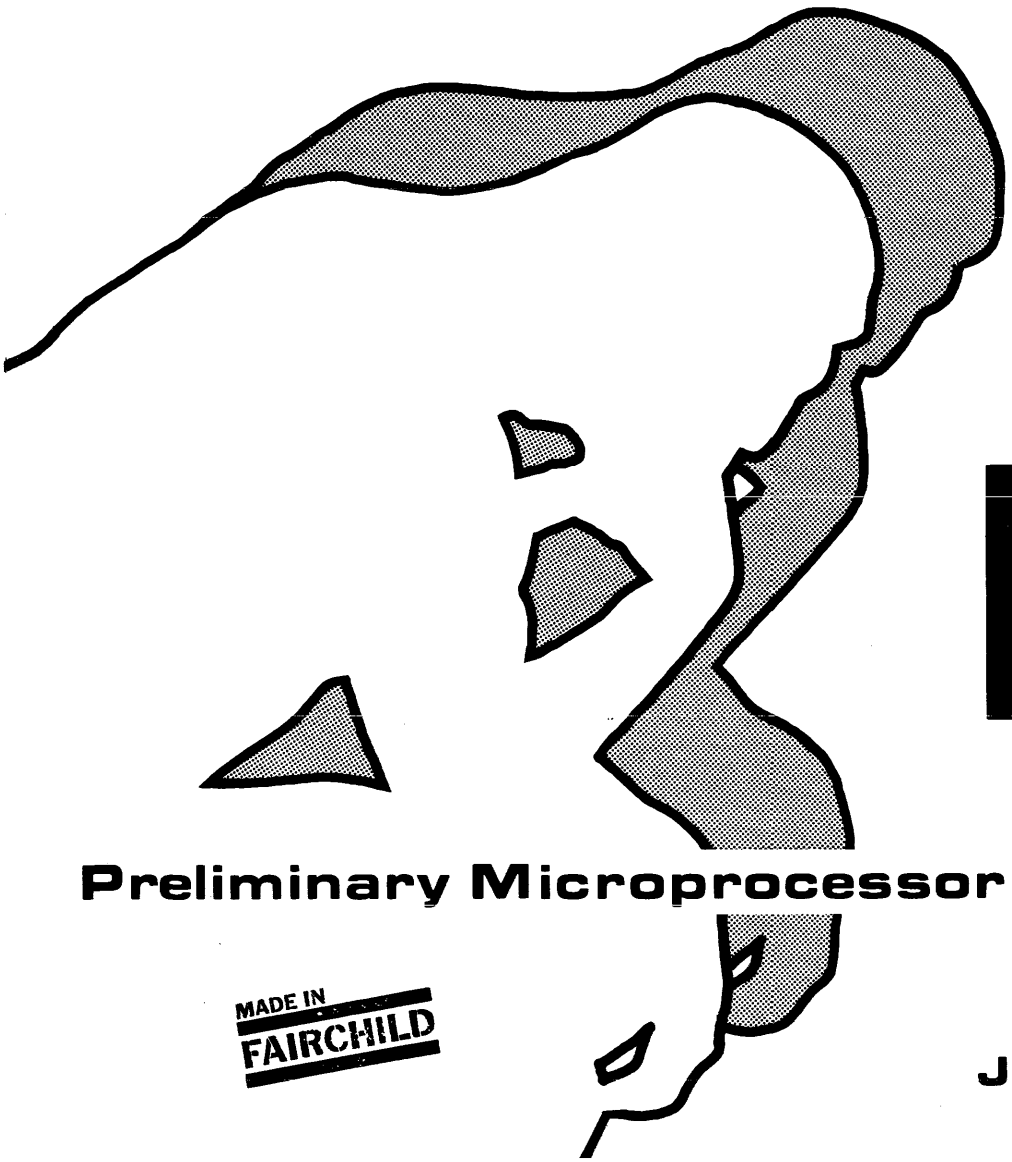
FAIRCHILD SEMICONDUCTOR



F8

Preliminary Microprocessor User's Manual

MADE IN FAIRCHILD

January 1975

Fairchild

F8

Microprocessor

INDEX

1. INTRODUCTION

A microprocessor differs from a specifically designed logic network in that the microprocessor contains generalized logic units (registers, counters, control elements, et al) which are used for many different tasks during the operation of the system. Thus, a register designated as an ACCUMULATOR may receive the results of an arithmetic operation, or may be used to input from or output to some other system; or may have its contents shifted or modified as appropriate during the operation of the system. The control sequence that determines how each of the logic units is used during the operation of the microprocessor is called the *PROGRAM*.

The program is composed of individual steps, or *INSTRUCTIONS*, that moves or modifies information in a specific way as it is passed through the microprocessor. A task, (for example: adding two (10) digit numbers) may require that a program be designed that contains just a few instructions or as many as several thousand, (depending on the application). Each instruction is performed or *EXECUTED* sequentially.

The most significant differences between a microprocessor based system design and a dedicated logic based system design is in the area of performance and the amount of logic units required. A few generalized logic units of the microprocessor design are used repetitively during the execution of the programmed instructions. More time is required to process the information to the system specifications. Thus, the microprocessor based system trades off speed (or performance) for fewer logic elements.

Since performance is sacrificed in designing a microprocessor based system, the justification for using a microprocessor must be lower overall cost for implementing the system. There are two areas of cost savings in designing a microprocessor based system:

- Fewer Logic Packages
- Shorter Design and Development Schedules

Writing a program for a microprocessor based system is generally simpler than designing the interconnections for discrete logic networks. There is a finite set of instructions available in a microprocessor that is most usually limited to no more than a few hundred instructions. Most one chip microprocessors generally contain less than one hundred unique instructions which are easy to learn and

apply. The design of logic networks, however, implies infinitely more ways in which to interconnect a family of logic components together to complete the required design. Not only must the designer be aware of the rules for logic interconnections, but, signal interface compatibility, signal race conditions and logical polarities must be accounted for. Additionally, unexpected interferences (or noise) must be ferreted out of the design during the debugging of the first system. All of these problems are greatly reduced when using a microprocessor. The fewer total packages of a microprocessor design produce fewer design and debugging problems. Secondly, a program consisting of instructions is easier to write. All of these advantages mean reduced design time and costs.

The manufacturing of microprocessor based systems is also less costly because fewer total integrated circuits are used, and this requires fewer printed circuits, less power and smaller and simplier housings.

The F8 System is designed to provide the engineer the maximum benefits possible in the area of reduced costs and high performance. There are approximately 70 instructions available most of which are single byte in length. More complicated functions and addressing modes are eliminated to compact total circuit size and thereby reduce costs. The instructions, as a consequence, are easy to learn and simple to use facilitating easier program design and shorter development time.

The design of the F8 circuits has been carefully planned to provide the most useful functions possible in the limited chip area of reasonably producable circuits. Partitioning of the internal logic has been arranged to provide the maximum utilitarian use of the available external circuit interconnections. For example, the program counter has been located in the memory circuits eliminating the need for 16 address connections between F8 circuits. Input/Output ports have been added to both the CPU and the F8 ROM circuits and brought out to the now available 16 pins on each circuit to give the designer more utilitarian functions available for implementing his design.

A fully functional F8 microprocessor based system can be fabricated with as few as only (2) F8 circuits and some passive components (resistors and capacitors). Typical applications for the 2 circuit configuration are noncomplicated control

or business machine applications. Yet, the F8 has the power and flexibility to be used in more complex systems having up to 65,536 bytes of total addressable memory, direct memory access and multiple level priority interrupts.

Additional circuits are included to add flexibility and minimize the total system package count. The clock generator circuit and power-on detect circuits are included in the CPU circuit to simplify system design. Each memory chip contains a programmable timer to facilitate generating real time events or delays without disabling the processor. (The usual method of creating time delays ties up the processor counting non-functional instructions in a loop until the desired elapsed time has occurred.)

A RAM of sixty-four words of 8 bits each is included in the CPU as a fast operating scratchpad register array. Instructions that use the RAM (ADD, LOAD, OR, AND, Decrement, et al) all execute within 2 microseconds*.

The F8 system of components is designed to provide you with the most functional value at the lowest system cost possible consistent with current technology.

* The Decimal ADD requires 4 usec. and the Decrement REQUIRES 3 μsec..

2. ORGANIZATION OF THE MANUAL

The format for information provided in this publication is organized to be used effectively as a training guide and also as a reference guide to persons interested in a FAIRCHILD F8 Microprocessor System of Components. As a training guide, all information is developed in the sequence that a user not having familiarity with the product might require as he developes his understanding of the system and its applications. During the discussion of particular technical areas, questions might logically arise that would require the introduction of topics not yet presented. References are provided where this occurs to direct the reader's attention to the section where the new concepts are discussed.

This organization facilitates using the publication as a reference guide by users who are familiar with the operating concepts of the F8 system and need to refer to particular topics. Each topical section is as brief as possible, only discussing the main point of that section. Additional or supplementary information relevant to an overall understanding of the topic under review are referred to where applicable.

Chapter 1 discusses the highlights of the F8 architecture and the Fairchild philosophy for the organization of the F8.

Chapter 2 is an introduction to the organization of the User's Manual.

Chapter 3 describes the organization of the F8 circuits. The functional block diagrams are discussed; and, the functional relationship of each logical unit within each of the circuits is discussed at the processor system level. Examples are provided where possible to suggest some of the possibilities for the use of the F8 circuits.

Certain logical functions such as the *PROGRAM COUNTER* and *INDEX REGISTER* (Data Counter) are contained in more than one circuit when using multiple circuit configurations. To avoid repetition and to develop the description of the circuits as directly as possible, references are inserted in the appropriate areas of discussion to direct the reader to a more detailed discussion.

CHAPTER 4 contains the static and dynamic interface specifications for the F8 system and circuits. Timing diagrams are provided where appropriate.

CHAPTER 5 lists all of the machine instructions with sufficient detail to permit a reader not familiar with the F8 system to program a system with a minimum of difficulty.

A condensed listing of the F8 instructions at the end of this section provides a quick reference to the function of the instructions.

CHAPTER 6 discusses programming in assembly language. Complete instructions for using the assembler are included. However, the procedures for accessing the cross assembler will depend on the system in which it resides (i.e. resident IBM 370 System, Timeshare Service, et al) and is not included in this section.

CHAPTER 7 discusses the use and operation of the Cross Simulator written in FORTRAN IV. A computer program that simulates the operation of the F8 with an applications program is called the Cross Simulator. This section discusses the features, use and function of the F8 Simulator Program. Instructions for specifying or setting up the Simulated Input/Output Signals, Interrupts and all test and monitor points (called TRACE & BREAK POINTS) are reviewed in detail.

This publication is intended to provide all of the information needed by the reader to design an effective F8 system. Both the F8 instructions and electrical interface specifications are discussed in detail.

Our objective in preparing the content of this publication is to provide sufficient detail so that anyone having experience in designing a digital logic system can effectively incorporate a microprocessor in his design with minimum other special training. If you find the organization of this material difficult or any material lacking, please pass your suggestions along. A return mailer is provided in the back of this publication for convenience in submitting your comments.

3.0 F8 CIRCUIT ORGANIZATION

3.1 GENERAL

The F8 family of microprocessor circuits are manufactured using N Channel Isoplanar MOS technology. Some of the features and characteristics of this microprocessor set are:

- o 8-Bit Data Organization
- o 2 μs Instruction Cycle Time
- o Over 70 Instructions
- o 64 General Purpose Read Write Registers
- o Binary and Decimal Arithmetic and Logic Functions
- o Internal Clock Generation
- o Internal Power-on and Reset
- o I/O Capability Included on the Circuits
- o Multi-level Interrupt System
- o Programmable Interval Timers

A complete microprocessor system constructed with F8 components requires as little as two integrated circuits. Larger, more power, systems may be built with additional F8 components. The family of F8 parts contains:

- o 3850 F8 Central Processing Unit (CPU)
- o 3851 F8 Read Only Memory (ROM)
- o 3852 F8 Dynamic Memory Interface (MI)
- o 3853 F8 Static Memory Interface (MI)
- o 3854 F8 Direct Memory Access (DMA)

These circuits may be combined to form an 8-bit microprocessor. Additional semiconductor components such as memories may easily be added to expand the full system capability.

3.1	INTRODUCTION
3.2	F8 Circuits
3.2.1	3850 CPU
3.2.2	3851 ROM
3.2.3	3852 Dynamic Memory Interface (MI)
3.2.4	3853 Static Memory Interface (MI)
3.2.5	3854 Direct Memory Access (DMA)
3.3	Central Processing Unit
3.3.1	CPU Data Path Organization and Control Circuits
3.3.2	Major CPU Registers
3.4	ROM and Memory Interface Circuits
3.4.1	Data Path Organization and Control Circuits
3.4.2	Major Register of the 3851, 3852, and 3853
3.5	Local Timer
3.6	Interrupt System
3.7	Input/Output
3.8	Initialization Requirements
3.9	Dynamic Memory Interface - 3852
3.9.1	Memory and DMA Interface
3.9.2	Control Circuits for Dynamic RAMs
3.9.3	Description of Inputs and Outputs - 3852
3.10	Static Memory Interface - 3853
3.10.1	Static Memory Interface Organization
3.10.2	Description of Inputs and Outputs - 3853

TABLE 3.1

ORGANIZATION OF SECTION 3

A detailed description of each circuit is given in this chapter. Table 3.1 lists the contents of each paragraph following. Data sheets for the F8 circuits are in Chapter 4. A description of the microprocessor instruction set is reserved until Chapter 5.

3.2 F8 CIRCUITS

This section is an overview of each member of the F8 circuit family. A detailed description of the operation of the system components is contained in Section 3.3.

3.2.1 3850 F8 CENTRAL PROCESSING UNIT

The 3850 CPU circuit controls the entire microprocessor operation and executes the functions called for by each instruction. The main functions of the CPU are:

- receive and decode an instruction
- transmit all necessary control signals to cause all other microprocessor circuits to execute required functions in synchronism with the CPU.
- execute the functions indicated by the instruction.
- handle interrupts
- complete an input from or output to an I/O port

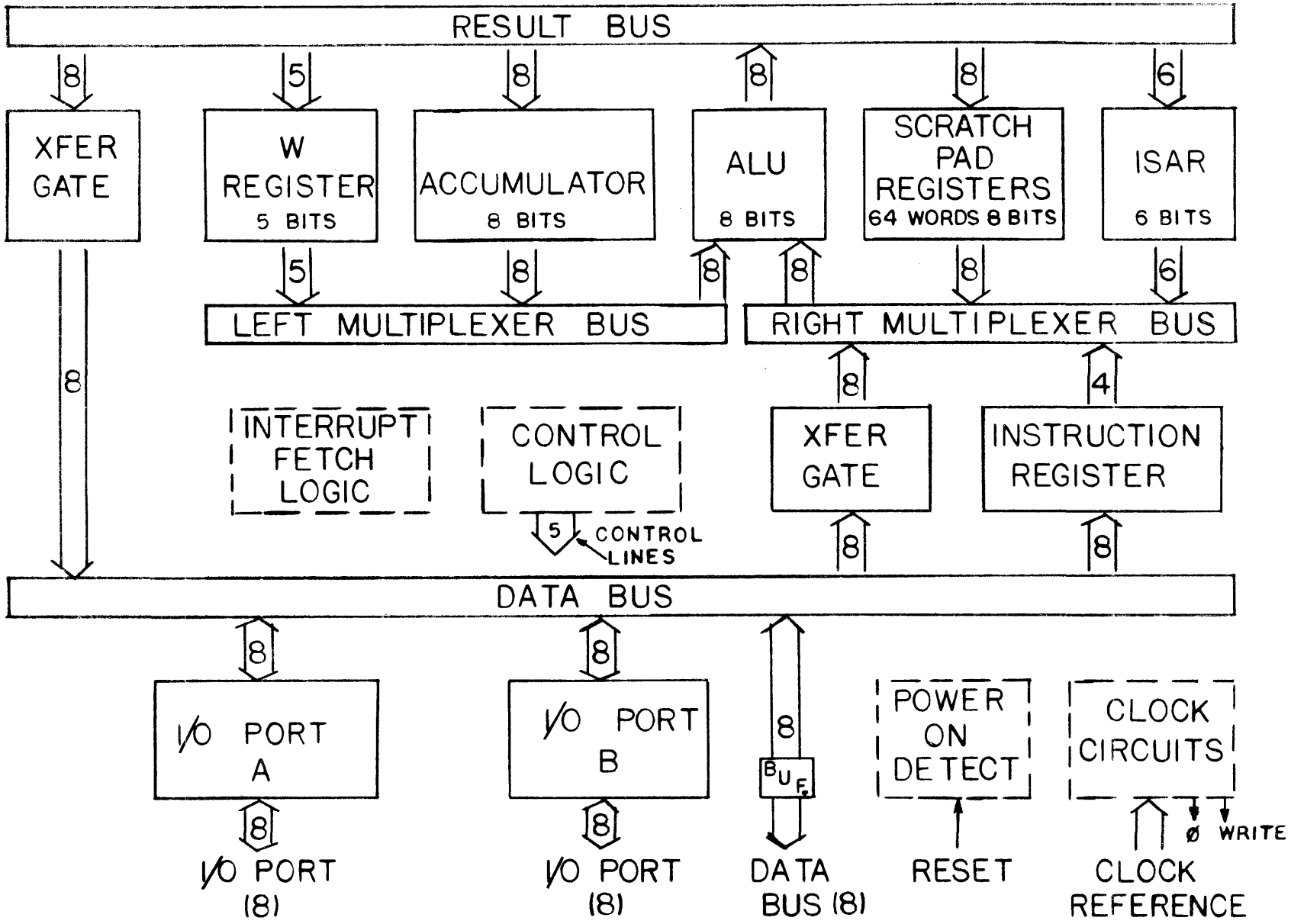
Figure 3.1 is the functional block diagram of the 3850 CPU circuit. The discussion of the operation of circuits internal to the CPU and of those contained in other F8 components is contained in Section 3.3.

3.2.2 F8 READ ONLY MEMORY (ROM)

The F8 ROM circuit contains 1024 bytes of read only memory for storing the program instructions. Additional circuitry is contained in the ROM so that it may be attached to the F8 data bus.

The principle functions of the ROM circuit are:

- Access Memory for either an instruction or a constant
- Maintain the program counter, stack register and data counter
- Control a local interrupt level
- Control two input/output ports



EXTERNAL DATA CONNECTIONS

CPU CIRCUIT: DATA PATH ORGANIZATION

Locating memory addressing circuits in the ROM minimizes the circuit interconnections necessary to control and access memory locations. These address registers are discussed in Section 3.4.2.

The functional description of the 3850 ROM circuit is shown in Figure 3.2.

The description and operation of the internal circuits of the ROM is contained in Section 3.2.

A discussion of the interrupt circuits appears in Section 3.6.

Each 3851 ROM contains two 8-bit I/O ports. A hardware description of the F8 I/O structure is discussed in section 3.7 while a software (programming) description is given in Chapter 5, Section 5.13.

In addition to these circuit features, each 3851 ROM has a programmable timer. The operation of this register is discussed in Section 3.5

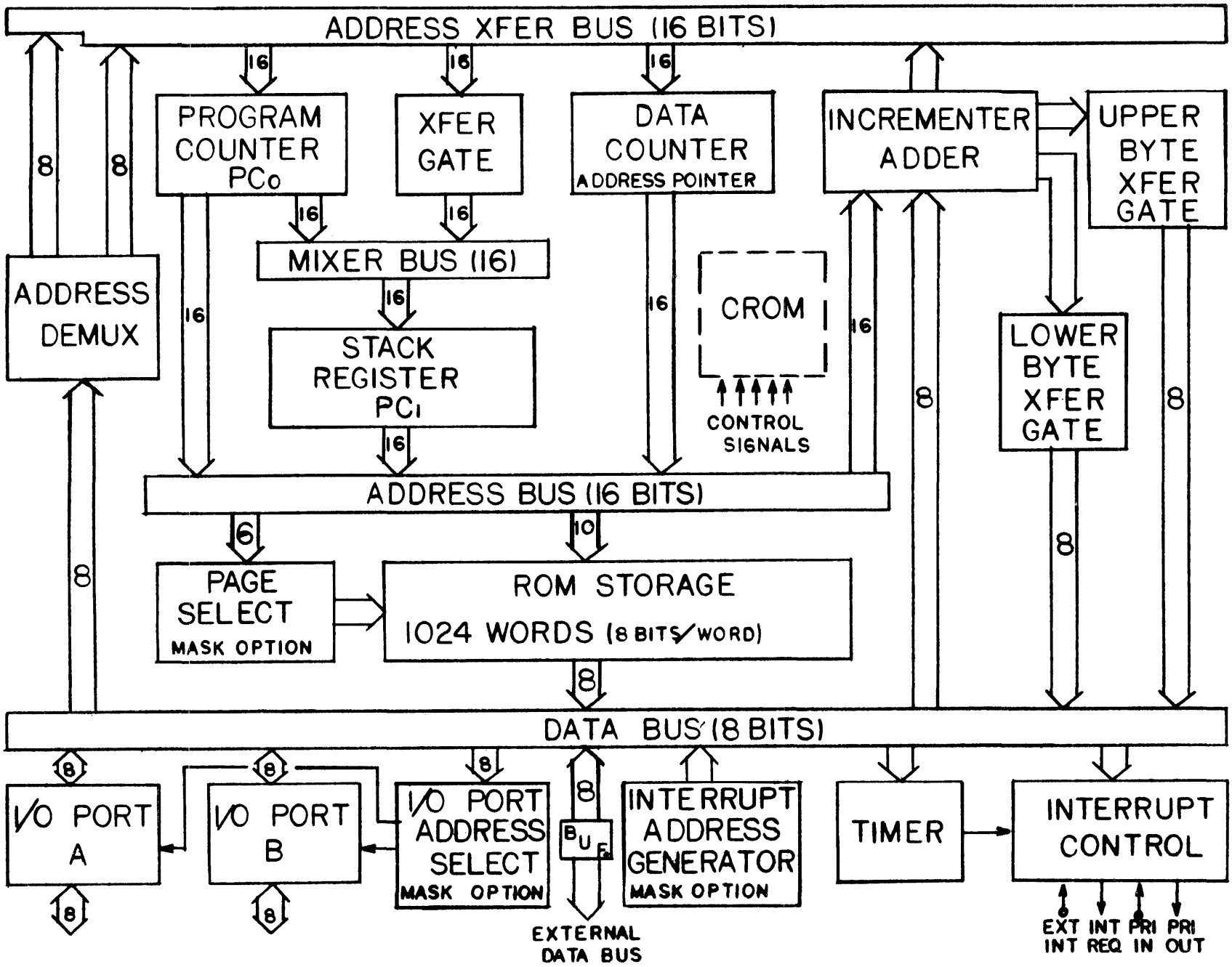
3.2.2.1 CUSTOM ROM SPECIFICATIONS

Each 3851 ROM is made to meet customer specifications. The custom ROM pattern is produced with a custom diffusion mask. Five items must be specified at the time the proprietary mask is produced. These are:

- 1024 x 8-bit ROM pattern
- Memory page address (See Below)
- ROM Port Addresses (Refer to Section 3.4.2.5)
- Interrupt Address Vector (Refer to Section 3.6)
- I/O Port Electrical Options (Refer to Chapter 4)

The F8 uses 16 bits to specify a memory address. As such, the 1024 Byte ROM uses the lower ten bits to select the addressed byte. The remaining six bits specify the active memory page.

Internal to the ROM chip is page selection logic (Figure 3.2). For each memory access, this logic



BLOCK DIAGRAM OF ROM CIRCUIT

decodes the upper six bits by comparing them to those specified by the customer. In this way the memory page is decoded.

The interrupt address vector is discussed in Section 3.6. Briefly, it is the address to which the program counter will be set after an interrupt has been acknowledged.

3.2.3 3852 DYNAMIC MEMORY INTERFACE (MI)

The Dynamic Memory Interface circuit provides the 16 address lines and control signals necessary to interface standard memory circuits to an F8 system. Both static and dynamic memories may be used, dynamic memory refresh is performed automatically and without any degradation in system performance. Standard memories such as the Fairchild 8101 (256 x 4) or the 2102 (1024 x 1) MOS RAMS may be used with the 3852 MI. Dynamic memories, such as the Fairchild 4096 may also be used. Figure 3.3 is a functional diagram of the 3852.

In addition to controlling dynamic memories, the 3852 MI may be used in conjunction with the 3854 DMA to allow direct memory access channels to be established between a peripheral and F8 memory.

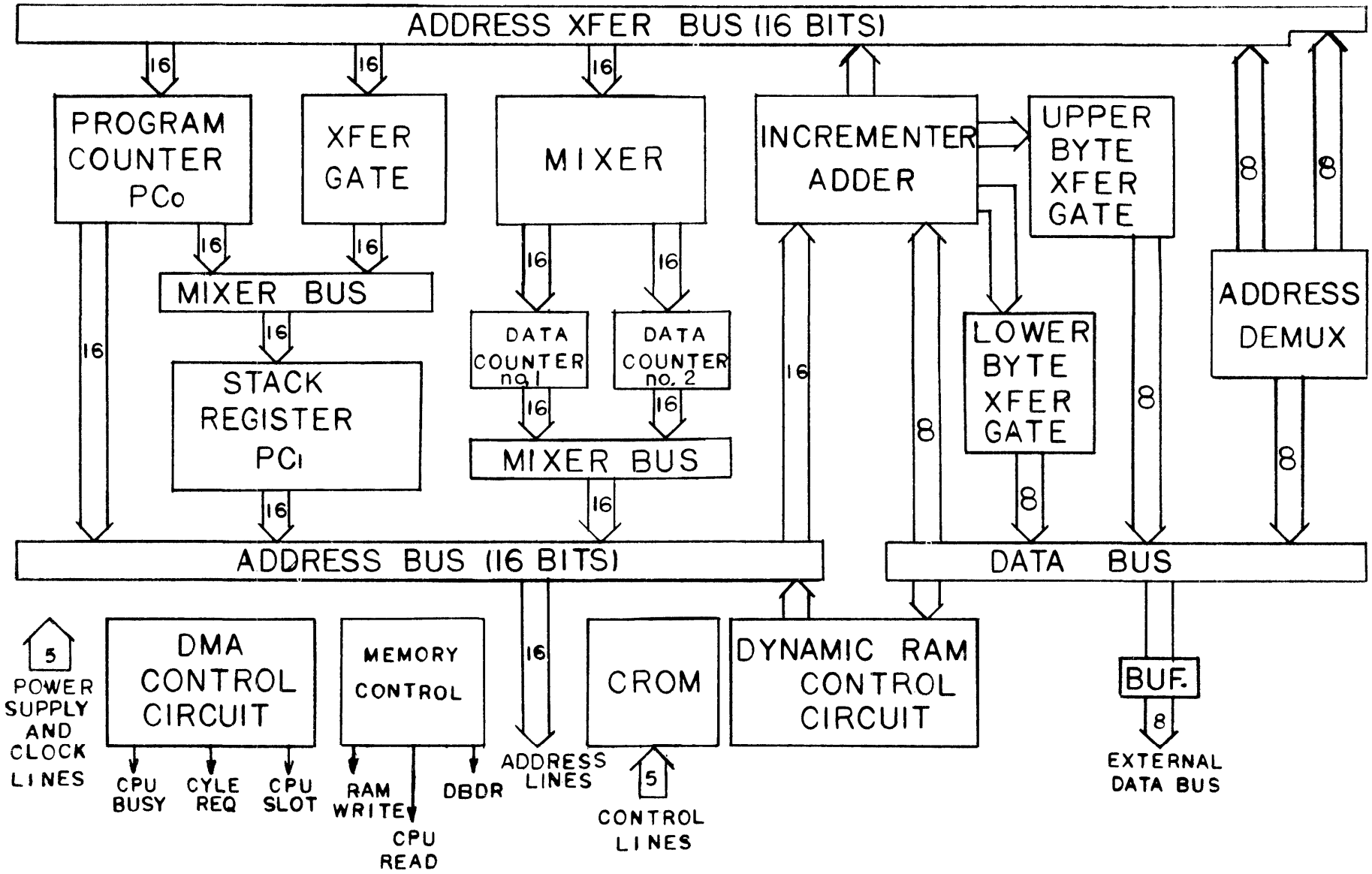
A functional description of the 3852 circuit appears in Section 3.4.

A discussion of the 3852 MI controlling dynamic memories is included in Section 3.9.

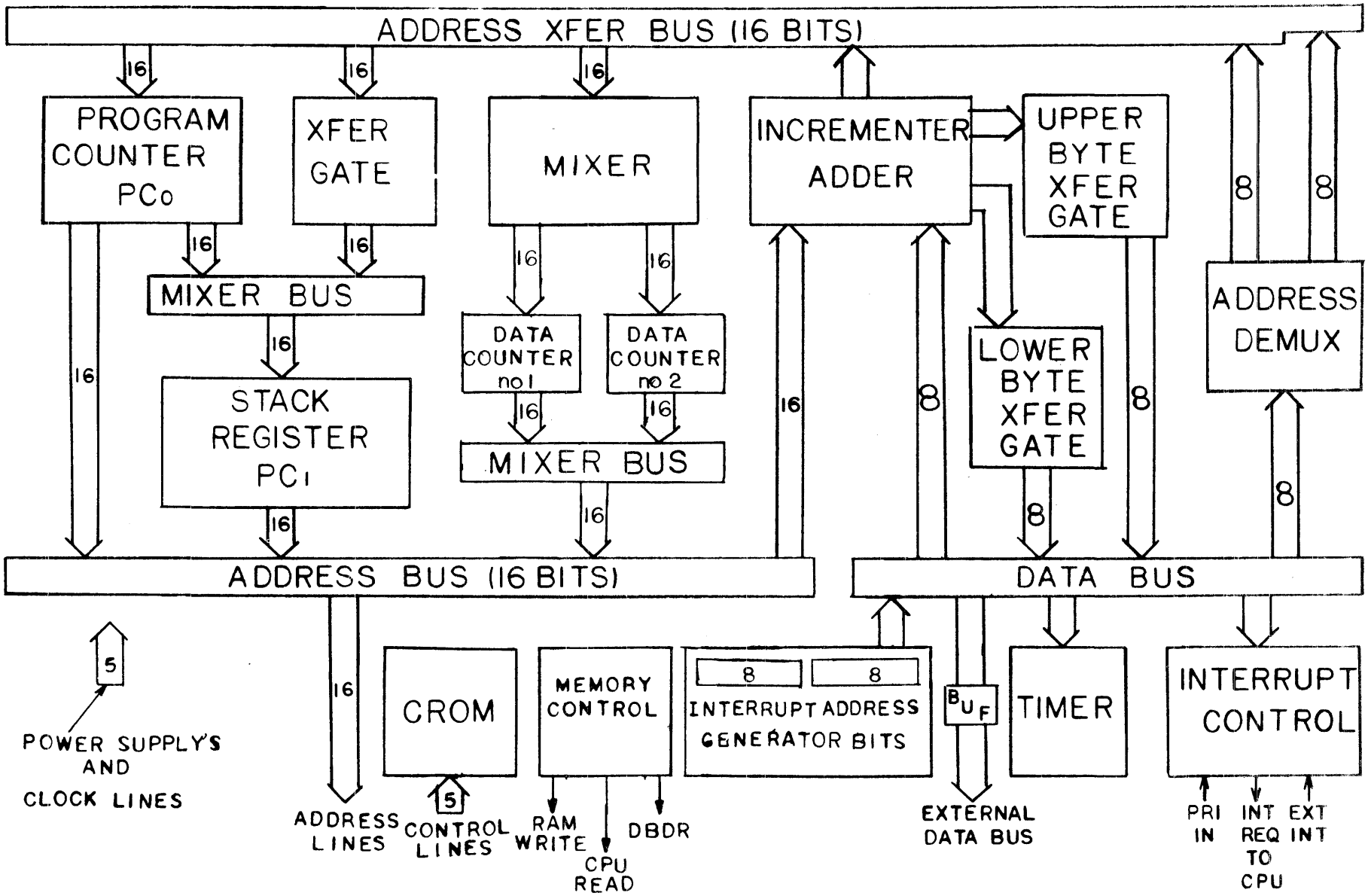
3.2.4 3853 STATIC MEMORY INTERFACE (MI)

The 3853 Static Memory Interface provides the necessary address lines and control signals to operate standard static RAM circuits such as the Fairchild 8101 or 2102. In addition to these MOS memories, bipolary memory will also interface with the 3853.

The functional organization of the 3853 Static Memory Interface circuit is shown in Figure 3.4. As the figure shows, this circuit includes interrupt control logic.



F8 MEMORY INTERFACE CHIP: 3852



F8 MEMORY INTERFACE CHIP: 3853

3.2.4 3853 STATIC MEMORY INTERFACE (MI) (Cont.)

Refer to Section 3.6 for a full description of the interrupt circuit. Included with the interrupt circuit is a programmable interval timer. This is discussed in Section 3.5.

3.2.5 3854 DIRECT MEMORY ACCESS (DMA)

The 3854 Direct Memory Access circuit sets up a high speed data path between F8 memory and a high speed peripheral, a FIFO, or another F8. The transfer is initiated by the CPU under program control. Once started, the DMA transfer will continue without CPU intervention. The CPU can sense the flag line of the DMA to determine the completion of a transfer.

3.3 CENTRAL PROCESSING UNIT (CPU)

The 3850 Central Processing Unit (CPU) is the heart of the F8 family of compatible microprocessor circuits. Its organization is shown in Figure 3.1. In addition to performing processing functions, the CPU generates timing and control signals for F8 system operation. The main processing functions are:

- o receiving and decoding the instruction
- o transmitting necessary control signals to cause all other microprocessor circuits to execute required functions in synchronism with the CPU
- o executing the function indicated by the instruction
- o handling interrupts

CPU contains the following features:

- o accumulator
- o 64 scratchpad registers
- o 2 I/O ports
- o interrupt control circuitry
- o power-on detect

- clock circuits that can be operated in one of three modes:

  an RC network
  crystal control
  external master frequency

This section describes the CPU circuits and all of its major registers. Section 3.3.1 is a description of the major busses and central circuits of the 3850. The major registers of the central processor are explained in Section 3.3.2. The ROM and Memory Interface circuits are discussed in the next sections of this chapter. System functions such as the interrupt and I/O structure are described in later sections of Chapter 3.

3.3.1 CPU DATA PATH ORGANIZATION AND CONTROL CIRCUITS

This section describes the major busses in the 3850 CPU. These busses transfer bytes of data between CPU registers and other F8 system components. The gating of these busses and the operation and timing of the CPU circuit is performed by the CPU control circuit, also described in this section.

3.3.1.1 F8 DATA BUSSES

Figure 3.1 shows the data path organization of all major functional blocks in the CPU. Data within this chip is transmitted between blocks via four busses. These are:

- The Result Bus: The result bus receives the results of the arithmetic logic unit operation and makes them available for storage in one of the receiving registers associated with the system. Data from the result bus is passed on to either the W register (status), the accumulator, the scratchpad register, the indirect scratchpad address register, or is gated to the data bus through the transfer gate(s) for availability in the I/O port or for transmission over the external data bus to the rest of the circuits in the F8 system.

