THE DESIGN PROPOSAL OF A 16-BIT MICROPROGRAMMED STACK MACHINE

by

DON RHEA HUSH

B. S., Kansas State University, 1980

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:

[illegible]

Major Professor

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

ACKNOWLEDGEMENTS

First and foremost I would like to thank Dr. E. Haft for his guidance and hard work that helped make this thesis possible. I think through our combined effort we managed to learn a great deal about the design of computers that is often times overlooked in the literature. I would also like to thank the Electrical Engineering Department of Kansas State University for its financial and emotional support.

1 Statement of Objectives

This paper presents the design of a relatively simple, yet complete computer system. The following is a list of objectives set forth in the design of this system.

1. The primary objective is to develop a complete computer system that can be built and tested by students interested in computer design. Once built, the system would provide an excellent educational tool in such areas as microprogramming, machine instruction sets, and computer design.

2. The hardware design should favor that of a stack machine.

3. The Control Unit should be microprogrammed with a writable Control Store. In addition, provisions should be made so that this memory can be easily modified from a Console Terminal, thus providing for ease of firmware development.

4. The hardware design should be relatively straight forward and easy to follow.

The system that follows contains some characteristics seen mostly in large computers, and others seen only in minicomputers and microprocessors. Although it may appear to have many restrictions, it can be seen to exhibit some very powerful features. A strong attempt has been made to use the power of simplicity to develop a complete system that runs at a reasonable speed.

2 Definitions and Terminology

The following defines the symbols and abbreviations used throughout this paper. Many of them will reflect no useful operation to the reader until they're encountered in the text. All are presented here, however, to provide a universal reference.

Logic Definitions

$\wedge$ = Logical AND

$\vee$ = Logical OR

high = 1 = TRUE

low = 0 = FALSE

GO = active high signal, the function indicated by the signal name, GO , is activated when the signal is a 1.

$\overline{\text{GO}}$ = active low signal

Abbreviations

msb - most significant bit

lsb - least significant bit

MSB - most significant byte

LSB - least significant byte

IR - Instruction Register, or the Macroinstruction Register (they are the same)

PLR - Pipeline Register (could also be called the micro-instruction register)

CS - Control Store, also referred to as the Microprogram Memory

ALU - Arithmetic and Logic Unit; Throughout this paper the term ALU is most often used in reference to the entirety of hardware designed to help perform arithmetic and logic functions, i.e. the Stack Tender/Register Select circuit, the Data Select/Sign Extend logic, and the 2901s are all part of the ALU.

ALFU - Arithmetic and Logic Function Unit; the ALFU is a set of logic within the 2901s which actually performs arithmetic and logical operations.

I/O - Input/Output

PC - Program Counter

SP - logical Stack Pointer

MSP - Memory Stack Pointer

TOS - A 2-bit counter which points to the top of the Fast Stack

TOS register - the 2901 register that currently holds the value at the top of the Fast Stack.

IX - Index Register

MAR - Memory Address Register

CPU - Central Processing Unit; In this paper the term CPU encompasses both the Control Unit and the ALU.

FS - Fast Stack

EA - Effective Address

NA - Next Address; the NA is the address of the next microinstruction fetched from the CS.

ns - nanoseconds, or 10^{-9} seconds

Definitions

cycle - one microinstruction cycle, 300ns

period - one of three time divisions of a cycle, i.e. each cycle is divided into three 100ns periods

F_i - the ith bit in the F field

$F_{i,j}$ - the ith and jth bit in the F field

F_{i-j} - bits i through j in the F field

3 An Overview

The computer presented in this paper is a 16-bit machine. That is, all registers in the Arithmetic and Logic Unit (ALU) are 16 bits wide. There are a total of 16 such registers. Four of these function as a "Fast Stack" (FS), which is like a small cache memory for the top four locations in the stack. The remaining 12 registers perform the functions of a Program Counter, Stack Pointer, Index Registers, and any other functions required by the machine instruction set. ALU operations include the basic arithmetic functions of addition and subtraction along with the logical functions OR, AND, EXCLUSIVE OR, and EXCLUSIVE NOR. Integer multiplication and division are easily implemented in firmware due to the presence of an additional ALU register called the Q-register. The Q-register can easily be chosen to function as a 16-bit extension of any ALU register. Hardware multiplexing provides the capability of performing ten basic shift and rotate operations, each of which can be performed on both single and double width operands.

The Control Unit in this machine is based on a microprogrammed architecture. Thus, machine language instructions, (hereafter referred to as macroinstructions), are carried out via a sequence of micro-instructions. Microinstructions in this machine exhibit a dominantly horizontal structure, which is reflected in the relatively large width of the Control Store word. A microinstruction sequencer supplies the microprogram memory (Control Store) with a 12 bit address, thus providing up to 4K possible microinstructions. The basic microinstruction cycle is 300ns, comparable to many of today's mini-computers.

