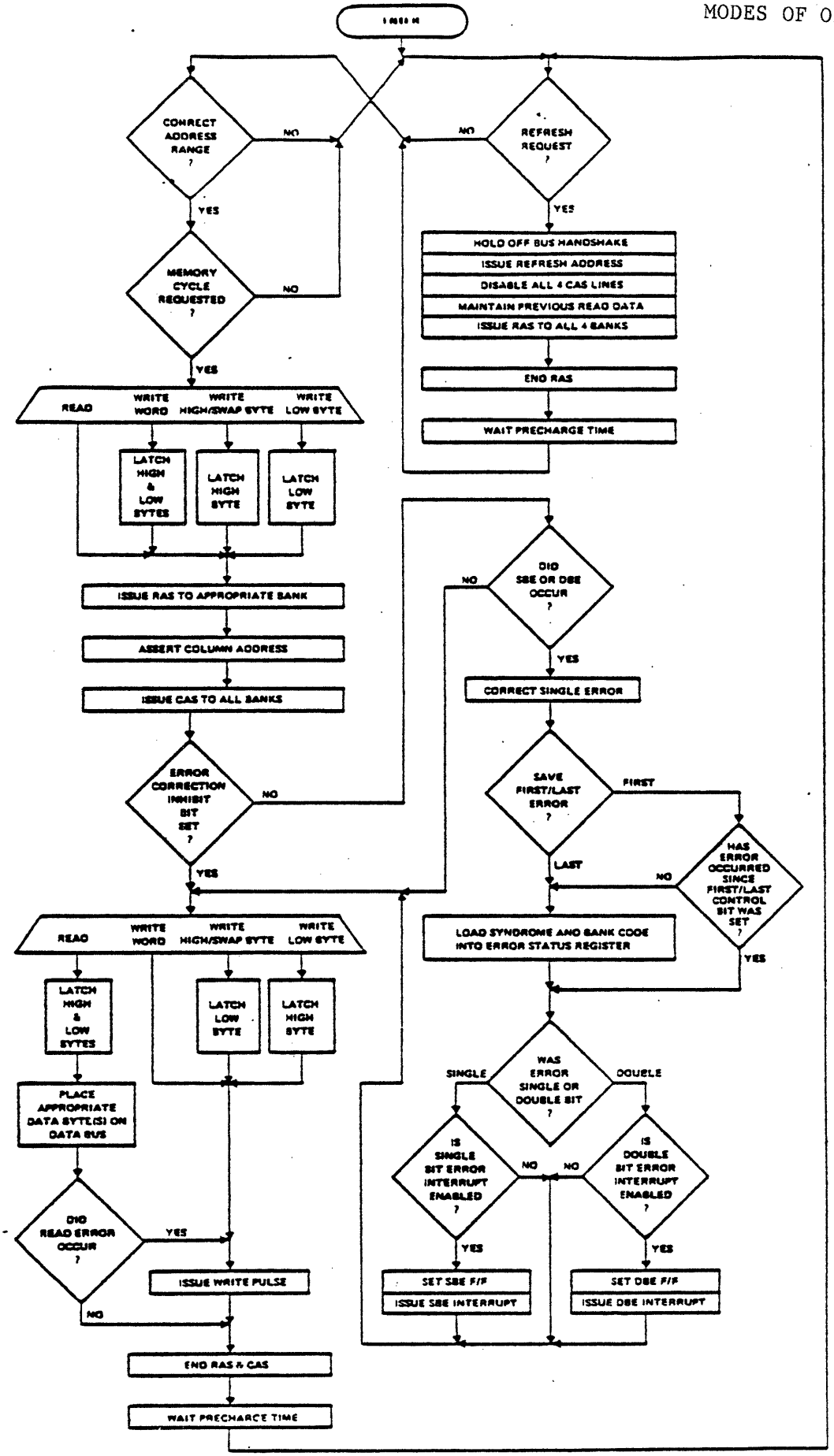


Figure 4-7 (A)
Memory Cycle Flow Chart

MEMORY MODULE
MODES OF OPERATION

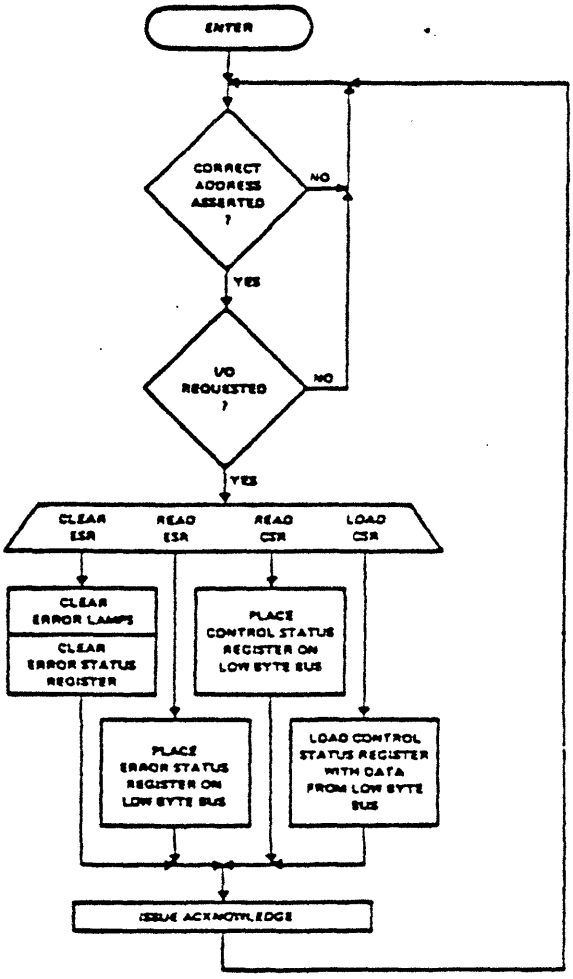


Figure 4-7 (B)

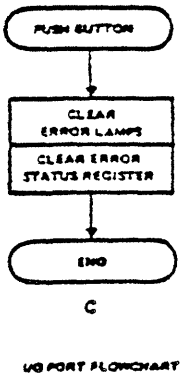


Figure 4-7 (C)

Figure 4-8 summarizes the 8-bit and 16-bit data paths used.

(Figure 4-8 is on page 4-19A)

4.6.1 Read Operations

Data from a specified address is obtained from memory and sent uncorrected to the EDAC logic for validation and correction. If EDAC is inhibited, uncorrected data are immediately available at the bus as either a 16-bit or 8-bit word. Otherwise, data are checked, and error flags are set if an error is detected. If an SBE occurs, corrected data are rewritten to memory. If enabled, error flags are latched into the CSR, and interrupts are sent to the bus master. The CSR will contain a value that can be interpreted to find the failing DRAM.

4.6.2 Write Operations

If EDAC is enabled, 16-bit and 8-bit words for a specified address are sent to the EDAC logic to generate checkbits, and the data in memory are checked for errors, corrected, and stored in registers. If 8-bit data are written, the input replaces the unwanted byte of the corrected word in the data register. A new checkbit word is generated, and the data word in the register is written back into memory.

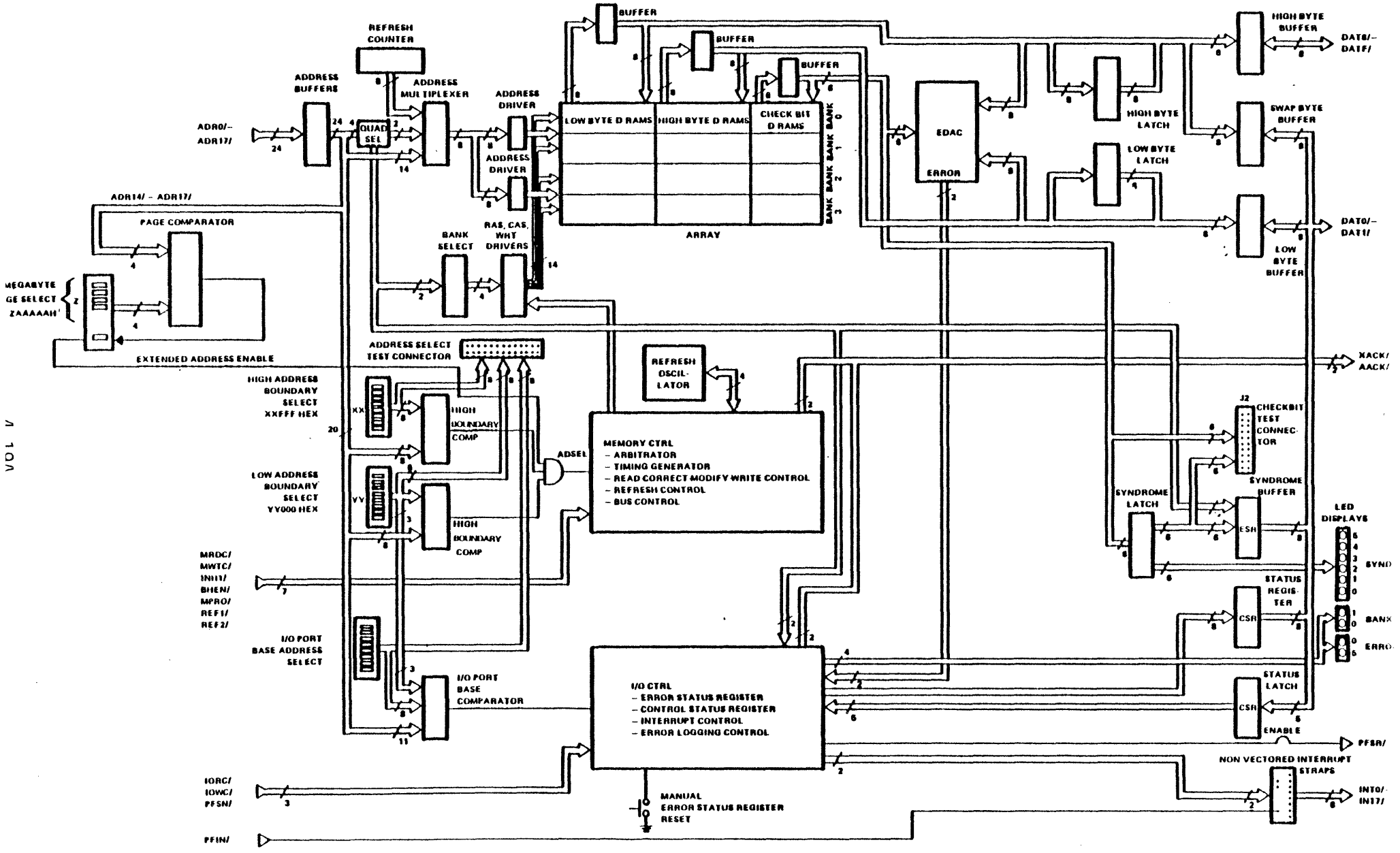


Figure 4-8
Memory Cycle Data Paths

4.6.3 Byte Swap

A byte swap buffer is included to maintain compatibility with 8-bit bus masters. This buffer exchanges even bytes between the 8-bit system bus and the high-byte in memory when BHEN/ is high.

4.6.4 Refresh

Refresh permits you to retain data in dynamic memory. A Row Address Storage (RAS) only cycle is used for refresh and a minimum of 256 cycles must occur every four milliseconds. A different row address is accessed for each cycle of the 256 cycles.

4.7 CONFIGURING THE MEMORY MODULE

Three things happen on powerup,

1. The CSR is initialized so that error detection and correction are enabled,
2. The ESR captures the last error,
3. DBE and SBE interrupts are disabled.

For initial checkout, you need set only the four address switches.

4.8 LOCATION OF ADDRESS SWITCHES

Turn the module so that the switches are facing you. From left to right the switches facing you are:

STARTING ADDRESS, ENDING ADDRESS, I/O PORT BASE ADDRESS

And in the upper right-hand corner, **MEGABYTE PAGE ADDRESS.**

NOTE

The MSB is on the left of each switch. ON equals a logical zero (0). OFF equals a logical one (1).

4.8.1 Starting Address

The starting address is WX000 hexadecimal. WX represents two hex digits (eight bits) that correspond directly to the eight bits of the starting address switch, S2. Consider the example in Figure 4-9.

Desired Starting Address (hex)	Switch S2								
	WX	-1	-2	-3	-4	-5	-6	-7	-8
00000	00	0	0	0	0	0	0	0	0
12000	12	0	0	0	1	0	0	1	0
3F000	3F	0	0	1	1	1	1	1	1
~~C3FFF~~ 80000	80	1	0	0	0	0	0	0	0

Figure 4-9
Starting Address Settings

4.8.2 Ending Address

The ending address is YZFFF hexadecimal. YZ represents two hex digits (eight bits) that corresponds directly to the eight bits of the ending address switch, S3. Consider the example in figure 4-10.