- Left Multiplexer Bus: The left multiplexer bus gates the accumulator into one of the operand ports in the arithmetic logic unit.

When the contents of the status register are transferred, it is gated through the ALU from the left multiplexer to the result bus.

- Right Multiplexer Bus: The right multiplexer bus transfers the second operand required by the ALU. This operand may be taken from the scratchpad registers, from the scratchpad address register, from the instruction register (used only when immediate operands are contained in the instruction), or from the data bus via the transfer gates. The data bus permits the operands to be extracted either from other chips associated with the microprocessor system or from the input/output ports contained on the CPU.

- The Data Bus: The data bus is the principal path for transmitting 8 bit bytes between the CPU and other circuits in a microprocessor system. It also transmits data between the CPU input/output ports and the accumulator. Data entering the CPU may originate from a ROM, a RAM or from the memory interface circuits. Three types of information are transmitted on the data bus:

  - data (or operands)
  - I/O port addresses used for activating an I/O port designated by an I/O instruction.
  - Program address transmitted during branch or interrupt functions.

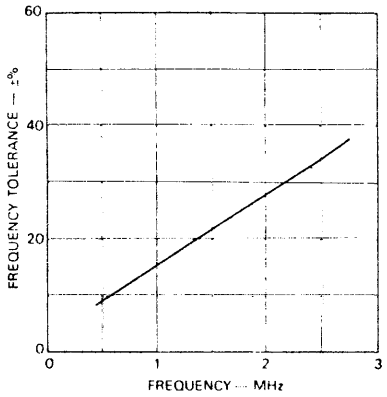
  Information travels into and out of the CPU over the data bus.

3.3.1.2 CPU TIMING AND CONTROL

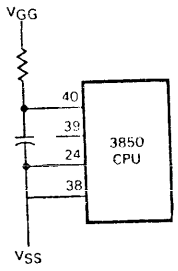
Clock Circuits: The clock circuits generate two clock signals used by all circuits in the microprocessor system. The reference frequency of the CPU may be selected from one of three modes, as shown in Figure 3.5. In Figure A, the clock frequency is established with the selection of an external RC network. When a precise operating frequency is required, an external crystal may be used to select a frequency. This is shown in Figure B. For applications requiring a controlled clock, an external clock connection is made to the microprocessor, as shown in Figure C.

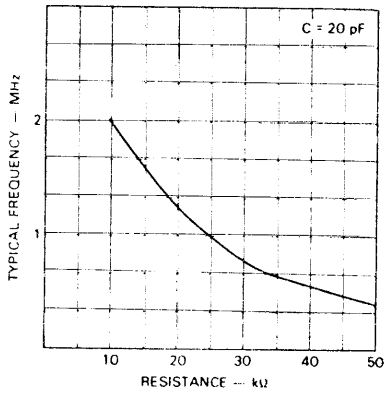
CLOCK CONSIDERATIONS

RC Mode

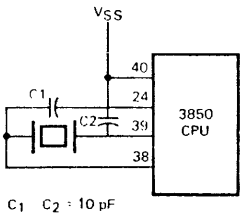


Timing Considerations





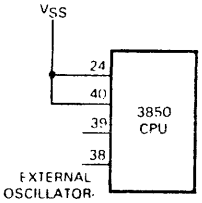
Crystal Oscillator



C_1 C_2 = 10 pF

C1 and C2 are not needed for crystals in the range of 1 2 MHz They do, however, give an added degree of frequency control for systems requiring a high degree of frequency stability.

External Oscillator



3.3.1.2 CPU TIMING AND CONTROL (Cont.)

The internal clock generator circuits of the CPU create two clock signals to control the internal registers of the CPU as well as other F8 circuits. The two clock signals generated by the CPU are Ø and WRITE. In a system with a 2MHz reference frequency, the Ø signal will occur every 500 ns and the WRITE pulse will have a 2 us or 3 us period, depending on the instruction. These signals are fully described in Chapter 4.

Control Circuits: The control circuits of the CPU generate the necessary register gating and control signals for the entire microprocessor system. The control circuits decode the instruction and pace all the system timing and data gathering to cause the instruction to be executed properly. The circuit consists of the following parts:

- o Instruction Register
- o State Generator
- o Control ROM

The operation of the CPU control circuits is sequential. An instruction is placed into the instruction register and decoded. The instruction decode is performed by the control ROM. Therefore, the instruction op code is used to address the necessary control signals. The control circuit combines the control and timing signals to execute each instruction. In addition, the control circuit generates five control lines to other F8 circuits connected to the data bus.

F8 instructions may either be one, two, or three bytes long. In all instructions, the op code is contained in the first instruction byte. The remaining bytes are used either as operands or addresses. Thus, the second or third byte of multiple byte instructions are never routed to the instruction register; rather, they are used in the appropriate data registers or program counters in the F8 microprocessor system.

3.3.2 MAJOR CPU REGISTERS

Figure 3.1 shows the major CPU registers and their relationship with respect to the CPU busses. These

CPU registers are:

- o The Accumulator
- o The ALU
- o The W (STATUS) Register
- o The Scratchpad Registers and ISAR
- o The I/O Ports

Each of these registers is described below.

3.3.2.1 THE ACCUMULATOR

The accumulator is an 8-bit storage register. It retains the result of ALU operations and transfers information into and out of the scratchpad memory, bulk memory, and I/O ports. The contents of the accumulator may be either shifted or complemented.

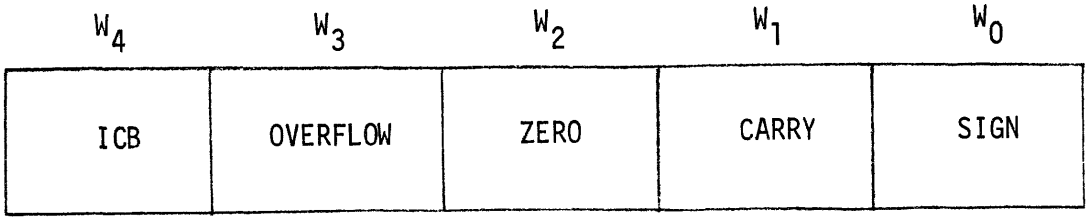
3.3.2.2 ARITHMETIC LOGIC UNIT (ALU)

The ALU is an eight bit parallel logic network used in the execution of the F8 instructions. Binary and decimal ADD operations may be executed. In addition, complement, logical AND, logical OR, logical XOR, increment, compare, and decrement operations may also be executed. One operand of the ALU is usually supplied by the accumulator. The other operand may be either scratchpad memory data, bulk memory data, the ISAR, or the instruction register. Outputs from the arithmetic logic unit are placed in the accumulator via the result bus.

3.3.2.3 W (STATUS) REGISTER

The W register stores the status indications resulting from an arithmetic or logic operation. Figure 3.6 shows the bit assignments for the five status bits. The status register instructions transfer the W register to and from the scratchpad (See Section 5).

Table 3.2 lists the conditions for which each bit is set. These five bits are explained below.



ICB - Interrupt Control Bit

Figure 3.6 The W Register

$$\text{OVERFLOW} = \text{CARRY}_7 \oplus \text{CARRY}_6$$

$$\text{ZERO} = \overline{\text{ALU}_7} \wedge \overline{\text{ALU}_6} \wedge \overline{\text{ALU}_5} \wedge \overline{\text{ALU}_4} \wedge \overline{\text{ALU}_3} \wedge \overline{\text{ALU}_2} \wedge \overline{\text{ALU}_1} \wedge \overline{\text{ALU}_0}$$

$$\text{CARRY} = \text{CARRY}_7$$

$$\text{SIGN} = \overline{\text{ALU}_7}$$

TABLE 3.2

THE STATUS REGISTER

The *OVERFLOW BIT* is the exclusive OR of the carry propagated from the 2^6 and 2^7 stage of the ALU. If 2's complement notation is used, in which the most significant bit (2^7) represents the sign ($\pm$) of the number the overflow bit detects a result exceeding the boundries of two's complement notation.

The *CARRY BIT* is set whenever a carry out of the 2^7 stage is progagated as a result of an arithmetic operation.

The *ZERO BIT* is set whenever the results of an arithmetic or logic operation is ØØØØ ØØØØ. Since this bit is set on the results of the arithmetic logic unit rather than the contents of the accumulator, the state of the zero flip-flop does not necessarily represent the current contents of the accumulator. For example, if an arithmetic add is performed such that the entire eight bits of the result are Ø, the zero flag bit is set in the W register. If the subsequent instruction loads the accumulator with a new byte from the memory or scratchpad, the zero flag still remains set indicating that the previous arithmetic function performed had resulted in a zero condition.

3.3.2.3 W (STATUS) REGISTER (Cont.)

The *SIGN BIT* always assumes the opposite polarity of the bit contained in the 2^7 stage of an arithmetic or logic operation. Thus, if a function is performed such that the result is either 0 or positive (the 2^7 bit is 0), the sign bit will be set. The sign bit is reset if the result is negative (2^7 bit is one).

The *INTERRUPT CONTROL BIT* (ICB) is used to mask the CPU interrupt sequence. If the ICB is set, the CPU may be interrupted. However, a ZERO in the ICB causes interrupts to be ignored. A detailed interrupt discussion is contained in Section 3.6.

3.3.2.4 THE SCRATCHPAD REGISTERS AND THE INDIRECT SCRATCHPAD ADDRESS REGISTER (ISAR)

The CPU contains 64, eight bit registers. Systems requiring more than 64 bytes of read/write memory may be expanded with multiple sourced memories such as the 2102 connected to the CPU via an F8 memory interface (MI) circuit.

There are two modes of addressing the scratchpad registers. These are:

- o direct addressing of the lowest order 12 bytes of scratchpad.
- o indirect addressing of any of the 64 bytes of scratchpad.

Direct address is performed by a one byte instruction. The first four bits are the op code while the remaining bits are the scratchpad memory location. Indirect addressing of scratchpad memory locations is also performed in one byte. In this case, however, the indirect scratchpad address register (ISAR) points to the desired scratchpad location.

Figure 3.7 is a diagram of the organization of the scratchpad registers. Notice that only the lower registers of the scratchpad may be directly addressed.

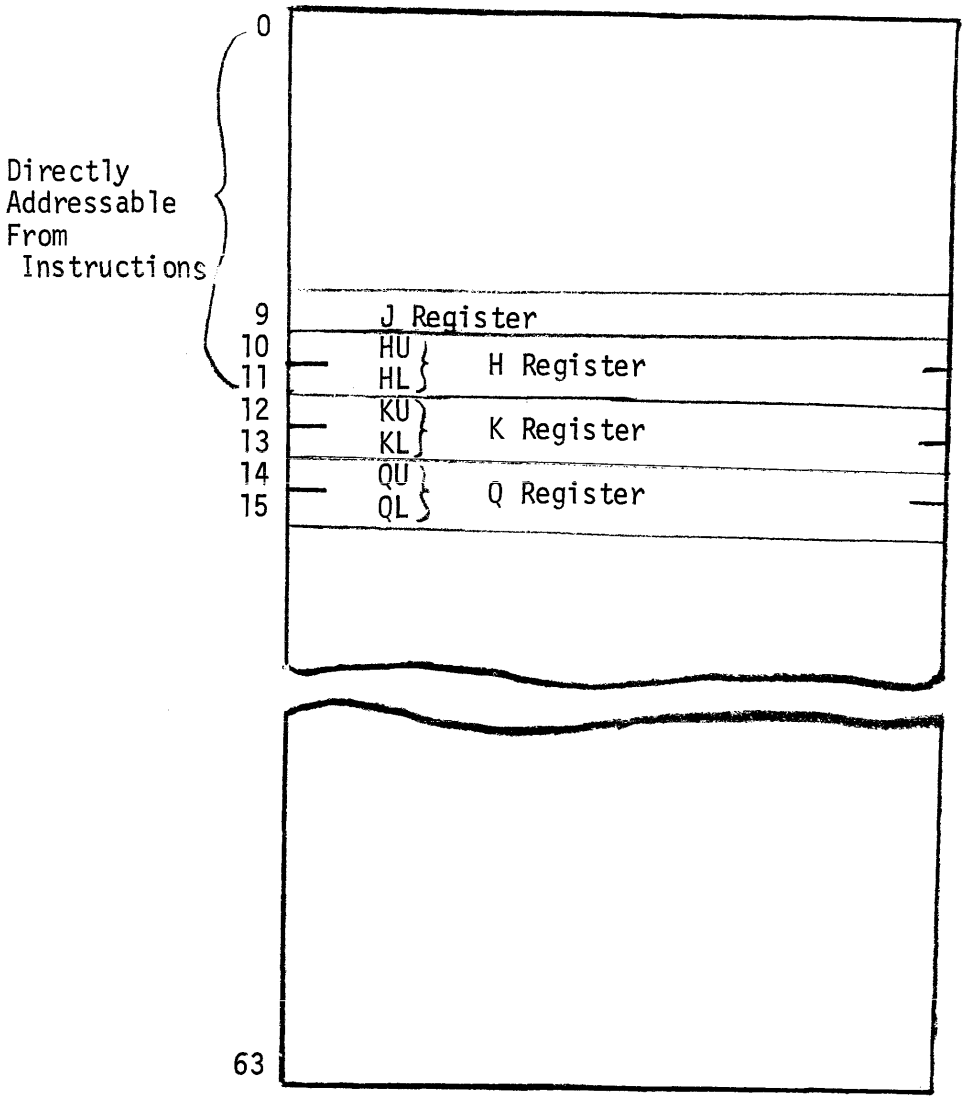


Figure 3.7 The Scratchpad Registers

3.3.2.4 THE SCRATCHPAD REGISTERS AND THE INDIRECT SCRATCHPAD ADDRESS REGISTER (ISAR) (Cont.)

All 64 registers in the scratchpad may be indirectly addressed using the 6-bit indirect scratchpad address register (ISAR).

Special assignments in the scratchpad are used by specific instructions to link the program counter, the stack register, the status register and the data counter to the scratchpad. This allows a multiple level software system under interrupt control. The Q and the K locations are generally used for storing address vectors of 16 bits from either the program counter or the stack register. The Q registers are scratchpad registers 14 and 15. The K registers are address locations 12 and 13. Scratchpad location 10 and 11 are designed as the H registers while location 9 is the J register.

Memory reference instructions that access information storage somewhere in 65,536 bytes of bulk memory use a 16-bit address contained in the data counters in the memory chips. Addresses may be loaded to the 16-bit data counters from one of two memory locations designated as either H or Q in the scratchpad.

Scratchpad location 9 may be used for storing the status register when handling interrupt routines.

Instructions in which the operation code designates that the scratchpad is addressed by the 6-bit content of the ISAR have one of three modes of execution:

- o indirect address by the ISAR.
- o indirect addressing by the ISAR followed by incrementing the ISAR (lower 3 bits) at the conclusion (completion) of the instruction execution.
- o indirect addressing by the ISAR followed by decrementing the ISAR (lower 3 bits) at the conclusion of the instruction execution.

Thus, the ISAR is essentially an incrementing or decrementing pointer to the scratchpad registers. The lower three bits of the ISAR form a modulo eight counter, when the incrementing and decrementing modes are used. Thus, if the ISAR contains 30_8,

an incrementing instruction will reference this location and increase the ISAR to 31_8 while a decrementing instruction will reference the same location and decrement the lower three bits of the ISAR, to yield 37_8.

3.3.2.5 INPUT/OUTPUT PORTS

The CPU has two bidirectional I/O ports. Each port may be used for either gathering data from the external electronics or for outputting data to other circuits. Latches in the outbound side hold the eight bits of output data. The CPU I/O ports are directly selected by the control circuits when the OUTS or INS instruction is executed with an operand of zero or one. Because the ports are directly selected, input and output transfer to these are faster than for other I/O ports. The CPU I/O ports are designated by the 4-bit address 0000 and 0001. The F8 I/O configuration is fully described in Section 3.8. The electrical characteristics of the CPU output bits may be found in Chapter 4.

3.4 ROM AND MI CIRCUITS

While the CPU carries out the processing, the ROM and MI circuits supply memory to an F8 system. To do this, several registers are necessary to link onto the F8 data bus. These are:

- o Program Counter (PC0)
- o Stack Register (PC1)
- o Data Counter (DC)
- o An Incrementer/Adder

Each circuit, the 3851 ROM, 3852 Dynamic Memory Interface, and the 3853 Static Memory Interface, contains these functional registers described in Section 3.4.2. In addition, the 3851 ROM and the 3853 Static Memory Interace each contains an interrupt level.

The F8 interrupt structure is described in Section 3.6. Both the 3852 MI and the 3853 MI have two data counters for flexible memory referencing. The data path organization and the control circuits of these three circuits are described in the next section.

3.4.1 DATA PATH ORGANIZATION AND CONTROL CIRCUITS

This section describes the major busses in the 3851 ROM, 3852 MI and 3853 MI. These busses transfer bytes of the data or address between the CPU, registers, each other, and memory. The gating of these busses and the operation and timing of the circuits is performed by the control circuitry of each part, also described in this section.

3.4.1.1 DATA PATH ORGANIZATION

The three circuits (3851 ROM, 3852 Dynamic MI, and the 3853 Static MI) have very similar data path organization, as figures 3.2, 3.3, and 3.4 reveal. The major registers of each of these circuits are connected by three 16-bit busses and one 8-bit bus. These busses, and the function they perform, are:

- o ADDRESS TRANSFER BUS: This 16-bit bus transfers an address from the data bus to the program counter, stack register, or data counter(s). In addition, the contents of the incrementer/adder and the address bus may also be transferred to these registers via the address transfer bus.

- o ADDRESS BUS: The address bus receives the program counter, stack register, and data counter(s) and passes these addresses to the incrementer/adder. In addition, the 16 lines of the address bus are actually used for memory reference from either the program counter or data counter(s). In the ROM, the upper 6 bits of the address bus are used for selecting ROM pages (one of

64, 1024 byte pages). The lower 10 bits are used as the ROM address. In the Memory Interface circuits, the 16-bit address bus is outputted for use by external memory. There is no paging performed by the MI circuits.

- o MIXER BUS: The mixer bus selects either the program counter or the address transfer bus to be gated into the stack register.
- o DATA BUS: The 8-bit data bus is the principal path for transmitting 8-bit bytes between the other circuits in the F8 system. Each of the three circuits, 3851, 3852, and 3853 contain four port assignments. (Refer to Section 3.4.2.5) These are linked by the data bus to the accumulator of the CPU. The data bus also passes 8-bit addresses between F8 circuits in a system. The four busses are gated by signals produced by the control circuitry, discussed in the next section.

3.4.1.2 TIMING AND CONTROL CIRCUITRY

The 3851 ROM, 3852 MI, and 3853 MI are sequential circuits. The logic states of these circuits are selected by the five control lines generated by the CPU. The central processor also supplies the timing for these circuits (the Ø and WRITE signals). For each instruction cycle (defined by the period of the WRITE signal) the five control lines select a location of the control ROM located in each circuit. The contents of the CROM address are the control lines used by internal gating of the logic. The clock lines are used to sequence the circuit through its logic states.

3.4.2 MAJOR REGISTERS OF THE 3851, 3852, AND 3853

The circuitry of the 3851 ROM, 3852 MI, and the 3853 MI are similar. Each part contains a program counter, stack register, data counter (the MI circuits each have two) an incrementer/adder and four addressable ports.

These registers are discussed in detail in this section. In addition, memory paging in the ROM and MI chips are described.

3.4.2.1 PROGRAM COUNTER (PC0)

The program counter contains the address of the next instruction byte to be fetched from memory. After a fetch cycle executes, the program counter is automatically incremented. There are four ways to modify the PC0 under program control:

- o From the CPU via the data bus using branch instructions.
- o From a memory address using JMP or PI instructions.
- o From the stack pointer with a POP instruction.
- o From the scratchpad memory using PK or LR instructions.

Chapter 5 details changing the contents of PC0 with the instruction set.

3.4.2.2 STACK REGISTER (PC1)

The stack register is directly linked to the program counter. It receives the contents of the program counter whenever an interrupt is generated or when the program counter is pushed to the stack register with a push instruction (PI) or PK instruction. It is the function of the stack register to simplify the creation of address stack. Two sources of data exist for the stack register. Information is normally pushed from the program counter via the mixer bus to the stack register during the push instruction. The stack register may also be loaded directly from the address transfer bus by a program instruction. This facilitates loading the stack register prior to executing a POP instruction for returns from sub-routines back to the next higher level programs in a multi-level program system. Instructions exist to transfer the stack register to and from scratchpad.

3.4.2.3 DATA COUNTER (DC)

The F8 contains a data counter for referencing memory addresses. Because it is 16 bits long, the DC may address up to 64K bytes of memory. A group of instructions exist which use the DC to point to their operands in the memory space. The data counter is incremented by one at the conclusion of the memory access cycle; thus it will be pointing to the next location in memory. This is convenient because control loops for transferring data fields need not contain an extra instruction to increment the data counter. Specific instructions link the data counter to the two bytes of storage located in the CPU scratchpad designated as locations H and Q. Data field locations may be stored in either of these locations and may be transferred to the data counter prior to initiating a routine that will fetch a field of data from bulk memory. The 3852 and 3853 Memory Interface circuits each have two data counters--one active and one inactive. Both are 16 bits wide. The active data counter is used to access memory and communicate to the CPU. The active data counter is loaded directly from an instruction operand using the DCI instruction. In addition, the data counter may be transferred to and from scratchpad with the LR DC, H, LR DC, Q, LR H, DC, and LR Q, DC instructions. The instruction XDC swaps the contents of the active and inactive DC's. The active data counter is incremented after every memory reference while the inactive data counter is not.

3.4.2.4 THE INCREMENTER/ADDER

Each time the program counter or data counter is used to fetch a byte from storage, it is either incremented or modified, depending on the type of instruction to be executed. The incrementer/adder receives the contents of the counter from the address bus, increments it and transfers the result back to the appropriate counter. The incrementing is done over a full 16-bit field. During the execution of branch and add to data counter instructions, eight bits (a displacement vector) are added to the present contents of the register in the incrementer/adder and then returned. In both cases the 8-bit vector is in 2's complement notation; hence, the displacement vector may range from -128 to +127.

3.4.2.5 ADDRESSABLE PORTS

The 3851, 3852, and 3853 each have four addressable ports. These are 8-bit registers connected to the data bus. They are linked to the accumulator of the CPU by the instruction set. Each port is referenced by an 8-bit address. The upper six bits of the address refer to the circuit on which the ports are located while the lower two bits select one of the four ports. Thus, the port addresses are referred to as X0, X1, X2, and X3; where X is a six-bit binary number. Each port on the circuit may be written into with output instructions. The contents of the I/O ports may be read with input instructions. These instructions transfer contents between ports and the accumulator on the CPU. In the ROM circuit, two ports are used as 8-bit I/O ports (refer to Section 3.7 for I/O operation). The remaining two ports are the 8-bit timer and the local interrupt control circuitry (refer to Section 3.5 for timer operation and Section 3.6 for F8 interrupt operations). In the 3852, dynamic memory interface, one port is used for controlling the dynamic refresh circuitry and to select the state of the DMA control circuit. One other port may be used as an 8-bit storage register. The remaining ports are not assigned. In the 3853, static memory interface circuit, one port is assigned to the timer while another is assigned to the local interrupt control circuitry. In addition, two ports are used as a programmable interrupt address vector. Table 3.3 lists the addressable ports and their respective function.

3.4.2.6 PAGE SELECT LOGIC

Page select logic is required in the memory of an F8 system for memory references and to drive the data bus. The ROM circuit performs the page select internally, comparing the upper six address bits with a 6-bit page number. This page number is selected by the user; it is a mask option. The most significant six address lines are compared against the ROM page number in the page select block and a decision is made to activate or deactivate the fetch control signals. The page select logic is external to the 3852 MI and 3853 MI circuits allowing users total flexibility for memory addressing.

3.4.2.6 PAGE SELECT LOGIC (Cont.)

The internal logic of the memory interface circuits, however, do require a page select signal. This is used to drive the data bus whenever the CPU requests the contents of the program counter or data counter. Several instructions transfer PC1 and DC to the scratchpad memory. If the address in the requested register lies within the memory space of the MI (determined by the page select line) then that circuit will respond. This also holds true for the ROM, however, the page selection is performed internally. Both the 3852 MI and 3853 MI have a pin, REGDR. The page select line should be connected to this pin. The REGDR pin is bidirectional. It may simultaneously be used to control the enabling of a buffer/driver for the data bus. The ROM circuit has a pin, DBDR, to control the enabling of a buffer/driver for the data bus.

3851 ROM

PORT ADDRESS	FUNCTION
X00	ROM I/O Port A
X01	ROM I/O Port B
X10	ROM Local Interrupt Control
X11	ROM Timer

X is a 6-bit binary number. It cannot be 0.

3852 DYNAMIC MEMORY INTERFACE

PORT ADDRESS	FUNCTION
0C	8-Bit Register
0D	Control Bits for refresh and DMA
0E	Not Assigned
0F	Not Assigned

3853 STATIC MEMORY INTERFACE

PORT ADDRESS	FUNCTION
0C	MI Interrupt Vector, upper byte
0D	MI Interrupt Vector, lower byte
0E	MI Local Interrupt Control
0F	MI Timer

TABLE 3.3

3.5 LOCAL TIMER

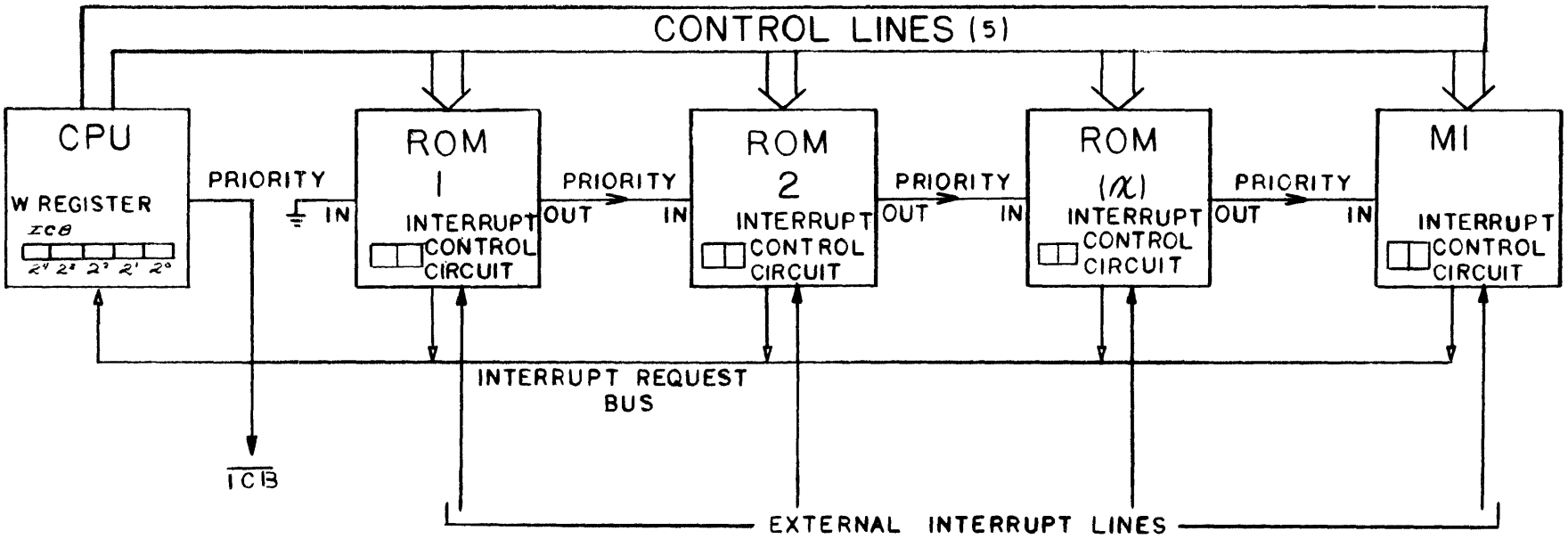
Each memory circuit (ROM and MI) has a local timer to generate program initiated delays. To the programmer, the timer is an 8-bit register, addressable via F8 output instructions to the specified timer port address. Delay codes, calculated by the assembler, are loaded into the accumulator and then transferred to the timer (a polynomial shift register). An output instruction to the timer port number performs this function. After it is loaded, the timer counts down. A table of delay codes matched to delay times appears in the Appendix.

The timer runs continuously. It signals the interrupt control circuitry after every timer cycle (3.953 ms). However, when an OUTS instruction is executed with the timer port number as the operand, the timer is jammed with a specific count and the local interrupt control logic clears any stored timer interrupt. The timer continues from that count to count down and generate an internal interrupt request. Again, the timer continues to cycle every 3.953 milliseconds (for a 2MH system). If the ICB of the CPU is not set, or if the local interrupt control logic is not set for timer interrupts, a timer initiated interrupt will be stored by the local interrupt control circuitry. When the local interrupt control logic is finally set, the memory chip will request interrupt service; the request will be serviced when the ICB of the CPU is set.

Time delays between 0 and 254 counts may be chosen. The timer is decremented once every 31 Ø clock cycles. Therefore, the counter may count as high as 7905 clock cycles. (For a system at 2MHz, a Ø clock cycle occurs every 500 ns.) Longer durations are achieved by counting multiple interrupts as they occur. The timer may be stopped by loading it with all ONES.

3.6 INTERRUPT

Figure 3.8 is a block diagram of the interrupt interconnection for a typical F8 system. Both the 3851 ROM and 3853 Static memory Interface have the capability for either of two types of interrupts, internal or external. The internal interrupts may be generated by the programmable timer while the external interrupt is stimulated by the outside world. A local interrupt control circuit containing two latches is built on each chip.

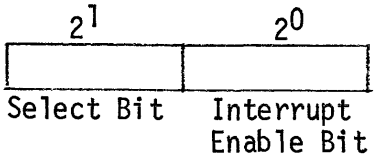


F8 INTERRUPT INTERCONNECTION

Fig 3.8

These latches are the Select Bit and the Interrupt Enable Bit.

LOCAL INTERRUPT CONTROL BITS



These two bits have four possible states:

Select Bit	Interrupt Enable Bit	Function
0	0	No Interrupt
0	1	External Interrupt Enabled
1	0	No Interrupt
1	1	Timer (Internal) Interrupt Enabled

These control latches are loaded under program control using an output instruction. This loading clears the interrupt control logic, execpt for any pending Timer Interrupt. The operand for the OUT or OUTS instruction must be the predefined port number of the interrupt control circuit. (Section 3.4.2.5 Addressable Ports) The two control bits allow each interrupt circuit to have independently controlled enable/disable capability; if enabled, the select bit may choose either internal (timer generated) interrupts or external interrupts.

Each ROM has a $\overline{\text{PRIORITY IN}}$ and a $\overline{\text{PRIORITY OUT}}$ line so that they may be daisy chained together, in any order, to form a priority level of interrupts. The first circuit in line should have its $\overline{\text{PRIORITY IN}}$ line tied to V_{SS} (or tied to the $\overline{\text{ICB}}$ pin of the CPU chip). The MI only has a $\overline{\text{PRIORITY IN}}$ line, so it must be at the end of the chain. All memory chips are tied back to the CPU chip via the interrupt request bus. When an interrupt requires servicing, the local interrupt logic signals the central processor via the interrupt request bus and does not pass on the $\overline{\text{PRIORITY OUT}}$ signal. If the memory chip has no interrupt, it simply passes the $\overline{\text{PRIORITY OUT}}$ signal on to the next memory chip in line. Again, the same decision is made by that unit.

By daisy chaining the priority lines, the interrupt levels are set up. The first memory chip in line has the highest interrupt level and the last units (Figure 3.8) are the lowest. The MI in an F8 system will always be at the lowest priority level.

To generate a timer interrupt, the timer must be set under program control. An interrupt occurs when the timer times out AND the interrupt control has been set (Select Bit = 1, Enable Bit = 1) AND ICB of the CPU is set. The timer may time out before the Interrupt Control Bit is set or the Local Interrupt Control is enabled for internal interrupts; even so, an interrupt will still be initiated after the required conditions have been met. Any pending timer interrupt is cleared whenever the timer of the memory chip is reloaded. The ICB is always cleared after the CPU has acknowledged an interrupt request.

The generation of an external interrupt request is also controlled by the local interrupt control circuit. If the Select Bit is set to zero and the Enable Bit is set to one, then the control logic of the memory chip will be responsive to external interrupts. To guarantee an interrupt, the external interrupt line must drop from V_{DD} to V_{SS} (1 to 0), and stay down for a minimum of two WRITE clock times (4 μs for a 2 μs system clock). The ICB may or may not be set when this occurs. If it is not set, the request will be stored by the local interrupt control logic until the ICB is reenabled. However, the stored external interrupt request will be lost whenever the control bits are reloaded. The ICB will be cleared after the CPU has acknowledged the interrupt request. The stored external interrupt request will be cleared after the interrupt is serviced.

The operation of the local interrupt control circuitry is complex The above word description may be an oversimplification of this crucial function.

Within each local interrupt control circuit is a 16-bit interrupt address vector. This vector is the address to which the program counter will be set after an interrupt is acknowledged. Thus, it is the address of the first executable instruction of the interrupt routine. The 3851 ROM has a mask programmed interrupt

address, selected by the user. It is another mask option of the ROM (refer to Section 3.2). Fifteen bits are selected by the designer. These are bits 0 thru 6 and 8 thru 15. Bit seven (2^7) is dependent upon the type of interrupt. This bit will be a Ø for internal timer generated interrupts and a 1 for external interrupts. The interrupt address vector of the 3853 Static Memory Interface is programmable via output instructions. Port 0C of the 3853 MI is assigned to hold the upper eight bits of the 16-bit address (bits 2^8 thru 2^{15}) while port 0D hold the seven lower bits (bits 2^0 thru 2^6). Bit seven (2^7) of the interrupt address vector (bit 7 of port 0D) is controlled by the local interrupt hardware, just as in the F8 ROM chip. It will be a Ø if timer interrupts are enabled and a 1 if external interrupts or no interrupts are enabled. The programmable interrupt address vector of the 3853 MI is assigned the following bits:

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	Port 0C
7							0	
2^{7*}	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Port 0D
7							0	

* This bit is controlled by the local interrupt control circuit.

The interrupt vector may be loaded using an OUT or OUTS instruction. It may be read back into the accumulator using an IN or INS instruction with port 0C or 0D as operands.

When the interrupt logic sends an interrupt request signal and the CPU is enabled to service it, the normal state sequence of the CPU is interrupted at the end of an instruction. The CPU signals the interrupt circuits via the five control lines. The requesting local interrupt circuit sends a 16-bit interrupt address vector (from the interrupt address generator) onto the data bus in two consecutive bytes. The address is made available to the program counter via the address demultiplexer circuits while simultaneously, it is made available to all other circuits on the data bus. It is the address of the next instruction to be executed. The program counter (PC0) of each memory chip is set with this new address while the stack register (PC1) is loaded with the previous contents of the program counter.