Main memory consists of up to 64K bytes of static semiconductor storage. Memory is accessed via a 16 bit wide communication Bus which carries both address and data. Data can be accessed in both "word" (16 bits) and "byte" modes. To accomplish this, the hardware treats all memory addresses as byte pointers. When memory is accessed in the word mode the least significant bit of the address is ignored. Thus, all words are accessed at even byte boundaries. In the byte mode, all Bus information is transferred over the lines that carry the 8 least significant bits of data. Information transferred between the most significant byte of memory in this mode is multiplexed onto the least significant data lines at the memory - Bus interface. When writing a byte to memory, the 8 most significant data lines are ignored. When reading from memory these lines are 3-stated. Special hardware at the ALU input allows these lines to be either set to zero, or serve as a sign extension of the least significant data byte.

The type of I/O presented in this paper is most commonly referred to as Direct or Programmed I/O, which implies that all I/O operations are carried out under CPU control. Direct I/O devices in this machine are said to be non-memory mapped. That is, a separate Bus control line is used to signify a transaction with one of them as opposed to main memory. Each I/O device is addressed according to its device code. Input and output devices are treated separately and thus require different device codes. I/O devices are capable of interrupting the CPU from any one of the 16 interrupt levels. Interrupts are serviced on a priority basis which is enforced in hardware. The design of a Direct Memory Access, (DMA), device for this computer requires further development which is beyond the scope of this paper.

4 A Block Diagram View

4.1 Arithmetic and Logic Unit

A block diagram of the computer is shown in Figure 4-1. The 16-register ALU is centered around the 2901A, which is a 4-bit slice. Each slice contains 16 general purpose registers, a multi-function ALU, and an additional Q-register which is used primarily for multiplication and division routines. Carry look-ahead is implemented by the 74S182 look-ahead carry generator chip. Various shifts and rotates are created by simply multiplexing the shifted inputs and outputs of the 2901s. Most of the ALU status signals are provided by the 2901. These include the carry, sign bit, 2's complement overflow, and zero result.

A special circuit called the Stack Tender is designed to make the first four 2901 registers function as a Fast Stack. This means that as many as 4 of the top entries in the stack may actually reside in fast ALU registers. Thus, the average speed of stack operations is greatly increased.

The logic network entitled "Data Select/Sign Extend" provides for input to the ALU and its registers from a variety of sources. The only 16-bit source is the system Bus. All other sources (8-bit sources) enter the ALU through the least significant byte. At times the least significant byte of the Bus will be the only part carrying valid information. This happens when a byte is read from memory or an input is received from an 8-bit device. Any 8-bit source to the ALU can be sign extended to 16-bits. If it's not sign extended, the upper 8 bits are set to zero.