Desired Ending Address (hex)	YZ	Switch S3 -1	-2	-3	-4	-5	-6	-7	-8
3FFFF	3F	0	0	1	1	1	1	1	1
56FFF	56	0	1	0	1	0	1	1	0
9EFFF	9E	1	0	0	1	1	1	1	0
C3FFF	C3	1	1	0	0	0	0	1	1

Figure 4-10
Ending Address Settings

4.8.3 Enabling Extended Address Lines

To enable the four extended address lines, locate the 5 position DIP switch, S5, in the upper right-hand corner when the board component is side up and the gold finger are away from you. The leftmost switch, S5-1, enables or disables the four additional address lines as shown here:

Switch	Position	Function
S5-1	0	enables ADR14/ through ADR17/
S5-1	1	disables ADR14/ through ADR17/

4.8.4 Setting Address For Extended Lines

The board's location will be one of 16 possible one-megabyte pages. From the remaining four positions of the 5 position DIP switch S5, select one page as shown in Table 4-4.

One-megabyte page	Switches S5-2	S5-3	S5-4	S5-5
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table 4-4
Switch Setting for Each Page

NOTE

You cannot place the board across megabyte boundaries. Thus, if the board is in the 24 address line mode and if S5 is set to page 12 (C hex), then the combination of S2 and S3 setting may not extend the starting address below C00000 hex the ending address above CFFFFF hex

4.8.5 I/O Port Address

The I/O port base address is **PQR** hexadecimal,

where: **PQ** represents two hex digits (eight bits) in direct correspondence to the eight

bits of the I/O port base address switch, S4

R is a hex digit comprising the three more significant bits of **W** (from the module starting address) plus the state of ADR0/ from the bus.

Table 4-5 shows an example of how to set the I/O port base address.

PQ Switch S4 I/O Port Setting								R Switch S2 Start Address			Logical ADR0	(PQR) I/O Port Addr	REG
-1	-2	-3	-4	-5	-6	-7	-8	-1	-2	-3			
0	0	0	0	0	0	0	0	0	0	0	L	000	CSR
0	0	0	0	0	0	0	0	0	0	0	H	001	ESR
0	1	0	1	0	1	1	0	0	0	0	L	560	CSR
0	1	0	1	0	1	1	0	0	0	0	H	561	ESR
1	0	1	0	0	1	0	1	0	0	1	L	A52	CSR
1	0	1	0	0	1	0	1	0	0	1	H	A53	ESR
1	1	1	1	1	1	1	1	1	0	0	L	FF8	CSR
1	1	1	1	1	1	1	1	1	0	0	H	FF9	ESR

Table 4-5
Address Settings for I/O Port

CHAPTER 5

WD1000 AND COUPLER/FLOPPY BOARD SET

5.1 INTRODUCTION

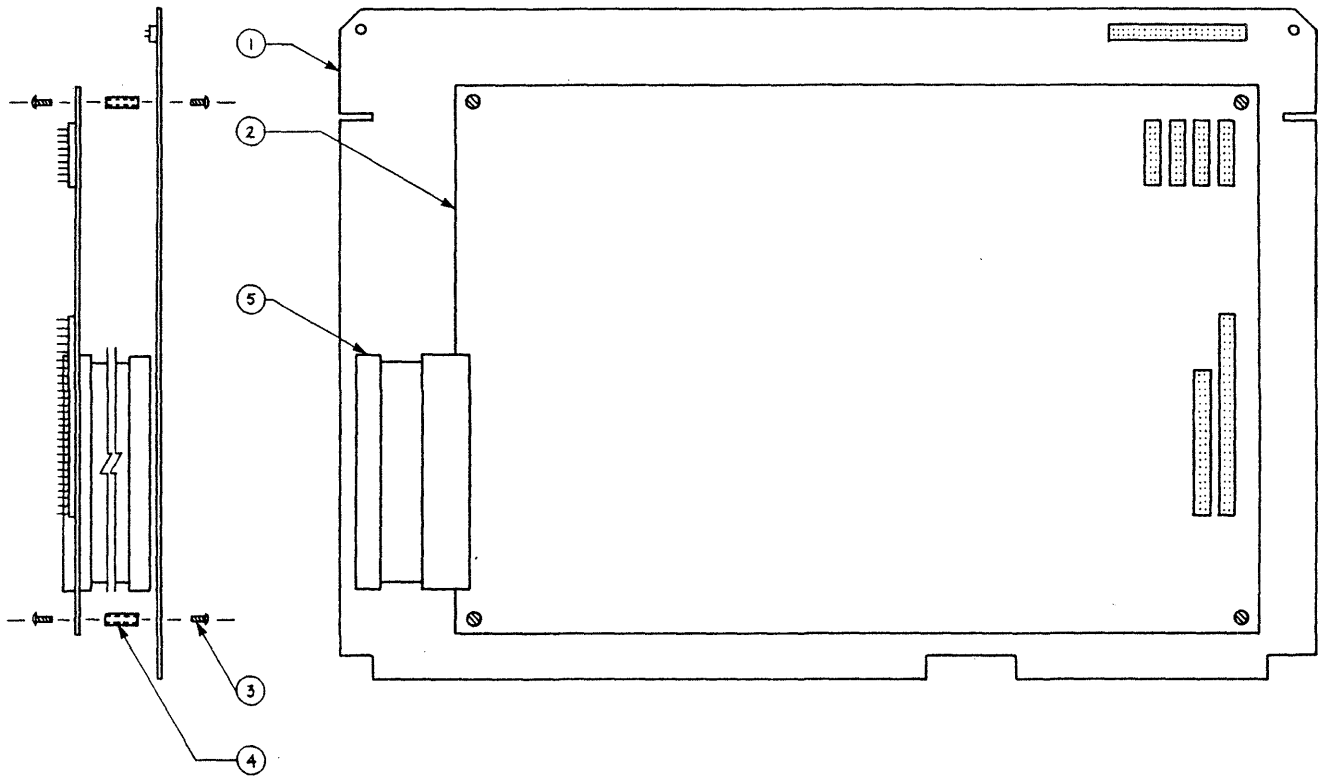
This chapter describes the WD1000 and Coupler/Floppy Board Set, showing its configuration and explaining its functions and features.

5.2 DEFINITION AND FEATURES

Floppy disk control is handled by the WD1000 and Coupler/Floppy Board set designed by Western Digital. The set is formed by physically and logically connecting the WD1000 board and the Coupler/Floppy board. Figure X-1 shows the board set configuration.

5.3 WD1000 AND COUPLER/FLOPPY BOARD SET CONFIGURATION

The WD1000 Coupler/Floppy Controller board inserts into the mother board in the bottom of the chassis. The WD1000 board is attached to the coupler by four three-eighths inch standoffs. The circuitry of the two boards interface via the coupler cable, as shown in Figure 5-1.



Legend:

1 - WD1000 Coupler Board
2 - WD1000 Board
3 - Screw
4 - Standoff
5 - WD1000 Coupler Cable

Figure 5-1
WD1000 and Coupler/Floppy Board Set Configuration

5.4 COUPLER/FLOPPY CIRCUITRY AREAS

The circuitry of the WD1000 Coupler/Floppy board is divided into two areas. The first area acts as a coupler and Direct Memory Access (DMA) controller for the WD1000 Disk Controller, and has five areas of circuitry (see Section 5.4). The second area is a 5 1/4" Floppy Disk Drive controller and has two areas of circuitry (see Section 5.5).

Figure 5-2 shows the circuitry areas on the Coupler/Floppy Board.

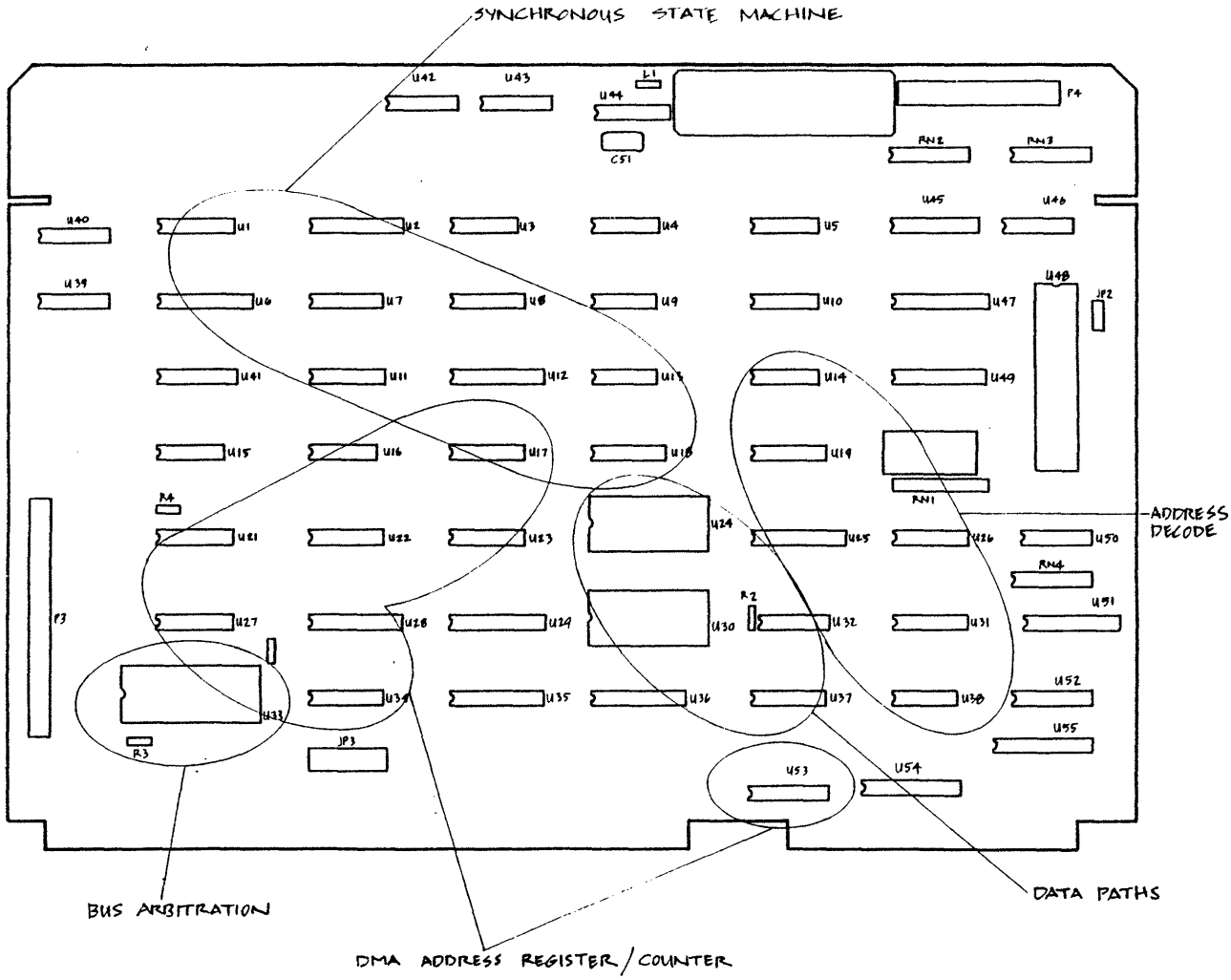


Figure 5-2
WD1000 Coupler/Floppy Board Circuitry

The circuitry areas are logically distinct and are discussed in the following sections.

5.4.1 WD1000 Coupler/DMA Controller

The heart of the coupler portion of the board is an 8 MHz clock synchronous state machine, which allows the flexibility and speed required to interface between the WD1000 board and the IEEE 497 bus.

The coupler has five major circuitry areas:

- A synchronous state machine (Section 5.3.1.1)
- Address decode (Section 5.3.1.2)
- DMA address register (Section 5.3.1.3)
- Bus arbitration (Section 5.3.1.4)
- Data Paths (Section 5.3.1.5)

5.4.1.1 Synchronous State Machine -

The following shows the integrated circuits involved in the state machine and their functions. (Refer to Figure 5-2.)

NOTE

In the following examples, a "U" followed by a number (such as U2, U33, etc.) refers to a specific integrated circuit on the state machine. Notations such as U33-25 refer to a specific pin location on that integrated circuit.

Functions	IC's
State register	U2
Input forming logic	U7,U11
Decision variable selection	U18 (state machine inputs)
Input Synchronization	U13
Output forming logic	U1,U8
Output latching	U6,U12 (state machine outputs)

5.4.1.2 Address Decode -

The address decoding is performed by the following:

Functions	IC's
Board select	U14,U19,U26
Onboard selects	U31,U38

5.4.1.3 DMA Address Register/Counter -

Direct Memory Access (DMA) allows direct access to main memory for the transfer of data without using the CPU. Those IC's on the board that accomplish this function are indicated below.

The DMA address is 24 bits wide and is generated by:

Functions	IC's
DMA register/counter	U23,U17,U22,U27,U34,U53
Address buffer/driver	U29,U28,U35,U54

5.4.1.4 Bus Arbitration -

Bus arbitration (see Section 1.X Bus Theory) is accomplished using the Intel 8218 Integrated Circuit (IC) U33. When the state machine wants the bus it brings the signal Bus Control Request (BCR) U33-25 (IC number U33, pin number 25) high. When the 8218 has received the bus it brings ADEN (address enable) U33-19 low.

5.4.1.5 Data Path -

The data path is a transfer bus for input/output and data handling operations. Data going to the WD1000 passes through U30. ICs U24 and U30 are used as data latchs to interface a 16-bit bus to an 8-bit bus. IC U37 is used to force the WD1000 address lines low during DMA.