The information in PC1 is lost. Thus, the next instruction to be executed is determined by the value of the interrupt address vector.

The Interrupt Control Bit (ICB) of the CPU (loaded in the W register) allows interrupts to be recognized. Clearing the ICB prevents acknowledgement of interrupts. The ICB is cleared during power on, external reset, and after an interrupt is acknowledged. The interrupt status of the ROM chips are not affected by the execution of the DISABLE INTERRUPT (DI) instruction. At the conclusion of most instructions, the fetch logic checks the state of the Interrupt Request Line. If there is an interrupt, the next instruction fetch cycle is suspended and the system is forced into an interrupt sequence. The CPU allows interrupts only after certain instructions.

The exceptions are the following F8 instructions.

(PK)	PUSH K
(PI)	PUSH IMMEDIATE
(POP)	POP
(JMP)	JUMP
(OUTS)	OUTPUT SHORT (Excluding OUTS 01 and 02)
(OUT)	OUTPUT
(EI)	SET ICB
(LR W,J)	LOAD THE STATUS REGISTER FROM SCRATCHPAD
POWER ON	

Therefore, it is possible to perform one more instruction after these CPU operations without being interrupted. This is especially useful during routines which perform interrupt housekeeping operations. For instance, whenever an interrupt occurs, the program counter is pushed into the stack register (PC1). Most likely, this value should be saved for later use. If there were no way to block interrupts, another one could occur before PC1 has been safely stored into memory. Thus, the ability to disable interrupts is essential.

3.7 INPUT/OUTPUT

A typical F8 system will have at least one CPU with one or more ROM memory circuits and perhaps a Memory Interface Circuit connected to the data bus. An F8 I/O operation involves the movement of data between the accumulator and an I/O port. This may be accomplished with one of four instructions (refer to Chapter 5). These instructions are also used to reference the interrupt control blocks and the timer registers (also referred to as "ports". Refer to Section 3.5 for timer and

Section 3.6 for interrupt). An example of an output instruction is:

```
OUT   aa
```

In this example, byte aa refers to port address aa. Each port (I/O, interrupt control block, timer) in an F8 system has a predefined output port number. The two CPU ports are always labeled 00 and 01; they are accessed by the INS and OUTS instruction only.

PORT ADDRESS
H'00' CPU circuit, I/O port 0
H'01' CPU circuit, I/O port 1
H'02' not assigned
H'03' not assigned

The selection of the four ROM port addresses is a customer's mask option while the Memory Interface port numbers are preassigned. Each circuit, ROM or MI, will be given four sequential port addresses; refer to Section 3.4.2.5 for an explanation of address assignments.

The transfer of data from the accumulator to the I/O port is completed with an OUT or OUTS instruction. Sampling of the I/O ports may be done with either an IN or INS instruction.

The OUTS and INS instructions are one byte in length; the first four bits are the op code and the remaining four bits are the port address. Thus, only 16 ports may be referenced with these instructions. These are the lower 16 ports, 00 through 0F. The OUT and IN instructions are two bytes long. The first byte is the op code while the second byte is the port number. Any one of 256 ports may be referenced by this instruction. Ports 00 and 01 are addressable only with the OUTS and INS instructions.

Each I/O port has an 8-bit latch on the output side; therefore, it will retain the data of the last output instruction. These ports are bidirectional, so data may also be read into the accumulator from the same port. The only restriction is that the output port bit latch must have a zero in it for input data to be valid. This is a consequence of the wired-OR connection of the I/O ports (refer to Chapter 4). So, if a full 8-bit byte is to be transferred into the accumulator, the I/O port buffer must have previously been set to all zeros.

Unused I/O ports may also be used as data latches. There can be OUTPUT and INPUT bits on the same port.

The ROM chip offers three I/O port options:

- o Standard pullup
- o Open drain
- o Driver pullup

Chapter 4 details these configurations.

3.8 INITIALIZING REQUIREMENTS

The power on detect circuit for an F8 system is located in the CPU. This circuit insures that all critical control circuits and registers are in a valid operating condition when power is first applied. It performs the following functions:

- o Pushes previous contents of the program counter to the stack register
- o Resets the program counter to address "0000"
- o Resets the Interrupt Control Bit (ICB)
- o Sets control block on the 3852 MI circuit

When power is connected to the circuit or the reset line goes low, the CPU clears the program counter (PC0), pushing its previous contents into the stack register (PC1). Therefore, the instruction in location zero is executed first. The interrupt control bit is also cleared at this time. The rest of the F8 system is initialized under program control. The local interrupt block of the individual memory chips must be loaded before allowing any interrupts to occur. Output latches must be reset to zero before they may be used to input data.

3.9 DYNAMIC MEMORY INTERFACE CIRCUITS 3852

The 3852 memory interface circuit allows dynamic memory to interface to an F8 system. This circuit will perform memory

refresh between CPU memory accesses. Memory refresh is accomplished without degradating system performance. The 3852 memory interface circuit also interfaces to the 3854 direct memory access circuit to control DMA transfers into and out of an F8 system. The principal features of this circuit are:

- o Dynamic and Static Memory Interface
- o 16 Bit Program Counter
- o 16 Bit Stack Register
- o Two 16 Bit Data Counters
- o On chip Incrementer/Adder
- o DMA Control

Figure 3.3 shows a block diagram of the 3852 MI circuit.

3.9.1 MEMORY & DMA INTERFACE

The MI circuit requires the use of an external tristate buffer with storage to link memory to the F8 data bus. Three control signals are supplied by the MI circuit to control this buffer.

These signals are:

CYCLE REQ — A signal to identify the start of a memory access. This is useful for generating strobes for dynamic RAMs.

CPU SLOT — This line signals a time period during which the CPU will perform a memory access.

CPU READ — This line identifies a time period during which the CPU requests a byte of data from memory i.e. the CPU is reading a byte.

The interrelation of these signals to control the RAM-CPU interface buffer is:

CYCLE REQ	AND*	CPU SLOT	CPU READ	STATE OF RAM-CPU INTERFACE BUFFER
0			0	Float outputs to F8 Data Bus
1			0	Float outputs to F8 Data Bus, load storage
1			1	Drive outputs to F8 Data Bus, from memory
0			1	Drive outputs to F8 Data Bus, from buffer storage

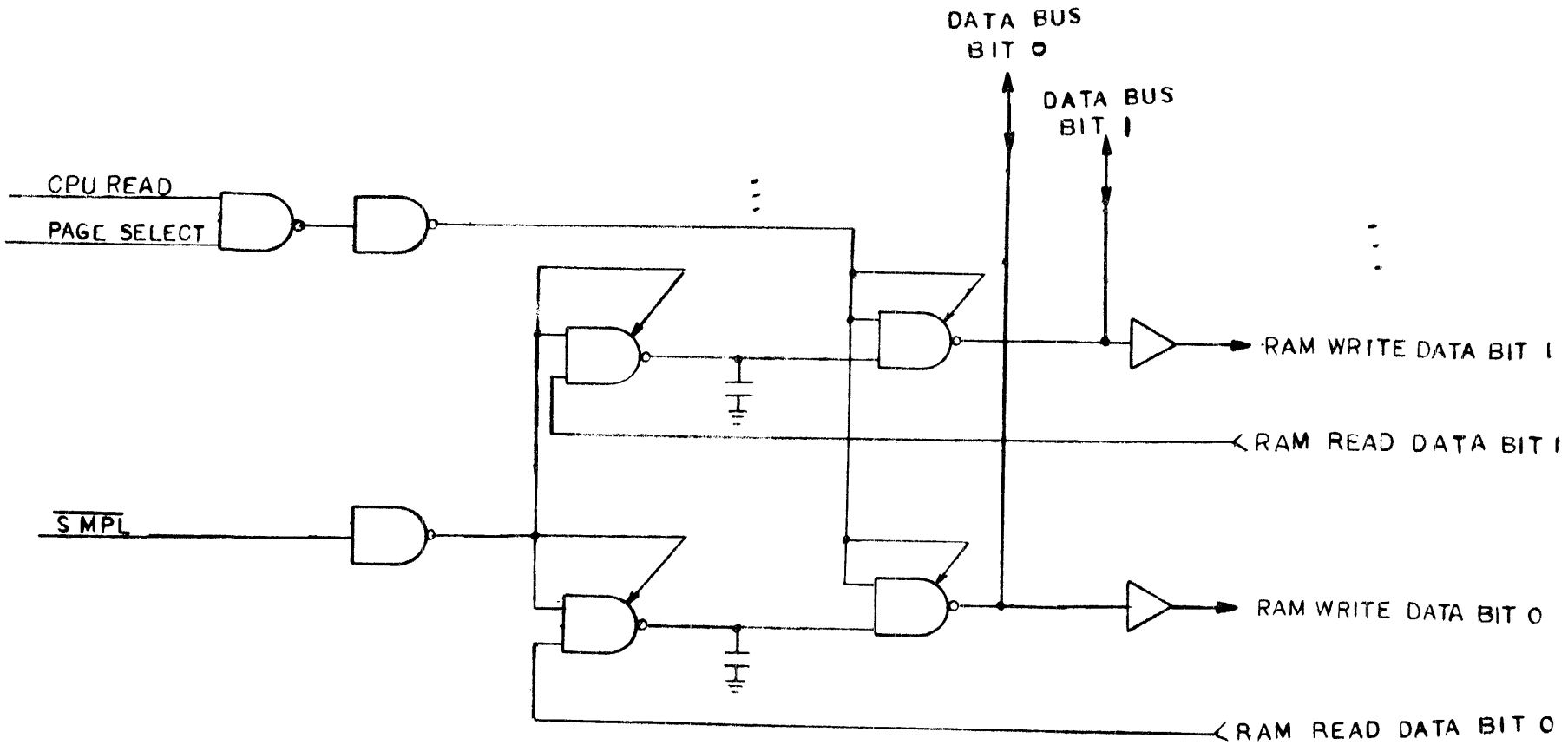
* Logical AND

Figure 3.9 is a block diagram of a typical buffer. A Read/Write signal is supplied by the MI to control the operation of the memory chips.

The MEM IDLE pin of the 3852 MI is used to control the DMA channel. The MEM IDLE line is at V_{DD} when memory is available. Memory is available for DMA use during every cycle except for one of the two cycles of the store instruction, ST.

3.9.2 CONTROL CIRCUITS FOR DYNAMIC RAMS

The 3852 memory interface chip provides a strobe signal at the start of each memory cycle and contains circuitry for refreshing dynamic memory devices. This is implemented using two counters. One counter cycles through 64 addresses doing a read in each, while the other keeps track of the number of available DMA slots. Every fourth or eighth DMA slot is used to refresh one memory address. A complete refresh cycle will take 1.5 ms with a worst case program running on a 2 μs instruction cycle time and using every eighth available slot for refresh. The signal MEM IDLE is held at V_{SS} level during a DMA slot that is used for refreshing.



DATA BUS BUFFER
FOR 3852 M I

Fig 3.9

3.9.2 CONTROL CIRCUITS FOR DYNAMIC RAMS (Cont.)

The high to low transitions of MEM IDLE signals the start of a memory access. There are either two or three memory accesses during each CPU cycle. The number of accesses during a cycle is a function of the CPU state.

The memory interface has a control block to set its function under program control. This 3-bit register may be referenced with an output instruction using OD_{16} as the port address operand. The function of these bits are:

Bit #	Function	Logical Value 1	Logical Value 0
0	DMA	OFF	ON
1	Refresh	ON	OFF
2	Refresh	Mode 1	Mode 2

Mode 1 - Every 4th DMA slot is used for refresh
Mode 2 - Every 8th DMA slot is used for refresh

The three bits are initialized to 1's during the power-on reset.

The interfacing between an F8 system and the Fairchild 4K RAM is shown in Figure 3.10. Figure 3.11 is the external circuitry necessary for generating the necessary control strobe. A detailed description of the timing of the output signals is contained in Chapter 4.

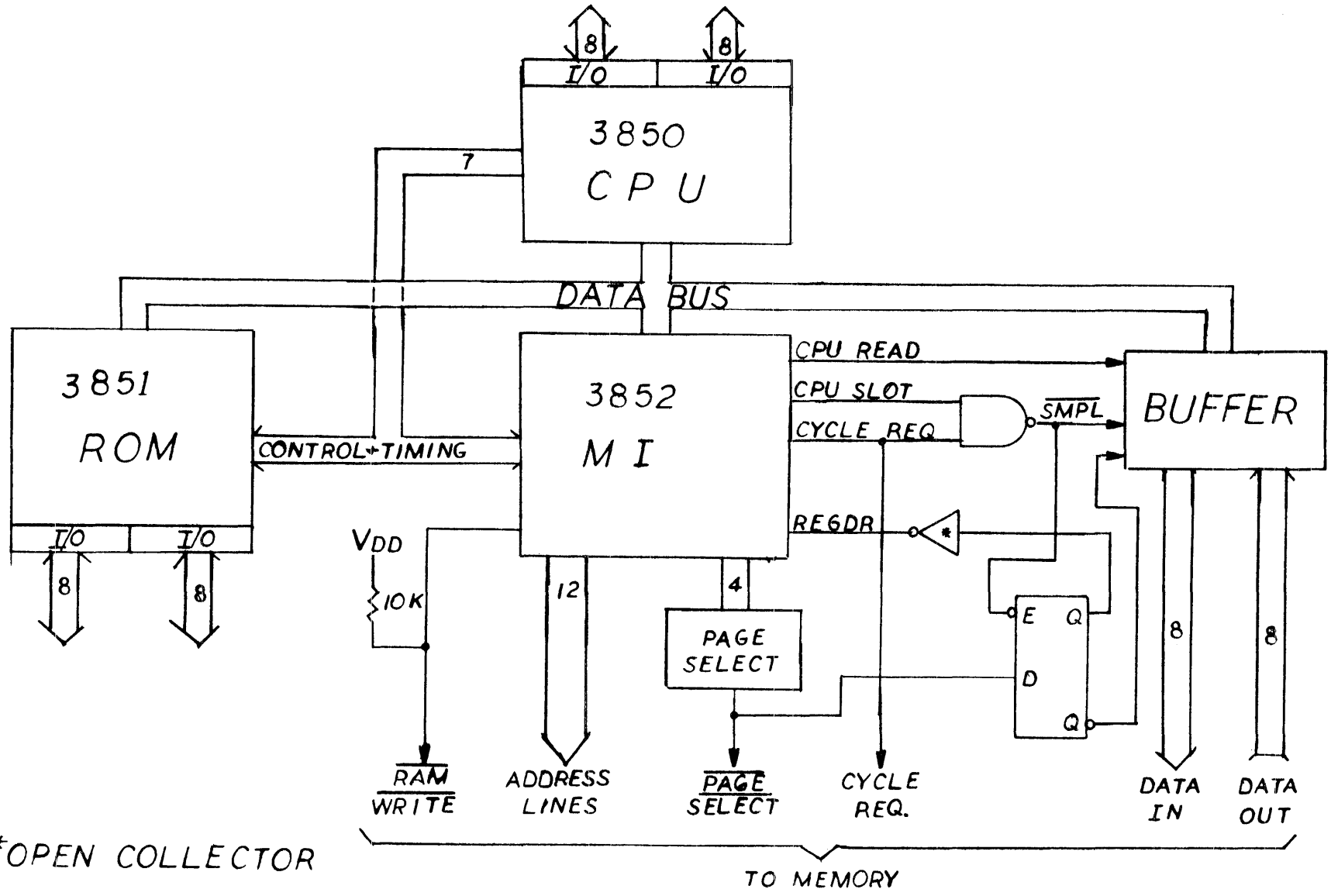
3.9.3 DESCRIPTION OF PINS OF THE 3852 MI CHIP

Data Bus - 8 Lines, Bidirectional: These lines form the main communication bus for the F8 system.

ROMC 5 Lines, Input: ROMC0 - ROMC4 are the five control lines originating from the CPU.

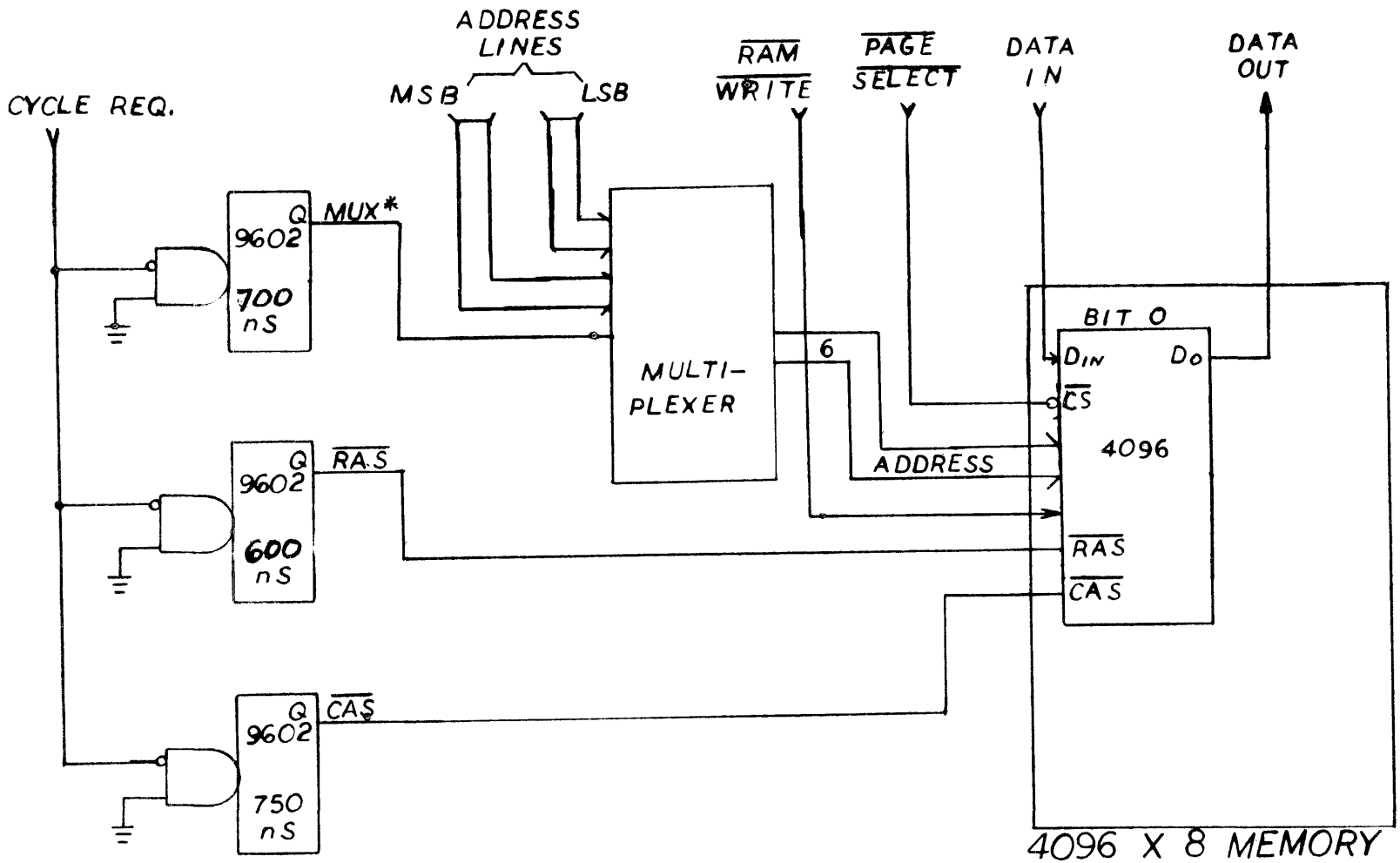
Ø WRITE - 2 Lines, Input: These are the timing signals generated by the CPU.

REGDR - 1 Line, Bidirectional: REGDR controls the passing of



3852 MEMORY INTERFACE IN AN F8 SYSTEM

Fig. 3.10



*WHEN MUX IS HIGH, THE MSB BITS ARE SELECTED

3852 MEMORY REFRESH WITH 4096 DYNAMIC RAM

Fig. 3.11

the PC1 or DC registers onto the F8 data bus during specific instructions. An output from the memory page select logic should be tied to this pin through an open collector gate. This line is wired-AND ed with a second signal to internally control data bus drivers.

RAM ADDR - 16 Lines, Output: These lines are the memory address lines used to select the desired byte in memory.

$\overline{\text{RAM WRITE}}$ - 1 Line, Output: The $\overline{\text{RAM WRITE}}$ line specifies the mode of memory addressing.

1 = Read from memory

0 = Write into memory

CYCLE REQ - 1 Line, Output: Signals the start of each memory access and controls the memory buffer.

CPU SLOT - 1 Line, Output: This line controls the memory buffer. It identifies a CPU memory access time period.

CPU READ - 1 Line, Output: This line controls the memory buffer. It identifies a CPU memory read time period.

MEM IDLE - 1 Line, Output: MEM IDLE indicates when memory is available for DMA use.

3.10 STATIC MEMORY INTERFACE CIRCUIT 3853

The 3853 memory interface circuit is another compatible component of the F8 microprocessor family. This chip allows standard memory elements to be incorporated into an F8 system; it is used for interfacing with static memory elements such as the 2102 (1024 x 1 static RAM) and has full interrupt capability.

The principal features of the Static Memory Interface Circuit, 3853, are:

- Static memory interface
- 16-Bit Program Counter
- 16-Bit Stack Register
- Two 16-Bit Data Counters

- o Local Interrupt Control Circuitry
  (Independently set and reset under program control, programmable interrupt vector)
- o On-Chip Incrementor/Adder
- o Real Time Counter For Timing Functions

3.10.1 STATIC MEMORY INTERFACE: 3853

The MI has a 16-bit wide address bus. These lines may be used for accessing up to 64K bytes of memory. Two signals are used to control the added memory circuits. One line, CPU READ, controls the external buffers; this signal is True during instruction cycles that reference memory or fetch instructions. A buffer must be externally provided to connect memory data onto the F8 data bus. The other control line, $\overline{\text{RAM WRITE}}$, determines whether the data will be written into or read from the memory.

Figure 3.12 is an application showing the 3853 MI in a memory intensive system. A detailed description of the timing of the output signals is contained in Chapter 4.

3.10.2 DESCRIPTION OF PINS OF THE STATIC CIRCUIT - MI 3853

Data Bus - 8 Lines, Bidirectional: Thes lines form the main communication bus for the F8 system.

ROMC- 5 Lines, Input: ROMC0 - ROMC4 are the five control lines originating from the CPU.

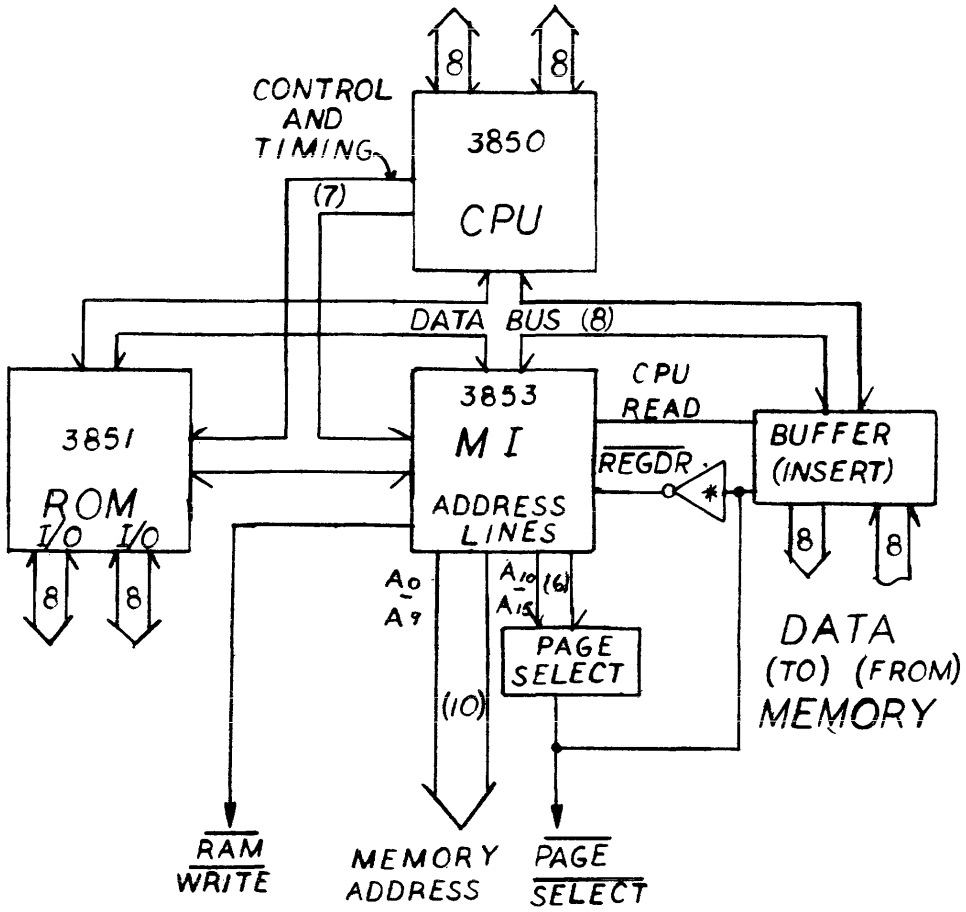
Ø, WRITE - 2 Lines, Input: These are the timing signals generated by the CPU.

REGDR- 1 Line, Bidirectional: REGDR controls the passing of the PC1 or DC registers onto the F8 data bus during specific instructions. An output from the memory page select logic should be tied to this pin through an open collector gate with a pull-up resistor. This line is wire ANDed with a second signal to internally control data bus drivers.

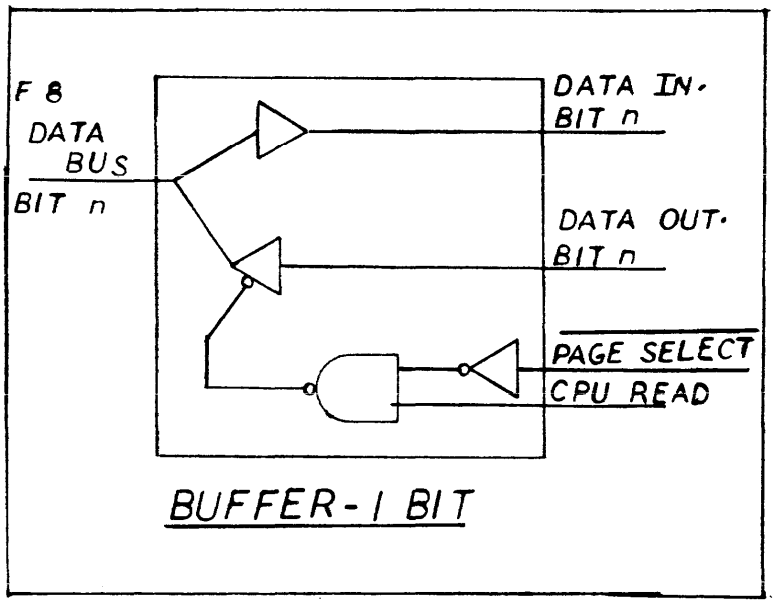
RAM ADDR - 16 Lines, Output: These lines are the memory address lines used to select the desired byte in memory.

$\overline{\text{RAM WRITE}}$ - 1 Line, Output: The $\overline{\text{RAM WRITE}}$ line specifies the mode of memory accessing.

1 = Read From Memory
0 = Write Into Memory



*OPEN COLLECTOR



F 8 SYSTEM WITH 3853 STATIC MEMORY INTERFACE

Fig. 3.12

CPU READ - 1 Line, Output: This line helps to control the memory buffer.

$\overline{\text{PRIORITY IN}}$ - 1 Line, Input: This line is part of the interrupt logic of the 3853 Memory Interface circuit.

$\overline{\text{INTERRUPT REQ}}$ - 1 Line, Output: This line is also part of the interrupt control.

$\overline{\text{EXT INT}}$ - 1 Line, Input: This line is part of the MI interrupt control.

V_{DD}, V_{SS}, V_{GG}: These supply power to the chip.

MOS PRELIMINARY SPECIFICATION

Dec. 1974

3850

F8 CENTRAL PROCESSING UNIT

GENERAL DESCRIPTION: The 3850 is the Central Processing Unit of the F8 Microprocessor family. When connected to an external clock, a crystal, or an RC network, the 3850 generates all the necessary timing and control signals to make a functioning system. The CPU can decode and execute a complete set of over 70 machine instructions operating on 8-bit bytes of information. The 3850 is manufactured using n-channel Isoplanar MOS technology.

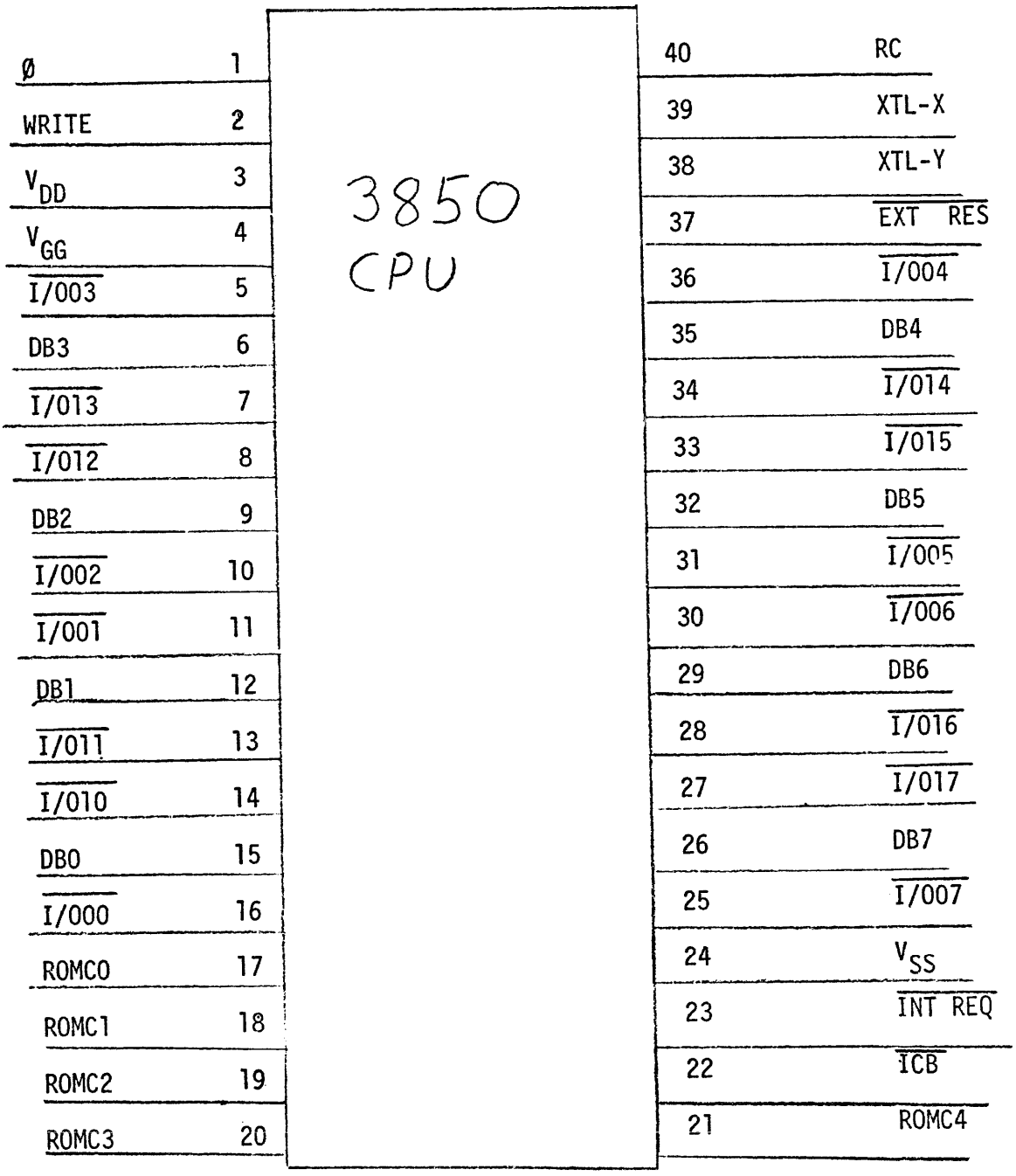
The 3850 CPU is designed to interconnect to the other chips in the F8 family to build an 8-bit microprocessor.