Computer Block Diagram

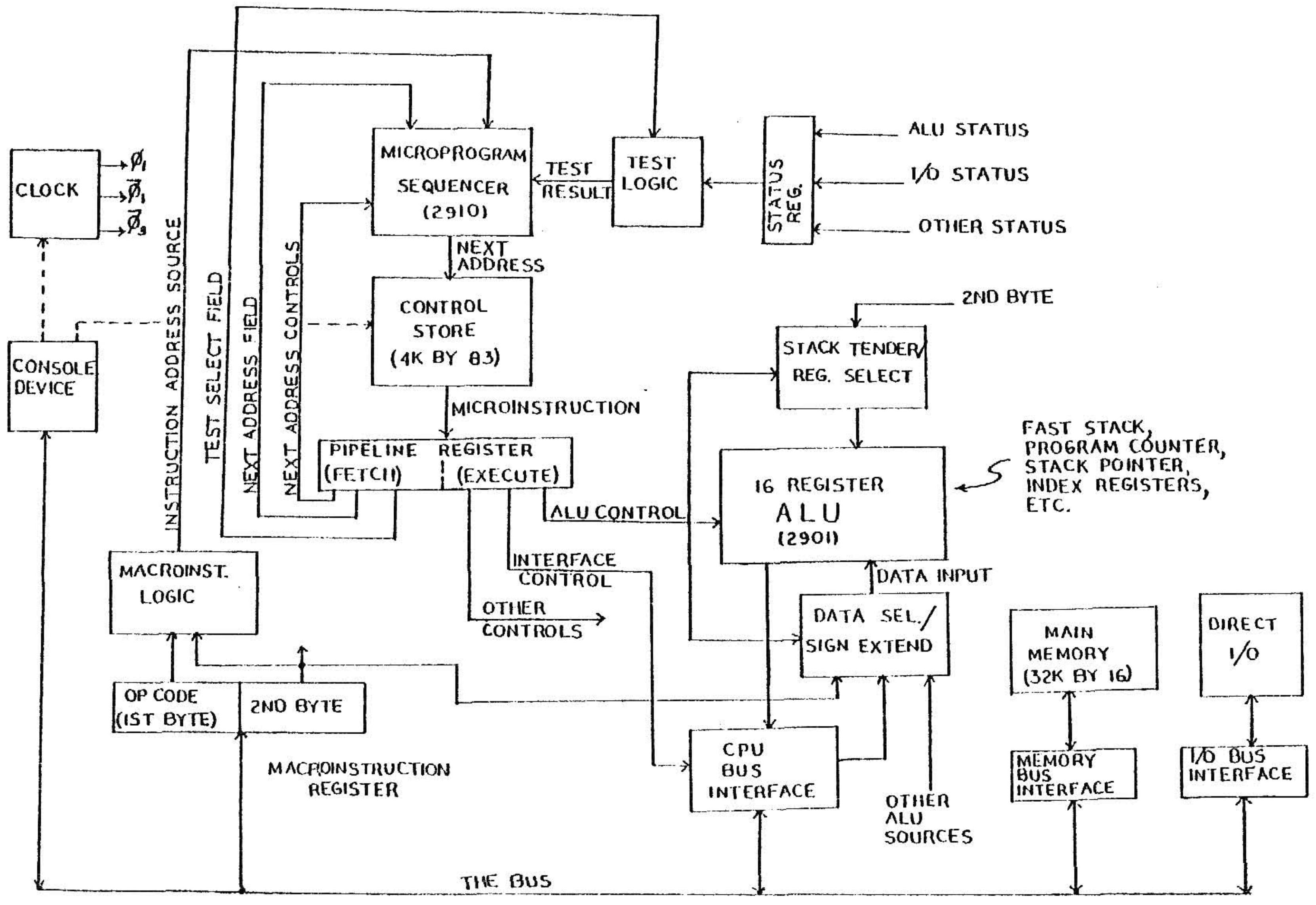


Figure 4-1

4.2 Main Memory

The main memory can have up to 32K 16-bit locations. This memory is designed using Hitachi's HM6116P-3 chip, a 2K word by 8-bit high speed static CMOS RAM with an access time of 150ns. The memory design includes a 16-bit Memory Address Register (MAR). The upper 15 bits are always interpreted as a word address, while the least significant bit (lsb) is used to choose between bytes of the selected word when the byte mode is specified. In the word mode the lsb is ignored. This forces words to be accessed at even byte boundaries independent of the lsb value. There is no Memory Buffer Register for this system. Memory is accessed without a handshake, much like typical microprocessor systems. To read, the CPU sends an address in one cycle and receives data in the next. Writing is similar in that the address is sent in one cycle followed by the data in the next. The timing for this memory has been carefully worked out with the 6116's. Although this type of design lacks generality, it simplifies the CPU, Bus, and memory design significantly.

4.3 The Console

There are two types of I/O pictured in Figure 4-1. The Console I/O device allows the user to communicate directly with main memory. It also has such features as run/halt and single step. These are implemented at the microinstruction level rather than the macroinstruction level. When the machine is halted, the Console has the capability of reading and writing directly to the Control Store. In fact, it is through the Console that this memory is initialized during start-up. All of the operations performed by the Console are carried out with the CPU in halt mode. Only one Console device exists in the system.

4.4 Direct I/O

The other type of I/O is Direct I/O. Devices like those shown in Figure 4-1 are the target of all programmed I/O. Many such devices may exist within the system. The CPU communicates with these devices over the system Bus. A simple handshake circuit is required to synchronize data transfer. Although these devices are connected to the system Bus, they are unable to communicate directly with memory. All data transfers between I/O and memory must first pass through the CPU. Each Direct I/O device has the capability of interrupting the CPU on 1 of 16 interrupt levels. The CPU can however mask interrupts from any or all interrupt levels by enabling the appropriate interrupt mask bits. Interrupts are acknowledged by the CPU on a priority basis. This is accomplished in hardware with a daisy chain. Associated with each device is a Device Code with which it responds once its interrupt has been acknowledged. Input and Output devices are treated separately, each has its own Device Code. The CPU uses the Device Code to determine the source or destination of all I/O operations. An I/O device is selected by placing its Device Code on the Bus concurrently with the appropriate set of Bus control signals. I/O data transfers occur in the cycle directly following the Device Code, much in the same way memory data transfers follow memory addresses.

4.5 The Control Unit

The Control Unit in this machine is microprogrammed. That is, the sequence of operations required to fetch and execute macroinstructions is carried out by a series of microinstructions. Microinstructions are kept in the microprogram memory, often referred to as the Control Store, or simply CS (see Figure 4-1). The CS in this machine has 4K locations, each of which is 88 bits wide. It is composed of the same type of memory chips used in main memory, 6116's, which for this purpose is slow but adequate. When a microinstruction is to be executed, it is placed in the Pipeline Register.

Every microinstruction has two major parts. The first part provides the fundamental control signals required by the other units in the computer. Examples of such signals include the shift, carry, add, and read/write controls for the ALU and memory. The second part of every microinstruction helps determine the CS address for the microinstruction to follow. For simplicity, this address will be referred to as the Next Address. The operation is such that the fetch of the next microinstruction is overlapped with the execution of the current one, hence the concept of pipelineing.

The Next Address is a 12-bit number supplied by the Microinstruction Sequencer, also called the Next Address Generator. The Next Address Controls of the current microinstruction, coupled with the selected Test Result provide all the information necessary for the Sequencer to choose the Next Address. The Microinstruction Sequencer in this machine is a bipolar LSI device, the 2910, which is a member of the 2900 microcomputer logic family.