5.4.2 Two-mode Operation Of Coupler

The coupler operates in two modes: register interface and DMA controller modes.

5.4.2.1 Register Interface -

While in register interface mode the coupler acts as a bus slave allowing the host system to read and write to the WD1000 internal registers. In this mode the state machine converts the interface signals of the Multibus to the signals required by the WD1000, ensuring that the timing requirements of both interfaces are met. These registers pass all status, parameters, and commands to and from the WD1000 disk and controller.

5.4.2.2 DMA Controller -

The state machine services all WD1000 board requests for a DMA cycle. During a DMA cycle the coupler board becomes the system bus master, reads or writes a 16-bit word to system memory and writes two 8-bit bytes to the WD1000 internal sector buffer. Bus control is relinquished between each DMA cycle and returned to the processor.

5.4.3 Floppy Controller Circuitry

The second circuitry area of the Coupler Board controls the 5 1/4" floppy disk drive. The Western Digital 1795 chip works with two other WD chips: the 1691 and the 2143 (see Figure 5-1) that do the data separation. The floppy controller interfaces with the Multibus using a simple logic. The floppy interface excludes a DMA controller, and therefore the system processor must do the system interface.

5.4.3.1 Data Separator -

The 1691 chip, used with a 74LS269 Voltage-controlled oscillator (VCO), performs the data separation.

5.4.3.2 Write Precompensation -

Each track on the disk contains the same number of sectors; therefore, the inner tracks are more compressed. Read errors can occur because of data that has shifted location because of magnetic interference between the bits. The 1691 chip, used with the 2143, "precompensates" the data for the inner tracks by anticipating the deviation and writing to that anticipated location. The 2143 supplies four timing signals to the 1691.

5.5 WD1000 BOARD CONFIGURATION

The WD1000 is a stand-alone, general purpose Winchester Controller board that interfaces up to four Winchester disk drives to a host processor. All necessary buffers and receivers/drivers are included on the board to allow direct connection to the drive. Both 34 pin (5-1/4" drive) and 50 pin (8" drive) connectors are provided, as well as four 20 pin data connectors.

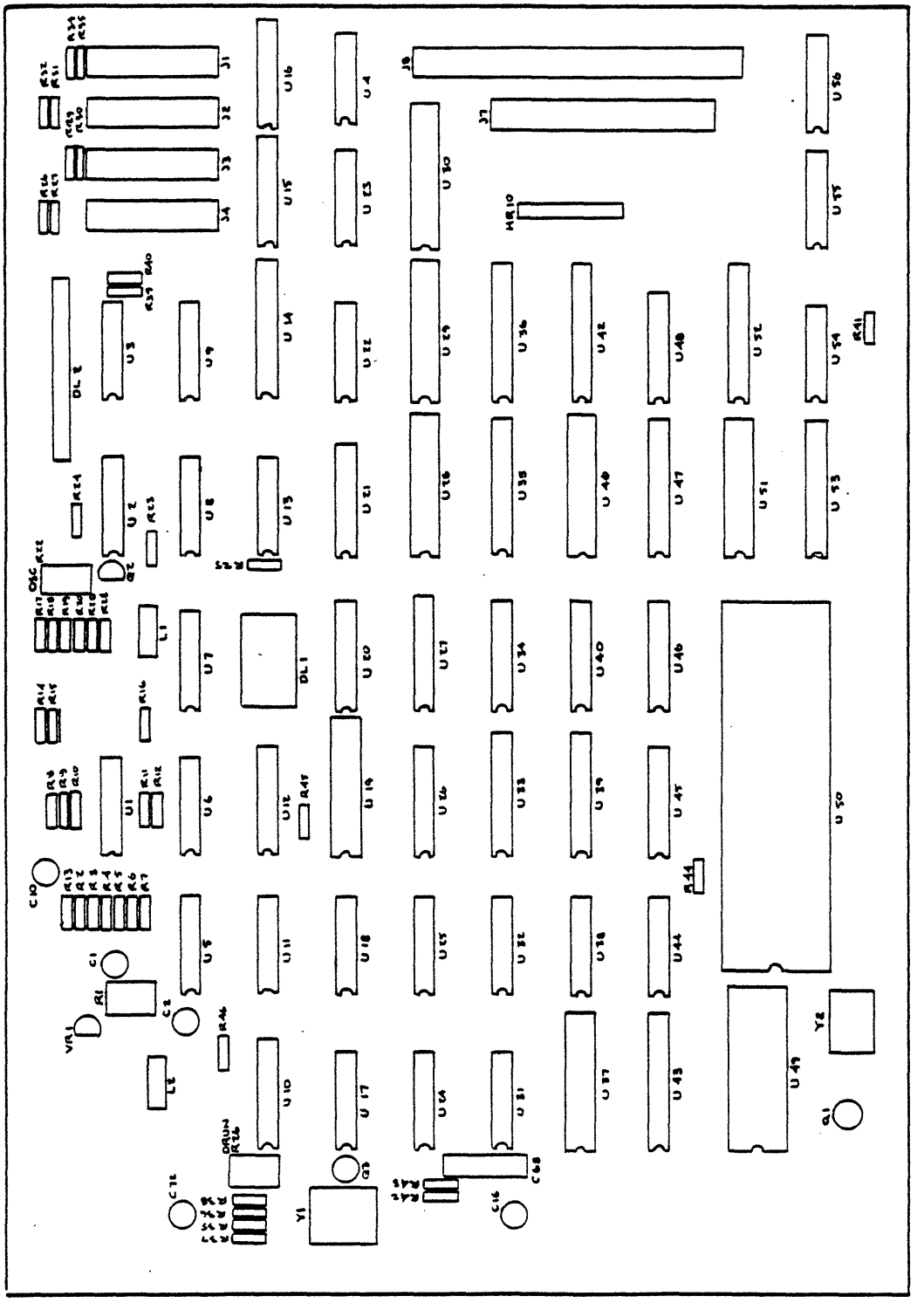


Figure 5-3
WD1000 Board Circuitry Areas

5.6 BOARD SET ELECTRICAL DATA

The following is a list of electrical data and specifications that pertain to the WD1000 Coupler/floppy and the WD1000 Disk Controller as a board set.

ITEM	SPECIFICATION
Winchester Controller	
Encoding Method:	Modified Frequency Modulation (MFM)
Cylinders per Head:	Up to 1024
Sectors per Track:	Up to 256
Heads per Drive:	8
Drive Selects:	4
Step Rate:	10 uS to 7.5 mS (0.5 mS increments)
Write Precomp:	Yes
Sectoring:	Soft
Drive Cable Length:	8 ft. max
DMA Address:	24 bits
DMA Data:	16 bits
Floppy Controller	
Sectoring:	Soft
Density:	Double (MFM)
Format:	IBM System 34
Sector Length:	128, 256, 512, 1024
Cylinders per Head:	Up to 256
Heads per Drive:	2
Drive Selects:	4
Write Precomp:	Yes
Drive Cable Length:	8 ft.
Voltages	+ 5 volts + 12 volts - 12 volts
Board address	Decodes lower 12 address lines Base address selectable on 32-byte boundary

5.7 MECHANICAL DATA

The WD1000 and Coupler board set interfaces with the Extended-Multibus. The board-set meets all electrical specification of the bus, but it does not meet the mechanical size specification. The following are the mechanical specifications for the board-set.

ITEM	SPECIFICATION
WD1000 board:	6.83" X 9.88"
Coupler board:	12" X 8.3"

5.8 ENVIRONMENTAL DATA

The following is the environmental specification for the board set.

ITEM	SPECIFICATION
Ambient Temperature operating:	0 C to 50 C
Relative Humidity operating:	20% to 80% noncondensing

5.9 SOFTWARE INTERFACE DATA

Thirty-two registers communicate the command and status information. The registers are divided into two groups of 16, one group for the Winchester disk drives and one group for the floppy disk drives. The registers are offset from a base address that can be located on any 32-byte boundary in the I/O space. The following is an I/O map for the controller's registers. (Note: The addresses are 68000 addresses. The least significant address on the 68000 (ADDR0) is inverted to that of the Multibus.

5.10 BASE ADDRESS

System 150 (mapped) - F00180 hex

5.11 WINCHESTER REGISTERS

I/O port Address	Input Command I/O Read	Output Command I/O Write
Base + 0	Error Register	Write Precomp
Base + 1	Data Register	Data Register
Base + 2	Sector Number	Sector Number
Base + 3	Sector Count	Sector Count
Base + 4	Cylinder High	Cylinder High
Base + 5	Cylinder Low	Cylinder Low
Base + 6	Status Register	Command Register
Base + 7	Size/Drive/Head	Size/Drive/Head
Base + 8	Reserved	Reserved
Base + 9	Reserved	Reserved
Base + 10	Reserved	Reserved
Base + 11	Reserved	Reserved
Base + 12	Not Used	DMA Addr. 9-16
Base + 13	Not Used	DMA Addr. 1-8
Base + 14	Reserved	DMA R/W
Base + 15	Not Used	DMA Addr. 17-23

5.12 FLOPPY REGISTERS

I/O port Address	Input Command I/O Read	Output Command I/O Write
Base + 16	Status Register	Command Register
Base + 17	Track Register	Track Register
Base + 18	Sector Register	Sector Register
Base + 19	Data Register	Data Register
Base + 20	Reserved	Reserved
Base + 21	Reserved	Reserved
Base + 22	Reserved	Reserved
Base + 23	Reserved	Reserved
Base + 24	Interrupt status	Drv Sel/Intr Dis
Base + 25	Reserved	Reserved
Base + 26	Reserved	Reserved
Base + 27	Reserved	Reserved
Base + 28	Reserved	Reserved
Base + 29	Reserved	Reserved
Base + 30	Reserved	Reserved
Base + 31	Reserved	Reserved

5.12.1 Description Of Floppy Registers

The following is a brief description of the function of each of the Floppy registers. Motor control is transparent to the software because of the head load and head load timing signal of the 1795 chip.

5.12.1.1 Status Register (Read Only) -

The status register is updated after each command. The meaning of the bits of the status register may change following the different commands. The status is not valid until 28 uSec. after a write to the command register in double density, and 56 uSec. in single density.

5.12.1.2 Command Register (Write Only) -

Commands are written to this register. Upon a write signal to the register the command is started. Each command has several flags, set as follows:

V = 0	No verify
h = 1	unload head at beginning
T = 1	Update track register
a0 = 0	Normal data address mark
U	is set to the desired head 0 or 1
E = 1	15 ms head settle time
L = 1	for IBM compatability
m = 0	single record

5.12.1.3 Track Register (Read/Write) -

The track register contains the present position of the drive heads. If a new drive is to be selected this register has to be saved and the new drive's track position written to the register.

5.12.1.4 Data Register (Read/Write) -

This register is the port through which all the data passes to and from the disk drives. Once a data transfer begins, this register must be read or written to every 23 uSec.

5.12.1.5 Drive Select/Interrupt Register -

This register is used to select the floppy drive. It is not part of the 1795 but is located on the coupler board. The following hexadecimal pattern corresponds to each of the drives.

Drive Select with Interrupts Disabled

DRIVE0 = FE hex
DRIVE1 = FD hex
DRIVE2 = FB hex
DRIVE3 = F7 hex
NO DRIVE = FF hex

Drive Select with Interrupts Enabled

DRIVE0 = 7E hex
DRIVE1 = 7D hex
DRIVE2 = 7B hex
DRIVE3 = 77 hex
NO DRIVE = 7F hex

5.13 SIGNAL DEFINITIONS

A0 - A17	Address 0 - Address 17
ADEN	Address Enable (means coupler has bus)
ADDR0 - ADR17	Address 0 - Address 17 (DMA)
BCLK	Bus Clock
BCR	Bus Control Request
BHEN	Bus High Enable
BOARD SEL	Board Select
BPRN	Bus Priority In
BPRO	Bus Priority Out
BREQ	Bus Request
BUSY	Bus Busy
CONTR SEL	Controller Select (WD1000)
D0 - D7	Internal Data Bus
DATA REG SEL	Data Register Select
DMA R/W	DMA Read / Write
DRIVE LOAD	Drive Select Load
DRQ	DMA Request
FSEL	Floppy Controller Select
HBIC	High Byte Input Control (internal side)
HBOC	High Byte Output Control (internal side)
HWC	High Write Control (multibus side)
INC	Increment
INIT	Initialize (reset)
INT0 - INT7	Interrupt 0 - 7
INT DIS	Interrupt Disable
INTRQ	Interrupt Request
IORC	I / O Read Control
IOWC	I / O Write Control
LBIC	Low Byte Input Control (internal side)
LBOC	Low Byte Output Control (internal side)
LOAD0 -LOAD3	DMA Register Load 0 - 3
LWC	Low Write Control (multibus side)
MR	Master Reset
MRDC	Memory Read Control
MWTC	Memory Write Control
Q0 - Q7	Present state 0 - 4
RC	Read Control (multibus side)
READ	Internal Read Strobe
VSEL0 - VSEL2	Variable Select 0 - 2
WAIT	Wait (WD1000 Handshake line)
WD1000 SEL	WD1000 Select
WRITE	Internal Write Strobe
XACK	Transfer Acknowledge
XACKBAK	Transfer Acknowledge Back
8MHZ	8 MHz Clock

APPENDIX A

I/O BOARD JUMPER PIN LOCATIONS BY PORT

The following chart arranges the jumper pins according to their respective ports.