FEATURES:

- o 2 μs CYCLE TIME
- o 64-BYTE RAM
- o TWO I/O PORTS-WITH OUTPUT LATCHES
- o 8-BIT ALU
- o BINARY AND DECIMAL ARITHMETIC
- o INTERRUPT CONTROL
- o +5 V AND +12 V POWER SUPPLIES
- o LOW POWER DISSIPATION - TYPICALLY LESS THAN 300 mW
- o POWER ON RESET
- o MASTER CLOCK
- o CLOCK CIRCUITS CAN BE OPERATED IN ONE OF 3 MODES
  - AN RC NETWORK
  - CRYSTAL CONTROL
  - EXTERNAL MASTER FREQUENCY
- o OVER 70 INSTRUCTIONS

PIN NAMES

Pin Names		Type
DB0 - DB7	Data Bus	Bidirectional
$\overline{I/O00}$ - $\overline{I/O07}$	I/O Port Zero	Input/Output
$\overline{I/O10}$ - $\overline{I/O17}$	I/O Port One	Input/Output
ROMC0 - ROMC4	Control Lines	Output
RC	RC Timing Input	Input
XTL-X	Crystal Clock Input	Input
XTL-Y	External Clock Input	Input
EXT RES	External Reset	Input
ø,WRITE	Clocks	Output
$\overline{ICB}$	Interrupt Control Bit	Output
$\overline{INT\ REQ}$	Interrupt Request	Input
V_{DD} V_{SS} V_{GG}	Power	Input


3850
CPU
Ø 1
WRITE 2
V_{DD} 3
V_{GG} 4
$\overline{I/003}$ 5
DB3 6
$\overline{I/013}$ 7
$\overline{I/012}$ 8
DB2 9
$\overline{I/002}$ 10
$\overline{I/001}$ 11
DB1 12
$\overline{I/011}$ 13
$\overline{I/010}$ 14
DB0 15
$\overline{I/000}$ 16
ROMC0 17
ROMC1 18
ROMC2 19
ROMC3 20
40 RC
39 XTL-X
38 XTL-Y
37 $\overline{EXT\ RES}$
36 $\overline{I/004}$
35 DB4
34 $\overline{I/014}$
33 $\overline{I/015}$
32 DB5
31 $\overline{I/005}$
30 $\overline{I/006}$
29 DB6
28 $\overline{I/016}$
27 $\overline{I/017}$
26 DB7
25 $\overline{I/007}$
24 V_{SS}
23 $\overline{INT\ REQ}$
22 $\overline{ICB}$
21 ROMC4

ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (Above which useful life may be impaired)

V_{GG}	+15 V to -0.3 V
V_{DD}	+7 V to -0.3 V
All Inputs and Outputs	+15 V to -0.3 V
Storage Temperature	-55°C to +150°C
Operating Temperature	0°C to + 70°C

Note: All Voltages With Respect to V_{SS}

DC CHARACTERISTICS: V_{SS} = 0.0 V, V_{DD} = +5 V ± 5%, V_{GG} = + 12 V ± 5%, T_A = 0°C to + 70°C (Note 1)

SUPPLY CURRENTS

SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
I_{DD}	V_{DD} Current		30	80	mA	f = 2MHz
I_{GG}	V_{GG} Current		10	20	mA	f = 2MHz

DATA BUS

SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
V_{IH}	Input Voltage HIGH	3.5		V_{DD}	V	
V_{IL}	Input Voltage LOW	V_{SS}		0.8	V	
V_{OH}	Output Voltage HIGH	3.9		V_{DD}	V	I_{SOURCE} = 100μA
V_{OL}	Output Voltage LOW	V_{SS}		0.4	V	I_{SINK} = 100μA
I_L	Leakage Current			1.0	μA	V_{IN}=6V, Output In 3-State Mode

I/O PORTS

V_{IH}	Input Voltage HIGH (Hysteresis Input*)	2.9		V_{DD}	V	Internal Pull-up to V_{DD} provides TTL Compatability
V_{IL}	Input Voltage LOW	V_{SS}		.8	V	
V_{OH}	Output Voltage HIGH	2.9		V_{DD}	V	I_{SOURCE}=100μA
V_{OH}	Output Voltage HIGH	3.5		V_{DD}	V	I_{SOURCE}=1μA
V_{OL}	Output Voltage LOW	V_{SS}		0.4	V	I_{SINK} =2.0mA
I_{IL}	Input Current LOW			1.2	mA	V_{IN}=V_{SS}

*Hysteresis input provides 0.5V noise immunity

CONTROL LINES

SYMBOL	CHARACTERISTIC	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
V_{OL}	Output Voltage LOW	V_{SS}		0.4	V	I_{SINK} = 100 uA
V_{OH}	Output Voltage HIGH	3.9		V_{DD}	V	I_{SOURCE} = 100uA

EXTERNAL CLOCK INPUT

V_{IH}	Input Voltage HIGH	4.0		V_{DD}	V	
V_{IC}	Input Voltage LOW	V_{SS}		0.8	V	
I_{LI}	Input Leakage Current			1.0	uA	V_{IN} = 6V

EXTERNAL RESET

V_{IH}	Input Voltage HIGH	3.5		V_{DD}	V	Internal Pull-up to V_{DD}
V_{IL}	Input Voltage LOW	V_{SS}		0.8	V	
I_{IL}	Input Current LOW			0.3	mA	V_{IN} = V_{SS}

CLOCK LINES

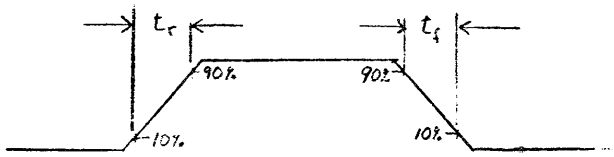
SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
V_{OH}	Output Voltage HIGH	4.4		V_{DD}	V	I_{SOURCE} = 100 µA
V_{OL}	Output Voltage LOW	V_{SS}		0.4	V	I_{SINK} = 100 µA
		INTERRUPT CONTROL BIT				
V_{OH}	Output Voltage HIGH	3.9		V_{DD}	V	I_{SOURCE} = 100 µA
V_{OL}	Output Voltage LOW	V_{SS}		0.4	V	I_{SINK} = 100 µA
		INTERRUPT REQUEST				
V_{IH}	Input Voltage HIGH	3.5		V_{DD}	V	Internal Pull-up to V_{DD}
V_{IL}	Input Voltage LOW	V_{SS}		0.8	V	
I_{IL}	Input Low Current			1.0	mA	V_{IN} = 0.4V

NOTE 1: Junction Leakage Current, all pins:

1.0 µA max. when V_{IN} = 6V, V_{DD} = V_{GG} = V_{SS}

AC Characteristics: V_{SS} = 0.0V, V_{DD} = 5.0V ± 5%, V_{GG} = 12V ± 5%, T_A = 0°C to 70°C

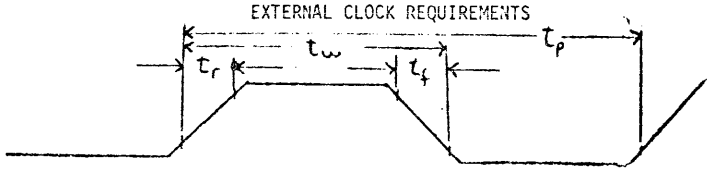
f = 200 KH2 to 2 MH2, (Notes 2, 3)



OUTPUT VOLTAGE WAVEFORM

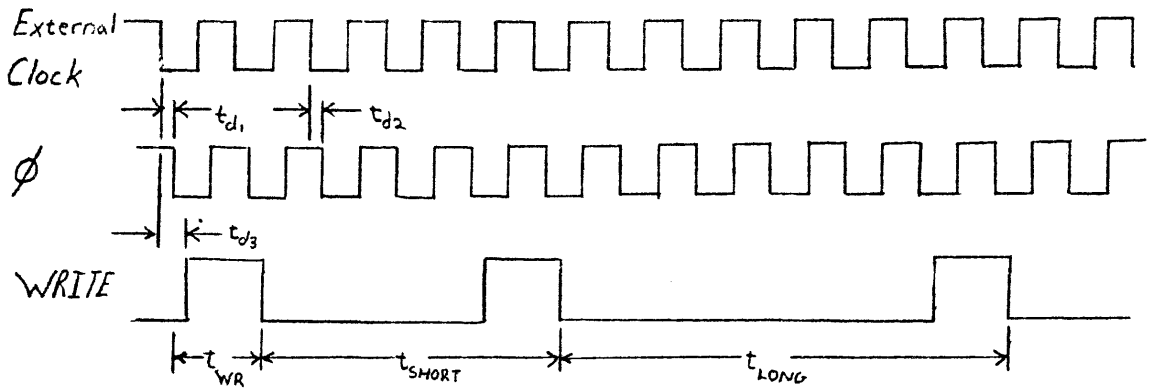
PIN NAME	tr MAX.	tf MAX.	TEST CONDITIONS
ø	120 ns	80 ns	C_L = 100 pf
WRITE	120 ns	80 ns	C_L = 100 pf
I/O	180 ns	140 ns	C_L = 100 pf

EXTERNAL CLOCK REQUIREMENTS



PARAMETER	MIN.	MAX.
tr	0	50 ns
tf	0	50 ns
tw	(.4) X tp	(.6) X tp
tp	[illegible]	10 µs

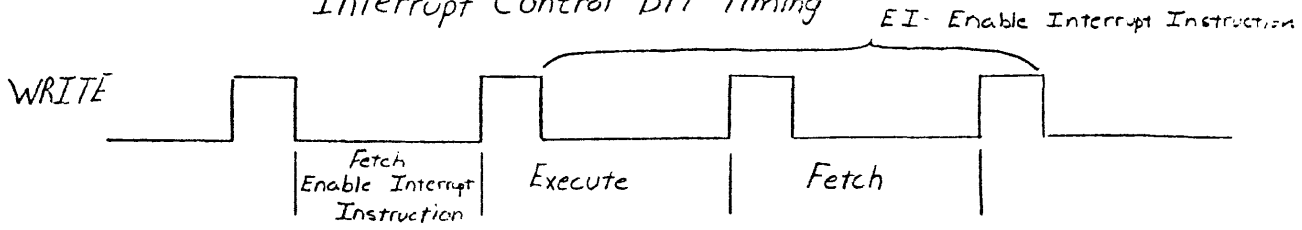
F8 Clock Timing

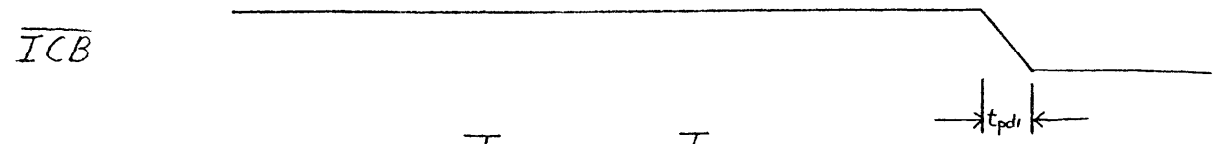


Long Cycle = 6 Clock Pulses
Short Cycle = 4 Clock Pulses

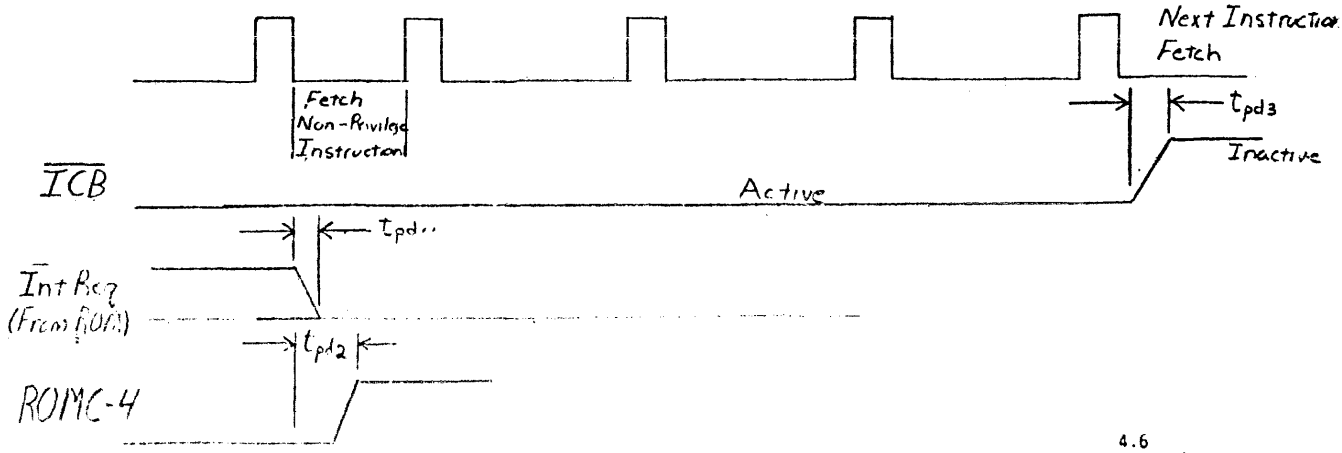
PARAMETER	MIN.	TYP.	MAX.	UNITS
t	0.5		5	uS
t_{d1}	0		200	nS
t_{d2}	0		200	nS
t_{d3}	0		400	nS
t_{SHORT}	2		20	uS
t_{LONG}	3		30	uS
t_{WR}	0.5		5	uS

Interrupt Control Bit Timing

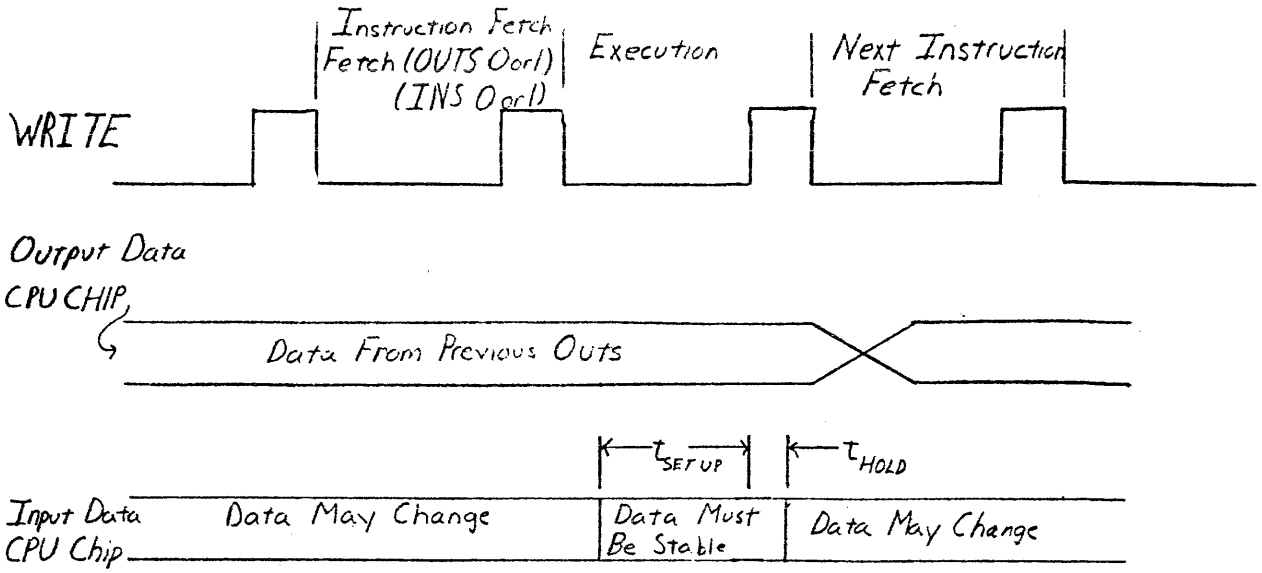




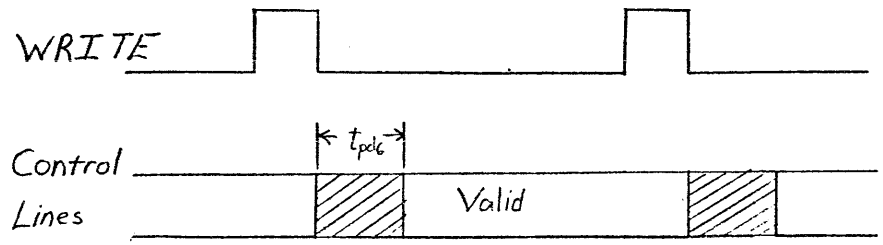
Interrupt Timing



CPU I/O Timing



Control Line Timing



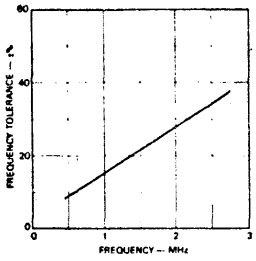
PARAMETER	MIN.	TYP.	MAX.	UNITS
t_{pd1}			450	nS
t_{pd2}			815	nS
t_{pd3}			310	nS
t_{pd4}			585	nS
t_{pd5}			600	nS
t_{pd6}			710	nS
t_{setup}	800			nS
t_{hold}	160			nS

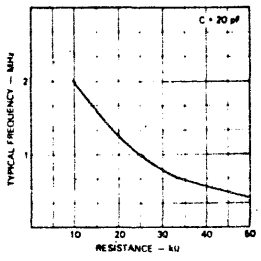
Note 2. Timing delays are referenced from valid input to valid output. This includes input and output rise and fall times.

Note 3. Maximum Capacitive Loading for 2 MHz system;
- 50 pf on $\overline{ICB}$ pin.
- 100 pf on all other pins

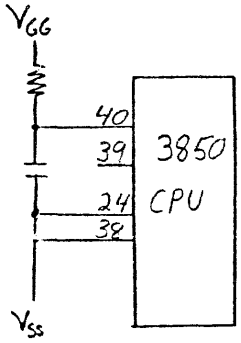
Clock Considerations

RC Mode

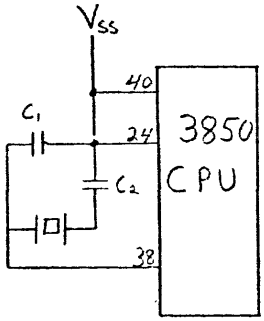




RC Oscillator

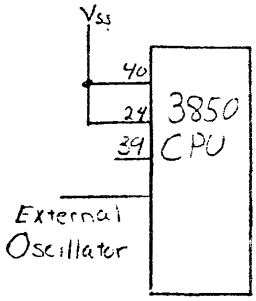


Crystal Oscillator



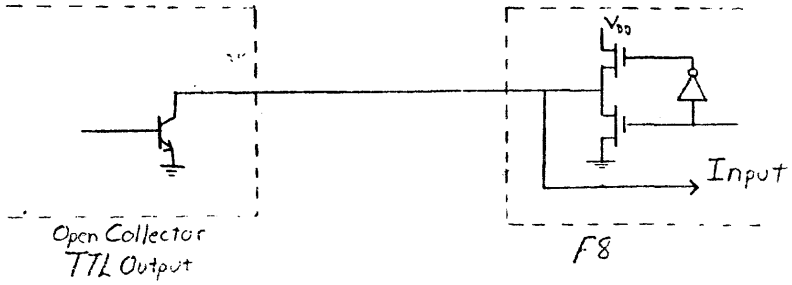
$C_1 = C_2 = 10pf$

External Oscillator

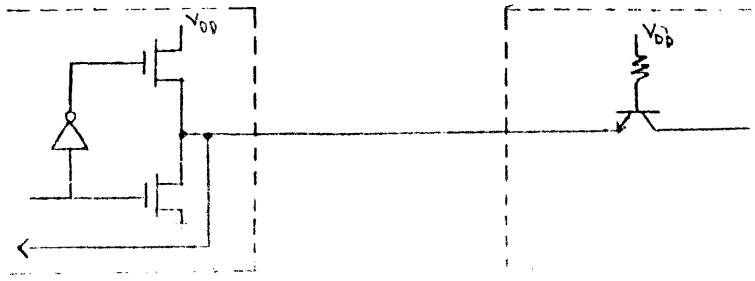


I/O Configurations

TTL To F8



F8 To TTL



3851

F8 ROM
FAIRCHILD ISOPLANAR MOS INTEGRATED CIRCUIT

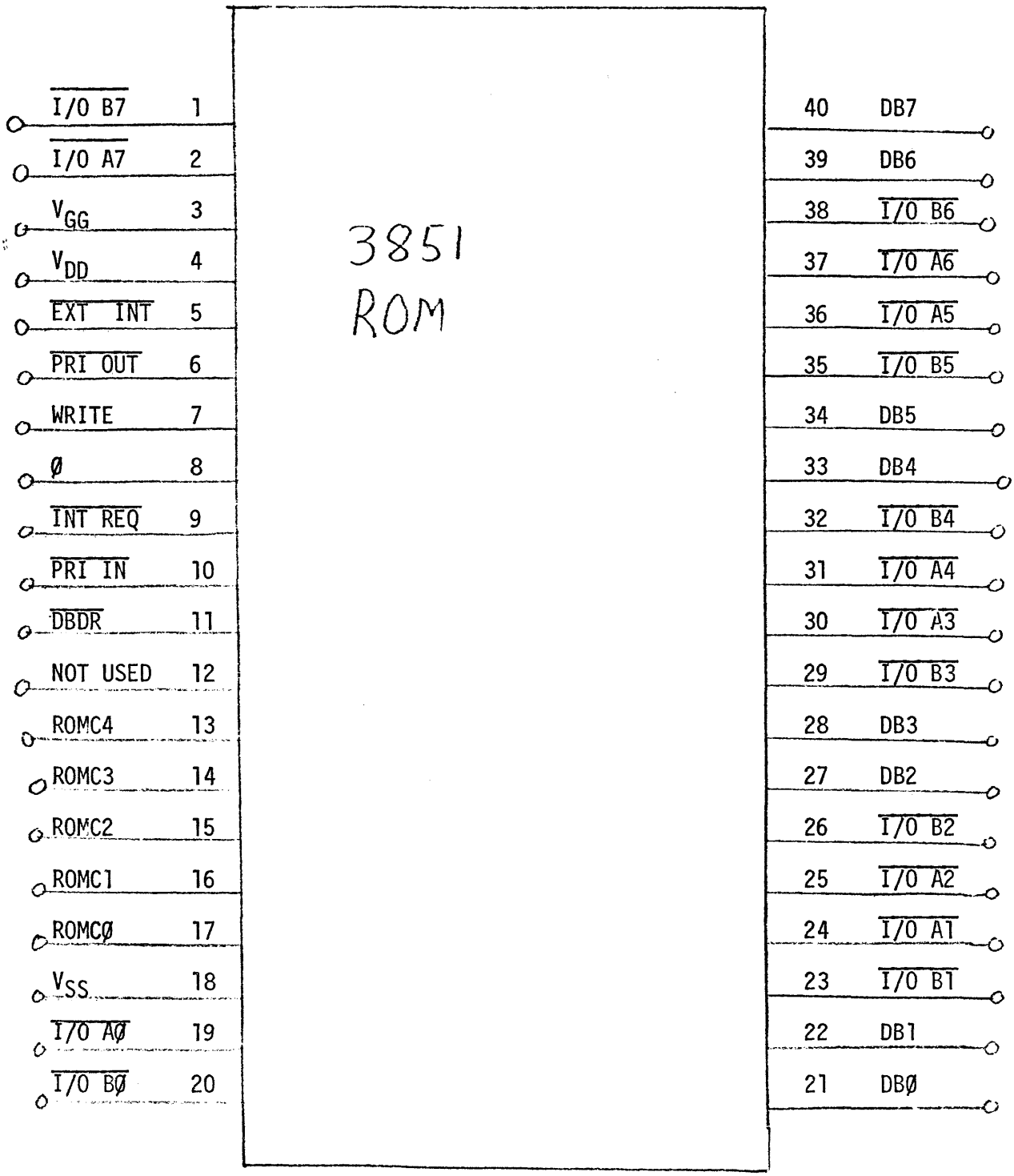
GENERAL DESCRIPTION: The 3851 provides 1K bytes ROM storage for an F8 microprocessor system. Manufactured using n channel isoplanar MOS technology, the 3851 requires two voltage supplies, +5 volts and +12 volts. The F8 ROM receives control signals from the F8 CPU; an 8-bit wide bidirectional data bus transfers data words between the CPU and the ROM. The I/O ports may be selected from one of three mask options for compatibility with external signals.

FEATURES:

- o 1024 8-bit ROM Storage
- o 16 Bit Program Counter
- o 16 Bit Stack Register
- o 16 Bit Data Counter
- o Interrupt Control Circuitry
- o Programmable Timer
- o Two, 8-bit I/O Ports
- o +5 volt and +12 volt power supply
- o Incrementer/Adder
- o Low Power Dissipation - Typically less than 300 mW

PIN NAMES:

Pin Names		Type
$\overline{\text{I/O A0}}$ - $\overline{\text{I/O A7}}$	I/O Port A	Input/Output
$\overline{\text{I/O B0}}$ - $\overline{\text{I/O B7}}$	I/O Port B	Input/Output
DB0 - DB7	Data Bus	Bidirectional
ROMC0 - ROMC4	Control Lines	Input
Ø, WRITE	Clock Inputs	Input
$\overline{\text{EXT INT}}$	External Interrupt	Input
$\overline{\text{PRI IN}}$	Priority In	Input
$\overline{\text{PRI OUT}}$	Priority Out	Output
$\overline{\text{INT REQ}}$	Interrupt Request	Output
$\overline{\text{DBDR}}$	Data Bus Driver	Output
V_{DD}, V_{SS}, V_{GG}	Power	


3851
ROM
$\overline{\text{I/O B7}}$ 1
$\overline{\text{I/O A7}}$ 2
V_{GG} 3
V_{DD} 4
$\overline{\text{EXT INT}}$ 5
$\overline{\text{PRI OUT}}$ 6
WRITE 7
Ø 8
$\overline{\text{INT REQ}}$ 9
$\overline{\text{PRI IN}}$ 10
$\overline{\text{DBDR}}$ 11
NOT USED 12
ROMC4 13
ROMC3 14
ROMC2 15
ROMC1 16
ROMCØ 17
V_{SS} 18
$\overline{\text{I/O AØ}}$ 19
$\overline{\text{I/O BØ}}$ 20
40 DB7
39 DB6
38 $\overline{\text{I/O B6}}$
37 $\overline{\text{I/O A6}}$
36 $\overline{\text{I/O A5}}$
35 $\overline{\text{I/O B5}}$
34 DB5
33 DB4
32 $\overline{\text{I/O B4}}$
31 $\overline{\text{I/O A4}}$
30 $\overline{\text{I/O A3}}$
29 $\overline{\text{I/O B3}}$
28 DB3
27 DB2
26 $\overline{\text{I/O B2}}$
25 $\overline{\text{I/O A2}}$
24 $\overline{\text{I/O A1}}$
23 $\overline{\text{I/O B1}}$
22 DB1
21 DBØ

ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS (Above which useful life may be impaired)

V_{GG}	+15 V to -0.3 V
V_{DD}	+ 7 V to -0.3 V
All Inputs and Outputs	+15 V to -0.3 V
Storage Temperature	-55°C to +150°C
Operating Temperature	0°C to + 70°C

Note: All Voltages With Respect to V_{SS}

DC CHARACTERISTICS: V_{SS} = 0.0 V, V_{DD} = +5 V ±5%, V_{GG} = +12 V ±5%, T_A = 0°C to +70°C (Note 1)

SUPPLY CURRENTS

SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
I_{DD}			30	70	mA	f = 2MHz
I_{GG}			10	18	mA	f = 2MHz

EXTERNAL INTERRUPT

V_{IL}	Input Voltage LOW	V_{SS}		1.2	V	
V_{IH}	Input Voltage HIGH	3.5		V_{DD}	V	
I_{IL}	Input LOW Current			.5	mA	$V_{IN} = V_{SS}$

CONTROL LINES AND PRIORITY IN

V_{IL}	Input Voltage LOW	V_{SS}		0.8	V	
V_{IH}	Input Voltage HIGH	3.5		V_{DD}	V	
I_L	Input Leakage Current			1	μA	V_{IN} = 6V

PRIORITY OUT

V_{OL}	Output Voltage LOW	V_{SS}		0.4		I_{SINK} = 100 μA
V_{OH}	Output Voltage HIGH	3.9		V_{DD}		I_{SOURCE} = 100 μA

DATA BUS

SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
V_{IH}	Input Voltage HIGH	3.5		V_{DD}	V	
V_{IL}	Input Voltage LOW	V_{SS}		0.8	V	
V_{OH}	Output Voltage HIGH	3.9		V_{DD}	V	I_{SOURCE} = 100uA
V_{OL}	Output Voltage LOW	V_{SS}		.4	V	I_{SINK} = 100 uA
I_L	Leakage Current			1	uA	V_{IN} = 6V; Output in 3-state mode

INTERRUPT REQUEST

V_{OL}	Output Voltage LOW	V_{SS}		.4	V	I_{SINK} = 1 mA
V_{OH}	Output Voltage HIGH	3.9		V_{DD}	V	Open drain output Pullup resistor on CPU

DATA BUS DRIVER

V_{OL}	Output Voltage LOW	V_{SS}		.4	V	I_{SINK} = 2.5 mA
V_{OH}	Output Voltage HIGH	3.9		V_{DD}	V	External 1.8K Pullup to V_{DD}

WRITE and ϕ

V_{IH}	Input Voltage HIGH	4.0		V_{DD}	V	
V_{IL}	Input Voltage LOW	V_{SS}		.8	V	
I_L	Input Leakage Current			1	uA	V_{IN} = 6V

NOTE 1 Junction Leakage Current, all pins:

1.0 uA max. when V_{IN} = 6V, V_{SS} = V_{DD} = V_{GG}

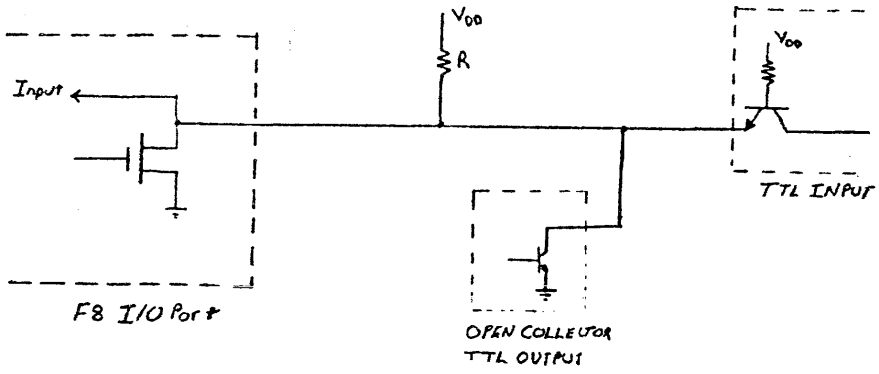
I/O PORT OPTION A

OPEN DRAIN

SUMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
V_{OH}	Output Voltage HIGH	-		V_{gg}		$I_{Leakage}$ @ $V_{OH\ max}$ = 1 μA max
V_{OL}	Output Voltage LOW	V_{ss}		.4	V	I_{SINC} = 2 mA
T_r	Rise Time					Dependent on External Pull-up Resistor
T_f	Fall Time			140	nS	C_L = 100 pf
V_{IH}	Input Voltage HIGH	2.9		V_{DD}	V	
V_{IL}	Input Voltage LOW	V_{SS}		.8	V	
I_{IH}	Input Current HIGH			1.0	uA	

OPEN DRAIN CONFIGURATION

BIDIRECTIONAL APPLICATION

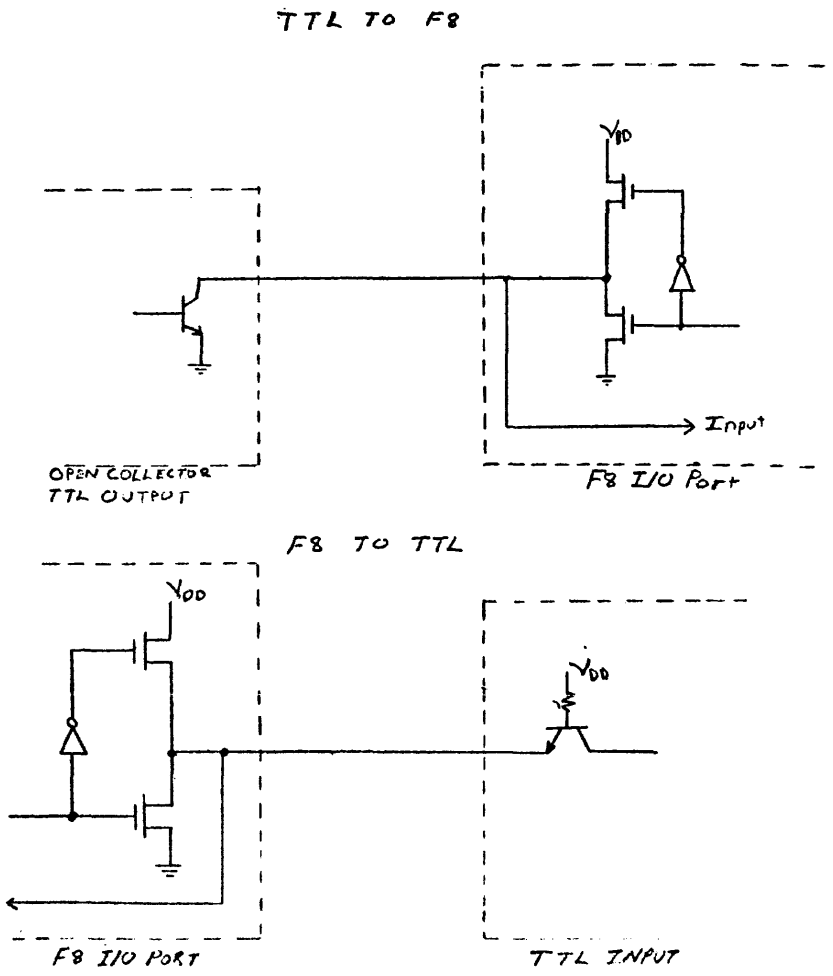


I/O PORT OPTION B

STANDARD PULL UP

SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
V_{OH}	Output Voltage HIGH	3.5		V_{DD}	V	I_{SOURCE} = 1 uA
V_{OH}	Output Voltage HIGH	2.9		V_{DD}	V	I_{SOURCE} = 100 uA
V_{OL}	Output Voltage LOW	V_{SS}		.4	V	I_{SINK} = 2 mA
T_r	Rise Time			180	nS	C_L = 100pf
T_f	Fall Time			140	nS	C_L = 100pf
V_{IH}	Input Voltage HIGH	2.9		V_{DD}	V	Internal Pullup to V_{DD} Provides TTL Compatability
V_{IL}	Input Voltage LOW	V_{SS}		.8	V	
I_{IL}	Input Current LOW			1.2	mA	V_{IN} = V_{SS}

STANDARD PULL UP CONFIGURATION



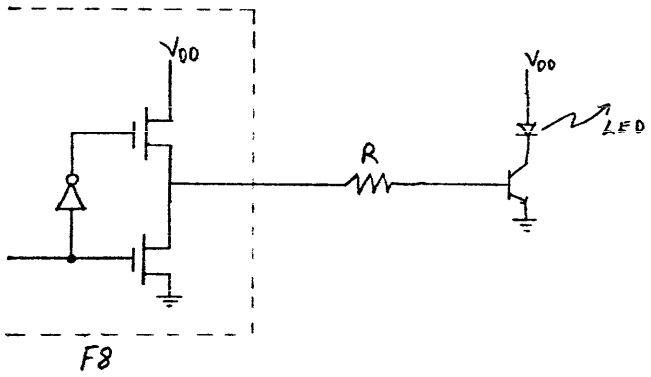
I/O PORT OPTION C

DRIVER PULL UP

SYMBOL	CHARACTERISTICS	MIN.	TYP.	MAX.	UNITS	TEST CONDITIONS
V_{OH}	Output Voltage HIGH	V_{DD} -1		V_{DD}	V	I_{SOURCE} = 1 mA
V_{OL}	Output Voltage LOW	V_{SS}		.4	V	I_{SINK} = 2 mA
T_r	Rise Time			120	nS	C_L = 100 pf
T_f	Fall Time			140	nS	C_L = 100 pf