There are a total of 5 possible sources for the Next Address. Two of these are shown in Figure 4-1; the Instruction Address Source (from the Macroinstruction Logic), and the Next Address Field (from the Pipeline Register). The other three reside in the 2910. The first of these is referred to as the MicroProgram Counter, or μPC. It performs a similar function to that of the Program Counter for main memory. The 2910 also contains a 4 register stack, providing for the capability of microprogram subroutines. The third source of the Next Address in the 2910 is a general purpose counter/register. Although there are a total of 5 Next Address sources, only 2 of the 5 are potential candidates at any given time. The 2910 uses the Test Result to determine which of the two will actually supply the Next Address. The Test Result may be any one of the machine status bits, selected by a field in the Pipeline Register via a hardware multiplexer. The decision making capability of the entire machine is based on the ability of the 2910 to make a 2-way jump in a microprogram routine based on the selected Test Result. A detailed explanation of the 2910 operation is given when the Control Unit hardware is presented in detail.

4.6 The Clock and Timing

This machine has a microinstruction cycle time of 300ns. This cycle is divided into 3 equal periods of 100ns. The clock signals for the system are as shown in Figure 4-2. These signals, ϕ_1, ϕ_2, and ϕ_3, along with their complements (which are also generated, but not shown), provide the necessary timing for the entire system.

The rising edge of ϕ_1 signifies the beginning of a microinstruction

System Clocks

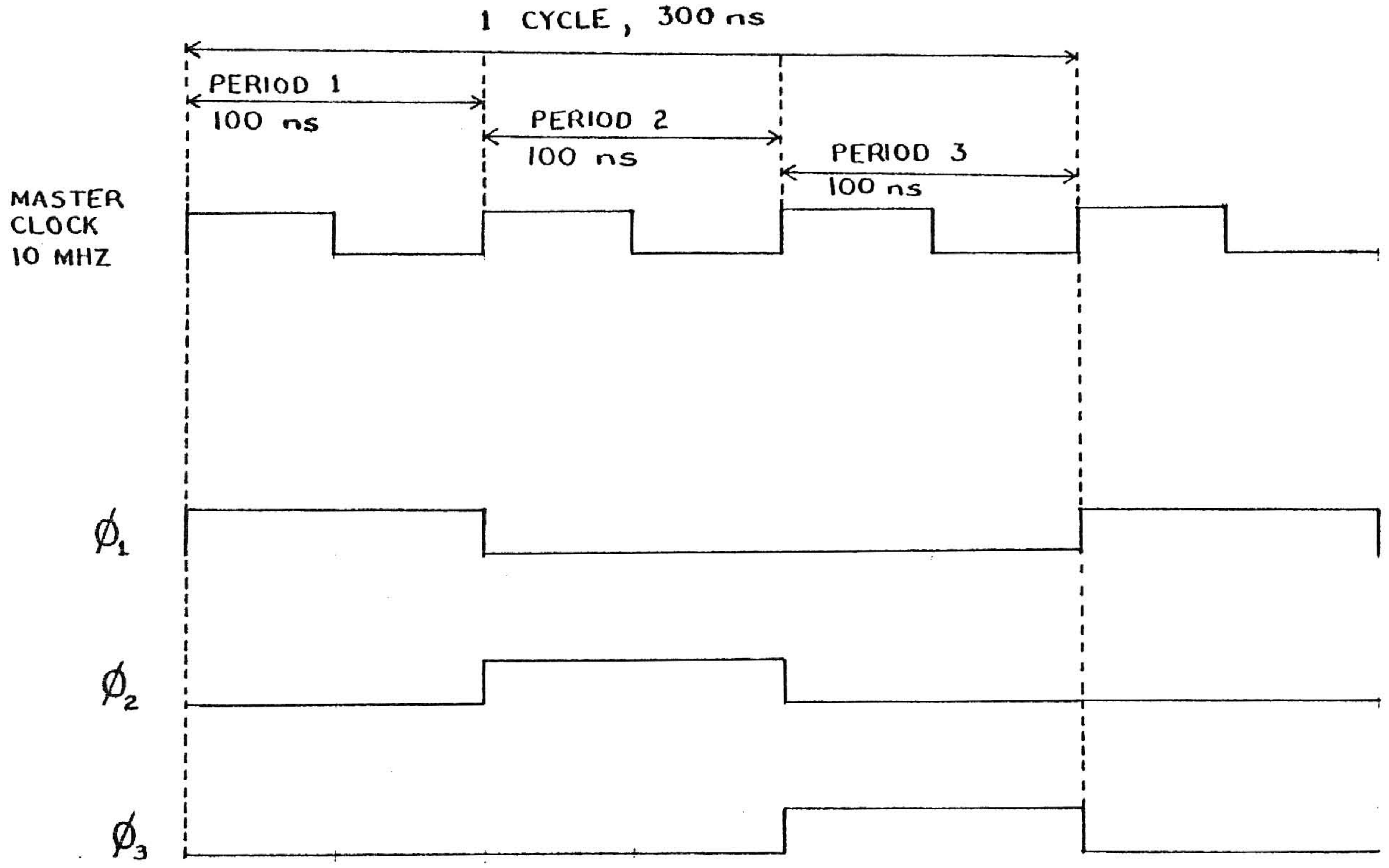


Figure 4-2

period. This edge is used to load the Pipeline Register with the new microinstruction. It is also used to load the Status Register with the results of the previous microinstruction. In addition, $\emptyset_1$ and its complement, $\overline{\emptyset}_1$, are used to synchronize all information transfer on the system Bus (see Figure 4-3). Information is transferred on the Bus during periods 2 and 3 (most likely by enabling a set of 3-state buffers). As soon as $\emptyset_1$ goes high, signifying the beginning of period 1 for the next microinstruction, controls are provided to disable these 3-state buffers, and hence remove the information from the Bus. These controls are delayed slightly with respect to $\emptyset_1$ due to some unavoidable control logic. The 3-state buffers themselves contribute an additional delay. Thus, the information placed on the Bus by Device #1 will not be effectively removed until sometime during period 1 of the next microinstruction. Therefore period 1 is set aside to compensate for these delays. It is not until period 2 that Device #2 will begin to place its information on the Bus. In this way, the potential overlap of information from both devices on the Bus is avoided.