Port 0	Port 1	Port 2	Port 3	Port 4	Port 5
JP1	JP2	JP3	JP4	JP5	JP6
	JP19	JP15	JP11	JP13	JP17
	JP20	JP16	JP12	JP14	JP18

APPENDIX B

JUMPER PIN INPUT/OUTPUT SIGNALS

The following chart shows the signals associated with each jumper pin, their functions, default state (on/off), and the status of the signal (input/output).

Pin #	Signal	Function	Defaut State	Input/Output
JP1	TxEMT		Off	
JP2	"		"	
JP3	"		"	
JP4	"		"	
JP5	"		"	
JP6	TxEMT		Off	
JP7				
JP8				
JP9				
JP10				
JP11	CTS/RTS		On	Both
JP12	DCR/DTR		"	"
JP13	"		"	"
JP14	CTS/RTS		"	"
JP15	DCR/DTR		"	"
JP16	CTS/RTS		"	"
JP17	"		"	"
JP18	DSR/DTR		"	"
JP19	"		"	"
JP20	CTS/RTS		"	"
JP21	"		"	"
JP22	DSR/DTR		"	"

APPENDIX C

WICAT DIAGNOSTIC MONITOR (DIAMOND)

C.1 INTRODUCTION

DIAMOND is a general purpose interactive program which incorporates the capabilities of a compiler, assembler, debugger, loader, and operating system within a single architecture.

Some of the attributes of DIAMOND are:

- core-efficiency
- high running speed
- extreme flexiblity

The flexiblity of the language permits the user to develop a working vocabulary of subroutines tailored to his specific application.

The most prominent feature of DIAMOND is its principal data structure, called the **dictionary.** The dictionary is an ordered list of entries called **words.** Associated with the dictionary entry for each word is a **name.** A legal name for a word is any string of up to 255 ASCII characters. All printing ASCII characters, including letters, numbers, and special characters may be freely used within a name, except for the SPACE and RUBOUT characters.

NOTE

1. Do not use non-printing characters or control characters in a name.

2. If you create a name that looks like a number (in any radix), DIAMOND will assume that you are entering a number. To avoid this problem, make sure there is at least one character in the name that is not part of the set "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F".

3. Although the RUBOUT character may not be used as part of a name, pressing the RUBOUT (DELETE on some terminals) key deletes the character to the left of the cursor and may be used to correct a typing error.

C.2 LITERALS

A **literal** is a sequence of characters which describes a constant. DIAMOND supports two types of literal: 32-bit **integer** and **string.**

C.2.1 Integer Literals

An **integer literal** is a sequence of digits optionally preceded by a plus or minus sign, in accordance with these rules:

1. All digits must be less than the current **radix** (also known as the **base**). For example, if the current radix is "DECIMAL", the digits 0 through 9 are valid. DIAMOND allows you to use BINARY, OCTAL, DECIMAL, or HEX as the radix.

   NOTE

   Although the default radix is 16 (HEX); to avoid confusion, the examples presented in this manual use a default radix of 10 (DECIMAL), unless otherwise specified.

2. No spaces may be embedded within an integer literal.
3. Integer literals must be in the range of -2,147,483,654 to 2,147,483,653 if signed or 0 to 4,294,967,308 if unsigned.
4. A sign is optional. No sign means the literal is unsigned.

EXAMPLES:

-1234	is a legal literal (unless the current radix is BINARY)
+-100	is not a legal literal (the second sign is illegal)
-AFC0	is a legal literal if the current radix is HEX (if not, the line is ignored and an error message is printed)

C.2.2 String Literals

String literals may take one of two forms:

1. A string enclosed in double quotes:

   "STRING"

   NOTE

   A carriage return before the second double quote will terminate the string.

2. A string preceded by one single quote and terminated by a SPACE, TAB, carriage RETURN, or FORM FEED:

   'STRING

EXAMPLES:

"THIS IS A STRING LITERAL"

'HELLO_THERE

C.3 SYNTAX

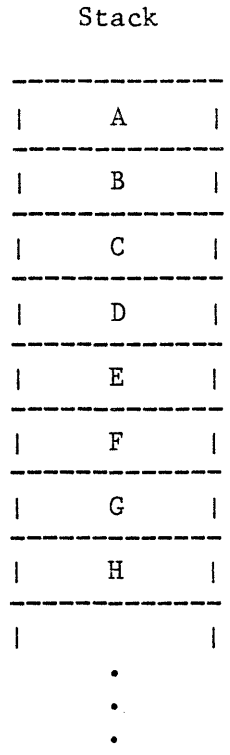
DIAMOND **syntax** is quite simple. A legal **command line** consists of a sequence of literals and/or names of words separated by spaces or tabs, and is terminated by a carriage RETURN.

Programming in DIAMOND consists primarily of defining a set of new words based on words which have already been defined. An initial vocabulary of about one hundred words called the KERNEL enables the user to get started.

C.3.1 The Stack

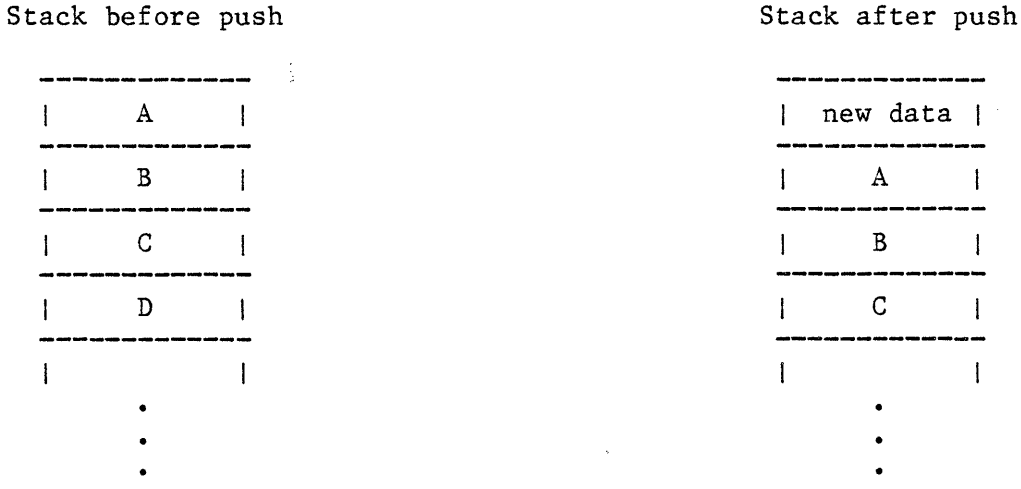
The principal vehicle for communication between words is the **parameter stack,** frequently called the **stack.** A stack is a common programming tool which allows the programmer to store information on a **last-in, first-out** method. An example of a stack is the tray dispenser in a cafeteria, where the only available tray is the top one, which must be removed before any of the other trays are accessible. Similarly, when clean trays come from the kitchen, they are placed on top of the trays already there. Thus, the most recently added trays are the first ones available.

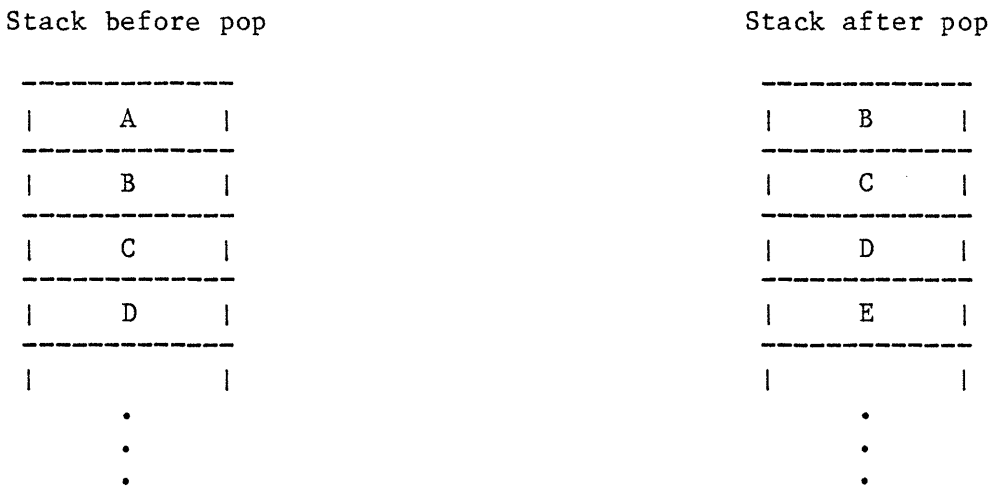
In many of the examples in the following sections, a picture like this will represent the stack:



In this example, "A" represents the entry at the top of the stack, "B" represents the next entry, and so on.

In computer terminology, adding a parameter to the top of the stack is called **pushing** the parameter, and removing the top parameter is called **popping**.





Typically, the parameters upon which a word will operate are pushed on the stack. The word pops its parameters from the stack and pushes its results on the stack.

C.3.2 Variables In Fixed Locations

DIAMOND also uses communication through variables in fixed locations (not on the stack). In most cases, to use a variable which is in a fixed location, place the address of the variable on the stack, then execute a word that takes the address off the stack and operates on the variable at this address.

C.3.3 Reverse-polish Notation

DIAMOND uses **reverse-polish notation (RPN)** for all operations. This means that all operands precede their operators and parentheses are never necessary.

Some hand-held scientific calculators use this method of entering instructions. For example:

1 1 + 2 *	in RPN is	(1+1)*2	in algebraic notation
1 2 3 * -	in RPN is	1-(2*3)	in algebraic notation

C.3.4 Addresses Versus Contents

Unlike most high-level languages, DIAMOND enables the user to manipulate **addresses** and **data**. It is important, however, for the user to remain aware of the distinction between an **address** and its **contents**.

Three common types of words which push numbers on the stack: literals, constants, and variables.

- A reference to a **literal** or a **constant** causes its **value** to be pushed on the stack.
- A reference to a **variable** causes its **address** to be pushed on the stack.

The operators "@" and "!" are used to obtain and modify the value of a variable. They are defined on the following pages.

NOTE

1. In the examples on the following pages, you will see the characters "0] " at the left end of some of the lines. This is the **prompt message** which DIAMOND prints to show that it is ready for input. This prompt is described in section A.7 below.

2. If you are familiar with RPN, you may be confused by the use of an equals sign (=) in some of the examples. As described in section A.4.4, DIAMOND uses the equals sign to output the top number on the stack. The calculations specified are carried out on the stack whether or not the equals sign is present.

C.3.4.1 @

This word is used to load the contents of a memory location (byte) onto the stack. It replaces the address on the top of the stack by the contents of that address.

For example:

0] 341 @ = prints the value of the byte at location 341

C.3.4.2 @W

This word is used to load the contents of the word (2 bytes) beginning at the memory location on top of the stack. It replaces the address on the top of the stack by the contents of that address. If the address on top of the stack is an odd byte location, an address error occurs.

For example:

0] 342 @W = prints the value of the word which begins at location 342

C.3.4.3 @L

This word is used to load the contents of the longword (4 bytes) beginning at the memory location on top of the stack. It replaces the address on the top of the stack by the contents of that address. If the address on top of the stack is an odd byte location, an address error occurs.

For example:

0] 340 @L = prints the value of the longword which begins at location 340

C.3.4.4 !

Store at the address on the top of the stack the number (byte) next to top of the stack. Both numbers are removed from the stack.

(In this set of examples, A, B, and C represent constants, and X, Y, and Z represent variables.)

For example:

0] 100 X !	set value of X to 100
0] X Y !	set the value of Y to the address of X
0] X 10000 !	store the address of X in location 10000
0] X @ Y !	set the value of Y to the value of X
0] X @ Y @ + Z !	add the values of X and Y and store the result in Z
0] X A + Y !	store (address of X)+A in Y
0] X A B + + @ Y !	set value of Y to the contents of location X+A+B

C.3.4.5 !W

Store at the address on the top of the stack the word (2 bytes) contained in the entry next to the top of the stack. Both numbers are removed from the stack. If the address on top of the stack is an odd byte location, an address error occurs.

For example:

0] 100 X !W set value of X to 100

C.3.4.6 !L

Store at the address on the top of the stack the longword (4 bytes) contained in the entry next to top of the stack. Both numbers are removed from the stack. If the address on top of the stack is an odd byte location, an address error occurs.

For example:

0] 100 X !L set value of X to 100

C.3.4.7 .!<- OPERATOR

C.4 FIXED POINT OPERATORS

DIAMOND provides the user with a large number of fixed-point operators. Unless otherwise specified, all numbers used as arguments to these fixed point operators are 32-bit (4 byte) integers.

C.4.1 UNARY OPERATORS

The following operators replace the number on top of the stack (usually called "A") with their result.

C.4.1.1 MINUS

Negates A.

For example:

```
0] 1 MINUS =
-1
```

C.4.1.2 .!ABS

C.4.1.3 **NOT**

Computes the logical complement of A.