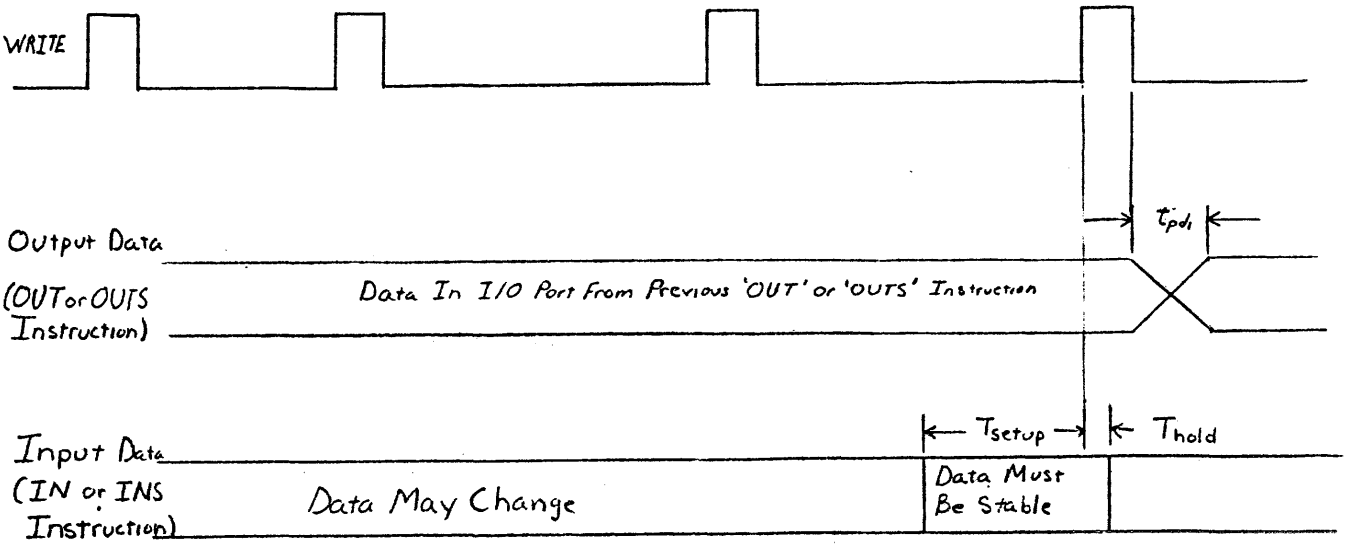
DRIVER PULL UP CONFIGURATION

OUTPUT ONLY

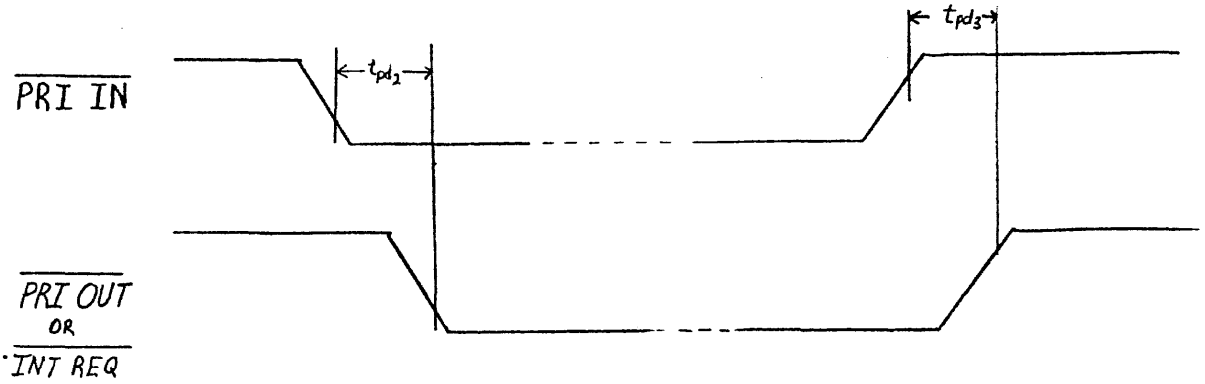


F8

ROM I/O TIMING



INTERRUPT TIMING (Note 2)

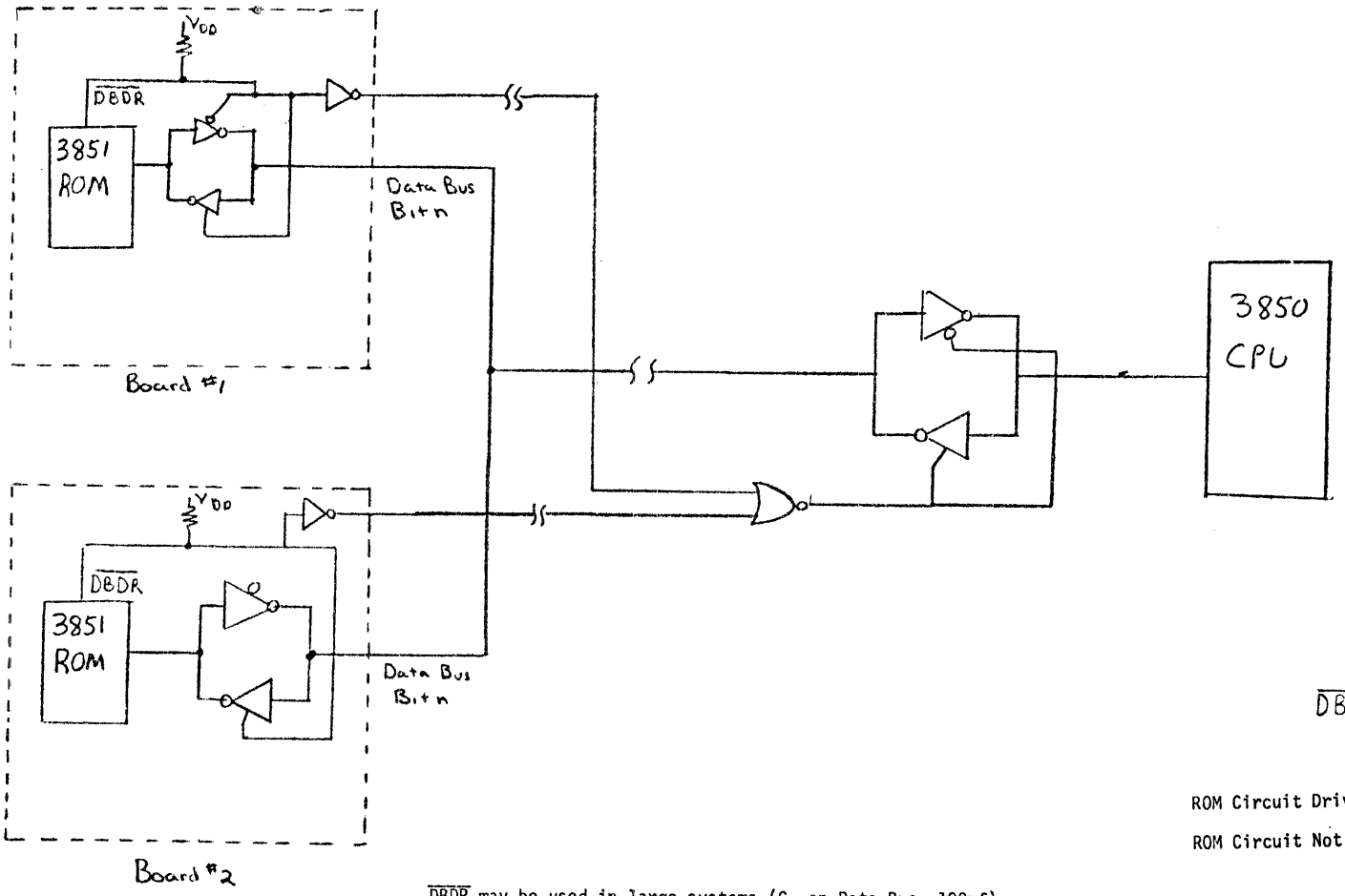


Parameter	Min.	Typ.	Max.	Units
t_{pd_1}	0		600	ns
t_{pd_2}			270	ns
t_{pd_3}			360	ns
T_{hold}	160			ns
T_{setup}	1.3			μs

Note 2. $\overline{\text{PRI OUT}}$; $C_L = 50pf$ max

$\overline{\text{INT REQ}}$; $C_L = 100pf$ max

Use of $\overline{DBDR}$ Data Bus Driver in a Large System



$\overline{DBDR}$ Levels

	$\overline{DBDR}$
ROM Circuit Driving Data Bus	V_{SS}
ROM Circuit Not Driving Data Bus	V_{DD}

$\overline{DBDR}$ may be used in large systems (C_L on Data Bus 100pf) to control Data Bus Buffers. It is not required for systems where C_L 100pf on Data Bus Lines.

5.0 MACHINE INSTRUCTIONS

5.1 GENERAL

This chapter describes each of the process instructions used when programming the F8. The instructions are given in the *ASSEMBLY LANGUAGE* format since this is the most basic form of the instructions usually used when writing a program. The description for each instruction also gives the hexadecimal equivalent called the *MACHINE LANGUAGE* format for convenient reference when debugging programs.

Formats for writing the processor instructions are first explained in Sections 5.2 and 5.3. Section 5.2 describes the format used for instructions coded in machine language format. The assembly language format for instructions is described in Section 5.3.

A symbolic notation is included in the description of each instruction. This permits the user who is familiar with the instructions to quickly review the functions performed.

The description of each instruction also indicates how the Status Register is affected by the instruction. Refer to Section 3.3 for an explanation of the Status Register.

The instructions are divided into functional groupings. Table 5.1 lists the instruction groups and the section which contains the detailed descriptions of each.

INSTRUCTION GROUP	FUNCTION	SECTION
Accumulator	Describes instructions that only affect the contents of the Accumulator (ACC)	5.5
Status Register	Describes instructions in which data moves between the Status Register and a Scratchpad Register	5.6
Indirect Scratchpad	Describes instructions to load the Indirect Scratchpad Address Register (ISAR) or load the Accumulator from ISAR	5.7
Scratchpad Register	Describes instructions in which data moves between a designated Scratchpad Register and the Accumulator or the ALU	5.8
Data Counter Reference	Describes instructions in which data moves between the Data Counter and the Accumulator or designated Scratchpad Registers	5.9
Memory Reference	Describes instructions in which data moves between main memory and the ALU or Accumulator	5.10
Program Counter	Describes all instructions which use or modify the contents of the program counter (PC_0) or the Stack Register (PC_1) except branch instructions. Includes call to subroutine instructions.	5.11
Branch	Describes all Conditional and Unconditional Branch instructions	5.12
Input/Output	Describes the four Input/Output instructions	5.13
No Operation	Describes the No-Op instruction	5.14
Interrupt Control	Describes the master interrupt control instructions	5.15
Condensed Listing	Condensed listing of all instructions with the function stated in symbolic notation	5.16

TABLE 5.1

5.2 MACHINE LANGUAGE FORMATS

All instructions are actually stored as binary numbers in the memory area assigned to store the programs. This binary representation of instructions is known as *MACHINE LANGUAGE*. All binary numbers may be grouped in 4 bit binary units called *HEXADECIMAL DIGITS*. Each hexadecimal digit is assigned a single symbol as shown in Table 5.2

Decimal Symbol	Hexadecimal Symbol	Binary Code MSB LSB
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
NA	A	1010
NA	B	1011
NA	C	1100
NA	D	1101
NA	E	1110
NA	F	1111

TABLE 5.2 NUMBER REPRESENTATIONS

Expressing binary numbers in hexadecimal notation simplifies interpretation of the binary value and generally simplifies the writing and reading of binary numbers.

For example; the binary number:

```
10010110
```

can be divided into 2 fields

```
1001 0110
```

which is assigned a hexadecimal notation of:

```
9 6
```

Through the rest of this manual, hexadecimal notation will be used, except where clarity can be maintained by retaining the binary form.

All instructions are encoded in multiples of eight bit units called *BYTES*. An instruction may be a one byte, two byte or three byte instruction depending on the function to be performed. Table 5.3 shows the formats for the one, two, and three byte instructions.

INSTRUCTION LENGTH	FORMAT (HEXADECIMAL) MEMORY LOCATION*			NUMBER OF BITS
	P	P + 1	P + 2	
1 BYTE	MSB LSB 1 A			8
2 BYTE	MSB LSB 2 4	MSB LSB F F		16
3 BYTE	MSB LSB 2 5	MSB LSB A A	MSB LSB 0 0	24

* P is the address location of the instruction.

TABLE 5.3 ORGANIZATION OF INSTRUCTIONS

5.2.1 SCRATCHPAD MEMORY

The scratchpad memory consists of sixty-four 8-bit bytes of RAM. Scratchpad memory instructions set up data paths between scratchpad and the accumulator. Seven bytes of scratchpad have been assigned special functions with specific instructions to link these bytes to other registers. These scratchpad locations are:

Register	Function	
9	J Register	linked to status register
10	HU H Register	linked to data counter
11	HL H Register	
12	KU K Register	linked to program counter
13	KL K Register	
14	QU Q Register	linked to stack register to the data counter
15	QL Q Register	

The first 12 scratchpad registers are directly addressable while the entire 64 bytes may be indirectly addressed via the 6-bit Indirect Scratchpad Address Register (ISAR). For example, the instruction LR A,5 loads the accumulator with the contents of scratchpad register 5. The directly addressed scratchpad location (5) is contained in the one byte instruction. When using the ISAR to indirectly address a scratchpad location, one of three addressing modes may be used. In all instances, the ISAR supplies the register address. The three modes are:

S - The ISAR contents are unchanged

I - The ISAR is incremented by one after execution

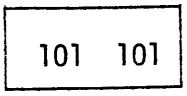
D - The ISAR is decremented by one after execution

For instance, if the ISAR contains

101	100

ISAR

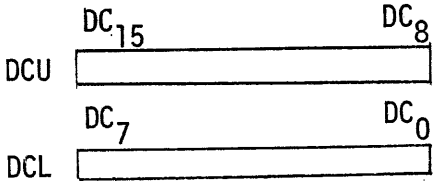
the instruction LR A, I will load the accumulator with the contents of scratchpad register 54_8 and increment the ISAR. After execution of this instruction, the ISAR will contain:



When incremented and decremented, the three least significant bits of the ISAR form a modulo 8 counter; the three most significant bits of ISAR remain unmodified when addressing with the I or D format. A group of instructions loads the ISAR.

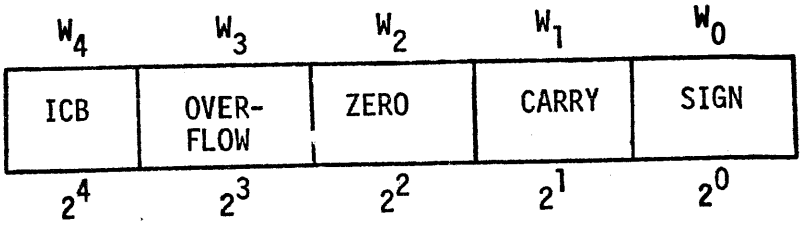
5.2.2 DATA COUNTER

The F8 microprocessor contains a 16-bit data counter to indirectly address the 64K bytes of memory. Thus, all memory reference instructions need no operand. An instruction, LM, will automatically transfer the contents of the byte specified by the data counter from memory to the accumulator. At the end of all memory reference instructions, the data counter is automatically incremented. The data counter instructions modify and transfer the contents of the data counter.



5.2.3 STATUS REGISTER

The W Register in the CPU contains the following status bits.



The ICB is the *INTERRUPT CONTROL BIT*. Instructions exist to set and reset this bit. The four remaining bits are dependent on the results of ALU operations. Boolean equations for these bits are:

$$\text{OVERFLOW} = \text{CARRY}_7 \oplus \text{CARRY}_6$$

$$\text{ZERO} = \overline{\text{ALU}_7} \wedge \overline{\text{ALU}_6} \wedge \overline{\text{ALU}_5} \wedge \overline{\text{ALU}_4} \wedge \overline{\text{ALU}_3} \wedge \overline{\text{ALU}_2} \wedge \overline{\text{ALU}_1} \wedge \overline{\text{ALU}_0}$$

$$\text{CARRY} = \text{CARRY}_7$$

$$\text{SIGN} = \overline{\text{ALU}_7}$$

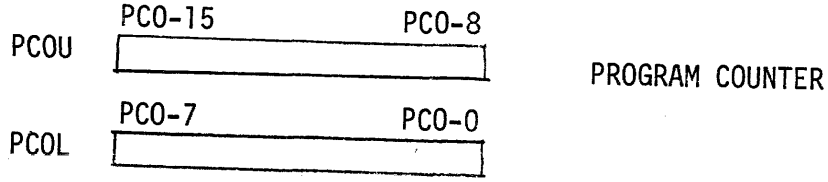
Where ALU_n are the eight bits resulting from ALU operations CARRY_7 is the carry bit from adding the seventh stage. CARRY_6 is the carry generated from adding the sixth stage.

A set of instructions transfer data between the W register and r9 of the scratchpad. These instructions are useful for storing and retrieving the CPU state during interrupts and subroutine jumps.

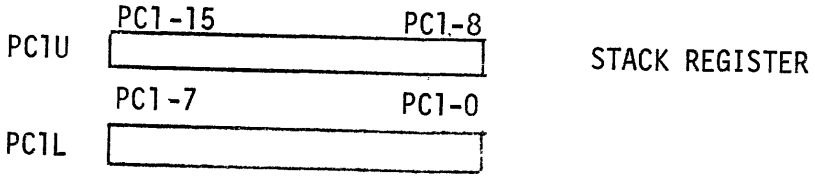
5.2.4 PROGRAM COUNTER AND STACK REGISTER

The *PROGRAM COUNTER* contains the address of the next instruction to be executed. Linked directly to the program counter is the *STACK REGISTER*; the stack register is loaded with the contents of the program counter during interrupts and subroutine jumps.

The program counter (PC0) contains the location of the next instruction to be executed.



Program counter instructions load PC0 with the contents of the scratchpad registers or directly from an instruction. The contents of the program counter may be transferred to and from the stack register (PC1).



The stack register is useful for saving and restoring the program counter during interrupts and subroutine jumps. It is linked to scratchpad registers K and Q by special instructions.

5.3 ASSEMBLY LANGUAGE FORMATS

The scope of this section is to familiarize the reader with the format of the assembly language form of the F8 processor instructions. The rules for programming in assembly language are described in greater detail in Chapter 6.0, CROSS ASSEMBLER. The descriptions of all the instructions in the following paragraphs will be stated in terms of the assembly language format.

There are two parts to an instruction written in assembly language form. These are:

- o OPERATION CODE, or OP CODE
- o OPERAND(S)

The two parts of an instruction are written in assembly language in the following format.

OP CODE	OPERANDS
LR	A,KU

The *OP CODE* specifies what function is to be performed. For example: the OP CODE may specify that an addition is to be performed; or that a register is to be loaded; or than an input port is to be read and the data present at the input stored in the accumulator. OP CODES are always written with two to four alphabetic letters only.

The second part of an assembly language instruction is called the *OPERAND(S)*. (Some instructions contain two operands). OPERANDS specify either (1) where the data that is to be used in the execution of the instruction is to be found (OPERAND *ADDRESS);* OR (2) contain the data content directly *(IMMEDIATE OPERAND)*. An F8 instruction will contain either one, two,or no operands depending on the function to be performed.

For example: A load register instruction (LR A,QL) contains 2 operands. The first operand specifies that one byte of data is to be loaded into the accumulator, and the second operand specifies that the byte is to be obtained from Scratchpad Register #15 (QL).

The shift right instructions (SR 4) contains one operand which specifies that the data stored in the accumulator, is to be shifted right toward the LSB four times.

5.4 SYMBOLIC NOMENCLATURE

A functional notation is used in the descriptions of all instructions to demonstrate what takes place when the instruction is *EXECUTED* or performed. Use of this notation can replace the longer verbal descriptions of the function of the instructions.

Throughout this chapter, symbols and variables are used to define instructions.

Table 5.4 lists each of the symbols used and their definitions.

SYMBOL	DEFINITION
←	Is Replaced By. The entire expression to the rear of the arrow replaces that at the head of the arrow.
( )	The Contents Of.
+	Addition. The symbol is used for both binary and decimal addition.
Λ	Logical And.
V	Logical Or.
⊕	Logical Exclusive Or.
-	Subtract.
" "	The Value contained in quotes.

Table 5.4. Symbols Used In Descriptions Of Instructions And The Definition.

Definitions of operand variables are given in Table 5.5

Operand Variable*	DEFINITION
i	Immediate Value
rn	Scratchpad Register n; n is between 0 to 63
x	Source Register
Y	Destination Register
a	Address Digit
t	Test Condition; used in conditional Branch instructions
d	Digit Value; used in Table 5.6 format only. Refer to Table 5.2 for acceptable digits.

Table 5.5 Definition of Variables

Table 5.6 lists the formats of constants which are used.

The correct format for expressing each constant is shown in the column labeled OPERAND CONSTANT. If no format is given, a decimal value will be assumed.

Operand Constant *	Definition	Alternate Form of Operand
H'd'	Hexadecimal Digit; d may be one of the digits 0 thru 9, A,B,C,D,E,F	
D'd'	Decimal Digit; d may be one of the digits 0 thru 9	d*
O'd'	Octal Digit; d may be one of the digits 0 thru 7	
C'd'	Character; 8 bits which are one of the ASCII characters. Refer to Appendix	
B'd'	Binary Digit; d may be either "1" or "0"	
T'd'	Timer Counts	

Table 5.6 Definition & Formats for Operand Constants

*Each character represents 4 binary bits. More than one digit "d" may be used to represent the full value of the operand as required.

Example: A 16 bit binary number 0110 0011 0100 1001 may be represented in hexadecimal notation as:

H'6349'

5.5 ACCUMULATOR GROUP INSTRUCTIONS

Accumulator Group Instructions are instructions that affect only the contents of the accumulator. Data moves from the Accumulator to the ALU for modification and back to the accumulator for restorage.

Operands used during the execution of these instructions originate either in the Accumulator or are constants called *IMMEDIATE VALUES* contained in the instruction word directly.

INSTRUCTIONS WITH OPERANDS ORIGINATING IN THE ACCUMULATOR:

OP. CODE	OPERAND	MACHINE FORMAT	Description	Bytes	Cycles*
SR	1 or 4	1 S	Shift the contents of the accumulator right 1 or 4 binary bits. The right most bit in the accumulator (ACC_0) is not saved. The most significant bit of the accumulator(ACC_7) is filled with zero.	1	1

Value of S	Number of Bits Shifted
2	1
4	4

The Status bits are modified by execution of this instruction*

OVF	ZERO	CARRY	SIGN
0	1/0	0	1

* A 1/0 box of the status register implies that the status bit may become either a 1 or a 0 as a result of the instruction execution.

Shift Right Example:

	Accumulator	Instruction	Status Register
Before	10110101	SR 4	OVF ZERO CRY SIGN
After	00001011		0 0 0 1

* Throughout this chapter, a cycle equals two microseconds for an F8 with a 2MHz system clock.

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
SL	1 or 4	1 S	Shift the contents of the accumulator left "1" or "4" binary bits. The left most bit in the accumulator, ACC_7, is not saved. The least significant bit (ACC_0) is filled with 0.	1	1

Value of S	Number of Bits Shifted
3	1
5	4

The status bits are set by execution of this instruction as follows:

OVF	ZERO	CARRY	SIGN
0	1/0	0	1/0

	Accumulator	Instruction	Status Register
Before	11110010	SL 1	OVF ZERO CRY SIGN
After	11100100		0 0 0 0

COM	---	1 8	COMPLEMENT ACC ← (ACC) ⊕ H 'FF' The accumulator is loaded with the binary "1"s complement of the original contents of the accumulator.	1	1

The status bits are set by execution of this instruction as follows:

OVF	ZERO	CARRY	SIGN
0	1/0	0	1/0

Complement Example:

	Accumulator	Instruction	Status Register
Before	1000 1011	COM	OVF ZERO CRY SIGN
After	0111 0100		0 0 0 1

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
LNK	---	1 9	LINK CARRY TO THE ACCUMULATOR ACC ← (ACC) + CB A binary addition of the Carry Bit (CB) to the current accumulator contents is performed. This instruction facilitates programming multiple precision arithmetic by allowing carry from a lowered valued byte to be added into the byte of next higher power. The status bits are set by execution of this instruction as follows: OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0	1	1
INC	---	1 F	INCREMENT ACCUMULATOR ACC ←(ACC) +1 The accumulator value is increased by one binary count. The status bits are set by execution of this instruction as follows: OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0	1	1
INSTRUCTIONS WITH IMMEDIATE OPERANDS CONTAINED IN THE INSTRUCTION					
LIS	i	7 i	LOAD IMMEDIATE SHORT ACC ← H 'i' The hexadecimal digit 'i' is loaded into the four least significant bits of the accumulator, ACC_0 through ACC_3. The most significant four bits, ACC_4 through ACC_7 are set to "0".	1	1

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			No status bits are modified by execution of this instruction.		
CLR	---	7 0	CLEAR ACCUMULATOR ACC ← H'00' All bits of the accumulator are reset. No status bits are modified by execution of this instruction.	1	1
LI	ii	2 0 i i	LOAD IMMEDIATE ACC ← H 'ii' The 8-bit binary value (hexadecimal) value ii contained in the second byte of the instruction is placed in the accumulator. No status bits are modified by execution of this instruction.	2	2.5
NI	ii	2 1 i i	AND IMMEDIATE ACC ← (ACC) Λ H'ii' The accumulator contents are replaced by the present contents of the accumulator "AND" ed with the 8-bit binary value (hexadecimal ii) contained in the second byte of the instruction. The status bits are modified by execution of this instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0	2	2.5
OI	ii	2 2 i i	OR IMMEDIATE ACC ← (ACC) V H'ii' The contents of the accumulator are replaced by the present contents of the accumulator "OR"ed with the 8-bit binary value	2	2.5

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			(hexadecimal ii) contained in the second byte of the instruction. The status bits are set by execution of this instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0		
XI	ii	2 3 i i	SELECTIVE COMPLEMENT (EXCLUSIVE "OR") ACC ← (ACC) ⊕ H 'ii' The contents of the accumulator replaced with the present contents of the accumulator exclusive "OR"ed with the 8-bit binary value (hexadecimal ii) contained in the second byte of the instruction. The instruction selectively complements those bits in the accumulator which correspond to the binary "1"s in the 8-bit binary value (hexadecimal ii) contained in the second byte of the instruction. The status bits are modified by execution of this instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0	2	2.5

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES

Example: Load Immediate Short

	Accumulator	Instruction	Status Register
Before	1010 1100	LIS 5	unchanged
After	0000 0101 (5)		

Example: Exclusive OR Immediate

	Accumulator	Instruction	Status Register			
			OVF	ZERO	CRY	SIGN
Before	1011 0010	XI H'A3'	0	0	0	0
After	0001 0001		0	0	0	1

H'A3' = 1010 0011
A 3

```
  1011  0010
⊕ 1010  0011
  0001  0001
```

AI	ii	2 4 i i	ADD IMMEDIATE ACC ← (ACC) + H 'ii' This is a binary addition of 8 bits. The accumulator contents are replaced by the result formed by the binary addition of the 8 bit binary value (hexadecimal ii) contained in the second byte of this instruction to the current contents of the accumulator. The status bits are modified as follows: OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0	2	2.5

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
CI	ii	2 5 i i	COMPARE IMMEDIATE: H'ii' + ($\overline{ACC}$) + 1 The status bits of the "W" register are determined by the comparison of the accumulator with the 8 bit binary value (hexadecimal 'ii') contained in the second byte of the COMPARE IMMEDIATE INSTRUCTION. This instruction is useful when screening fields of bytes to detect the presence of a specific binary number. The accumulator register is not altered with the results of the compare. All status bits are modified: OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0	2	2.5

5.6 STATUS REGISTER INSTRUCTIONS

Instructions in which data flows between the Status Register and the Scratchpad Register are called STATUS REGISTER INSTRUCTIONS. The J Register in the scratchpad is used for storing the STATUS BITS.

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
LR	W,J	1 D	LOAD W FROM r9 W ← (r9) The contents of scratchpad register (r9) are loaded to the status register (W). The five least significant bits of r9 have the following bit assignments. W_4 W_3 W_2 W_1 W_0 ICB | OVF | ZERO | CARRY | SIGN r9 2^4 2^3 2^2 2^1 2^0 An interrupt will not be accepted by the CPU at the completion of this instruction.	1	2
LR	J, W	1 E	LOAD r9 FROM W r9 ← (W) The contents of the status register (W) are stored in scratchpad register (r9). The original contents of W are not modified. The three most significant bits of r9 are set to "0".	1	1

5.7 INDIRECT SCRATCHPAD ADDRESS REGISTER INSTRUCTIONS

INDIRECT SCRATCHPAD ADDRESS REGISTER instructions are used either to move data between the Indirect Scratchpad Register (ISAR) and the Accumulator; or to load the ISAR with an immediate address contained in the instruction.

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
LR	A,IS	0 A	LOAD ACCUMULATOR FROM ISAR ACC ← (ISAR) The 6-bit content of ISAR is loaded into the least significant six bits of the Accumulator. Bits ACC_6 and ACC_7 of the Accumulator are reset to "0". The original contents of ISAR are not changed. Status bits are not modified during execution of this instruction.	1	1
LR	IS,A	0 B	LOAD ISAR FROM ACCUMULATOR ISAR ← (ACC) The least six significant bits of the Accumulator are transferred to the ISAR. Bits Acc_6 and Acc_7 of the accumulator are not used. The original contents of the accumulator are not changed. Status bits are not modified during execution of this instruction.	1	1
LISU	a	6 0 a *	LOAD "a" TO ISAR UPPER OCTAL DIGIT (ISAR U) ISAR U ← a The 3-bit octal digit "a" is placed in the three most significant bits of the ISAR. The three least significant bits are not altered. Address character "a" must be one of the octal digits 0, 1, 2, 3, 4, 5, 6, or 7 only. Status bits are not changed by the execution of this instruction. *Note that the 2^3 bit of the instruction word is binary "0".	1	1

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
LISL	a	0110 1 a *	LOAD "a" to ISAR LOWER OCTAL DIGIT (ISAR L) ISAR L ← a The 3-bit octal digit "a" is placed in the three least significant bits of the ISAR. The three most significant bits are not altered. Address character "a" must be one of the octal digits 0, 1, 2, 3, 4, 5, 6 or 7 only. Status bits are not modified by execution of this instruction. *Note that the 2^3 bit of the instruction word is a binary "1".	1	1

5.8 SCRATCHPAD REGISTER INSTRUCTIONS

Instructions in which the data flows principally between the scratchpad register and the accumulator or ALU are grouped together as Scratchpad Register Instructions. The scratchpad register is also linked by instructions to the program counter and data counter; those instructions will be treated in later sections.

OP. CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
LR	y, x	-	Load Register $y \leftarrow (x)$ The contents of register x (source register) are transferred to register y (destination register). The contents of the source register x are <u>not</u> altered. This is a general format for transfer of data between the accumulator register and a designated register in the scratchpad. The LR op code is the same for all load register instructions, however, the operands will specify the specific transfer to be performed. All load register instructions of the scratchpad group are one byte instructions and execute in one cycle. Table 5.7 shows the valid operands for the Load Register Instruction. Any load register instruction of the Scratchpad Register Instruction group <u>must use</u> the Accumulator as either a source or destination register. The status bits are not modified by the Load Register Instruction.		

OPERAND y	OPERAND x	MACHINE FORMAT	DESCRIPTION	
A	r*	4r	Load Accumulator from r#	A←(r#)
A	KU	00	Load Accumulator from Register 12	A←(r12)
A	KL	01	Load Accumulator from Register 13	A←(r13)
A	QU	02	Load Accumulator from Register 14	A←(r14)
A	QL	03	Load Accumulator from Register 15	A←(r15)
r*	A	5r	Load Register r from Accumulator	r←(A)
KU	A	04	Load Register 12 from Accumulator	r12←(A)
KL	A	05	Load Register 13 from Accumulator	r13←(A)
QU	A	06	Load Register 14 from Accumulator	r14←(A)
QL	A	07	Load Register 15 from Accumulator	r15←(A)

TABLE 5.7

OPERAND FORMATS FOR LOAD REGISTER INSTRUCTIONS OF THE SCRATCHPAD GROUP

*Note: r is the implied address if the scratchpad register is used in the instruction. It may designate either a register location directly or that the register to be addressed is pointed to by the contents of the indirect scratchpad address register. Table 5.8 specifies the interpretation given to r.

FORMATS FOR r	ADDRESS FUNCTION PERFORMED
0 thru 11	r = direct address of the scratchpad register (registers #0 thru #11 may be addressed directly)
S 12	The scratchpad register address is supplied by the indirect scratchpad register (ISAR). ISAR contents are unchanged.
I 13	The scratchpad register address is supplied by the indirect scratchpad address register (ISAR). ISAR is incremented after the instruction is executed.*
D 14	The scratchpad register address is supplied by the indirect scratchpad address register (ISAR). ISAR is decremented after the instruction is executed.*
15	Not assigned.

TABLE 5.8

FUNCTIONS SPECIFIED BY THE OPERAND ADDRESS DESIGNATOR, r.

*Note: The three least significant bits of the Indirect Scratchpad Address Register (ISAR) forms a modolo eight counter when it is incremented or decremented. The three most significant bits of the ISAR remain unmodified by these instructions. These can only be modified by executing the LISU Instruction or by the LR IS,A instruction (refer to the Indirect Scratchpad Address Register Instructions in section 5.7).

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
AS	r	C r	BINARY ADDITION: ACC←(ACC) + (r) The eight bits of the operand implied by address r are added to the contents of the accumulator. The addition is binary. status bits are modified with the result of the addition. OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0 Table 5.8 shows the formats for r.	1	1
ASD	r	D r	DECIMAL ADDITION ACC←(ACC) + (r) ; decimal values The eight bits of the operand implied by address r are added to the eight bits of the accumulator. The result is stored in the accumulator. The addition is performed on two binary coded decimal digits stored in the location specified by r and two binary coded decimal digits stored in the accumulator. These decimal digits are coded in BCD code. Two machine cycles are required to execute the instruction. The first cycle performs a binary addition. The second cycle adjusts the result to form BCD digits in the accumulator. The addition of decimal numbers is performed in two steps. First, execute a binary addition of H'66' (hexadecimal 66) to one of the two decimal number operands. Next, execute the ASD instruction on the two operands. The inter-digit carrys and decimal adjustment will be performed automatically.	1	2

Example: ADD Bytes A and B to form C
Let A = 34 stored in BCD Format in r4
B = 72 stored in BCD Format in r5

The following subroutine will add the two decimal numbers to form the results (C) in the accumulator.

carry	accumulator	content			
x*	0011	0100	LR	A,4	Load A to accumulator
0	1001	1010	AI	H'66'	Add 66
1	0000	0110	ASD	5	Add B

* x is a don't care condition, its value is a result of the previous operand.

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			The status bits are modified at the end of the first cycle and before the decimal adjustment is made in the second cycle. The Status Bits are set by execution of this instruction as follows: OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0		
NS	r	F r	LOGICAL AND ACC←(ACC) ∧ (r) The contents of the register implied by r are "AND"ed with the contents of the accumulator. The result is returned to the accumulator. The status bits are modified by the AND instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0	1	1

5.8 SCRATCHPAD REGISTER INSTRUCTIONS (Cont.)

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
XS	r	E r	EXCLUSIVE OR ACC←(ACC) ⊕ (r) The contents of the register implied by r are "EXCLUSIVE OR"ed with the contents of the accumulator and returned to the accumulator. All status bits are modified when the XS instruction is executed as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0	1	1
DS	r	3 r	DECREMENT The scratchpad register implied by r is decremented by one binary count. The actual operation is r ← (r) + H'FF' The status bits are modified when the DS instruction is executed as follows: OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0	1	1.5

	Scratchpad Register 27	ISAR	Accumulator	Instruction	Comment
Before	0001 1010	011 011	1011 0101		
After	0001 1010	011 010	0001 0000	NS 14	'AND' Accumulator with r27 and decrement ISAR

STATUS REGISTER	OVF	ZERO	CARRRY	SIGN
	0	0	0	1

5.9 DATA COUNTER INSTRUCTIONS

Instructions in which data flows principally between the Data Counter and the Scratchpad or Accumulator are grouped as *DATA COUNTER INSTRUCTIONS*. The Data Counter functions as an indirect address register for referencing memory locations with one byte instructions. The Data Counter instructions facilitate modifying the address of memory to be used in memory reference instructions.