$\overline{\emptyset}_3$, the complement of $\emptyset_3$, is the clock signal used by the ALU. For this reason it is referred to as the Execution Clock.

The Console Device has ultimate control over the system clock signals. The Console can halt the CPU by simply stretching these signals, i.e. holding them at a fixed level. The CPU is always halted at the beginning of a microinstruction cycle. That is, $\emptyset_1$ is held high while $\emptyset_2$ and $\emptyset_3$ are held low. In the halt mode, a Single Step is carried out by allowing the clocks to proceed through one complete cycle, after which they are again halted. In the Run Mode, the clocks are normally allowed to proceed in an undisturbed manner. $\overline{\emptyset}_3$, the Execution Clock, is the only exception

Bus Timing Example

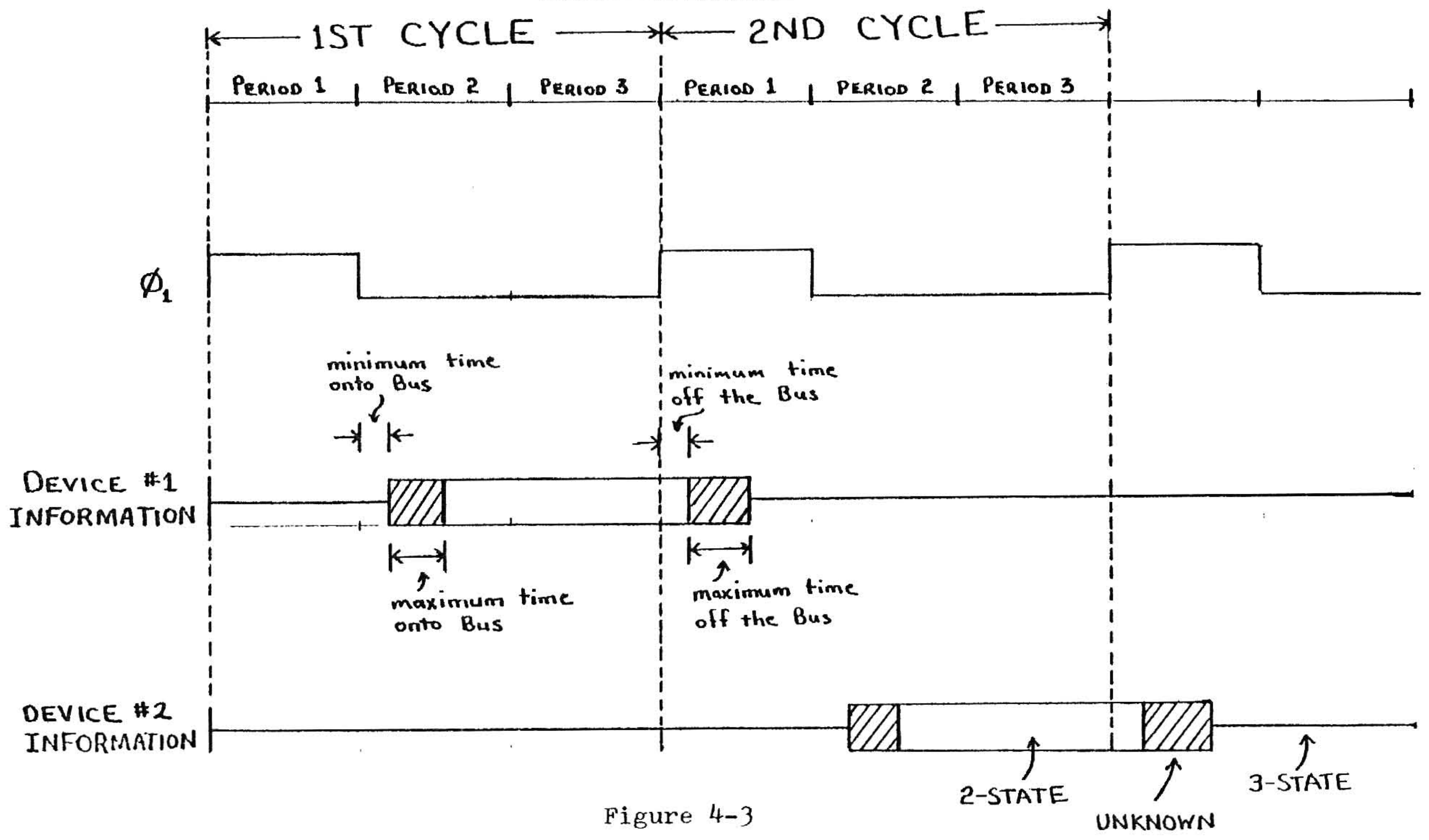


Figure 4-3

here. Under certain conditions the Stack Tender circuit will generate a hardware trap signal which will require a temporary modification of $\overline{\phi}_3$. The operation will be discussed in detail when the Stack Tender and Clock Circuit hardware designs are presented.