C.4.1.4 .!2*

C.4.1.5 .!2/

C.4.1.6 .!1+

C.4.1.7 .!1-

C.4.1.8 EQZ

Tests if A is equal to zero. Replaces A by 0 if A is equal to 0, -1 otherwise.

* = 0 or -1

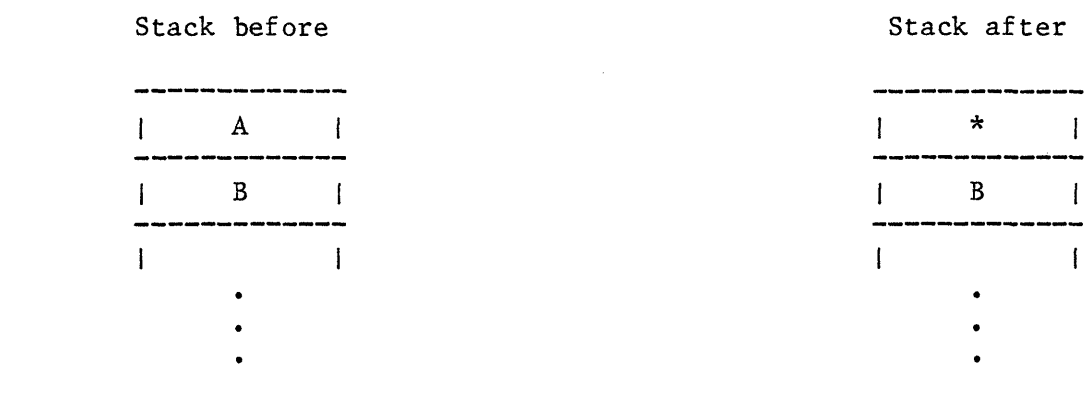
For example:

```
0] -1 EQZ =
-1

0] 0 EQZ =
0
```

C.4.1.9 NEZ

Tests if A is not equal to zero. Replaces A by 0 if A is not equal to 0, -1 otherwise.



* = 0 or -1

For example:

```
0] -1 NEZ =
0

0] 0 NEZ =
-1
```

C.4.1.10 LTZ

Tests if A is less than zero. Replaces A by 0 if A is less than 0, -1 otherwise.

* = 0 or -1

For example:

```
0] -1 LTZ =
0

0] 0 LTZ =
-1

0] 1 LTZ =
-1
```

C.4.1.11 LEZ

Tests if A is less than or equal to zero. Replaces A by 0 if A is less than or equal to 0, -1 otherwise.

* = 0 or -1

For example:

```
0] -1 LEZ =
0

0] 0 LEZ =
0

0] 1 LEZ =
-1
```

C.4.1.12 GEZ

Tests if A is greater than or equal to zero. Replaces A by 0 if A is greater than or equal to 0, -1 otherwise.

* = 0 or -1

For example:

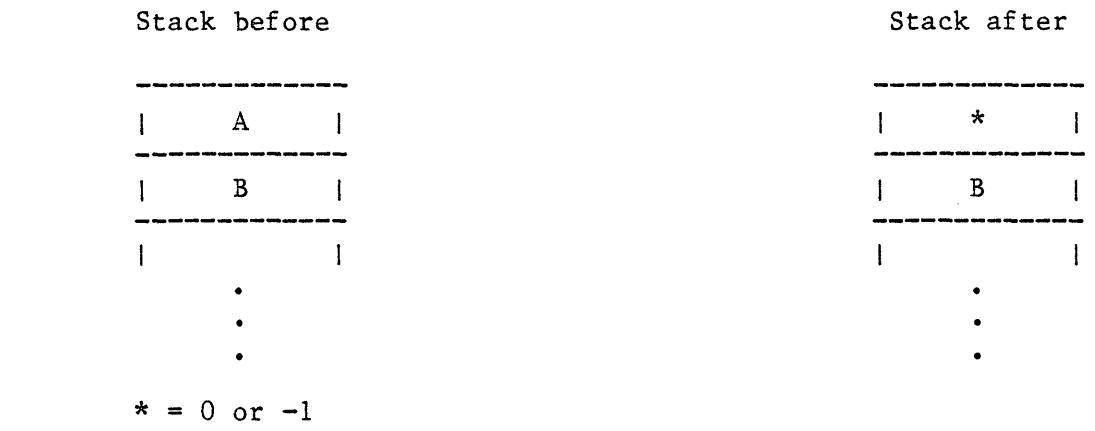
```
0] -1 GEZ =
-1

0] 0 GEZ =
0

0] 1 GEZ =
0
```

C.4.1.13 GTZ

Tests if A is greater than zero. Replaces A by 0 if A is greater than 0, -1 otherwise.



For example:

```
0] -1 GTZ =
-1

0] 0 GTZ =
-1

0] 1 GTZ =
0
```

C.4.1.14 SPLIT

Splits the longword on top of the stack into two words. The two words replace the top two stack entries. SPLIT sign-extends the results. In other words, the high order bit of each resulting word is propagated to the left to fill up the longword stack entry. If the first bit is 0, the stack entry will be left-filled with zeroes. If the first bit is 1, the stack entry will be left filled with Fs (hex). (See the example below).

For example:

```
0] 12345678 SPLIT = =
1234 5678

0] HEX FEEE7EEE SPLIT = =
FFFFFEEE 7EEE
```

(The second number is left-filled with zeroes, which do not print.)

C.4.1.15 SPLITB

Splits the word on top of the stack into two bytes. The two bytes replace the top two stack entries. SPLITB works just like SPLIT execpt it ignores the top word of the longword on top of the stack. SPLITB sign-extends the results as does SPLIT.

For example:

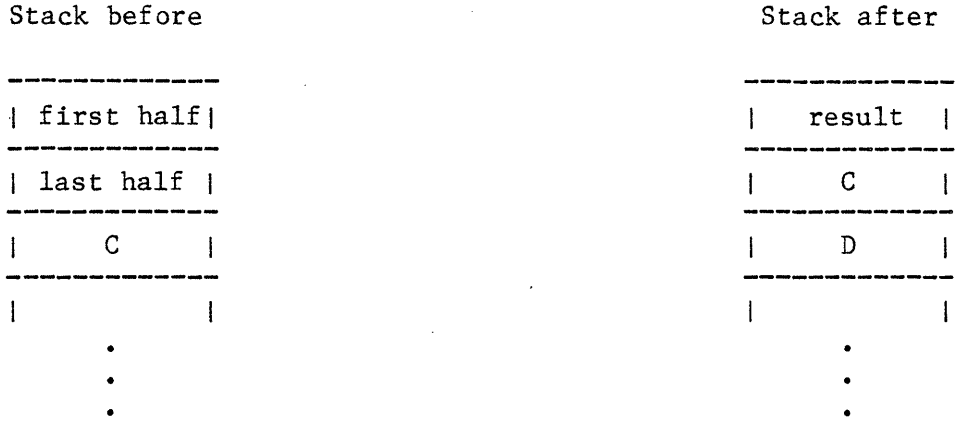
```
0] 1234 SPLITB = =
12 34

0] HEX FE7E SPLITB = =
FFFFFFFE 7E
```

(The second number is left-filled with zeroes, which do not print.)

C.4.1.16 JOIN

Combines the bottom words of the top two stack entries to form a new longword, which is placed on top of the stack.



For example:

```
0] 1234 5678 JOIN =
56781234
```

C.4.1.17 JOINB

Combines the bottom bytes of the top two stack entries to form a new word, which is placed on top of the stack.

For example:

```
0] 12 34 JOINB =
3412
```

C.4.2 BINARY OPERATORS

The following operators replace the top two numbers on the stack with their results.

C.4.2.1 +

Computes B+A.

For example:

```
0] 1 1 + =
2

0] 1 1 1 + + =
3
```

C.4.2.2 -

Computes B-A.

For example:

```
0] 2 1 - =
1
```

C.4.2.3 *

Computes B*A.

For example:

```
0] 2 3 * =
6
```

C.4.2.4 /

Divides B by A and returns the quotient and remainder.

For example:

```
0] 8 2 / = =
4 0

0] 7 2 / = =
3 1
```

"A modulo B" can be computed as:

```
0] A B / DROP
```

For example:

```
0] 5 3  / DROP =
2
```

C.4.2.5 /U

Performs an unsigned divide of B by A and returns the quotient and remainder.

For example:

```
0] 8 2 /U = =
4 0
```

C.4.2.6 MAX

Returns the signed maximum of B and A.

For example:

```
0] -1 5 MAX =
5
```

C.4.2.7 MIN

Returns the signed minimum of B and A.

For example:

```
0] -1 5 MIN =
-1
```

C.4.2.8 AND

Returns the logical AND of B and A.

For example:

```
0] 1 0 AND =
0

0] 1 1 AND =
1
```

C.4.2.9 OR

Returns the logical OR of B and A.

For example:

```
0] 0 0 OR =
0

0] 1 0 OR =
1
```

C.4.2.10 XOR

Returns the logical EXCLUSIVE OR of B and A.

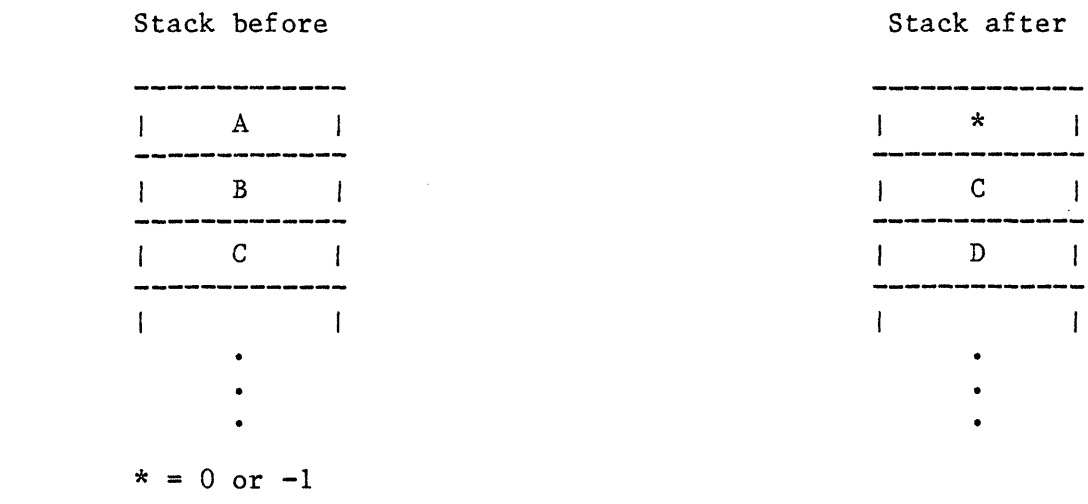
For example:

```
0] 1 0 XOR =
1

0] 1 1 XOR =
0
```

C.4.2.11 EQ

Tests if B is equal to A. Returns 0 if B is equal to A, -1 otherwise.



* = 0 or -1

For example:

```
0] 1 2 1 - EQ =
-1
```

C.4.2.12 NE

Tests if B is not equal to A. Returns 0 if B is not equal to A, -1 otherwise.

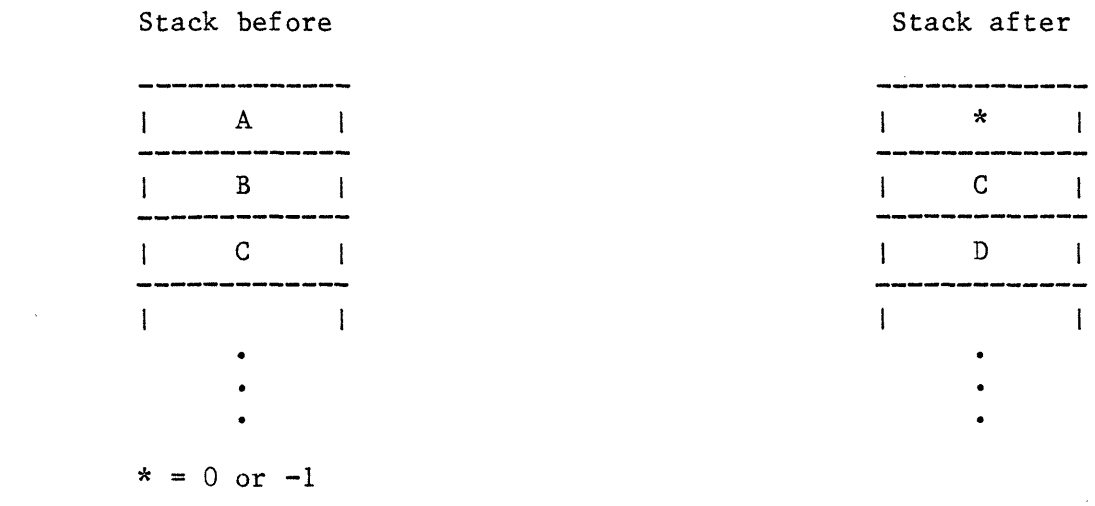
* = 0 or -1

For example:

```
0] 1 3 2 - NE =
-1
```

C.4.2.13 LT

Tests if B is less than A. Returns 0 if B is less than A, -1 otherwise.



* = 0 or -1

For example:

```
0] 5 4 LT =
0

0] 7 9 LT =
-1
```

C.4.2.14 LE

Tests if B is less than or equal to A. Returns 0 if B is less than or equal to A, -1 otherwise.