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
LR	y, x		LOAD REGISTER y FROM REGISTER x y ← x Table 5.9 lists the allowable operands (y,x) for Data Counter instructions. The Data Counter (DC) must always be specified as the destination or source address (y or x). The other operand must always be either scratchpad locations Q or H as follows: Q = {r14} and {r15} H = {r10} and {r11} The LR instruction of the Data Counter group always transfers two bytes. The status bits are not modified when any DATA COUNTER LOAD REGISTER instruction is executed.	1	4

OPERAND y	OPERAND x	MACHINE FORMAT		DESCRIPTION
Q	DC	0	E	The contents of the data counter is loaded to locations Q in scratchpad r14 ← (DCU); r15 ← (DCL)
H	DC	1	1	The contents of the data counter is loaded to locations H in scratchpad r10 ← (DCU); r11 ← (DCL)
DC	Q	0	F	The contents of scratchpad locations Q is loaded to the Data Counter DCU ← (r14); DCL ← (r15)
DC	H	1	0	The contents of scratchpad locations H is loaded to the Data Counter DCU ← (r10); DCL ← (r11)

TABLE 5.9

OPERAND FORMATS FOR DATA COUNTER GROUP LOAD REGISTER INSTRUCTION

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
ADC	--	8	E	Add Accumulator to DC DC ← (DC) + (ACC) A displacement address to the present location specified by the contents of DC is formed by adding 8 binary bits of the Accumulator to DC. The value of of the Accumulator is treated as a 2's complement number. A displacement address of plus 127 or minus 128 memory locations relative to the present data counter address can be formed with this instruction.	1	2.5

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
				The status bits are not modified by the execution of the ADC instructions.		
DCI	iiii	2	A	LOAD DC IMMEDIATE	3	6
		i	i	DC ← H'iiii'		
		i	i	The 16 bit binary number contained in bytes 2 and 3 of the DCI instruction are loaded into the data counter. Instruction byte 2 goes to DCU (bits DC_8 thru DC_{15}) Instruction byte 3 goes to DCL (bits DC_0 thru DC_7) No status bits are altered during execution of this instruction.		
XDC	--	2	C	EXCHANGE DATA COUNTERS DC_0 DC_1	1	2
				This instruction exchanges the contents of DC_0 with DC_1 on the Memory Interface Circuit. It is only operative when the Memory Interface Circuit is included in the system configuration.		
				The status bits are not modified by execution of this instruction.		

5.10 MEMORY REFERENCE INSTRUCTIONS

Instructions in which the data flows principally between a designated one byte storage location contained anywhere in memory (not designated as scratchpad) and the Accumulator or ALU are grouped together as *Memory Reference instructions*. All memory reference instructions are one byte instructions. Any location within a total storage capacity of 65,536 bytes may be accessed by a memory reference instruction. The 16 bit Data Counter points to the memory location accessed in memory reference instructions. The Data Counter is incremented at the end of each memory reference instruction.

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLE
LM	--	1	6	LOAD ACCUMULATOR FROM MEMORY ACC ← ((DC)) The contents of the memory addressed by the data counter is loaded into the accumulator. The data counter is incremented after the byte is accessed from memory. The status bits are not modified by the execution of the LM instruction.	1	2.5
ST	--	1	7	STORE TO MEMORY DC ← (ACC) The contents of the Accumulator are transferred to the memory location addressed by the contents of the data counter. The Data Counter is incremented after the memory location is loaded. None of the status bits are modified by the execution of the ST instruction.	1	2.5
AM	--	8	8	ADD MEMORY TO ACCUMULATOR, BINARY ACC ← (ACC) + ((DC)) {Binary ADD} All 8 bits of the memory location accessed by the address contained in the Data Counter are added to the contents of the Accumulator. The results are stored in the Accumulator. The Data Counter is incremented after the memory location is accessed. The addition performed is binary.	1	2.5

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
				The status bits are modified by the execution of the AM instruction. OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0		
AMD	--	8	9	ADD MEMORY TO ACCUMULATOR: DECIMAL ACC ← (ACC) + ((DC)) {DECIMAL ADD} All 8 bits of the memory location designated by the address contained in the data counter are added to the contents of the Accumulator. The results are stored in the Accumulator, and adjusted to provide a decimal number of two digits in the accumulator at the end of the execution of the AMD instruction.* The Data Counter is incremented after the memory location indicated is accessed. The status bits are modified by the execution of the AMD instruction. OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0	1	2.5
NM	--	8	A	LOGICAL AND FROM MEMORY ACC ← (ACC) $\wedge$ ((DC)) All 8 bits of the memory location indicated by the contents of the Data Counter are "AND"ed with the contents of the Accumulator. The result is stored in the Accumulator. The Data Counter is incremented after the memory location is accessed.	1	2.5

(Refer to ASP instruction, page 5.25, for a more detailed explanation.)

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
				The status bits are modified by the execution of the NM instruction. OVF ZERO CARRY SIGN 0 1/0 0 1/0 The NM instruction may be used as a selective mask instruction. The mask word will be the byte stored in the memory location indicated by the contents of the Data Counter.		
OM	--	8	B	LOGICAL "OR" FROM MEMORY ACC ← (ACC) ∨ ((DC)) The 8 bits contained in the memory location indicated by the contents of the Data Counter are logically "OR"ed with the contents of the Accumulator. The results are stored in the Accumulator. The Data Counter is incremented after the memory location indicated is accessed. All status bits are modified by the execution of the OM instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0	1	2.5
XM	--	8	C	EXCLUSIVE OR FROM MEMORY ACC ← (ACC) ⊕ ((DC)) The 8 bits contained in the memory location indicated by the contents of the Data Counter are logically "EXCLUSIVE OR"ed with the contents of the Accumulator. The result is stored in the Accumulator. The Data Counter is incremented after the memory location indicated is accessed.	1	2.5

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			The status bits are modified by the execution of the XM instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0		
CM	--	8 D	COMPARE MEMORY TO ACCUMULATOR ((DC)) + $\overline{ACC}$ + 1 The 8 binary bits in the memory location indicated by the contents of the Data Counter are compared to the 8 bits contained in the Accumulator. The status bits in the W register are set by the results of the comparison. The contents of the memory location referenced and the contents of the Accumulator are not altered by execution of the CM instruction. The Data Counter is incremented after the memory location is accessed. The status bits are modified by executing the CM instruction. OVF ZERO CARRY SIGN 1/0 1/0 1/0 1/0	1	2.5

5.11 PROGRAM COUNTER INSTRUCTIONS

Instructions which cause data to move between any two or more of the counters or registers designated as the Program Counter (PC_0), the Stack Register (PC_1), or the Scratchpad Registers are grouped together as *PROGRAM COUNTER INSTRUCTIONS*. Branch instructions are specifically treated as a separate grouping in Section 5.12.

Three kinds of functions are performed by the PROGRAM COUNTER INSTRUCTIONS:

- Link either the Program Counter (PC_0) or the Stack Register (PC_1) to the Scratchpad. (refer to paragraph 5.11.1)

- o Call to Subroutine (refer to paragraph 5.11.2)
- o Return from Subroutine (refer to paragraph 5.11.3)

Instructions that link PC_0 and PC_1 to the scratchpad registers facilitate handling of multi-level interrupts under program control. These instructions permit the programmer to create an address stack of indefinite length in bulk Read/Write storage, when used in conjunction with the Load Register instructions contained in the SCRATCHPAD REGISTER instruction group (refer to section 5.8).

The Call to Subroutine instructions allows the programmer to branch directly to a subroutine and automatically save the address of the next instruction to be executed in the main program.

The Return from Subroutine or POP instruction is a return to a higher level operating program from an immediately lower level subroutine.

5.11.1 PROGRAM COUNTER INSTRUCTIONS - LINK GROUPING

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
LR	K,P	0 8	LOAD "K" REGISTERS FROM THE STACK REGISTER (PC_1) r12 ← (PC_1U); and r13 ← (PC_1L). Register 12 and 13 in the scratchpad are loaded with 16 bits from the Stack Register (PC_1). Bits (PC_1) 8 through (PC_1) 15 are transferred to Register r12 and bits (PC_1) 0 through (PC_1) 7 are transferred to Register r13. The Stack Register contents are transferred in two consecutive bytes to the data bus. The first transfer moves the eight most significant bits of the Stack Register (PC_1U) to the Scratchpad Register r12. The second transfer moves the least significant eight bits (PC_1L) of the stack register to scratchpad register r13.	1	4

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			This instruction is used to save the main program return address in a multiple level program. When used after an automatic interrupt return can be stored into memory for storage in infinite levels. Status bits are not modified by execution of this instruction.		
LR	P,K	0 9	RESTORE "K" REGISTER CONTENTS TO THE STACK REGISTER (PC_1) $PC_1U \leftarrow$ (r12), and $PC_1L \leftarrow$ (r13) The contents of Scratchpad Registers r12 and r13 are moved to the Stack Register PC_1. The most significant 8 bits of PC, (PC_1U), bits (PC_1) 8 through (PC_1) 15, receive the contents of scratchpad register r12. The 8 least significant bits of the stack register (PC_1L), bits (PC_1) 0 through (PC_1) 7, receive the contents of scratchpad register r13. The move is completed in two consecutive transfers on the data bus. The most significant eight bits of (PC_1U) are transferred in the first move. The least significant eight bits (PC_1L) are transferred in the second move. This instruction is used to restore the main program return address to the stack register just before the POP instruction (refer to 5.11.3) is executed to cause return to the main routine. When using a program with multiple level interrupt, the program return address should be stored in memory. The return	1	4

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES

address will be pushed into the stack register (PC_1) automatically by the interrupt. This address should be saved in memory before the interrupt control bit is reactivated. This prevents a second unexpected interrupt from forcing the first interrupt address (placed in PC_1 from the first interrupt) into the stack register before the original program starting address was stored safely in memory.

When returning from an interrupt subroutine to the next higher program, it is suggested that the following procedure be observed:

- o Disable the interrupt system
- o Transfer the address of the routine to be entered to the stack register (PC_1)
- o Re-enable the interrupt
- o Execute POP

Disabling interrupt prevents an unexpected interrupt from occurring after the return address is loaded to the stack register and before the return is executed.

Example of Return:

OP CODE	OPERAND	COMMENT
DI		Disable interrupt
LR	P,K	Load return address to Stack Register
EI		Enable interrupt
POP		Return

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			Status bits are not modified by execution of this instruction.		
LR	P0,Q	0 D	RESTORE "Q" REGISTER CONTENTS TO THE PROGRAM COUNTER. $PC_0U \leftarrow (r14)$; and, $PC_0L \leftarrow (r15)$ The contents of scratchpad registers r14 and r15 are moved to the program counter (PC_0). Sixteen bits are moved to the program counter in two successive transfers of eight bits over the data bus. The eight bits in scratchpad register r15 are moved in the first transfer to the eight least significant bits of PC_0, bits (PC_0) 0 through (PC_0) 7. The eight bits in the scratchpad register r14 are moved in the second transfer to the eight most significant bits of PC_0, bits (PC_0) 8 through (PC_0) 15. This instruction affects a forced branch to the address indicated by the contents of Q (r14 and r15). It can be used in lieu of LR P,K to effect a return from an address stack to a higher level operating program. In this case, the return to a higher routine is completed with the LR P0,Q instruction without requiring use of the POP instruction. The LR P0,Q instruction can be used to perform indirect branches within a program. If the Q registers in the scratchpad are treated as an indirect program address registers, the actual branch address can be modified. The branch to the implied address is then determined by executing the LR P0,Q instruction. An example of this application follows:	1	4

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES

Example:

An intelligent terminal receives messages from a communication link with characters encoded in eight bit ASCII format. The next subroutine to be executed in the program depends on the nature of the next character to be received as follows:

CHARACTER RECEIVED	HEXADECIMAL CHARACTER CODE	FUNCTION TO BE PERFORMED	ADDRESS OF SUB-ROUTINE
SOH (Start of Header)	0E	Scan Header for Label	020E
STX (Start of Text)	02	Transfer text to output buffer	0202
EOM (End of Message)	19	Terminate Line Connection	0219

If the next character to be examined is known to be one of the above control characters, then indirect branch to the appropriate control routine can be expected as follows:

OP CODE	OPERAND	COMMENT
LIS	2	
LR	QU,A	Load upper byte of Branch Address
INS	0	Read next character in from I/O port "0"
LR	QL,A	
LR	P0,Q	

Status bits are not modified by execution of this instruction.

5.11.2 PROGRAM COUNTER INSTRUCTIONS: CALL TO SUBROUTINE

There are two call to subroutine instructions. These are CALL TO SUBROUTINE DIRECT (PK) and CALL TO SUBROUTINE IMMEDIATE (PI).

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
PK	--	0	C	CALL TO SUBROUTINE DIRECT $PC_1 \leftarrow (PC_0)$; and $PC_0U \leftarrow$ (r12); and $PC_0L \leftarrow$ (r13) The current contents of the program counter (PC_0) are transferred to the stack register (PC_1) and the contents of scratchpad K registers (r12 and r13) are transferred to the program counter. The sixteen bits contained in the K registers are moved to the program counter in two consecutive bytes on the data bus. The full sixteen bits of the program counter are transferred to the stack register (PC_1) and the contents of scratchpad register r13 are transferred to the least eight significant bits of PC_0, bits (PC_0) 0 through (PC_0) 7 on the first move. The contents of scratchpad register r12 are transferred in the second move to the most significant eight bits of PC_0, bits (PC_0) 8 through (PC_0) 15. Service for an interrupt is inhibited at the end of this instruction. (Refer to section 3.6 for discussion of the interrupt system.) The status bits are not modified by execution of this instruction.	1	4
PI	aaaa	2	8	CALL TO SUBROUTINE IMMEDIATE $PC_1 \leftarrow (PC_0)$; and $PC_0 \leftarrow$ H 'aaaa'	3	6.5
		a	a	The sixteen bits of the program counter (PC_0) are transferred to the stack register (PC_1) and the sixteen bit address (hexadecimal aaaa) is transferred to the Program Counter.		
		a	a			

5.11.2 PROGRAM COUNTER INSTRUCTIONS: CALL TO SUBROUTINE

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			All Program Counters are modified with the same sixteen bit address value. The immediate call address H 'aaaa' is transferred from the instruction memory location to all Program Counters over the data bus. All sixteen bits of the subroutine call address are fetched before the Program Counter is modified. The most significant eight bits of the subroutine call address are temporarily stored in the Accumulator while this instruction is being executed. Any previous results stored in the Accumulator are lost. The eight most significant bits of the call address are contained in the second byte of this instruction. The eight least significant bits of the call address are contained in the third byte of this instruction. Status bits are not modified during execution of this instruction. Service for an interrupt is inhibited at the end of this instruction. (Refer to section 3.6 for discussion of the interrupt system.)		

5.11.3 PROGRAM COUNTER INSTRUCTIONS: RETURN FROM SUBROUTINE

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
POP	--	1 C	RETURN FROM SUBROUTINE $PC_0 \leftarrow (PC_1)$ The sixteen bit contents of the Stack Register (PC_1) are transferred to the Program Counter (PC_0).	1	2

5.11.3 PROGRAM COUNTER INSTRUCTIONS: RETURN FROM SUBROUTINE (Cont.)

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			Status bits are not modified by execution of this instruction. Interrupt service is inhibited at the end of this instruction. (Refer to section 3.6 for discussion of the interrupt system.)		

5.12 BRANCH INSTRUCTIONS

The *BRANCH INSTRUCTIONS* also modify the operation of the program by altering the contents of the program counter (PC_0). They are grouped here separately from the program counter instruction group to facilitiate frequent reference by the programmer.

There are two types of branches executed in the F8 design:

- o Unconditional branch
- o Conditional branch

All branch instructions except the branch immediate (JMP) are two byte instructions in which the second byte is a relative address. When a branch is executed the relative address byte is added to the present contents of the program counter to obtain the new address of the next instruction to be fetched. The present contents of the program counter (P) will point to P + 1 where P is the first storage location of the Branch instruction. If no branch is to be taken, the next instruction to be fetched is indicated by the present program counter contents incremented (P+1).

The relative branch address is always formed by adding the eight bits in the second of the two byte relative branch instruction to the present contents of the program counter. The value of the second byte is treated as a 2's complement number. A relative branch can reach another instruction within the field of memory addresses bounded by an address 128 locations ahead of the current program counter address or 127 locations behind the current program counter address.

5.12.1 UNCONDITIONAL BRANCH INSTRUCTIONS

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
BR	aa	9	0	UNCONDITIONAL BRANCH RELATIVE	2	3.5
		a	a	$PC_0 \leftarrow (PC_0)$ +H 'aa'		
				Branch to the memory location found by adding the 8-bit 2's complement number (hexadecimal aa) to the present program counter address which is set to the location of the second byte of the two byte UNCONDITIONAL BRANCH RELATIVE instruction.		
				Status flags are not modified by execution of this instruction.		
JMP	aaaa	2	9	BRANCH IMMEDIATE	3	5.5
		a	a	$PC_0 \leftarrow$ H 'aaaa'		
		a	a	Branch to memory the location indicated by the 16 bit binary number (hexadecimal aaaa)		
				All sixteen bits of the address contained in the BRANCH IMMEDIATE instruction are fetched before the program counter is modified. The most significant eight bits of the address are temporarily stored in the accumulator while the instruction is executed. Any previous results contained in the accumulator are lost.		
				The most significant eight bits of the branch address are contained in Byte #2 of this instruction. The eight least significant bits of the branch address are contained in byte #3 of this instruction.		
				Status flags are not modified by execution of this instruction.		

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			Interrupt service is inhibited at the end of this instruction. (Refer to section 3.6 for discussion of the interrupt system.)		

5.12.2 CONDITIONAL BRANCH INSTRUCTIONS

All conditional branch instructions are two byte relative address instructions. The first byte contains the OP Code and the branch test conditions when applicable. The second byte contains an 8-bit binary branch vector. The new location (branch address) is formed by adding the contents of the branch vector (2's complement) to the present value of the program counter. Thus, all conditional branches are made relative to the present program counter location. Relative branching is performed within a range of 128 address locations forward (to a higher address) of the present program counter location* and 127 address locations behind (lower address) the present program counter location.

*NOTE: P is the address of the second byte of the conditional branch instruction.

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
BT	t,aa*	8 t	CONDITIONAL BRANCH TRUE	2	3.5** or
		a a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if any test is true. $PC_0 \leftarrow (PC_0) + 2$; if no test is true.*** The Conditional Branch True causes a program address modification when any one or more of the selected branch conditions is found to be active (true). Any combination of		3.0

* t is the operand specifying the test conditions

** The execution time is 3.5 cycles if the branch is taken and 3.0 cycles if the branch is not taken.

*** PC_0 contains the address of the instruction being executed.

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES

three possible tests may be made. The three test conditions are:

- o RESULT IS POSITIVE
- o RESULT IS ZERO
- o CARRY IS SET

Table 5.10 lists the Conditional Branch True instructions and the Branch conditions for each.

Status flags are not modified by execution of this instruction.

OPERAND	STATUS FLAGS TESTED			DEFINITION	COMMENTS
t	ZERO	CARRY	SIGN		
0	0	0	0	Non Functional	An effective 3 cycle NO-OP
1	0	0	1	Branch if positive	Same as BP
2	0	1	0	Branch on Carry	Same as BC
3	0	1	1	Branch if Positive or on Carry	
4	1	0	0	Branch if Zero	Same as BZ
5	1	0	1	Branch if Positive	Same as t=1
6	1	1	0	Branch if Zero or on Carry	
7	1	1	1	Branch if Positive or on Carry	Same as t=3

TABLE 5.10

5.12.2 CONDITIONAL BRANCH INSTRUCTIONS (Cont.)

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
BP	aa	8	1	BRANCH IF POSITIVE	2	3.5** or 3.0
		a	a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if the sign is positive $PC_0 \leftarrow (PC_0) + 2$; if the sign is negative A branch to the address formed by adding the eight bit binary displacement vector (hexadecimal aa) to the present program counter value is taken when the sign flag is binary ONE. (sign flag indicates the result is positive when set.) Otherwise fetch the next instruction. Note that a zero result always sets the sign flag indicating positive. Status flags are not modified by execution of this instruction.		
BC	aa	8	2	BRANCH ON CARRY	2	3.5** or 3.0
		a	a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if the carry flag is set $PC_0 \leftarrow (PC_0) + 2$; if the carry flag is reset. A branch to the address formed by adding the eight bit binary displacement vector (hexadecimal aa) to the present program counter value is taken if the carry flag is set. Otherwise fetch the next instruction. Status flags are not modified by execution of this instruction.		

** The execution time is 3.5 cycles if the branch is taken and 3.0 cycles if the branch is not taken.

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
BZ	aa	8	4	BRANCH ON ZERO	2	3.5** or 3.0
		a	a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if sign flag is set. $PC_0 \leftarrow (PC_0) + 2$; if sign flag is reset. A branch is taken if the zero flag is set. The next instruction is fetched if the zero flag is reset. Status flags are not modified by execution of this instruction.		
BF	t,aa	9	t	BRANCH IF FALSE	2	3.5** or 3.0
		a	a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if the selected status flags are all reset (binary "0"). $PC_0 \leftarrow (PC_0) + 2$; if any of the selected status flags are set (binary "1"). The Branch On False is a generalized branch instruction for testing the absence of all flags selected. Operand t is a 4-bit binary number that selects the status flags to be tested. It is specified as one of the 16 possible hexadecimal characters. Table 5.11 shows the possible combinations of tests that can be specified with the Branch False instruction and the condition in which the branch is taken. Operand aa is an 8-bit address displacement vector that forms the address of the instruction to be fetched when the branch is taken.		

** The execution time is 3.5 cycles if the branch is taken and 3.0 cycles if the branch is not taken.

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			The branch address is formed by adding the operand "aa" to the present program counter contents which is set to the location of the second byte of the instruction. The address displacement vector is a 2's complement binary value.		

OPERAND t	STATUS FLAGS TESTED OVF	ZERO	CARRY	SIGN	DEFINITION	COMMENTS
0	0	0	0	0	Unconditional branch relative	Refer: Sect. 5.12.1
1	0	0	0	1	Branch on negative	Same as BM
2	0	0	1	0	Branch if no carry	Same as BNC
3	0	0	1	1	Branch if no carry and negative	
4	0	1	0	0	Branch if not zero	Same as BZ
5	0	1	0	1		Same as t = 1
6	0	1	1	0	Branch if no carry & result is no zero	
7	0	1	1	1		Same as t = 3
8	1	0	0	0	Branch if there is no overflow	Same as BNO
9	1	0	0	1	Branch if negative and no overflow	
A	1	0	1	0	Branch if no overflow and no carry	
B	1	0	1	1	Branch if no overflow, no carry & negative	
C	1	1	0	0	Branch if no overflow and not zero	
D	1	1	0	1		Same as t = 9
E	1	1	1	0	Branch if no overflow, no carry & not zero	
F	1	1	1	1		Same as t = B

TABLE 5.11

BRANCH CONDITIONS FOR BF INSTRUCTION

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
BNO	aa	9	8	BRANCH IF NO OVERFLOW	3	3.5** or 3.0
		a	a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if OVF flag is reset $PC_0 \leftarrow (PC_0) + 2$; if OVF flag is set A branch is taken if no overflow is indicated. Otherwise the next instruction is fetched. Status bits are not modified by execution of this instruction.		
BM	aa	9	1	BRANCH ON NEGATIVE	2	3.5** or 3.0
		a	a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if sign flag is reset $PC_0 \leftarrow (PC_0) + 2$; if sign flag is set A branch is taken if the sign flag indicates a negative result; otherwise the next instruction is fetched. Status bits are not modified by execution of this instruction.		
BNC	aa	9	2	BRANCH IF NO CARRY	2	3.5** or 3.0
		a	a	$PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if carry flag is reset $PC_0 \leftarrow (PC_0) + 2$; if carry flag is set A branch is taken if the carry flag is reset, otherwise the next instruction is fetched. Status bits are not modified by execution of this instruction.		

** The execution time is 3.5 cycles if the branch is taken and 3.0 cycles if the branch is not taken.

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
BNZ	aa	9 4 a a	BRANCH IF NOT ZERO $PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if zero flag is reset $PC_0 \leftarrow (PC_0) + 2$; if zero flag is set A branch is taken if the zero flag indicates a non-zero result. Otherwise, the next instruction will be fetched. Status bits are not modified by execution of this instruction.	2	3.5** or 3.0
BR7	aa	8 F a a	BRANCH ON ISAR $PC_0 \leftarrow ((PC_0)+1)+H'aa'$; if ISAR L 7 $PC_0 \leftarrow (PC_0) + 2$; if ISAR L 7 If any of the lower 3 binary bits of the indirect scratchpad address register (ISAR L) are reset, a branch is executed. The next instruction will be fetched if an address in ISAR is XXX 111 (X is a don't care). The branch address is formed by adding the 8 bit binary displacement vector (2nd byte of instruction) to the present program counter contents. The displacement vector is a 2's complement binary value. Status bits are not modified by execution of this instruction.	2	2.5* or 2.0

** The execution time is 3.5 cycles if the branch is taken and 3.0 cycles if the branch is not taken.

* 2.5 cycles if the branch is taken and 2.0 cycles if no branch is taken.

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES

This instruction is useful for handling multiple bytes of data stored in the scratchpad under control of a program loop. The last byte of a data file may be assigned to a scratchpad location that is multiple of 8 storage locations such that the indirect scratchpad address is of the form X7 (where X is any octal value).

Example: Assume 4 bytes were stored in scratchpad locations 35, 36, 37, and 38. A program to form the exclusive "OR" of all four bytes can be written.

	OP CODE	OPERAND	COMMENT
	LISU	4	Load upper octal address into ISAR U.
	LISL	3	Load lower octal address into ISAR L.
	CLR		Clear Accumulator
Loop	XS	13	Form exclusive "OR" and increment ISAR.
	BR7	LOOP	Not finished, go back Loop.

5.13 INPUT/OUTPUT INSTRUCTION GROUP

The input/output instructions involve data movement between the I/O port and the accumulator. Input and output instructions are either:

- Short address, or one byte instructions
- Long address, or two byte instructions

Each input or output instruction contains two parts which are:

- OP CODE
- I/O port address

The input or output short address instructions have both the op code and the I/O port address contained in one byte. The four most significant bits of the short address form the op code while the four least significant bits form the I/O port address. Each circuit connected to the F8 data bus has four, 8-bit port addresses. In the ROM, for instance, two ports are I/O ports they are latches with outputs connected to the outside world. Another port in the ROM is called the *TIMER*. This is an 8-bit counter, pulsed every 31 clock cycles. The fourth, and last, port on this circuit holds two bits which determine the state of the local interrupt control circuitry. All four ports are accessible using I/O instructions, linking these registers with the accumulator. Table 5.12 lists the four ports on each circuit and their corresponding addresses. The four ports have sequential addresses, as indicated in the table.

CPU - 3850

Port Address	Location
00	CPU, I/O Port A
01	CPU, I/O Port B
02	Not Assigned
03	Not Assigned

ROM - 3851

Port Address	Location
X0 X4 X8 XC	ROM, I/O Port A
X1 X5 X9 XD	ROM, I/O Port B
X2 X6 XA XE	Local Interrupt Control
X3 X7 XB XF	ROM Timer

X is a four bit hexidecimal number.
The ROM port addresses are a user's mask option, thus, these four address are selected by the user.

DYNAMIC MEMORY INTERFACE 3852

Port Address	Location
0C	8 Bit Register
0D	RAM refresh and DMA control bits
0E	Not Assigned
0F	Not Assigned

STATIC MEMORY INTERFACE - 3853

Port Address	Location
0C	Interrupt Vector
0D	Interrupt Vector
0E	Local Interrupt Control
0F	Timer

TABLE 5.12

PORT FUNCTIONAL ADDRESS ASSIGNMENTS

The Input or Output short address instructions will only address I/O ports "00" through "0F".

The Input or Output long address instructions are each two bytes long. The OP CODE is contained in the first byte and the eight bit I/O port address is contained in the second byte. All I/O port addresses between "04" to "FF" may be addressed by the long address instructions.

Every I/O port is fully bidirectional and may be used alternately for input or output during the operation of a program. However, the latches used for an output function are electrically connected to the input bus used by the input instructions (INS or IN). If an I/O port had been used for an output in a previous instruction the output latches must be first cleared prior to executing the input instruction. Only those latches that are connected to the input lines which are actually used must be cleared. This permits the programmer to divide an I/O port to use one or more of the I/O port lines to actively drive while the others are used to sense an input simultaneously.

5.13.1 INPUT/OUTPUT INSTRUCTIONS

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
INS	a	A a	INPUT SHORT ADDRESS ACC← (INPUT PORT a) This instruction can address only the I/O ports with the lowest 16 address ("00" to "0F"). The status bits are set by execution of this instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0 *Note: Only 2 cycles when I/O port ADDRESS "00" or "01" is used.	1	4*

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
IN	aa	2 6 a a	INPUT LONG ADDRESS ACC ← (INPUT PORT aa) Transfer the contents of I/O port "aa" to the Accumulator. Any input port with an address between 4 and 255 can be monitored with this instruction. A full 8-bit byte is transferred from I/O port "aa" to the Accumulator. The status bits are set by execution of this instruction as follows: OVF ZERO CARRY SIGN 0 1/0 0 1/0	2	4
OUTS	a	B a	OUTPUT SHORT ADDRESS OUTPUT PORT "a" ← (ACC) Transfer the contents of the Accumulator to output port "a". One byte (8-bits) is transferred from the Accumulator to the output port addressed by the hexadecimal digit "a". The byte will be held in the output port latches, until the next OUT or OUTS instruction is executed for I/O port "a". The status bits are not modified by execution of this instruction. *Note: Only 2 cycles when I/O port address "00" or "01" is used.	1	4*
OUT	aa	2 7 a a	OUTPUT LONG ADDRESS OUTPUT PORT "aa" ← (ACC) Transfer the contents of the Accumulator to the Output Port "aa". One byte (8-bits) is transferred from the Accumulator to the output port	2	4

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
			addressed by the hexadecimal address "aa". The byte will be held in the output port latches until the next OUT or OUTS instruction is executed for I/O port "aa"only. I/O ports between address 4 through 255 may be addressed with this instruction. The status bits are not modified by execution of this instruction.		

5.13.2 PROGRAMMING THE TIMERS

One timer is available on each of the 3851 ROM or 3853 MI Model A circuits in an F8 system. The timers can be program controlled to implement timing functions without the need to add external circuits. Typical applications include driving printers, storing teletype input signals, generating telephone ring signals or other necessary time controlled functions.

The sequence for initiating the timer is:

- o Load the timer counter [Execute OUT(S) to the timer port]
- o Enable the internal timer control [Execute OUT(S) to the interrupt control port].

Example: Assume a given 3851 ROM chip has been given an I/O port group Address assignment:

ADDRESS	FUNCTION
"08"	I/O Port "A"
"09"	I/O Port "B"
"0A"	Local Interrupt Control
"0B"	Timer

5.13.2 PROGRAMMING THE TIMERS (Cont.)

The following subroutine uses the Timer with a duration before interrupt of 1.55 millisec (Refer to Appendix for duration codes.)

JP CODES	OPERAND	COMMENT
LI	T'100'	SET VALUE FOR 1.55 ms DELAY
OUTS	H'0B'	SET TIMER
LIS	03	SET TIMER CONTROL CODE
OUTS	H'0A'	ENABLE LOCAL INTERRUPT*

*Note: The Interrupt does not need to be enabled immediately but can be enabled anytime after the OUTS H'0A' instruction is executed and prior to the 1.55 ms elapsed time. The timer is initiated immediately after executing the OUTS H'0B' instruction.

Extended time delays are generated by initiating the timer and permitting it to continue to generate interrupts. Each time the interrupt occurs, the programmer may increment a memory location assigned to count interrupts for the terminal count. The first interrupt will occur at the preselected delay after the timer has been loaded. (The OUTS aa instruction is executed.) Each successive interrupt will occur at intervals of 3.968 millisec. (For a system running at 2MH.)

The minimum period for the timer is 15.5 us. (For a system running at 2MH.)
Operation of the timer generated interrupt is terminated by disabling the local interrupt or by resetting the INTERRUPT mode select bit. (Bit 2^1 in the local interrupt control circuit, paragraph 5.13.3, Table 5.13.)

5.13.3 PROGRAMMING THE LOCAL INTERRUPT CONTROL CIRCUIT

Each of the Local Interrupt circuits in the 3851 ROM or 3852 MI circuits may be individually activated or deactivated. Two bits are assigned to control the activation of the Local Interrupt Control and to select the mode of interrupt. (Internal timer or external interrupt)

Control is exercised by executing the following sequence:

- Load the accumulator with the required control word (refer to Table 5.13).
- Transfer the control word to the appropriate LOCAL INTERRUPT CONTROL circuit by executing an OUT(S) instruction. (Note: only I/O addresses "X2", "X6", "XA" or "XE" access the local interrupt control circuits.)

ACCUMULATOR CONTENT (Binary)		LOCAL INTERRUPT CONTROL FUNCTION
ACC_7	ACC_0	
XXXXXX	X0	Disable Local Interrupt
XXXXXX	01	Enable Local Interrupt; Select External Interrupt Mode
XXXXXX	11	Enable Local Interrupt; Select Internal Timer Mode

TABLE 5.13

LOCAL INTERRUPT CONTROL FUNCTION WORDS

5.14 INTERRUPT CONTROL INSTRUCTIONS

The Interrupt Control instructions turn on and off the Interrupt Control Bit (ICB) in the CPU circuit. When the interrupt control bit is reset the entire interrupt system is disabled.

OP CODE	OPERAND	MACHINE FORMAT		DESCRIPTION	BYTES	CYCLES
DI	-	1	A	DISABLE INTERRUPT The interrupt control bit is reset to zero.	1	2
EI	-	1	B	ENABLE INTERRUPT The interrupt control bit is set to one. Interrupt service is inhibited until the completion of the instruction that follows EI. Refer to section 3.6 for a description of the interrupt. Refer to section 5.13 for description of the local interrupt circuit programming.	1	2

5.15 NO OPERATION

OP CODE	OPERAND	MACHINE FORMAT	DESCRIPTION	BYTES	CYCLES
NOP	-	2 B	NO-OP No function is performed. The program counter is incremented.	1	1

All undefined OP CODES are executed as NOP instructions.