5 The System Bus

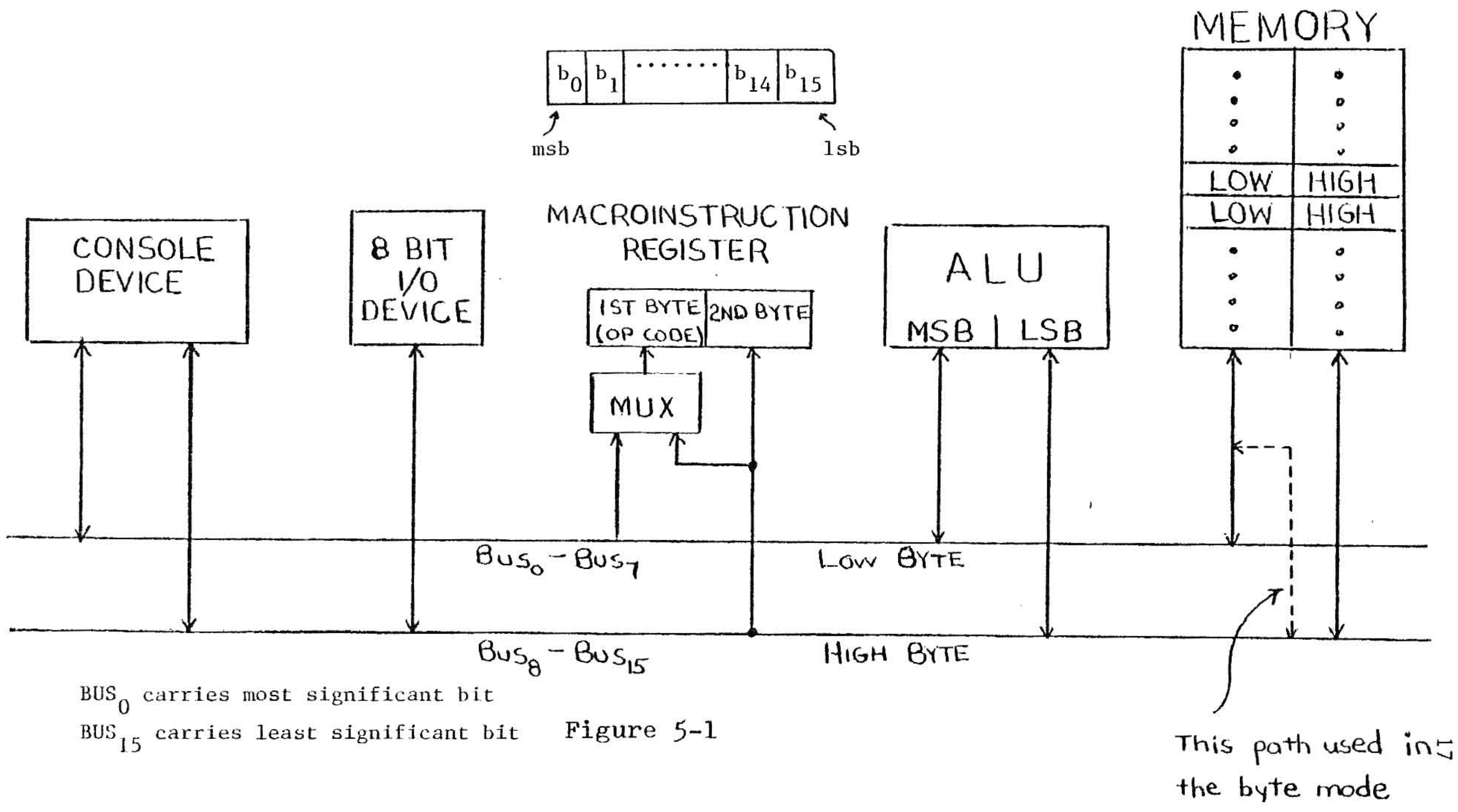
The system Bus consists of 16 information lines labelled BUS_0 through BUS_{15}. The information transferred over these lines include memory addresses, I/O Device Codes, instructions, and data. These lines are connected between various parts of the computer as shown in Figure 5-1. BUS_0 - BUS_7 is referred to as the low byte of the Bus, and BUS_8 - BUS_{15} as the high byte. Sequential bytes are accessed from memory locations in a "low byte", "high byte" fashion. The low byte is accessed with an even valued address, and the high byte with an odd address, one value larger. The low byte is shown on the left rather than the right for reasons related to the placement of variable length instructions in memory. This topic is discussed more thoroughly in the next section. As a consequence, the Most Significant Byte (MSB) of data to/from the ALU travels the low byte of the Bus, and the Least Significant Byte (LSB) travels the high byte. All single byte data, from I/O or memory, travels the high byte of the Bus so that it enters (exits) the LSB of the ALU.

In normal program mode, all Bus transfers are carried out via controls provided by the CPU. (When the CPU is in halt mode, the Console Device supplies the necessary controls). There are basically three types of control lines for the Bus; memory controls, I/O controls, and timing controls.