* = 0 or -1

For example:

```
0] 5 4 LE =
0
```

C.4.2.15 GE

Tests if B is greater than or equal to A. Returns 0 if B is greater than or equal to A, -1 otherwise.

* = 0 or -1

For example:

```
0] 5 4 GE =
0
```

C.4.2.16 GT

Tests if B is greater than A. Returns 0 if B is greater than A, -1 otherwise.

* = 0 or -1

For example:

```
0] 6 5 GT =
0
```

C.4.2.17 LSL

Performs a logical shift left of the data in B by the number of bits in A.

NOTE

Logical shifts should be used only on **unsigned** numbers.

For example:

```
0] HEX 7FFFFFFF 1 LSL =
FFFFFFFE

0] DECIMAL 1 3 LSL =
8
```

C.4.2.18 LSR

Performs a logical shift right of the data in B by the number of bits in A.

NOTE

Logical shifts should be used only on **unsigned** numbers.

For example:

```
0] HEX FFFFFFFF 1 LSR =
7FFFFFFF
```

C.4.2.19 ASL

Performs a arithmetic shift left of the data in B by the number of bits in A.

NOTE

Arithmetic shifts should be used only on **signed** numbers.

For example:

```
0] HEX 8FFFFFFF 1 ASL =
-00000002

0] HEX 8FFFFFFF 1 ASL =U
9FFFFFFF
```

C.4.2.20 **ASR**

Performs a arithmetic shift right of the data in B by the number of bits in A.

NOTE

Arithmetic shifts should be used only on **signed** numbers.

For example:

```
0] HEX FFFFFFFF 1 ASR =U
FFFFFFFF

0] HEX 7FFFFFFF 1 ASR =U
3FFFFFFF

0] HEX 8FFFFFFF 1 ASR =U
C7FFFFFF
```

C.4.3 Stack Operators

A number of operators are also provided whose sole function is to reorganize the elements of the stack:

C.4.3.1 **DUP**

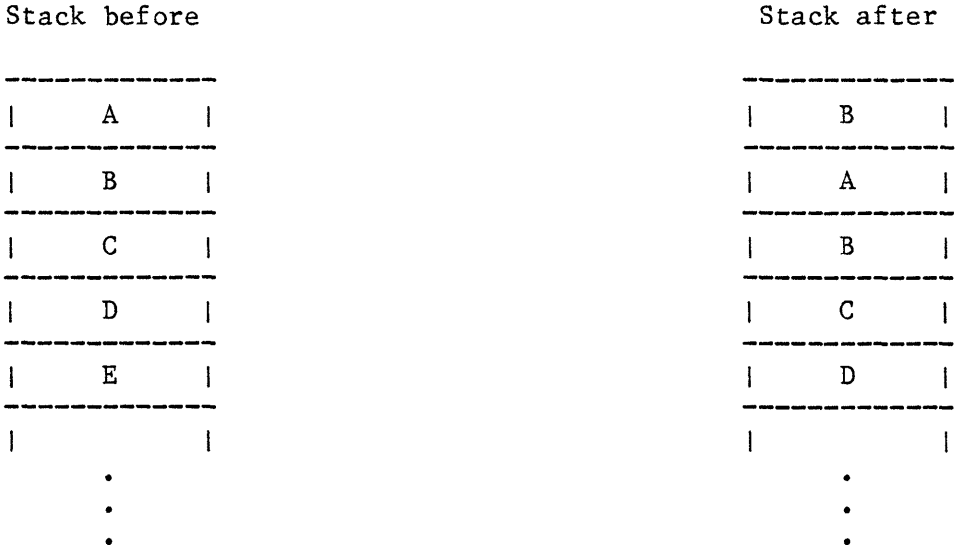
Places a duplicate of A on top of the stack. The rest of the stack is undisturbed.

For example:

```
0] 1 2 DUP = = =
2 2 1
```

C.4.3.2 **OVER**

Places a duplicate of B on top of the stack. The rest of the stack is undisturbed.



For example:

```
0] 1 2 OVER = = =
1 2 1
```

C.4.3.3 2OVER

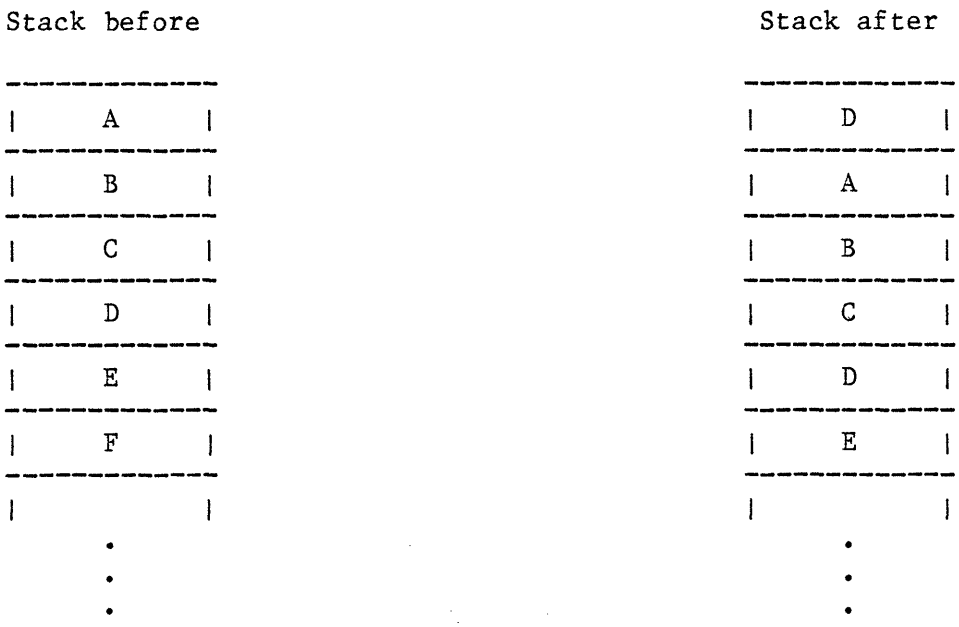
Places a duplicate of C on top of the stack. The rest of the stack is undisturbed.

For example:

```
0] 1 2 3 2OVER = = = =
1 3 2 1
```

C.4.3.4 3OVER

Places a duplicate of D on top of the stack. The rest of the stack is undisturbed.



For example:

```
0] 1 2 3 4 3OVER = = = = =
1 4 3 2 1
```

C.4.3.5 **UNDER**

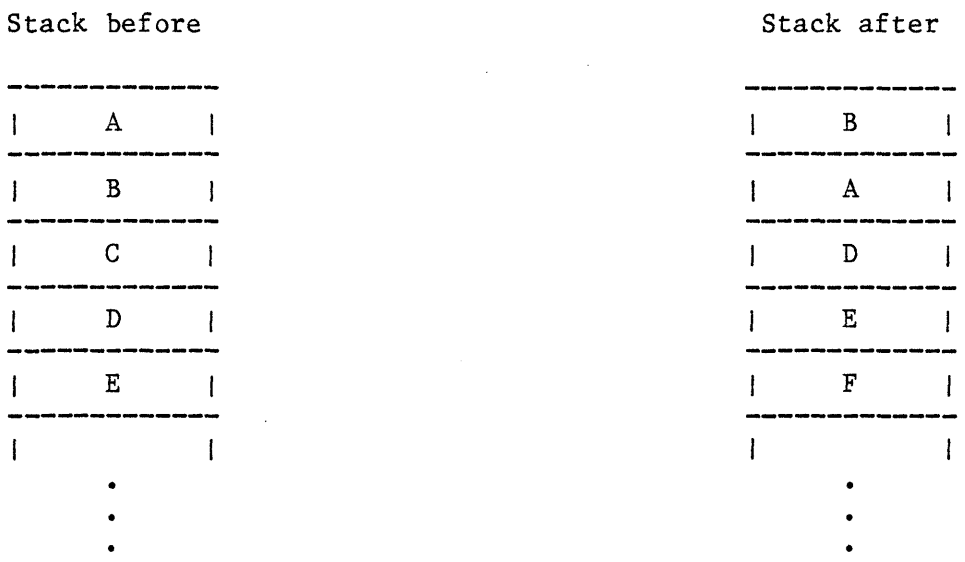
Replaces B by A and moves the stack pointer down one position.

For example:

```
0] 1 2 3 UNDER = =
3 1
```

C.4.3.6 2UNDER

Replaces C by A and moves the stack pointer down one position.

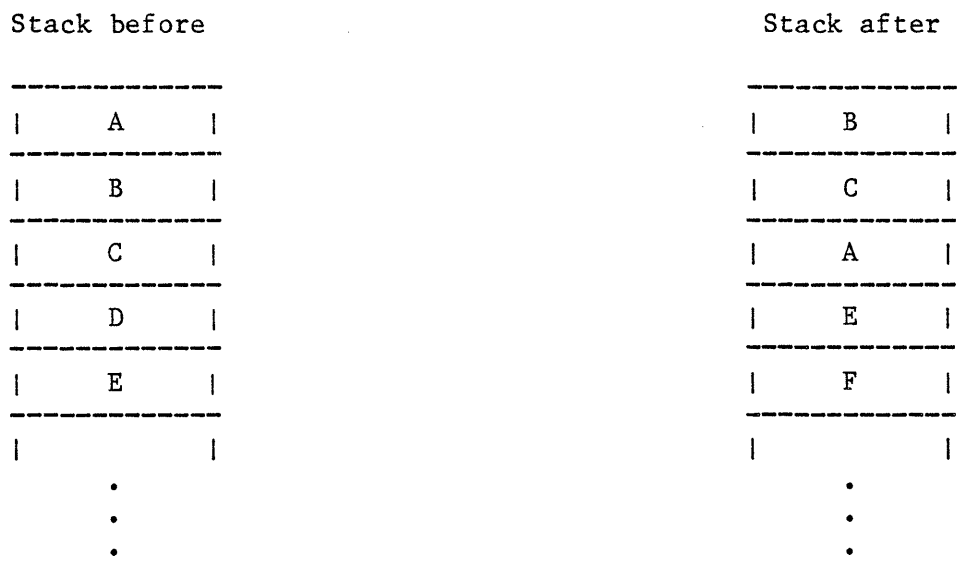


For example:

```
0] 4 3 2 1 2UNDER = = =
1 2 4
```

C.4.3.7 3UNDER

Replaces D by A and moves the stack pointer down one position.



For example:

```
5 4 3 2 1 3UNDER = = = =
2 3 1 5
```

C.4.3.8 DROP

Discards the number at top of the stack.

For example:

```
0] 1 2 3 DROP = =
2 1
```

C.4.3.9 SWAP

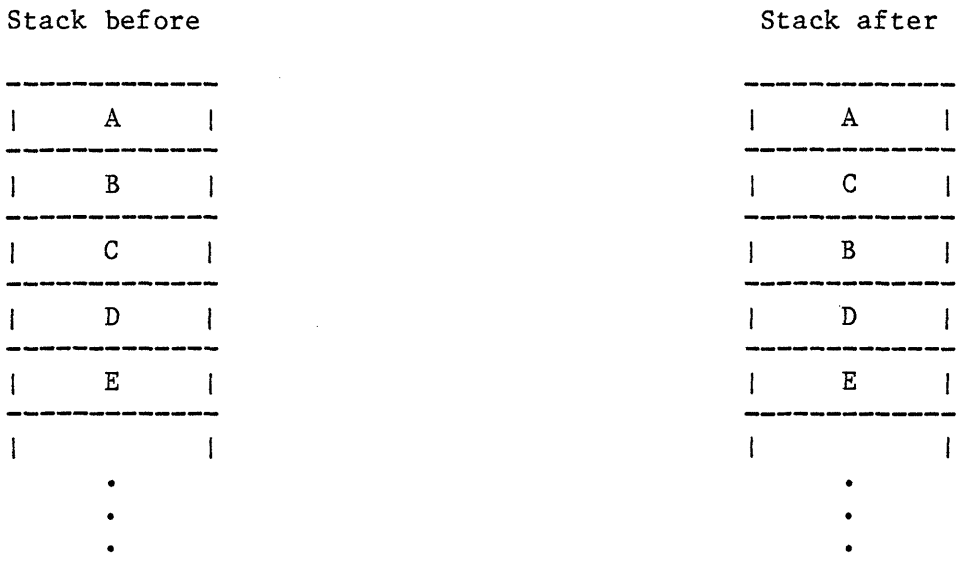
Exchanges the top two entries on the stack.

For example:

```
0] 1 2 SWAP = =
1 2
```

C.4.3.10 **2SWAP**

Exchanges B and C.