5.16 INSTRUCTION SET: CONDENSED LISTING

ACCUMULATOR GROUP INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF	ZERO	CARRY	SIGN	CYCLES
SR	1	12	SHIFT RIGHT ONE	0	1/0	0	1	1
SR	4	14	SHIFT RIGHT FOUR	0	1/0	0	1	1
SL	1	13	SHIFT LEFT ONE	0	1/0	0	1/0	1
SL	4	15	SHIFT LEFT FOUR	0	1/0	0	1/0	1
COM	--	18	ACC ← (ACC) ⊕ H 'FF'	0	1/0	0	1/0	1
LNK	--	19	ACC ← (ACC) + CB	1/0	1/0	1/0	1/0	1
INC	--	1F	ACC ← (ACC) + 1	1/0	1/0	1/0	1/0	1
LIS	i	7i	ACC ← H 'i'		-			1
CLR	--	70	ACC ← H '00'		-			1
LI	ii	20 ii	ACC ← H 'ii'		-			2.5
NI	ii	21 ii	ACC ← (ACC) ∧ H 'ii'	0	1/0	0	1/0	2.5
OI	ii	22 ii	ACC ← (ACC) V H 'ii'	0	1/0	0	1/0	2.5
XI	ii	23 ii	ACC ← (ACC) ⊕ H 'ii'	0	1/0	0	1/0	2.5
AI	ii	24 ii	ACC ← (ACC) + H 'ii' (Binary Add)	1/0	1/0	1/0	1/0	2.5
CI	ii	25 ii	H 'ii'+ ($\overline{ACC}$)+ 1	1/0	1/0	1/0	1/0	2.5

5.16 INSTRUCTION SET: CONDENSED LISTING (Cont.)

SCRATCHPAD REGISTER INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF	ZERO	CARRY	SIGN	CYCLES
LR	y,x		GENERAL LOAD REGISTER FORMAT ALLOWABLE OPERANDS LISTED BELOW					1
	A,r*	4r	ACC ← (r)		-			
	A,KU	00	ACC ← (r12)		-			
	A,KL	01	ACC ← (r13)		-			
	A,QU	02	ACC ← (r14)		-			
	A,QL	03	ACC ← (r15)		-			
	r,A	5r	r ← (ACC)		-			
	KU,A	04	r12 ← (ACC)		-			
	KL,A	05	r13 ← (ACC)		-			
	QU,A	06	r14 ← (ACC)		-			
	QL,A	07	r15 ← (ACC)		-			
AS	r	Cr	ACC ← (ACC)+(r)(Binary)	1/0	1/0	1/0	1/0	1
ASD	r	Dr	ACC ← (ACC)+(r)(Decimal)	1/0	1/0	1/0	1/0	2
NS	r	Fr	ACC ← (ACC) $\wedge$ (r)	0	1/0	0	1/0	1
XS	r	Er	ACC ← (ACC) $\oplus$ (r)	0	1/0	0	1/0	1
DS	r	3r	r ← (r) + H'FF'(Decrement)	1/0	1/0	1/0	1/0	1.5

* Operand r formats are:

Direct Addressing	Indirect Addressing		
0 through 11 (Decimal Form)	S	or	12
	I	or	13
H'0' through H'B' (Hexadecimal Form)	D	or	14

DATA COUNTER INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF ZERO CARRY SIGN	CYCLES
LR	Q,DC	0E	r14←(DCU) ; r15←(DCL)	-	4
LR	H,DC	11	r10←(DCU) ; r11←(DCL)	-	4
LR	DC,Q	0F	DCU←(r14) ; DCL←(r15)	-	4
LR	DC,H	10	DCU←(r10) ; DCL←(r11)	-	4
ADC	-	8E	DC←(DC) + (ACC)	-	2.5
DCI	iiii	2A ii ii	DC←H 'iiii'	-	6
XDC	-	2C	DC ⇄ DC_1 [Memory Interface Circuit Only]	-	2

INDIRECT SCRATCHPAD ADDRESS REGISTER INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF ZERO CARRY SIGN	CYCLES
LR	A,IS	0A	ACC ← (ISAR)	-	1
LR	IS,A	0B	ISAR ← (ACC)	-	1
LISU	a	01100a*	ISARU ← a	-	1
LISL	a	01101 a*	ISARL ← a	-	1

* a is 3 bits

MEMORY REFERENCE INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF	ZERO	CARRY	SIGN	CYCLES
LM	-	16	ACC ← ((DC))			-		2.5
ST	-	17	(DC)← (ACC)			-		2.5
AM	-	88	ACC←(ACC)+((DC)){Binary}	1/0	1/0	1/0	1/0	2.5
AMD	-	89	ACC←(ACC)+((DC)){Decimal}	1/0	1/0	1/0	1/0	2.5
NM	-	8A	ACC ←(ACC) $\wedge$ ((DC))	0	1/0	0	1/0	2.5
OM	-	8B	ACC ←(ACC) $\vee$ ((DC))	0	1/0	0	1/0	2.5
XM	-	8C	ACC ←(ACC) $\oplus$ ((DC))	0	1/0	0	1/0	2.5
CM	-	8D	((DC)) + ($\overline{ACC}$) + 1	1/0	1/0	1/0	1/0	2.5

STATUS REGISTER INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF	ZERO	CARRY	SIGN	CYCLES
LR	W,J	1D	W ← (r9)					2

W_4	W_3	W_2	W_1	W_0
INT	OVF	ZERO	CARRY	SIGN

(Privileged Instruction)*

LR	J,W	1E	r9 ← (W)			-		1

*Privileged Instructions inhibit interrupt service at the end of the instruction.

MISCELLANEOUS INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF ZERO CARRY SIGN	CYCLES
NOP	-	2B	NO OPERATION	-	1

PROGRAM COUNTER INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF ZERO CARRY SIGN	CYCLES
LR	K,P	08	r12←(PC_1U) ; r13←(PC_1L)	-	4
LR	P,K	09	PC_1U←(r12) ; PC_1L←(r13)	-	4
LR	PO,Q	0D	PC_0U←(r14) ; PC_0L←(r15)	-	4
PK	-	0C	PC_0U←(r12) ; PC_0L←(r13) and PC_1←(PC_0) Privileged Instruction*	-	4
PI	aaaa**	28 ii ii	PC_1←(PC_0) ; PC_0←H 'aaaa' Privileged Instruction*	-	6.5
POP	-	1C	PC_0←(PC_1) Privileged Instruction*	-	2

* Privileged Instruction inhibit interrupt service request at the end of the instruction.

** The contents of accumulator are destroyed.

5.16 INSTRUCTION SET: CONDENSED LISTING (Cont.)

BRANCH INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF ZERO CARRY SIGN	CYCLES
BR	aa	90 aa	$PC_0 \leftarrow ((PC_0)+1) + H'aa'$	-	3.5
JMP	aaaa****	29 aa aa	$PC_0 \leftarrow H'aaaa'$ Privileged Instruction*	-	5.5
BT	t,aa***	8t*** aa	$PC_0 \leftarrow ((PC_0)+1)+ H'aa'$ if any test is true $PC_0 \leftarrow (PC_0)+ 2$ if no test is true	-	3.5** or 3.0
BP	aa	81 aa	$PC_0 \leftarrow ((PC_0)+1)+ H'aa'$ if SIGN=1 $PC_0 \leftarrow (PC_0)+ 2$ if SIGN=0	- -	3.5 3.0
BC	aa	82 aa	$PC_0 \leftarrow ((PC_0)+1)+ H'aa'$ if CARRY=1 $PC_0 \leftarrow (PC_0)+ 2$ if CARRY=0	- -	3.5 3.0
BZ	aa	84 aa	$PC_0 \leftarrow ((PC_0)+1)+ H'aa'$ if ZERO=1 $PC_0 \leftarrow (PC_0)+ 2$ if ZERO=0	- -	3.5 3.0
BM	aa	91 aa	$PC_0 \leftarrow ((PC_0)+1)+ H'aa'$ if SIGN=0 $PC_0 \leftarrow (PC_0)+ 2$ if SIGN=1	- -	3.5 3.0
BNC	aa	92 aa	$PC_0 \leftarrow ((PC_0)+1)+ H'aa'$ if CARRY=0 $PC_0 \leftarrow (PC_0)+ 2$ if CARRY=1	- -	3.5 3.0
BNZ	aa	94 aa	$PC_0 \leftarrow ((PC_0)+1)+ H'aa'$ if ZERO=0 $PC_0 \leftarrow (PC_0)+ 2$ if ZERO=1	- -	3.5 3.0

STATUS BIT TESTS

2^2	2^1	2^0
ZERO	CARRY	SIGN

* Privileged Instructions inhibit interrupt service request at the end of the instruction.

** 3.5 cycles if branch is taken. 3.0 cycles if branch is not taken.

*** t is only 3 bits

**** The contents of the accumulator are lost.

BRANCH INSTRUCTIONS (Cont.)

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF ZERO CARRY SIGN	CYCLES
BF	t*,aa	9t* aa	$PC_0 \leftarrow ((PC_0)+1)+$ H'aa' if selected status bits are all "0"	-	3.5**
			$PC_0 \leftarrow (PC_0) + 2$ if any status bit is 1	-	3.0
BNO	aa	98 aa	$PC_0 \leftarrow ((PC_0)+1)+$ H'aa' if OVF=0 $PC_0 \leftarrow (PC_0)+ 2$ if OVF=1	- -	3.5 3.0
BR7	aa	8F aa	$PC_0 \leftarrow ((PC_0)+1)+$ H'aa' if ISAR≠7 $PC_0 \leftarrow (PC_0)+ 2$ if ISAR=7	- -	2.5*** 2.0

TEST CONDITIONS

2^3	2^2	2^1	2^0
OVF	ZERO	CARRY	SIGN

INTERRUPT CONTROL INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF ZERO CARRY SIGN	CYCLES
DI	-	1A	DISABLE INTERRUPT	-	2
EI	-	1B	ENABLE INTERRUPT Privileged Instruction****		

* t is four bits

** 3.5 cycles if branch is taken. 3.0 cycles if branch is not taken.

*** 2.5 cycles if branch is taken. 2.0 cycles if branch is not taken.

**** Privileged instructions inhibit interrupt service at the completion of execution of the instruction.

5.16 INSTRUCTION SET: CONDENSED LISTING (Cont.)

INPUT/OUTPUT INSTRUCTIONS

OP CODE	OPERAND (S)	MACHINE FORMAT	FUNCTION	STATUS BITS OVF	ZERO	CARRY	SIGN	CYCLES
INS	a	Aa	ACC←(INPUT PORT a) Input Ports 00 to 0F only	0	1/0	0	1/0	4*
IN	aa	26 aa	ACC←(INPUT PORT aa) Input Ports 04 through FF only	0	1/0	0	1/0	4
OUTS	a	Ba	OUTPUT PORT a ←(ACC) Output Ports 00 to 0F only			-		4*
OUT	aa	27 aa	OUTPUT PORT aa ← (ACC) Output Ports 04 through FF only			-		4

* 2 cycles when I/O port address is "0" or "1".

6.0 F8 CROSS ASSEMBLER

6.0.1 GENERAL

This chapter describes the F8 Cross Assembler. Written in Fortran IV, this routine will execute on a *HOST* machine. The first section of this chapter lays the foundation for using assemblers Section 6.2 describes the different fields of the F8 assembler instructions. The next section, 6.3, details each F8 assembler command and its resulting action. Section 6.4 briefly relates the F8 assembler to another F8 software package, the simulator. (The simulator is fully defined in the next chapter, Chapter 7.) Finally, Section 6.5 lists each assembler error statement and the probable cause of the error.

6.1 INTRODUCTION

The basic programming language for any computer is machine language; this involves writing out the one's and zero's that make up a set of instructions. Assembly language programming is more convenient than writing programs in machine language. It allows the software writer to use mnemonic codes instead of a bit configuration. For instance, a branch instruction may be written BR, instead of 1001 0000. The mnemonic listing of F8 instructions appears in Chapter 5. Assembly language allows addresses to be symbolic instead of absolute. Symbols are used to represent addresses or absolute values. Therefore, a branch instruction may be written:

```
BZ  LOC1.
```

The symbol, LOC1, is defined elsewhere in the program. Assembly language programs may be more easily read and understood than those written in machine language. Comments can be added throughout the program for documentation. Finally, assembly language programs ease the introduction of program data by providing instructions to define constants.

A program called an assembler is needed to translate a source program into an object program. The programmer, writing in assembly language, generates the source program. The assembler converts this to an object program, the machine language listing of the assembler. The assembler does the bookkeeping to keep track of the symbolic names used by the programmer. In addition, it converts the mnemonics to machine language, inserting the proper value of symbols and literals.

Using an assembler, the programmer can generate software in a shorter time period with fewer errors. Debugging a program is also easier using an assembler. Finally, assembly language programs are easier to document and maintain.

6.2 INSTRUCTION FIELDS

Assembler instructions have four fields: label, command, operand and comment. Instructions are written in *FREE FORM*. One or more blanks are used to separate fields. The restrictions are that the label field, if used, must start in the first position of a record and that multiple operands are separated by commas. An asterisk in column one indicates a comment statement and a nonblank character in column 72 indicates the following line is a continuation of the present statement.

There are two types of assembler commands: control instructions and F8 machine instructions. The control instructions are for assembler "bookkeeping". F8 instructions have a one to one correspondence with actual F8 machine commands. The assembler converts the F8 mnemonics into F8 machine language instructions.

The control instructions and the functions they perform are listed in Table 6.1 and explained below. F8 instructions and their assembler mnemonics are discussed in Chapter 5.

6.2.1 LABELS

Labels are used to identify lines of instruction. A label, if used, must begin in record position one. The first character of a label must be alphabetic; the remaining characters may be alphanumeric. While a label may be of any length, only the first four characters are retained by the assembler.

6.2.2 OPERANDS

The operand field may be either a constant, an address, or an expression. Each of these field types is described below.

6.2.2.1 Constant values can be specified in several number systems. The particular number system is indicated by a letter followed by the number in quotes. Table 6.2 lists the six types of constant formats. These constants may be used as the operand of both F8 and assembler instructions.

CONSTANT	DEFINITION
nn	If no letter appears, the default condition, a decimal number is assumed.
D'nn'	Decimal numbers are specified by the letter D.
H'nn'	Hexadecimal numbers are specified by the letter H.
B'nn'	Binary numbers are specified by the letter B.
O'nn'	Octal numbers are specified by the letter O.
C'nn'	ASCII characters are specified by the letter C.
T'nn'	F8 timer counts are specified by the letter T.
	The assembler convert the number of specified time delays into the appropriate F8 timer delay code. Delays between 0 and 254 may be specified in decimal notation.

TABLE 6.2

ASSEMBLER CONSTANT

6.2.2.2 Addressing fields refer either to program locations or to data in memory. The forms are:

1) Symbol - An address label defined elsewhere.

2) *+Constant
*-Constant - The location of the present instruction plus or minus the constant.

3) Constant - A constant value.

6.2.2.3 EXPRESSIONS

An operand may also be specified in terms of an expression. An expression must be enclosed in parenthesis. Five basic arithmetic operators may be used (+,-,*,/ and **) and up to ten levels of parenthesis; however, only decimal constants and symbols can appear in an expression. Multiple operators are translated according to the standard rules of algebra.

6.2.2.4 * COMMENT STATEMENT

An asterisk in position one of a statement denotes a comment statement. The line will be printed in the output but ignored by the assembler.

COMMAND	OPERAND	FUNCTION	SECTION
EQU	OPERAND	Equates symbols for programming responses	6.3.1
ORG	OPERAND	Set the value of the assembler location counter equal to the expression	6.3.2
DC	CONSTANT	Defines a one or two byte constants in memory. 2 bytes	6.3.3
TITLE	"HEADING"	The "Heading" will be printed at the top of each assembly page.	6.3.4
EJECT		Skip to next page in assembly listing	6.3.4
XREF		TA cross reference listing will be printed at the end of the listing.	6.3.5
SYMBOL		A symbol table will be printed at the end of the listing.	6.3.6
BASE	OCT,HEX, DEC	Selects the base of the numbers appearing in the output.	6.3.7
MAXCPU	OPERAND	Limits the amount of CPU execution time.	6.3.8
END		Signal the end of the program.	6.3.9

TABLE 6.1

ASSEMBLER INSTRUCTIONS

6.3 ASSEMBLER COMMANDS

Assembler commands assist the assembler by defining symbols and entering values into the program. The F8 assembler commands are listed in Table 6.1 and described below.

6.3.1 EQU - EQUATE SYMBOL

EQU instructions are used to define symbols; a symbol is assigned the value in the operand field of the instruction. The format for this instruction is:

LABEL	OPERATION	OPERAND
SYMBOL	EQU	EXPRESSION, SYMBOL, CONSTANT

The symbol is assigned the value of the expression. The assembler will insert this value wherever it encounters the symbol.

LABEL	OPERATION	OPERAND
BEGIN	EQU	15
LAST	EQU	(*+2)
MIDDLE	EQU	(LAST - 4)
R14	EQU	14

The EQU instruction can be used to equate symbols to scratchpad registers, memory locations or other specific values.

6.3.2 ORG - SET LOCATION COUNTER

An ORG instruction sets the value of the assembler's location counter to the expression. The format for the ORG instruction is:

LABEL	OPERATION	OPERAND
Must be blank	ORG	OPERAND (Expression, Symbol, or Constant)

All symbols used the expression must be previously defined. The location counter is an assembler counter. It keeps track of where in memory the next instruction will be

located. If the operand is omitted from an ORG instruction, the location is set to the next unused location.

LABEL	OPERATION	OPERAND
Must be blank	ORG	0
	ORG	(FIRST + 50)
	ORG	(*+20)

6.3.3 DC - DEFINE CONSTANT

DC instructions set up storage locations for constant values. The format for the DC instructions is:

LABEL	OPERATION	OPERAND
SYMBOL or BLANK	DC	CONSTANT

A symbol may be used to name a constant or the first constant of a table. The symbol can appear on the operand field of an instruction or, as a relative address. The DC instruction will set up one or two bytes of memory storage for definition of the symbol. Thus, the DC instruction actually reserves memory locations for use in a program.

6.3.4 TITLE, EJECT - FORMAT COMMANDS

These commands control the format of the output listing.

LABEL	OPERATION	OPERAND
	TITLE	"HEADING"

The TITLE command causes the operand to be printed at the beginning of each page of the listing. The EJECT command will skip the printer to the beginning of a new page.

6.3.5 XREF - CROSS REFERENCE LISTING

The XREF instruction will cause a cross reference listing to be printed at the end of a listing. This table lists each symbol, gives the source statement line item in which the symbol appeared as a label, and lists every reference to that symbol.

The format for this instruction is:

LABEL	OPERATION	OPERAND
Must be blank	XREF	

6.3.6 SYMBOL - SYMBOL TABLE LISTING

The Symbol instruction causes a symbol table to be printed out at the end of a listing. This table lists the symbol and the location in the object code to which the symbol refers. The format for this instruction is:

LABEL	OPERATION	OPERAND
Must be blank	SYMBOL	

6.3.7 BASE - SELECT OUTPUT BASE

This assembler command controls the output format of assembler number listings. There are three possible output modes: Octal, Decimal and Hexadecimal. If a base command does not appear, the output will be listed in decimal. The format for this instruction is:

LABEL	OPERATION	OPERAND
	BASE	HEX
	BASE	OCT
	BASE	DEC

6.3.8 MAXCPU

The MAXCPU instruction limits the amount of CPU time for each assembly run. This is to prevent unexpected excessive computer usage. The operand field lists the maximum amount of seconds allowed. Its format is:

LABEL	OPERATION	OPERAND
Must be blank	MAXCPU	CONSTANT

6.3.9 END - END ASSEMBLY

The END instruction is always the last statement in an assembly program. It is used to terminate the assembly

of a program. The format for an END instruction is:

LABEL	OPERATION	OPERAND
Must be blank	END	

6.4 ASSEMBLER INPUT/OUTPUT FILES

The F8 Cross Assembler treats the user's program as an input file. Written in Fortran IV, the F8 assembler passes through the input program twice. During the first pass, the F8 assembler defines symbols and sets up symbol tables. The second and final pass generates the F8 machine code listing. The assembler outputs two files, F8LIST and F8TXT. The first file is a listing of a program after it is assembled. The F8TXT file is intended as an input file to the F8 simulator.

6.5 ERROR MESSAGES

When the assembler reaches a point where it cannot interpret the user's input, an error will be flagged. F8 assembler errors are listed in Table 6.3 along with the section in which they are defined.

ERROR MESSAGE	SECTION
Syntax Error	6.5.1
Invalid Label	6.5.2
Label OV	6.5.3
Error ARG 1	6.5.4
Error ARG 2	6.5.5
Missing Comma	6.5.6
Missing 2nd Argument	6.5.7
Error Single Argument	6.5.8
Expression ER	6.5.9
Offset OV	6.5.10

TABLE 6.3

ERROR MESSAGES

When the assembler detects an error, it will note the error, output the faulty line, and put an X under the point of confusion. The assembler will complete the pass and note the number of detected errors at the end of the listing.

6.5.1 SYNTAX ERROR

A syntax error is the result of a statement which cannot be recognized. A spelling error may have occurred.

6.5.2 INVALID LABEL

A label was used which does not follow the proper label format. For instance, the label may have begun with a number.

6.5.3 LABEL OV

The symbol table has overflowed. Too many symbols were used in the assembly program.

6.5.4 ERROR ARG 1

An error appeared in the first argument of an instruction with two arguments in the operand.

6.5.5 ERROR ARG 2

An error appeared in the second argument of an instruction with two arguments in the operand.

6.5.6 MISSING COMMA

A comma between two arguments in the operand has been left out.

6.5.7 MISSING 2ND ARGUMENT

An instruction with two arguments in the operand has been used; however, the argument does not have two operands.

6.5.8 ERROR SINGLE ARGUMENT

An error appeared in the only argument of a one argument instruction.

6.5.9 EXPRESSION ER

An error occurred within an expression in the operand field of an instruction. The expression will be printed with an X below the point of confusion.

6.5.10 OFFSET OV

This error occurs whenever a branch instruction offset value is not within +128 or -127 bytes of the branch instruction itself.

```
LOC  OBJECT CODE     STMT    SOURCE STATEMENT

PASS 1 COMPLETE      0.56 SEC


                     * THIS PROGRAM WILL ADD TWO 16 DIGIT DECIMAL NUMBERS
                     *
                     * BOTH OPERANDS ARE STORED IN SCRATCHPAD
                     *      OPERAND A -   LOCATIONS 30 TO 37 (OCT)
                     *      OPERAND B -  LOCATIONS 40 TO 47 (OCT)
                     *      THE RESULT (R) WILL BE PLACED IN LOCATIONS 40 TO 47 (OCT)
                     *
                     *
                1          ORG     0
                2          BASE HEX          THE OUTPUT WILL APPEAR IN HEX NOTATION
0    21   0     3          N1   0            CLEAR THE CARRY BIT
2    63          4    LOOP  LISU 3            LOAD THE UPPER BITS OF ISAR TO 3
3    4C          5          LR   A,S          GET OPERAND A
4    19          6          LNK              ADD THE PREVIOUS CARRY TO OPERAND A
5    52          7          LR   2,A          STORE THE RESULT IN TEMPORARY REG 2
6    64          8          LISU 4            LOAD THE UPPER BITS OF ISAR TO 4
7    4C          9          LR   A,S          PUT OPERAND B INTO ACCUMULATOR
8    24   66    10          AI   H'66'        DO A DECIMAL ADJUST ON OPERAND B
A    D2         11          ASD  2            ADD DECIMAL OPERAND A AND OPERAND B
B    5E         12          LR   D,A          STORE THE RESULT INTO SCRATCHPAD
C    BF   F5    13          BR7  LOOP         BRANCH BACK IF ANY DIGITS ARE REMAINING
               14        SYMBOL              PRINT A SYMBOL TABLE
               15        XREF                PRINT A CROSS REFERENCE TABLE
               16          END
```

Figure 6.1 Page 1 of 3

File F8LIST Produced By The F8 Cross Assembler

SYMBOL TABLE

LOC	LABEL	LOC	LABEL	LOC	LABEL	LOC	LABEL
2	LOOP						

Figure 6.1 Page 2 of 3
File F8LIST Produced By The F8 Cross Assembler

03/11/75 14:39:13 F8A -- VERSION 4D -- 2/18/75

CROSS-REFERENCE PAGE 3

LABEL	STMT	REFERENCES
LOOP	4	13

PASS 2 COMPLETE 0.82 SEC

NO ERRORS IN ABOVE ASSEMBLY

Figure 6.1 Page 3 of 3
File F8LIST Produced By The F8 Cross Assembler

7.0 F8 CROSS SIMULATOR

7.0.1 GENERAL

This chapter describes the F8 Cross Simulator written in Fortran IV, this routine will execute on a host computer. The first section of Chapter 7 discusses simulators in general. Section 7.2 describes the different files that are needed by the cross simulator. F8 simulation commands are divided into two control segments: configuration control and simulation control. The configuration control instructions, defined in Section 7.3, set up the hardware configuration of the F8 system to be simulated. Once the F8 model has been organized, the simulation is ready to begin. Simulation control language instructions, described in Section 7.4, load the F8 registers with values, execute a specified instruction set, and monitor the F8, cycle by cycle. Section 7.5 describes the output files of the F8 simulator. The next section of this chapter, Section 7.6, defines every message of the F8 simulator. The final section of Chapter 7, section 7.7, gives examples of actual simulation runs. Table 7.1 lists the seven sections of this chapter and their contents.

7.1 INTRODUCTION

A very effective tool for the design engineer engaged in a microprocessor application is a software simulator. For several years, logic simulators have been used by logic designers to check their circuit design. In a similar fashion, a microprocessor simulator may be used to model a specific microprocessor. Thus, a user may check out the microprocessor programs to insure that they perform as expected. In addition, factors external to the microprocessor may be entered into the simulation. A microprocessor simulator allows designers to experiment with programs on another computer. In addition to simulating the I/O ports, the simulator monitors every major register in the microprocessor. Thus, the designer has access to register status that is not normally available.

The designer is able to simulate the environment in which the F8 will operate on a host computer. For instance, I/O information and external interrupts may be input at a specific machine cycle. In addition, the user's program may be entered into the simulator and modified where desired. The simulator will output the status of specific registers and memory locations. A variety of trace and dump options are available to monitor processor activity instruction by instruction.

TABLE 7.1

TABLE OF CONTENTS

7.2 INPUT FILES

7.2.1 F8TXT FILE

Two input files are required by the F8 Simulator. The first is the assembled program file, F8TXT. This file was created by the F8 cross assembler. It consists of a machine code listing of the user's instructions and a symbol table relating symbols used in the assembler to machine locations.

7.2.2 F8SCL FILE

The other simulator input file, the control file, directs simulator operation. This file sets up a model F8 system. Memory locations may be loaded or altered. F8 operation can be initiated and the operation of the processor monitored at every instruction. The control file also directs the output format. Section 7.3 and 7.4 describe the formation of the simulation control language file.

7.3 CONFIGURATION CONTROL

The configuration control section lays out the F8 model for the simulator. Using four control statements, ROM, RAM, PORT and INTERRUPT, a full description of the F8 model can be constructed. Once a system is described, the F8 simulator will be able to load the assembled program into the computer model and begin the analysis.

7.3.1 ROM MEMORY

ROM Memory locations may be reserved by ROM control instructions. The format for this command is:

OPERATION	OPERAND
ROM	*ADDRESS*

The only format restriction for this instruction is that a blank appears between the OPERATION and the OPERAND. The associated *ADDRESS* field may be either a constant or a symbol. The allowable constant formats are listed in Table 7.2. The assembler symbol table is transferred to the simulator in the assembled program file; thus, the

simulator can use the same symbols as the assembler. Each ROM operation will reserve 1024 F8 memory locations for read only memory storage. The memory assignment begins at the value of ADDRESS and extends through the next 1023 memory locations. Attempting to write into ROM memory locations will be flagged by the simulator (refer to Section 7.6 for simulator messages).

FORMAT	MEANING
Symbol	Address of assembly symbol value
nn	Decimal Number
D'nn'	Decimal Number
H'nn'	Hexadecimal Number
B'nn'	Binary Number
O'nn'	Octal Number
C'nn'	ASCII Character
*	Present Value of Program Counter
*+Constant	Present Value of Program Counter + Constant
*-Constant	Present Value of Program Counter - Constant

TABLE 7.2

ADDRESS FIELD FORMATS

7.3.2 RAM MEMORY

RAM configuration control instructions reserve RAM memory locations in the simulator. Their format is:

OPERATION	OPERAND
RAM	*ADDRESS*

As before, a space must appear between the operation and the operand. The address may be a symbol or a constant.

A RAM command reserves 256 bytes of F8 memory as read/write memory. This block may be written in to or read from. The starting location of the RAM block is selected by the operand and the block continues for another 255 address locations. The simulator will warn the user if a ROM and/or a RAM segment overlap.

OPERATION	OPERAND
RAM	0
RAM	START
RAM	H'00100'

7.3.3 PORT ADDRESS

I/O port assignments are made with a PORT instruction to assign an I/O port in the simulator model. Each F8 I/O port in a system requires one PORT command. The format for this instruction is:

OPERATION	OPERAND
PORT	*NUMBER*

The port number may take on any of the constant formats in Table 7.2. A reference to an unassigned port will result in an error message. (Refer to Section 7.6 for simulator messages.)

OPERATION	OPERAND
PORT	4

7.3.4 INTERRUPT ASSIGNMENT

F8 interrupts are either internal (timer generated) interrupts or externally initiated interrupts. Each interrupt has a unique interrupt address vector. The value of this address is specified by the INTERRUPT instruction. The format for this command is:

OPERATION	OPERAND
INTERRUPT	*NUMBER ADDRESS, NUMBER ADDRESS*

External Interrupt — Internal Interrupt

There must ba a space between the operation and the operand. Consecutive operands are divided by commas but not spaces. The number refers to the port assignment assigned to the local interrupt control block while the address is that of the associated interrupt address vector. For internal interrupts, the NUMBER is the timer port number. The first number and address of the OPERAND are for the external interrupt while the second number and address are those of the internal interrupt. The simulator will flag any deviation from the normal F8 configuration. Interrupt priority is determined by the order in which the INTERRUPT instructions appear. The first INTERRUPT instruction has the highest priority.

OPERATION	OPERAND
INTERRUPT	6 B'0000000000010010', 7 B'0000000010010010'
INTERRUPT	H'A' H'A540', H'B' H'A5B0'

7.3.5 END

The completion of the configuration control statements must be marked by an END instruction.

OPERATION	OPERAND
END	

7.4 CONTROL LANGUAGE

After reading the END statement from the configuration control section, the simulator turns on the power. The F8 program counter is set to zero; the processor registers and memory locations are set to a random number. Next, the simulator will load the assembled program into the model F8.

The simulator divides each F8 instruction into simulation cycles. Each simulation cycle is equivalent to 500 nsec. for an F8 running at 2MHz. Table 7.3 lists the instructions and their corresponding cycles. Input data and interrupts may be entered at selective machine cycles.

The simulator control statements are divided into eight general categories:

- CLEAR Instructions (Section 7.4.1)
- SET Instruction (Section 7.4.2)

- OUTPUT ON REFERENCE (Section 7.4.3)
- DUMP Instructions (Section 7.4.4)
- TRACE Instructions (Section 7.4.5)
- ENVIRONMENT Instructions (Section 7.4.6)
- RUN Instructions (Section 7.4.7)
- HOUSEKEEPING Instructions (Section 7.4.8)

The CLEAR and SET Instructions may be used to enter new data into the F8 registers or memory. To assist monitoring the F8 model, OUTPUT ON REFERENCE, DUMP, and TRACE Instructions will output microprocessors activity. ENVIRONMENT Instructions model the outside world interacting with the F8. They set data on the I/O ports and simulate F8 interrupts. RUN Instructions start and stop the F8 simulation. Finally, HOUSEKEEPING Instructions assist the operation of the simulation program.

7.4.0 Instruction Format

The Simulation Control Language Instructions are written in free form. The only restrictions are:

1) At least one space must appear between the operation mnemonic and the operand field.

2) RANGE- This operand defines a range of addresses. The format is:

VALUE TO VALUE

where value may be a defined symbol, a constant or an asterisk. The * symbol is equal to the present value of the F8 program counter. Relative addresses may be referenced with *+Constant or *-Constant. The address formats are listed in Table 7.2.

3) RNAME- These operands are mnemonics used to identify the CPU registers. They are listed in Table 7.4.

4) DATA- Data appearing in operands may be either assembly defined symbols or constants (listed in Table 7.5). Since the data will be entered into an 8-bit register, its binary value must lie within the range 00000000 to 1111 1111.

5) DATA LIST- This operand serially lists several DATA items. The DATA values must be separated by at least one space, but no commas.

6) CYCLE- This operand is used to reference cycle numbers. The cycle may be any constant from Table 7.2 or an asterisk. In this case, the * refers to the present cycle number. *+Constant and *-Constant may be used to represent offsets of the cycle.

7) NUMBER- Any constant format in Table 7.2 may be used for this operand.

8) COMMENT- An * in the first column indicates a comment statement.

MEANING	OPERAND
OPERAND	DEFINITION
ACC	ACCUMULATOR
W	STATUS REGISTER
DC	DATA COUNTER
ISAR	INDIRECT SCRATCHPAD ADDRESS REGISTER
PC_0	PROGRAM COUNTER
PC_1	STACK REGISTER

TABLE 7.4

RNAME OPERAND MNEMONICS

FORMAT	MEANING
nn	Decimal Number
D'nn'	Decimal Number
H'nn'	Hexadecimal Number
B'nn'	Binary Number
O'nn'	Octal Number
T'nn'	Timer Count (Refer to Appendix)
Symbol	Assembler Defined Symbol

TABLE 7.5

DATA CONSTANT FORMATS

7.4.1 CLEAR

The CLEAR instructions set one register or several registers to zero. The operand for this instruction may refer to a counter, a memory location, an I/O port, a register, or scratchpad registers. The format for this instruction is:

OPERATION	OPERAND
CLEAR ALL	
CLEAR COUNT	*NUMBER*
CLEAR MEM	*RANGE*
CLEAR PORT	*NUMBER*
CLEAR REG	*RNAME*
CLEAR SCRATCH	*RANGE*

Italics implies an optional operand.

7.4.1.1 CLEAR ALL

This instruction will clear the counters, the I/O ports, the entire memory, the processor registers, and the scratchpad.

7.4.1.2 CLEAR COUNT *NUMBER*

The clear count instruction loads an 8-bit counter with all zeros. The number in the operand field must correspond to a counter assigned in the configuration control section. If no operand appears, all of the timers are set to zero.

7.4.1.3 CLEAR MEM *RANGE*

All memory locations within the specified range are set to zero. If no range is listed, all memory locations will be cleared.

7.4.1.4 CLEAR PORT *NUMBER*

The I/O port called out by the operand will be cleared by this instruction. For instance,

OPERATION	OPERAND
CLEAR PORT	5

will clear I/O port 5. This port was identified in the configuration control section. If a number does not appear in the operand, all of the I/O ports will be cleared.

7.4.1.5 CLEAR REG *RNAME*

Each F8 register has a mnemonic abbreviation. These are listed in Table 7.3. To clear an F8 register, the CLEAR REG instruction may be used; RNAME identifies the register to be cleared. The default condition for this instruction (RNAME not given) will clear all six registers, ACC, W, DC, ISAR, PC0 and PC1.

7.4.1.6 CLEAR SCRATCH *RANGE*

The F8 has 64 scratchpad registers. Numerically, these are referred to by the numbers 0 to 63. The CLEAR SCRATCH instruction will set to zero all scratchpad registers within the specified range, inclusive. If no range is listed, all scratchpad registers will be cleared.