5.1 Memory Bus Controls

The Bus controls associated with main memory are defined as follows.

System Bus Connections



BUS_0 carries most significant bit

BUS_{15} carries least significant bit

Figure 5-1

1. LDMAR - Load Memory Address Register

   When high, this signal causes the information on the Bus to be placed in the Memory Address Register.

2. $\overline{\text{MEM SEL}}$ - Memory Select

   When low, this signal signifies a data transfer with main memory.

3. $R/\overline{W}$ - Read/Write

   If $\overline{\text{MEM SEL}}$ is low, this signal, when low, signifies a write to main memory. A read operation is performed when this signal is high.

5.2 I/O Bus Controls

Due to their asynchronous nature, I/O devices must communicate with the CPU via some sort of handshake. The handshake is accomplished by providing each I/O device with a Ready Flag. When this flag is high it signifies that the device is ready. Each time data is transferred from the CPU to an output device its Ready Flag is set low. As soon as the output device is ready to receive another piece of data it returns the Ready Flag to the high state. An input device signifies the presence of data available by setting its Ready Flag high. This flag is returned low when the CPU reads the input data. The Ready Flag status for all I/O devices is returned to the CPU via a 3-state Bus line called I/O READY.

I/O devices can interrupt the CPU from 1 of 16 interrupt levels. Interrupts from all levels are initiated by pulling low an open-collector interrupt line. Interrupts are acknowledged on a priority basis via a hardware daisy chain. The CPU has the capability to mask interrupts from any or all levels by setting the appropriate interrupt mask bits.

A total of 7 Bus control lines are provided to carry out the I/O operations specified above.

1. $\overline{\text{I/O SEL}}$ - I/O Select

   When low, the current Bus transaction is directed to an I/O device. NOTE: $\overline{\text{I/O SEL}}$ should never be low concurrently with LDMAR high or $\overline{\text{MEM SEL}}$ low.

2. I/O STAT/$\overline{\text{DATA}}$ - I/O Status/Data

   (If $\overline{\text{I/O SEL}}$ is high, this signal has no meaning.) I/O STAT/$\overline{\text{DATA}}$ high signifies that status information (the Ready Flag) from the selected I/O device should be placed on the I/O READY Bus line. Devices are selected by placing their Device Code on the Bus information lines concurrently with the activation of the $\overline{\text{I/O SEL}}$ line. I/O STAT/$\overline{\text{DATA}}$ low signifies that the current Bus transaction is 1 of the 2 operations required to transfer data between an I/O device and the CPU. These 2 operations are both performed with $\overline{\text{I/O SEL}}$ and I/O STAT/$\overline{\text{DATA}}$ in the low state, and must be carried out sequentially. The first operation entails the selection of the device by sending its Device Code down the Bus information lines. The actual I/O data transfer occurs during the second operation.

3. I/O READY

This is a 3-state Bus line shared by all I/O devices. When a device's status is requested via the I/O STAT/$\overline{\text{DATA}}$ line, the state of the Ready Flag for the selected device is placed on this line.

4. $\overline{\text{INTERRUPT}}$

This is an open-collector Bus line shared by all 16 interrupt levels. An interrupt is requested by pulling this line low.

5. $\overline{\text{INTACK}}$ - Interrupt Acknowledge

This line is pulled low by the CPU to acknowledge an interrupt. It serves as an input signal to a daisy chain network through which it propagates to the highest level interrupting device. Any interrupting device receiving an acknowledge should place its Device Code on the Bus information lines, thus identifying itself to the CPU as the interrupt source.

6. $\overline{\text{MSKEN}}$ - Interrupt Mask Enable

This signal is used by the CPU to mask interrupts from any or all of the 16 interrupt levels. It is pulled low concurrently with the placement of a 16-bit mask word on the Bus information lines. Interrupts are disabled by setting the corresponding bit of the mask word to 1.

7. $\overline{\text{BRST}}$ - Bus Reset

When low, this signal places all devices on the Bus in a known state. This signal should be asserted during system initialization.

5.3 Bus Timing Controls

The timing controls for the system Bus consist of the ϕ_1 and $\overline{\phi}_1$ clock signals generated in the CPU. These signals are used to gate all information on the Bus during periods 2 and 3, leaving period 1 as a transient period as previously discussed. ϕ_1 is used primarily with I/O, while $\overline{\phi}_1$ is used primarily with main memory.

6 Macroinstruction Formats

This machine has been designed to handle variable length instructions of 1, 2, 3, and 4 bytes. The actual operations performed by these instructions depends mostly on the microcode. Their formats, however, are largely a function of the hardware design. The hardware design places various restrictions on the Macroinstruction's Format and its placement in memory. The Op Code for all of these instructions must occupy the first byte. In addition, multi-byte instructions (2, 3, and 4 bytes) must be fetched from memory at an even byte address (i.e. a word boundary). Only 1-byte instructions can be fetched from an odd memory address. This implies that all instructions whose last byte occupies an even numbered address must be followed by a 1-byte instruction. In situations where the succeeding operation requires a multi-byte instruction, an 8-bit NO-OP must be inserted to satisfy the above criterion. This is similar to the way variable length instructions are handled in the HP3000 [1].