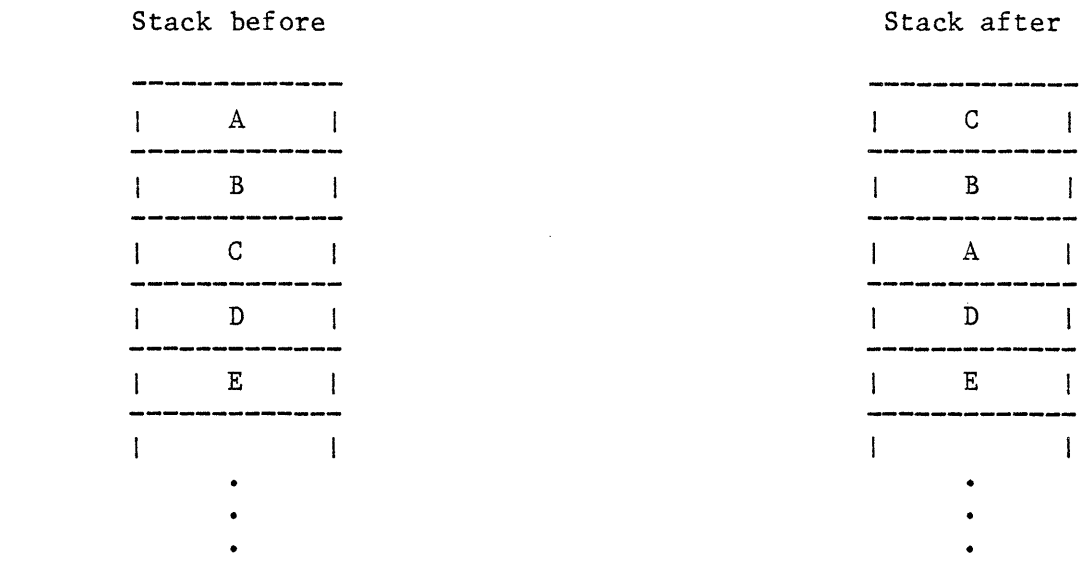


For example:

```
0] 1 2 3 4 2SWAP = = = =
4 2 3 1
```

C.4.3.11 FLIP

Exchanges A and C.

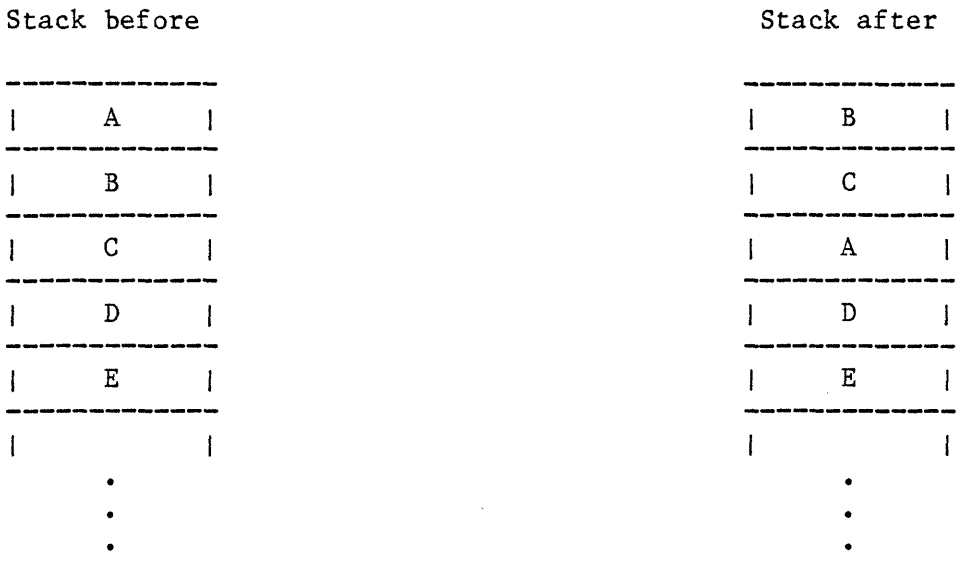


For example:

```
0] 1 2 3 4 FLIP = = = =
2 3 4 1
```

C.4.3.12 +ROT

Rolls top 3 stack entries up.

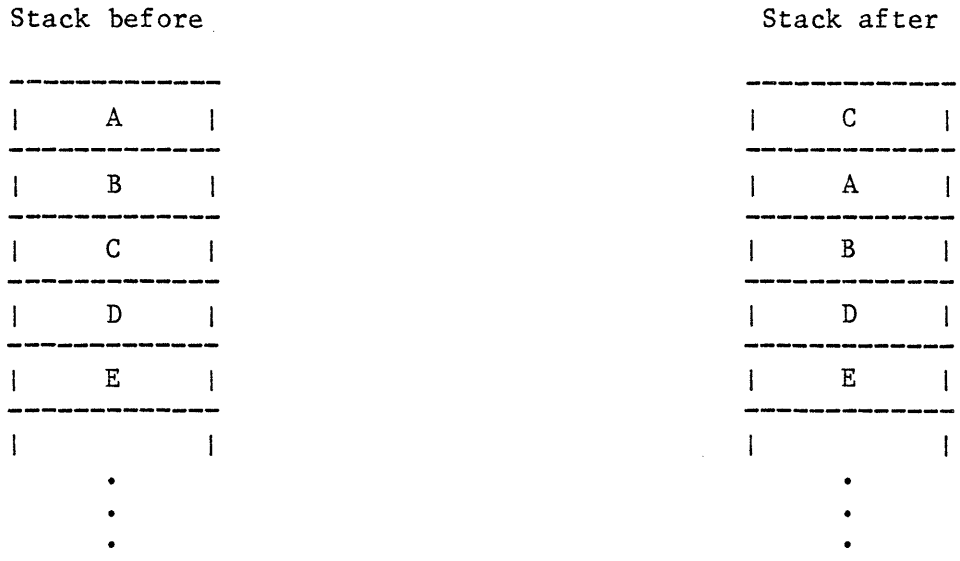


For example:

```
0] 1 2 3 +ROT = = =
2 1 3
```

C.4.3.13 -ROT

Rolls top 3 stack entries down.



For example:

```
0] 1 2 3 -ROT = = =
1 3 2
```

C.4.4 I/O WORDS

DIAMOND provides a number of words which are used for input and output.

C.4.4.1 TYO

Output the ASCII character in the rightmost 8 bits of A.

For example:

```
0] 33 TYO
!
```

(ASCII 33 is an exclaimation point.)

C.4.4.2 CR

Output a <CR><LF> (carriage return, line feed) combination.

For example:

```
0] 65 TYO CR 66 TYO
A
B
```

(ASCII 65 = "A", 66 = "B".)

C.4.4.3 SPACE

Output a space.

C.4.4.4 SPACES

Output A spaces.

For example:

```
0] 65 TYO 7 SPACES 66 TYO
A       B
```

C.4.4.5 TYI

Input a character and put it on top of the stack.

For example:

```
0] TYI =
                (press the"A" key)
61
0]
```

C.4.4.6 =

Output the number at top of the stack in the current radix followed by a space.

NOTE

The "=" operator removes the number from the stack.

For example:

```
0] 1 2 3 = = =
3 2 1
```

C.4.4.7 .!?

C.4.4.8 TYPE

Output string which starts at location stored at B.

```
0] "THIS IS A TEST" TYPE
THIS IS A TEST
```

C.4.5 WORDS WHICH CHANGE THE CURRENT RADIX

BINARY

Set current radix to BINARY.

OCTAL

Set current radix to OCTAL.

DECIMAL

Set current radix to DECIMAL.

HEX

Set current radix to HEXADECIMAL.

For example:

```
0] HEX 10 DUP =
10
0] DECIMAL DUP =
16
0] OCTAL DUP =
20
0] BINARY =
10000
```

C.4.5.1 .!-1<-

C.4.5.2 .!+!

C.4.5.3 .!1+!

C.4.5.4 .!1-!

C.4.5.5 .!MOVE

C.4.5.6 .!XCHG

C.4.5.7 .!MVBYTES

C.5 COLON DEFINITIONS

To define a new word in terms of previously defined words, use the **colon definition:**

'NEWWORD : WORD1 WORD2 ... WORDN ;

A colon definition consists of the following parts:

1. The name of the word being defined, preceded by a single quote (').
2. A colon (:), which separates the name of the word being defined from its definition.
3. The definition of the word.
4. A semicolon (;), which terminates the definition.

NOTE

The colon must be preceded and followed by a space or tab, and the semicolon at the end must be preceded by a space or tab.

This creates a new dictionary entry called NEWWORD which, when executed, will in turn execute WORD1, WORD2, ..., WORDN.

Each of the words WORD1, WORD2, ... WORDN must already exist as entries in the dictionary before definition of NEWWORD. If not, a fatal error message will be generated.

If, in the above example, WORD2 is not yet defined when NEWWORD is defined, the fatal error message will be:

WORD2

undefined, compiling... WORD2 ...in line
'NEWWORD : WORD1 WORD2 ... WORDN ;

A word may be redefined at any time. In this case, all **prior definitions** which referenced that word will still execute the **old** version. All **subsequent definitions**, however, will execute the **new** version.

NOTE

If the name of a word being redefined appears within the new definition, its old meaning will be used for purposes of the definition. Thus, the new version of the word will refer to the old version.

EXAMPLES:

```
'AVERAGE :  + 2 / ;
```

This defines the word "AVERAGE" which computes the average of the top two numbers on the stack.

```
0] 2 4 AVERAGE =
3
```

```
'SPACE :  32 TYO ;
```

This defines the word "SPACE" which types a space. (The ASCII code for a space is 32 decimal).

C.6 ITERATION

DIAMOND provides five means for iterative execution of a sequence of words, namely:

N (...)

Execute the words included in parentheses N times.

BEGIN ... END

Execute the words between "BEGIN" and "END" until a condition is satisfied.

BEGIN ... IF ... REPEAT

Execute the words between "BEGIN" and "IF."

If the condition is met, execute the words between "IF" and "REPEAT", then loop back to "BEGIN".

If the condition is not met, exit, skipping the words between "IF" and "REPEAT."

DO ... LOOP

Execute the words between "DO" and "LOOP," running index from a lower to an upper limit, incrementing by 1 each time.

DO ... N +LOOP

Execute the words between "DO" and "+LOOP," running an index from a lower to an upper limit, incrementing by N each time.

Iterations may be nested subject to the normal restrictions on overlapping ranges, i.e. any iteration which is initiated within the range of another iteration must be terminated within that same range.

C.6.1 N (...)

The following construction executes a sequence of words repetitively:

N (WORD1 WORD2 ... WORDN)

The sequence WORD1, WORD2, ... WORDN is executed N times where N is the number on the top of the stack. The value of N can be specified either when the iteration is defined or when it is executed. See the examples below.

If N is zero or negative, the sequence of words is not executed at all, and control passes to the word following the ")".

EXAMPLES:

'DINGDING : 2 (DING) ;

This definition is functionally equivalent to:

'DINGDING : DING DING ;

In either case, executing "DINGDING" causes the word "DING" to be executed twice. ("DING" must have been previously defined.)

'SPACES : (SPACE) ;

This is a definition of the word "SPACES". Thus, "20 SPACES" causes "SPACE" to be executed 20 times.

In this example, the value of "N" is specified at execution time.

C.6.2 BEGIN ... END

Use the **BEGIN ... END** iteration to execute a sequence of words and then, depending on a computed logical variable, either repeat the sequence or continue with the next instruction:

```
BEGIN WORD1 WORD2 ... WORDN END
```

The sequence WORD1, WORD2, ... is executed once. When the "END" is reached, the top of the stack is popped and tested. If it is true (0), control passes to the word following the "END". If it is false (not 0), control passes back to the word following "BEGIN".

EXAMPLE:

```
'EXAMPLE : 5 BEGIN 1 - DUP DUP = EQZ END DROP ;
```

This defines the word "EXAMPLE" which might be called as follows:

```
0] 5 EXAMPLE
4 3 2 1 0
```

Each time through the loop, the top of the stack (initially the number 5) is decremented, printed, and compared to zero. If it is not zero, the loop is repeated. When the top of the stack becomes zero, the loop terminates.

C.6.3 BEGIN ... IF ... REPEAT

BEGIN ... IF ... REPEAT is similar to BEGIN ... END except that the test is made in the middle of the loop rather than at the end:

1. The words from "BEGIN" to "IF" are executed.
2. If the top of the stack is true (0) when execution reaches the "IF," the words between "IF" and "REPEAT" are executed and control then passes back to the word following "BEGIN."
3. If the top of the stack is false (not 0) when execution reaches the "IF," control passes to the word following "REPEAT."

C.6.4 DO LOOPS

A **DO loop** facility is provided by DIAMOND for indexing through a sequence of words. There are two forms of DO loop:

```
HIGH LOW DO WORD1 WORD2 ...  WORDN LOOP

HIGH LOW DO WORD1 WORD2 ...  WORDN INCR +LOOP
```

The limits "HIGH" and "LOW" (the top two stack entries) are compared. If "HIGH" is less than or equal to "LOW", control passes to the word following "LOOP" or "+LOOP". Otherwise, the sequence WORD1, WORD2, ... WORDN is executed.

"LOOP" causes the lower limit ("LOW") to be incremented by 1 and compared to the upper limit ("HIGH"). If "LOW" is equal to or greater than "HIGH", the loop is terminated. Otherwise, another iteration is performed.

"+LOOP" is identical to "LOOP" with the exception that "LOW" is incremented by the word on the top of the stack ("INCR"). **"INCR" must be a positive number.**

NOTE

> In this release of DIAMOND, all DO loops will execute at least once, even if "HIGH" is initially less than or equal to "LOW". This will be changed in a future release.