7.4.2 SET

In addition to clearing registers, simulation control language allows registers to be set to a specified value. The format for set instructions is:

OPERATION	OPERAND
SET COUNT	*NUMBER DATA*
SET MEM	*RANGE DATALIST*
SET PORT	*NUMBER DATA*
SET REG	*RNAME DATA*
SET SCRATCH	*RANGE DATALIST*

For each instruction, all of the operands must appear.

7.4.2.1 SET COUNT *NUMBER DATA*

The 8-bit counters may be set to any value from 0 to 255. The counter is a polynomial counter; the count sequence is listed in the Appendix.

OPERATION	OPERAND
SET COUNT	07 T'34'

7.4.2.2 SET MEM *RANGE DATA*

Memory locations may be set to specific values with this instruction. The RANGE lists the address locations to be filled. The first data item is placed in the first memory address. All locations are filled in a sequential manner. A non-zero entry in column 72 will cause the next line of input to be a continuation of the previous record.

OPERATION	OPERAND
SET MEM	100 TO 103 H'A3' H'05' H'23' H'50'

The SET MEM instruction is equivalent to entering constants in the program.

7.4.2.3 SET PORT *NUMBER DATA*

This instruction loads the 8-bit F8 I/O port with the DATA entry. The port must be assigned by the configuration control section. This is equivalent to writing in an I/O port.

7.4.2.4 SET REG *RNAME DATA*

The REG referred to by this instruction are listed in Table 7.4. The SET command fills the appropriate register with the DATA item given.

OPERATION	OPERAND
SET REG	ACC B'00111101'

7.4.2.5 SET SCRATCH *RANGE DATA*

This instruction will set scratchpad registers to a given value. The first register to be set and the last register to be set are given by the RANGE operand. The data will be loaded sequentially, starting with the lowest scratchpad register number.

OPERATION	OPERAND
SET SCRATCH	5 TO 10 5 H'A5' B'00011011' 10 15

7.4.3 OUTPUT ON REFERENCE

This group of instructions is set by the programmer to flag processor activity within a specific region of memory.

7.4.3.1 ACCESS

The Access instruction will output whenever CPU references lie within a specified area of storage. An access may be either a read or a write into memory. The format for this instruction is:

OPERATION	OPERAND
ACCESS	*RANGE*

The simulator will print out the value of the program counter, the cycle number, the address accessed, and the memory modification (if any) for each store or fetch within the specified memory range. For example:

OPERATION	OPERAND
ACCESS	100 TO 1000

will monitor any memory read or memory write operations within memory locations 100 to 1000.

7.4.3.2 ALTER

To keep track of memory modification, the ALTER instruction is helpful. The format for this instruction is:

OPERATION	OPERAND
ALTER	*RANGE*

The simulator will print the value of the program counter, the cycle number, the memory address, and the new data for each store within this specified range.

7.4.3.3 SUPPRESS

Working in conjunction with the ACCESS and the ALTER instructions, the SUPPRESS command negates the activity of the ACCESS and ALTER instructions within the operand specified range. The format for the SUPPRESS instruction is:

OPERATION	OPERAND
SUPPRESS	*RANGE*

For instance, the user may want to monitor data stores within memory locations 5000 through 5400 except for addresses 5100 through 5150 and 5200 through 5250. Three instructions will accomplish this:

OPERATION	OPERAND
ALTER	5000 TO 5400
SUPPRESS	5100 TO 5150
SUPPRESS	5200 TO 5250

These instructions have the same effect as the following:

OPERATION	OPERAND
ALTER	5000 TO 5099
ALTER	5151 TO 5199
ALTER	5251 TO 5400

The SUPPRESS instruction reduces execution time by deleting unwanted output.

7.4.4 DUMP

At a particular point in a simulation, the user may want to look at the contents of memory locations or other registers. The DUMP instructions provide a snapshot picture of the F8. These instructions are:

OPERATION	OPERAND
DUMP ALL	
DUMP COUNT	*NUMBER*
DUMP MEM	*RANGE*
DUMP PORT	*NUMBER*
DUMP REG	*RNAME*
DUMP SCRATCH	*RANGE*

Italics imply an optional operand.

7.4.4.1 DUMP ALL

The DUMP ALL instruction will print out the value of the counters, I/O ports, CPU registers, scratch-pad registers, and memory locations.

7.4.4.2 DUMP COUNT *NUMBER*

During simulation of an F8 program, the contents of the counter can be obtained by issuing a DUMP COUNT instruction. The operand refers to a particular counter number; this number must correspond to a counter number defined in the configuration control section. If no operand is present, all counter values will be dumped.

7.4.4.3 DUMP MEM *RANGE*

The DUMP MEM instruction prints out the value of all locations within the specified memory range. For instance:

OPERATION	OPERAND
DUMP MEM	0 TO 100

will output the contents of memory locations 0 to 100. Sixteen memory locations are listed per line. A typical output format will be:

MEMORY ADDRESS 16

VALUE 90 34 12 26 75 23 56 46 98 26 38 48 50 03 33 23

This output means that memory address 16 contains 90, address 17 contains 34, address 18 contains 12, and so forth. The RANGE operand defines the section of memory locations that will be dumped. If no range is given, all memory locations defined in the configuration control section will be printed.

7.4.4.4 DUMP PORT *NUMBER*

If the programmer is interested in observing the contents of an I/O port, a DUMP PORT instruction

can be used. This will print out the present value that is contained in an I/O port. Each I/O port is bidirectional; therefore, the contents of the port is a logical "OR" of the value that is contained in the port buffer and the external connections to that port. The external connections are selected by a PORT instruction defined later in this section. If the operand is not present, all I/O ports defined in the configuration control section will be outputted.

7.4.4.5 DUMP REG *RNAME*

The contents of the accumulator, data counter, ISAR, and status register can be monitored using a DUMP REG instruction. The operand determines the register to be printed. The absence of an operand results in listing the content of all four registers. For instance,

OPERATION	OPERAND
DUMP	*W*

will output the value of the scratchpad register in the following format:

REGISTER *W* VALUE 0

7.4.4.6 DUMP SCRATCH *RANGE*

The scratchpad registers may be monitored with a DUMP SCRATCH instruction. As the name implies, the DUMP SCRATCH instruction outputs the value of the scratchpad registers within the specified range. If no range is given, all sixty-four scratchpad registers will be listed. For example,

OPERATION	OPERAND
DUMP SCRATCH	0 TO 16

will print the value of the first seventeen scratch-pad registers. The register values are listed sixteen to a line in the following format:

SCRATCH ADDRESS 0
VALUE 12 23 84 85 67 34 56 98 38 45 56 03 47 18 45 98

SCRATCH ADDRESS 16
VALUE 56

In this example, scratchpad register 0 contains 12, scratchpad register 1 contains 23, and soforth.

7.4.5 TRACE

The six TRACE commands track the flow of F8 instructions. The simulator offers three tracing modes. In the first, each instruction, regardless of type, is monitored. The simulator will print out after each instruction executes (the TRACE printout is described below). The second mode of trace instructions only tracks branch and I/O instructions. Finally, a trace instruction will monitor only the I/O instructions. The six trace instructions are:

OPERATION	OPERAND
TRACE ON	
TRACE ALL	*RANGE*
TRACE BRANCH	*RANGE*
TRACE IO	*RANGE*
TRACE CLEAR	*RANGE*
TRACE OFF	

Italics imply an optional operand.

7.4.5.1 STANDARD TRACE OUTPUT

All of the Trace instructions produce a standard trace output, as the example in Section 7.7 shows. The listing occurs after instruction execution. The BASE instruction selects the numeric format. The standard trace output includes:

DATA OUTPUT	COLUMN HEADING
1) The cycle number	CYCLE
2) The program counter (set to the next instruction)	PCO
3) The instruction mnemonic (of the completed instruction) An * next to the OP mnemonics implies that a branch was successful.	OP

DATA OUTPUT	COLUMN HEADING
4) The storage being accessed	STORAGE
a) Address of data	ADD
The data may be either from scratchpad (S) main memory (M) or an I/O port (P). For branch instructions, a C will appear for condition codes.	
b) Data Item	DATA
The data item is the 8-bit content of the data source register. For branch instructions, it is the condition code of the branch instruction.	
5) The content of the Accumulator (after instruction execution)	ACC
6) The status register (after instruction execution)	W
The status register will always appear as eight binary bits. The three MSB are zero. The remaining five bits are:	
a) Interrupt Control Bit	I
b) Overflow Bit	O
c) Zero Bit	Z
d) Carry Bit	C
e) Sign	+
7) The contents of the data counter after the instruction executes	DC
8) The contents of the indirect scratchpad address register after the instruction executes appears as two octal digits	ISAR
9) The contents of the stack register after instruction execution	PC1
10) The op code of the instruction	INSTRUCTION
This listing is the actual instruction bytes read from memory.	
11) The pending interrupts and timer contents	INTERRUPT
Pending interrupts are referenced by their priority number and the type of interrupt - I for external, C for the counter. A timer counting down will be listed along with its contents. The timer count down sequence is given in the Appendix. Only five interrupts will be listed in this section. Pending	

interrupts with highest priority are listed first, followed by the highest priority counting timers.

7.4.5.2 TRACE ALL *RANGE*

This instruction turns on the trace operation within the operand address range. For each instruction address within the range, the simulator will produce the standard trace output. If a range is not given, the TRACE ALL command will output after every F8 instruction.

7.4.5.3 TRACE BRANCH *RANGE*

To monitor F8 operation within a software loop, the TRACE BRANCH instruction may be used. This instruction produces a trace output for each branch instruction; in addition to tracing branches, the TRACE BRANCH instruction will also monitor I/O operations. An F8 I/O operation may be one of four instructions: IN, INS, OUT, and OUTS; therefore, writing into the local interrupt control blocks and the F8 counters will also be monitored. The output for the branch and I/O instructions is the same standard trace output. The operand of the TRACE BRANCH instruction is the range of instruction addresses within which the TRACE BRANCH instruction will function. If no range is listed, the TRACE BRANCH will operate for all branch and I/O instructions.

7.4.5.4 TRACE IO *RANGE*

The Trace IO instruction monitors the four F8 I/O instructions IN, INS, OUT, and OUTS. A standard trace output will be produced for each executed I/O instruction within the operand range. If a range is not listed, TRACE IO will monitor every instruction.

7.4.5.5 TRACE CLEAR *RANGE*

The TRACE CLEAR instruction may be used in conjunction with the TRACE ON. It has the effect of negating the trace function for instructions with addresses lying within the operand address range. If no range is listed, the simulator will not trace any instructions.

7.4.5.6 TRACE ON

This instruction will cause the trace output, which was turned off by a TRACE OFF command, to resume.

7.4.5.7 TRACE OFF

The TRACE OFF instruction stops trace printing; however, the setting of all previous TRACE instructions is maintained and may be resumed by a TRACE ON instruction.

If no TRACE instructions appear in the simulator control language, a trace will be performed for each instruction. The trace operation consumes considerable CPU time in the host machine. Thus, an efficient simulation program will only use trace instructions where they are needed.

7.4.6 ENVIRONMENT INSTRUCTIONS

At some point in the simulation, the system surrounding the F8 may interact with the microprocessor. This may be modeled using the environment instruction. Thus, the external world and its interaction with the F8 can be simulated. The environment instructions allow data to be entered into an I/O port and interrupts to be activated. The four environment instructions are:

OPERATION	OPERAND	
INTERRUPT	*NUMBER CYCLE*	(Section 7.4.6.1)
PORT	*NUMBER CYCLE DATA*	(Section 7.4.6.2)
EXTERNAL	*NUMBER* MATCH *DATA*	INTERRUPT*NUMBER CYC* (Section 7.4.6.3)
EXTERNAL	*NUMBER* MATCH *DATA*	PORT *NUMBER CYCLE D* (Section 7.4.6.4)

7.4.6.1 INTERRUPT *NUMBER CYCLE*

The INTERRUPT instruction will dynamically enter an external interrupt into the system. The interrupt structure was defined in the configuration control section. Each interrupt was assigned a port number corresponding to the port address of the interrupt control block. This port number is referenced by the *NUMBER* operand in the INTERRUPT instruction. The cycle during which the external interrupt enters the system is also selected by the operand of this instruction. Thus, when the simulator reaches the corresponding cycle number, an external interrupt request will appear at the corresponding interrupt.

Note that just an interrupt *REQUEST* will appear; the interrupt will be serviced according to the F8 interrupt structure.

7.4.6.2 PORT *NUMBER CYCLE DATA*

Data may be entered into the F8 model with a PORT instruction. The data is an 8-bit byte; this word will appear at the I/O port designated by the operand *NUMBER CYCLE* corresponds to F8 machine cycles. *DATA* will be put on the I/O port at the beginning of the next instruction executed on a cycle equal to or greater than the value of *CYCLE* appearing in the operand. The data will stay valid at the port until the next PORT instruction or EXTERNAL PORT instruction changes the input data. The actual information that is read into the accumulator by an input instruction is the result of a logical OR operation between the *DATA* item and the contents of the I/O buffer. Thus, for the full eight bits to be read without error, the output buffer must contain 00000000. When the simulator initializes the F8 system, the external lines into the I/O ports are set to zero.

7.4.6.3 EXTERNAL *NUMBER* MATCH *DATA* INTERRUPT *NUMBER CYCLE*

An EXTERNAL instruction can be used to enter an interrupt into the simulation dependent on results produced during the simulation. This instruction has an 8-bit byte, *DATA NUMBER*. It monitors a specific I/O port, designated by matching the *DATA* operand with the value contained in the I/O buffer. If, and when, the CPU writes a word into the I/O port which matches *DATA* contained in the instruction operand, an interrupt will be activated as soon as the simulator has reached the corresponding *CYCLE*. The interrupt will occur at the interrupt labeled *NUMBER*. Up to 13 EXTERNAL instructions that may appear in the simulation control text. In effect, if the data in the I/O buffer matches *DATA* in the operand, this instruction will operate as if it were the following:

INTERRUPT *NUMBER CYCLE*

7.4.6.4 EXTERNAL *NUMBER* MATCH *DATA* PORT *NUMBER CYCLE DATA*

This EXTERNAL instruction will enter data into an I/O port dependent on the results produced during operation. Once again, this instruction watches the appropriate I/O port (referenced by the first

NUMBER entry in the operand) waiting for the CPU or a PORT instruction to write into that port a word which matches the first *DATA* operand. If this occurs, the simulator will then put the second *DATA* operand on the I/O port referenced by the second *NUMBER* operand. This data will appear on the port at the beginning of the first instruction occurring on or after the *CYCLE* listed. It will remain valid until a PORT instruction or another EXTERNAL instruction changes the data.

7.4.7 RUN INSTRUCTIONS

The simulator will begin execution of the loaded program when a RUN command is issued. Once the F8 memory and registers have been loaded with the program and data, and after the programmer has selected monitoring modes, the simulation is ready to begin. The RUN instruction specifies the address of the first instruction to be executed, and the simulation begins. The model F8 will execute until the limit specified by the RUN statement (either cycle number or instruction address). When the simulation stops, all F8 registers and memory will remain intact. Therefore, the system can very easily be restarted by another RUN instruction. The three RUN instructions are:

OPERATION	OPERAND
RUN	*ADDRESS* TILL CYCLE *NUMBER*
RUN	*ADDRESS* TILL ADDRESS *VALUE*
RUN	*ADDRESS* TILL CYCLE *NUMBER* ADDRESS *VALUE*

7.4.7.1 RUN *ADDRESS* TILL CYCLE *NUMBER*

This RUN instruction will begin execution with the instruction in location *ADDRESS*. F8 observations during simulation are selected by the simulation control language instructions. Execution will continue until the simulator reaches the CYCLE limit.

RUN *ADDRESS* TILL ADDRESS *VALUE*

The simulator will begin execution with the instruction specified by the first *ADDRESS* listed in the operand and continue till it has performed the instruction specified by the second operand, *ADDRESS*. Thus,

the first operand *ADDRESS* sets the program counter (PCO). The last instruction simulated is contained at the location selected by the second *ADDRESS* operand.

7.4.7.2 RUN *ADDRESS* TILL CYCLE *NUMBER* ADDRESS *VALUE*

This instruction sets the simulator program counter to the address determined by the first *ADDRESS* operand. Execution will commence. The simulation will stop when it reaches either of two conditions:

1) the *CYCLE* limit has been reached

2) the execution of the instruction specified by the second *ADDRESS* operand has been completed.

7.4.8 HOUSEKEEPING INSTRUCTIONS

There are four simulator control instructions involved with controlling the output. These are:

OPERATION	OPERAND
BASE	*BIN, OCT, DEC, HEX*
TITLE	*'HEADING'*
MAXCPU	*VALUE*
END	

7.4.8.1 BASE *BIN, OCT, DEC, HEX*

This command controls the format of the numeric simulation output. The four choices are binary *BIN*, octal *OCT*, decimal *DEC*, or hexadecimal *HEX*. Except where noted in the text, the BASE instruction selects the output mode.

7.4.8.2 TITLE *'HEADING'*

The TITLE command will head the top of each output page with the *HEADING* appearing in the operand.

7.4.8.3 MAXCPU *VALUE*

This instruction limits the amount of time that the host computer executes. The *VALUE* operand is a decimal number representing the maximum seconds that the CPU can execute.

7.4.8.4 END

The completion of the simulation control language statements is marked by an END instruction.

OPERATION	OPERAND
END	

7.5 OUTPUT FILES

After the simulator has completed execution, it will generate one file, F8TRACE. This file is a listing of the simulator's output; it includes the input statement from the user as well as the resulting simulator output. The sample program in Section 7.7 are examples of the output files.

7.6 SIMULATOR MESSAGES

The simulator will output two types of error messages: warning messages and fatal errors. The warning messages alert the user that he is departing from standard F8 organization. They do not end the simulation. Fatal errors, however, pose a different problem for the simulator. These messages occur when the simulator cannot proceed with the information provided. Therefore, they result in a termination of the RUN.

7.6.1 WARNING MESSAGES

7.6.1.1 WARNING - DUPLICATE INTERRUPT NUMBER

Two interrupts with the same number were specified with INTERRUPT instructions.

7.6.1.2 WARNING - INTERRUPT NUMBER NONSTANDARD

The interrupt number operand in an INTERRUPT instruction does not conform to the standard F8 configuration.

7.6.1.3 WARNING - DUPLICATE PORT OR COUNTER NUMBER

The operand of a PORT instruction appeared in a previous PORT or INTERRUPT instruction.

7.6.1.4 WARNING - COUNTER NUMBER NONSTANDARD

The counter number specified in an INTERRUPT instruction does not conform to the F8 configuration.

7.6.1.5 WARNING - INTERRUPT OR COUNTER NUMBER NONSTANDARD

The operand of the INTERRUPT instruction does not conform to standard F8 configurations.

7.6.1.6 WARNING - PORT NUMBER NONSTANDARD

The operand of the PORT instruction does not conform to standard F8 configurations.

7.6.1.7 WARNING - SEQUENCE ER AFTER CARD

If the assembly text was sequenced and if the simulator detects an error in the sequence numbering, this warning will be issued.

7.6.1.8 WARNING - PROGRAM OVERLAY - *ADDRESS*

The user loaded two words into the same F8 memory address.

7.6.1.9 WARNING - PROGRAM LOADED INTO RAM

The user loaded a section of program into the RAM. Programs are normally loaded into ROM. (These ROM sections were detailed in the configuration control section.)

7.6.1.10 WARNING - OUTSIDE MEMORY

The CLEAR, DUMP, or SET instructions listed an address range which extends beyond the memory boundries configured.

7.6.1.11 WARNING - ALTER ATTEMPT ON ROM

The ALTER instruction was issued for a memory range which included ROM memory locations.

7.6.1.12 WARNING - INPUT CHANGED DURING INPUT OPERATION

The PORT instruction to input data was issued for a cycle during which an F8 INPUT instruction was being executed. In this case, the simulator does not use this new data.

7.6.1.13 PORT NOT ACCESSABLE

The I/O port was not listed in the configuration control instruction. The instruction will be treated like a NO OP.

7.6.2 FATAL ERRORS

7.6.2.1 SYNTAX ERRORS

All syntax errors result because the simulator can not comprehend the simulator instruction. The line following the message will include the syntax error. An X will appear under the character which the simulator did not recognize.

7.6.2.1.1 SYNTAX ERROR CFG

A syntax error appeared in a configuration control language instruction.

7.6.2.1.2 SYNTAX ERROR SCL

A syntax error appeared in the simulation control language instruction.

7.6.2.1.3 SYNTAX ERROR IP

A syntax error occurred during an INTERRUPT, PORT, or EXTERNAL instruction.

7.6.2.1.4 SYNTAX ERROR CDS

A syntax error occurred in either a CLEAR, DUMP, or SET instruction.

7.6.2.1.5 SYNTAX ERROR LOD

A syntax error occurred in the assembly input file.

7.6.2.2 LOAD TIME ERRORS

Load timer errors occur after the configuration control statements have been read. At this point, the simulator is loading the program into the simulation memory.

7.6.2.2.1 DF8 LOD ER

An error in the INPUT assembly text.

7.6.2.2.2 LABEL OV

The label table has overflowed (it holds only 1111 labels).

7.6.2.2.3 PROGRAM STORAGE OV

The user tried to load a program larger than the F8 configuration. The address for which the error occurred is given.

7.6.2.2.4 PROGRAM OUTSIDE STORAGE

The program is loaded into an address outside of the F8 configuration. The address at which the error occurred is given.

7.6.2.2.5 SIMULATOR STORAGE ERROR

The simulator does not have enough storage for the user.

7.6.2.3 CONFIGURATION MESSAGES

These messages occur when the programmer is configuring the F8 system to be modeled.

7.6.2.3.1 PORT OR COUNTER

The simulator will only handle eight port and/or counter numbers.

7.6.2.3.2 MEMORY OV

The simulator allows only 16K of memory.

7.6.2.3.3 MEMORY OVERLAP

The memory has been configured with two memory areas overlapping.

7.6.2.3.4 READ OV

Too many continuation cards appear in the input file.

7.6.2.4 RUN MESSAGES

RUN messages occur after the simulation program has set up the model F8 and has begun to simulate the execution of the user's program.

7.6.2.4.1 ATTEMPT TO REFERENCE OUTSIDE STORAGE

A simulation control command was issued with an operand address outside of storage.

7.6.2.4.2 ILLEGAL OP CODE

An F8 op code could not be understood. The op code will be given.

7.6.2.4.3 ADDRESS ERROR

An address in the program could not be interpreted. The address will be given.

7.6.2.4.4 CONSTANT ERROR

A constant in the program could not be understood. The constant will be given.

7.6.2.4.5 ATTEMPT TO ALTER ROM - ADDRESS - CYCLE

An attempt was made, during execution, to write data in a ROM location. The address and cycle at which this occurred will be listed.

7.6.2.4.6 ATTEMPT TO STORE VALUE GREATER THAN 255

A storage was attempted on a number which is larger than 8 bits.

7.6.2.5 PROGRAM ERRORS

If the simulator reaches a point where it does not know what to do next, one of the following error messages may be issued. The messages indicate an error has occurred in the register values.

F8 RA ER

F8 RB ER

F8 RC ER

F8 RD ER

F8 PC1 ER

F8 CC ER

DEC BIN ER

TRACE ER

7.6.2.5.1 I/O ER UNIT

An error occurred within the machine hardware resulting from an I/O unit.

7.6.3 CONCLUSION MESSAGES

At the end of a simulation run, one of the following error messages will appear.

RUN TIME EXCEEDED SEC XXXX

The run time as greater than the amount specified by the RUN instruction.

RUN COMPLETED SEC XXXX

The simulation run was completed without any fatal errors.

ERROR EXIT SEC XXXX

A fatal error occurred and the simulator terminated execution.

XXXX is the number of seconds used for execution.

INSTRUCTION	NUMBER OF CYCLES	INSTRUCTION	NUMBER OF CYCLES	INSTRUCTION	NUMBER OF CYCLES	INSTRUCTION	NUMBER OF CYCLES
ADC	10	LR A,QU	4	XI	10	IN	16
AI	10	LR QU,A	4	XM	10	INS	8 if ports 0 or 1 16 otherwise
AM	10	LR AL,A	4	XS	4	OUT	16
AS	4	LR AL,A	4	SL1	4	OUTS	8 if ports 0 or 1 16 otherwise
LNK	4	LR P,K	16	SL4	4		
COM	4	LR K,P	16	SR1	4		
CI	10	LR A,IS	4	SR4	4		
CM	10	LR IS,A	4	BR7	10 if jump 8 if not		
DS	4	LR PO,Q	16	BF	14 if jump 12 if not		
INC	4	LR Q,DC	16	BT	14 if jump 12 if not		
DCI	24	LR DC,Q	16	BM	14 if jump 12 if not		
LI	10	LR H,DC	16	BNC	14 if jump 12 if not		
LIS	4	LR DC,H	16	BNZ	14 if jump 12 if not		
LM	4	LR W,J	8	BNO	14 if jump 12 if not		
LISL	4	LR J,W	4	BP	14 if jump 12 if not		
PK	16	AMD	10	BZ	14 if jump 12 if not		
LR A,SR#	4	ASD	8	BR	14 if jump 12 if not		
LR SR#,A	4	NI	10	JMP	22		
LR A,KU	4	NM	10	PI	26		
LR KU,A	4	NS	4	POP	8		
LR A,KL	4	OI	10	DI	8		
LR KL,A	4	OM	10	EI	8		

TABLE 7.3

7.7 SIMULATION EXAMPLE

This section gives an example of a F8 Simulation Run. Two input files are required for the simulation. The first file, F8TXT, is generated by the F8 Cross Assembler. The second file, F8SCL, is produced by the user. The F8SCL file used in this example is shown in Figure 7.1. The F8 Cross Simulator generates an output file, F8TRACE, shown in Figure 7.2.

```
      ROM  0
      END
     BASE HEX
      SET SCRATCH 0'30' TO 0'35' H'22' H'76' H'89' H'54' H'54' H'45'
      SET SCRATCH 0'36' TO 0'37'  H'89' H'00'
      SET SCRATCH 0'40' TO 0'45' H'33' H'89' H'79' H'20' H'33' H'33'
      SET SCRATCH 0'46' TO 0'47'  H'27' H'45'
      DUMP MEM 0 TO 20
      DUMP SCRATCH
     TRACE BRANCH
      RUN  0 TILL CYCLE H'400'
      DUMP SCRATCH
      END
```

Figure 7.1

Figure 7.1 File F8SCL

ROM 0

Figure 7.2 Page 1of 3
File F8TRACE Produced By The F8 Cross Assembler

```
BASE HEX
SET SCRATCH O'30' TO O'35' H'22' H'76' H'89' H'54' H'54' H'45'
SET SCRATCH O'36' TO O'37' H'89' H'00'
SET SCRATCH O'40' TO O'45' H'33' H'89' H'79' H'20' H'33' H'33'
SET SCRATCH O'46' TO O'47' H'27' H'45'
DUMP MEM 0 TO 20
MEMORY ADDRESS 0 VALUE 20 0 6F 21 0 51 63 4C 24 66 D1 52 70 51 42 82
MEMORY ADDRESS 10 VALUE C 64 24 66 DC
DUMP SCRATCH
SCRATCH ADDRESS 0 VALUE 75 D8 F7 2A 4F 5C 5F EF A1 CC 22 9A 68 2 64 48
SCRATCH ADDRESS 10 VALUE 29 6D 1B CB CB A1 9D 4 22 76 89 54 54 45 89 0
SCRATCH ADDRESS 20 VALUE 33 89 79 20 33 33 27 45 BE D3 41 1E 6A 67 B2 87
SCRATCH ADDRESS 30 VALUE EC C6 54 FC F3 D7 7E 5E C7 56 1 2 FF E8 73 86
TRACE BRANCH
RUN 0 TILL CYCLE H'400'
```

CYCLE	PC0	OP		STORAGE ADD	STORAGE DATA	ACC	W IOZC+	DC	ISAR (OCT)	PC1	INSTRUCTION	INTERRUPT & COUNT
46	11	BT		C	2	0	00000101	7E8E	37	0	82 C	
6C	18	BT		C	2	45	00001001	7E8E	46	0	82 A	
78	6	BR7	*			45	00001001	7E8E	46	0	8F ED	
AC	11	BT		C	2	89	00000000	7E8E	36	0	82 C	
D2	21	BT	*	C	2	16	00000011	7E8E	45	0	82 A	
EC	18	BF	*	C	0	1	00000001	7E8E	45	0	90 F3	
FA	6	BR7	*			1	00000001	7E8E	45	0	8F ED	
12E	11	BT		C	2	46	00000001	7E8E	35	0	82 C	
154	18	BT		C	2	79	00000001	7E8E	44	0	82 A	
160	6	BR7	*			79	00000001	7E8E	44	0	8F ED	
194	11	BT		C	2	54	00000001	7E8E	34	0	82 C	
1BA	18	BT		C	2	87	00000000	7E8E	43	0	82 A	
1C6	6	BR7	*			87	00000000	7E8E	43	0	8F ED	
1FA	11	BT		C	2	54	00000001	7E8E	33	0	82 C	
220	18	BT		C	2	74	00000001	7E8E	42	0	82 A	
22C	6	BR7	*			74	00000001	7E8E	42	0	8F ED	
260	11	BT		C	2	89	00000000	7E8E	32	0	82 C	
286	21	BT	*	C	2	68	00000011	7E8E	41	0	82 A	
2A0	18	BF	*	C	0	1	00000001	7E8E	41	0	90 F3	
2AE	6	BR7	*			1	00000001	7E8E	41	0	8F ED	
2E2	11	BT		C	2	77	00000001	7E8E	31	0	82 C	
308	21	BT	*	C	2	66	00001011	7E8E	40	0	82 A	
322	18	BF	*	C	0	1	00000001	7E8E	40	0	90 F3	
330	6	BR7	*			1	00000001	7E8E	40	0	8F ED	
364	11	BT		C	2	23	00000001	7E8E	30	0	82 C	
38A	18	BT		C	2	56	00000001	7E8E	47	0	82 A	
396	1A	BR7				56	00000001	7E8E	47	0	8F ED	
39E	21	BF	*	C	0	56	00000001	7E8E	47	0	90 6	
3B8	18	BF	*	C	0	1	00000001	7E8E	47	0	90 F3	
3C6	1A	BR7				1	00000001	7E8E	47	0	8F ED	
3CE	21	BF	*	C	0	1	00000001	7E8E	47	0	90 6	
3E8	18	BF	*	C	0	2	00000001	7E8E	47	0	90 F3	
3F6	1A	BR7				2	00000001	7E8E	47	0	8F ED	

Figure 7.2 Page 2 of 3

File F8TRACE Produced By The F8 Cross Simulator

```
12/12/74    14:27:34                                                                      F8S -- VERSION  41 -- 11/08/74

   CYCLE    PCO    OP              STORAGE       ACC        W            DC        ISAR      PC1  INSTRUCTION                INTERRUPT & COUNT
                                ADD      DATA                 IOZC+                 (OCT)                                                PAGE   3
     3FE      21   BF     *         C  0          2        00000001      7E8E        47              0  90  6
                   DUMP SCRATCH
 SCRATCH ADDRESS            0     VALUE        75    2   23   2A   4F   5C   5F   FF   A1   CC   22   9A   68    2   64   48
 SCRATCH ADDRESS           10     VALUE        29   6D   1B   CB   CB   A1   9D    4   22   76   89   54   54   45   89    0
 SCRATCH ADDRESS           20     VALUE        56   66   68   74   87   79   16   45   BE   D3   41   1E   6A   67   B2   87
 SCRATCH ADDRESS           30     VALUE        EC   C6   54   FC   F3   D7   7E   5E   C7   56    1    2   FF   E8   73   86
                   END

 RUN COMPLETED       2.01 SEC
```

Figure 7.2 Page 3 of 3

File F8TRACE Produced By The F8 Cross Simulator

APPENDIX -TIMER COUNTS

CONTENTS OF COUNTER	COUNTS TO INTERRUPT
FE	254
FD	253
FB	252
F7	251
EE	250
DC	249
B8	248
71	247
E3	246
C7	245
8E	244
1D	243
3B	242
76	241
ED	240
DA	239
B4	238
68	237
D1	236
A3	235
47	234
8F	233
1F	232
3F	231
7E	230
FC	229
F9	228
F3	227
E6	226
CD	225
9B	224
36	223
6D	222
DB	221
B6	220
6C	219
D9	218
B2	217
64	216
C8	215
91	214
23	213
46	212
8D	211
1B	210
37	209
6F	208
DF	207
BE	206
7D	205
FA	204
F5	203
EA	202
D4	201
A9	200
52	199

CONTENTS OF COUNTER	COUNTS TO INTERRUPT
A4	198
49	197
92	196
25	195
4A	194
94	193
29	192
53	191
A6	190
4D	189
9A	188
34	187
69	186
D3	185
A7	184
4F	183
9E	182
3C	181
78	180
F0	179
E0	178
C1	177
82	176
04	175
09	174
12	173
24	172
48	171
90	170
21	169
42	168
85	167
0A	166
14	165
28	164
51	163
A2	162
45	161
8B	160
17	159
2E	158
5D	157
BB	156
77	155
EF	154
DE	153
BC	152
79	151
F2	150
E4	149
C9	148

CONTENTS OF COUNTER	COUNTS TO INTERRUPT
93	147
27	146
4E	145
9C	144
38	143
70	142
E1	141
C3	140
86	139
0C	138
18	137
31	136
63	135
C6	134
8C	133
19	132
33	131
67	130
CE	129
9D	128
3A	127
74	126
E9	125
D2	124
A5	123
4B	122
96	121
2D	120
5B	119
B7	118
6E	117
DD	116
BA	115
75	114
EB	113
D6	112
AD	111
5A	110
B5	109
6A	108
D5	107
AB	106
56	105
AC	104
58	103
B1	102
62	101
C4	100
88	99

CONTENTS OF COUNTER	COUNTS TO INTERRUPT
11	98
22	97
44	96
89	95
13	94
26	93
4C	92
98	91
30	90
61	89
C2	88
84	87
08	86
10	85
20	84
40	83
81	82
02	81
05	80
0B	79
16	78
2C	77
59	76
B3	75
66	74
CC	73
99	72
32	71
65	70
CA	69
95	68
2B	67
57	66
AE	65
5C	64
B9	63
73	62
E7	61
CF	60
9F	59
3E	58
7C	57
F8	56
F1	55
E2	54
C5	53
8A	52
15	51
2A	50
55	49

CONTENTS OF COUNTER	COUNTS TO INTERRUPT
AA	
54	48
A8	47
50	46
A0	45
41	44
83	43
06	42
0D	41
1A	40
35	39
6B	38
D7	37
AF	36
5E	35
BD	34
7B	33
F6	32
EC	31
D8	30
B0	29
60	28
C0	27
80	26
00	25
01	24
03	23
07	22
0F	21
1E	20
3D	19
7A	18
F4	17
E8	16
D0	15
A1	14
43	13
87	12
0E	11
1C	10
39	9
72	8
E5	7
CB	6
97	5
2F	4
5F	3
BF	2
7F	1
FE	0