Within the range of the loop, the current value of the loop index is available by using the word "I". If DO loops are nested, "I" always contains the value of the innermost index. The next outer indices are available using the words "J" and "K". The word "I'" is used to obtain the value of "HIGH"+"LOW"-I-1. This is used to run an index backwards from "HIGH"-1 to "LOW." The words "J'" and "K'" are similarly defined.

When parentheses are nested with "DO" loops, they count as one level of indexing. "I" used within the range of a parenthesis iteration will return the current value of the iteration count (which runs from its initial value downwards to one).

For example:

```
0] 3 0 DO CR 5 ( I = ) LOOP
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
```

C.6.4.1 EXIT and LAST_I

The word "EXIT" causes the innermost loop in which it is embedded to unconditionally terminate on the next cycle, whether a DO loop or a parenthesis loop.

The word "LAST_I", if executed immediately after leaving a loop, will push onto the stack the value of "I" at the time the word "EXIT" was executed.

If the word "EXIT" was never executed, LAST_I will push the value of "HIGH".

EXAMPLES:

```
0]5 0 DO I = LOOP
1 2 3 4
```

```
0] 4 0 DO 4 0 DO J 4 * I + = LOOP CR LOOP
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
```

```
0] 5 0 DO I' = LOOP
4 3 2 1 0
```

```
0 21 1 DO I + DUP = 2 +LOOP DROP
1 4 9 25 36 49 64 81 100
```

Suppose you have a DO loop that uses "I" (or "J" or "K") in conjunction with "+LOOP" to create an **ascending** set of indices. If you want to change the loop to produce the same indices in **descending** order, replace "I" by "I'" wherever it occurs and replace the value you used for "HIGH" by "HIGH"-"INCR"+1.

EXAMPLES:

```
0] 24 0 DO I = 4 +LOOP
0 4 8 12 16 20


0] 24 0 DO I' = 4 +LOOP
23 19 15 11 7 3


0] 24 4 - 1 + 0 DO I' = 4 +LOOP
20 16 12 8 4 0
```

C.6.5 CONDITIONALS

DIAMOND has a powerful **IF ... ELSE ... THEN** construction which allows complicated logical tests to be performed. The normal restrictions apply to nested conditionals, i.e. any conditional which is initiated within the range of another conditional must be terminated within that same range. The same restrictions apply to nesting of mixed conditionals and iterations.

For the purposes of the conditional, "TRUE" is considered to be zero (0) and "FALSE" is any non-zero value.

```
N IF T1 T2 ... TN THEN
```

The top of the stack, "N" is tested.

If true (0) the words T1, T2, ... TN are executed.

If false (not 0) control passes to the word following "THEN".

```
N IF T1 T2 ... TN ELSE F1 F2 ... FN THEN
```

The top of the stack, "N" is tested.

If true (0) the words T1, T2, ... TN are executed; control then passes to the word following "THEN", so the words F1, F2, ... FN are skipped.

If false (not 0) control passes to the word following "ELSE". The words F1, F2, ... FN are executed in this case and the words T1, T2, ... TN are skipped.

EXAMPLES:

```
'ABS : DUP LTZ IF MINUS THEN ;
```

This defines the word "ABS" which replaces the top of the stack with its absolute value.

```
'MAX : DDUP GT IF DROP ELSE UNDER THEN ;
```

This defines the word "MAX" which compares the top two stack entries and leaves the larger of the two.

C.7 USING DIAMOND FROM THE KEYBOARD

When activated, DIAMOND types a prompt consisting of the current nesting depth (see below), followed by a right bracket (]), followed by a space, to indicate that it is awaiting keyboard input.

When you see this prompt, type a **command line.**

NOTE

> When typing in a command from the keyboard, use the RUBOUT key to delete the last character.

As soon as you press RETURN, DIAMOND compiles the command line and, in the absence of compilation errors, executes it.

After command execution, the compiled code from the last command is discarded. DIAMOND again types its prompt message and waits for the next command line.

C.8 NESTING DEPTH AND CONTINUATION LINES

DIAMOND maintains a **nesting depth.** The nesting depth is used for **syntax checking** and to determine when a **multi-line command has been completed** and is ready to execute.

Initially, the nesting depth is set to zero. It is **incremented** whenever any of the following words are encountered during compilation:

```
IF
ELSE
(
BEGIN
DO
:
```

The nesting depth is **decremented** by the following words:

```
THEN
ELSE
)
END
LOOP
+LOOP
;

REPEAT (decrements nesting depth by 2)
```

A fatal "SYNTAX ERROR" occurs if either of the following happens:

- the nesting depth ever becomes negative
- the nesting depth is non-zero either at the beginning or at the end of a colon definition.

After compiling a line, DIAMOND checks the nesting depth. If it is zero, the line is executed. If it is non-zero, compilation continues on the next line.

For example:

```
0] 3 0 DO
1] 2 0 DO
2] I =
2] LOOP
1] CR
1] LOOP
0 1
0 1
0 1
0]
```

Thus, the execution of the DO loop is automatically postponed until the nesting depth returns to zero; i.e., when the "LOOP" matching the first "DO" is encountered.

Similarly, a multi-line colon definition is extended to include all words up to the matching ";".

C.8.1 Postponing Execution

Execution of compiled code may be postponed even if the nesting depth is zero by using the word "^".

NOT IMPLEMENTED

The "^" feature is not yet implemented in DIAMOND.

C.9 REPEATING THE LAST COMMAND LINE

Typing a line feed causes DIAMOND to recompile and re-execute the last command line executed.

NOT IMPLEMENTED

The LINE FEED word is not yet implemented in DIAMOND.

C.10 DEFINING CONSTANTS, VARIABLES, AND ARRAYS

C.10.1 CONSTANTS

A **constant** is a dictionary entry which causes a 32-bit integer to be pushed on the parameter stack. Once a constant is defined, its value is not intended to be changed at run time.

To define a constant, use the word "CONSTANT":

```
VALUE 'NAME CONSTANT
```

Here, "VALUE" is the number on the top of the stack and "NAME" is the name to be assigned to the constant. When you execute "NAME", "VALUE" is pushed on the stack.

EXAMPLE:

```
5 'FIVE CONSTANT
```

This sets up a dictionary entry with name "FIVE". Executing the word "FIVE" causes a 5 to be pushed on the stack.

C.10.2 VARIABLES

A **variable** is a dictionary entry which contains a 32-bit integer as its value. The value of a variable can be changed during program execution. When executed, it causes the **address of its value** to be pushed on the parameter stack.

Variables are defined as follows:

```
VALUE 'NAME VARIABLE
```

"VALUE" is the number on the top of the stack and "NAME" is the name to be assigned to the variable. "VALUE" is used to set the initial value of the variable. It is **not** an address where you want the variable to be stored.

EXAMPLE:

```
100 'X VARIABLE
```

This defines a variable "X" with an initial value of 100.

C.10.3 ARRAYS

Although DIAMOND has no built-in array handling facility, its ability to perform address arithmetic makes subscripting possible.

There are several methods for setting aside storage for an array. The simplest is to use the word "ARRAY":

```
LENGTH 'NAME ARRAY
```

This defines and zeros an array whose length (in 32-bit words) and name are specified. The array is just a variable with extra storage locations reserved. Referencing an array causes the address of the zeroth element to be pushed on the stack. (The elements run from 0 thru "LENGTH"-1.)

EXAMPLE:

```
100 'BUFFER ARRAY
```

This defines and zeroes a 100-word array named "BUFFER".

C.10.3.1 REFERENCING ARRAY ELEMENTS

To reference an element of an array, all that is necessary is to add an appropriate offset to the address of the zeroth element.

NOTE

Since the first element has offset zero, the Nth element has offset N-1.

EXAMPLE:

```
10 'X ARRAY
10 0 DO I X I 4* + !  LOOP
```

The above code defines a 10 element array "X" and fills it with the numbers 0 to 9.

Note that since addresses are in bytes, the index must be multiplied by 4.

Multidimensional subscripting is handled in a similar fashion.

EXAMPLE:

```
100 'X ARRAY
10 0 DO 10 0 DO I J + J 10 * I + 4* X + !  LOOP LOOP
```

This example sets up a 100 element array "X" which is treated as a 10 by 10 matrix and then stores I+J in the element (I,J). A general 10 by 10 matrix can be thought of as:

```
X(1,1)   X(1,2)  . . .  X(1,10)

X(2,1)   X(2,2)  . . .  X(2,10)

  .
  .
  .

X(10,1)  X(10,2)  . . .  X(10,10)
```

Another way of considering the array is as a one-dimensional array with 100 elements:

```
X(1,1)   X(1,2)  . . .  X(1,10)
  1        2              10

X(2,1)   X(2,2)  . . .  X(2,10)
  11       12             20

  .
  .
  .

X(10,1)  X(10,2)  . . .  X(10,10)
  91       92             100
```

It is easy to see that the index of an array element in one scheme is related to that in the other scheme by:

```
<one-dim index> = ( 10 * <row number> ) + <column number>
```

Or, calling the row number J and the column number I:

<index> = 10*J + I

Translating this last expression into RPN, the index can be written as:

J 10 * I +

which is the expression in the middle of the second line of the example:

10 0 DO 10 0 DO I J + J 10 * I + 4* X + ! LOOP LOOP

Since each element of the array is a longword (4 bytes long), the displacement from the beginning of the array is 4*<index> bytes, so this displacement is computed and added to "X", which is the starting address of the array, to get the address where an element should be stored:

10 0 DO 10 0 DO I J + J 10 * I + 4* X + ! LOOP LOOP

The following part of the statement computes I+J:

10 0 DO 10 0 DO I J + J 10 * I + 4* X + ! LOOP LOOP

So, the statement is of the form:

10 0 DO 10 0 DO <value> <address> ! LOOP LOOP

where:

<value> = "I J +"	(the value to be stored at location I,J)
<address> = "J 10 * I + 4* X +"	(the location of the I,J element)

The nested DO LOOPs run I and J through the values 1 through 10.

C.11 THE DICTIONARY

The **dictionary** starts in low core and grows upward toward the top of memory. As each definition is made, it is appended to the high memory end of the dictionary.

The following word gets the address of a dictionary entry:

```
'NAME ADDRESS
```

This word pushes two values on the stack if its execution is successful:

1. The address of "NAME". (Placed next to top of the stack)
2. A value which reflects the success or failure of the operation. (Placed on top of the stack)

For example:

If the word "GORK" is defined:

```
0] 'GORK ADDRESS = =
0 204078
```

If the word "GUCK" is not defined:

```
0] 'GUCK ADDRESS =
-1
```

(There is no second element.)

C.12 STRING HANDLING

Executing a string literal causes a pointer to the length word of the string to be pushed on the stack.

EXAMPLE:

```
'LARK :  'NONSENSE TYPE ;
```

Executing "LARK" causes "NONSENSE" to be typed.

C.13 NUMBER OUTPUT CONVERSION

DIAMOND performs number conversion using a small but powerful set of words which permit a variety of output formats to be generated.

RADIX Variable which contains current input and output radix.

TYO Convert the number on the top of the stack to an ASCII character.

Higher level words are used to provide number output in a default format:

=U Convert and output the number on the top of the stack. (unsigned)

= Convert and output the number on the top of the stack. (signed) stack. (signed)

=F Convert and output the number next to the top of the stack, displaying the number of digits specified by the number on top of the stack. (signed) If the number to be displayed requires fewer digits than specified, the output will be left zero filled.

=UF Convert and output the number next to the top of the stack, displaying the number of digits specified by the number on top of the stack. (unsigned) If the number to be displayed requires fewer digits than specified, the output will be left zero filled.

C.14 FORGET

Entire sections of the dictionary may be deleted by using the following word:

'NAME FORGET discards the named dictionary entry and all subsequent entries.

FORGET is useful when trying out definitions from the keyboard. The usual procedure is to first make a dummy definition:

```
0] 'TEST : ;
```

Next, test definitions are made (typically by loading a program). If they are unsuccessful, "'TEST FORGET" will delete them from the dictionary, and the process is repeated. For convenience, the dummy definition may be placed at the beginning of the program.

"FORGET" may also be used to provide an overlay facility. If a program has been loaded, and is no longer needed, it may be deleted using "FORGET". The dictionary space is then free to load another program.

C.15 DEBUGGING TECHNIQUES

The following techniques may prove useful in debugging new definitions:

1. To test a word, feed in arguments on the stack, execute the word, and examine the results using "=".

2. If a word fails, type in the words which make it up, one at a time, examining the stack as you go along and restoring it by typing the parameters back in, in reverse order.

3. Keep track of the radix you are using. This is a common source of errors. Within a program being executed, save the radix, set it to the value the program expects, and when done, restore the saved value.

   For example:

   ```
   0] RADIX @ DECIMAL  . . . RADIX !
   ```

   This saves the current radix on the stack and sets the radix to DECIMAL. At the end, the old radix is restored (assuming the stack has not been disturbed).

4. The proper selection of lower level words has an enormous effect on all subsequent higher level definitions. Thus, it pays to design lower level words very carefully.

5. When debugging a program, test all lower level words thoroughly before testing the words which call them.

6. Be especially careful with any word which modifies memory. Make sure the word is modifying only those locations you intend, and not part of the program.