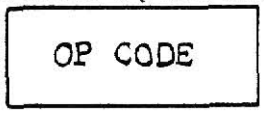
The Instruction Register, IR, in this machine holds 16 bits, i.e. the Op Code and one additional byte. An instruction fetch from an even byte address will place a full 16-bits in the IR. A fetch from an odd numbered address will place only an 8-bit Op Code in the IR. This is due primarily to the manner in which information is read from memory. In the 3 and 4 byte instructions, the information contained in the last one or two bytes is always destined for the ALU. Thus, this portion of the instruction is fetched during the execution phase and transferred directly to the ALU, completely by-passing the IR. Typical Macroinstruction Formats for this machine are shown in Figure 6-1.

Stack Operations consist of a single 8-bit Op Code. These instructions

MACROINSTRUCTION FORMATS

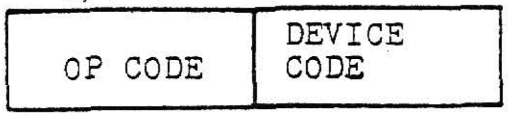
1-BYTE Instruction Format

Stack Operations

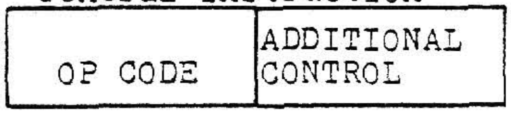


2-BYTE Instruction Formats

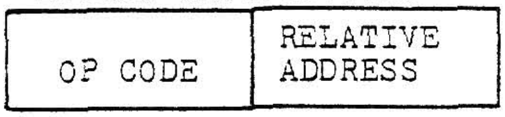
I/O Instructions



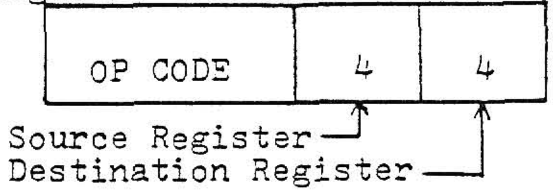
General Instruction



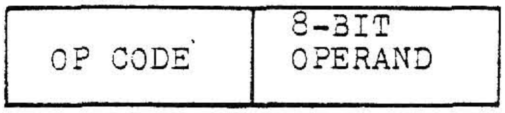
Branch Instructions



Register Reference Instruction



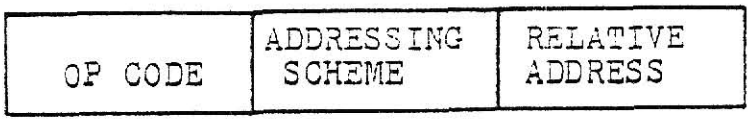
ALU Instruction



3&4-BYTE Instruction Formats

(Memory Reference Instructions)

3-BYTE



4-BYTE

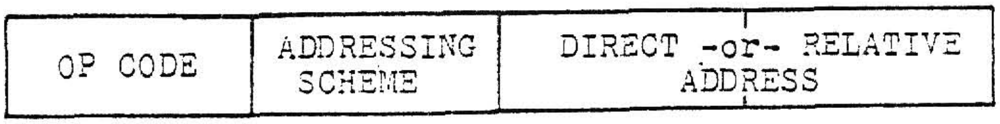


Figure 6-1

perform a variety of ALU functions including addition, subtraction, and logical operations. Operands for these instructions lie in various locations at the top of the stack. These instructions typically make up a significant portion of the total Macroinstruction Set.

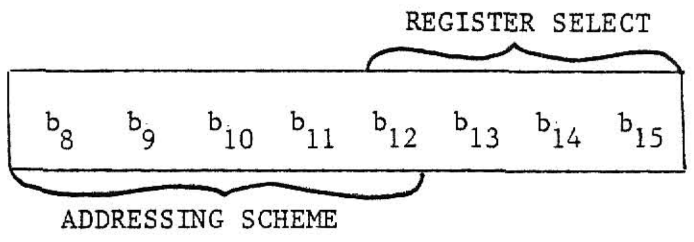
A variety of instruction types fit into the 2-byte catagory. Here the second byte is used to supplement the Op Code with additional information pertinent to the instruction. For example, I/O instructions require a Device Code field and Branch instructions require a relative address field. Other instructions may require an 8-bit operand or even additional control information. In the classical register machine, ALU operations require register select fields in the instruction format to determine the source and destination registers. This machine has been designed so that it can be operated in just this manner. That is, the stack hardware can be disabled (via microcode), in which case the second byte of the Instruction Register is divided into two register select fields, each capable of accessing any one of the 16 ALU registers. To perform the various functions described above, the second byte of the IR serves as an input to the Stack Tender/Register Select circuit, the Data Select/Sign Extend logic, and the Macroinstruction logic (see Figure 4-1).

Three and four byte instructions in this machine are designed as Memory Reference Instructions. The second byte of these instructions is used to specify various addressing schemes used to calculate Effective Addresses for memory operands. The remaining byte (s) function as an address field. For 3-byte instructions the last byte supplies an 8-bit relative address. In the 4-byte instruction, the last two bytes can be used to access any location in memory, either directly or in a relative fashion.

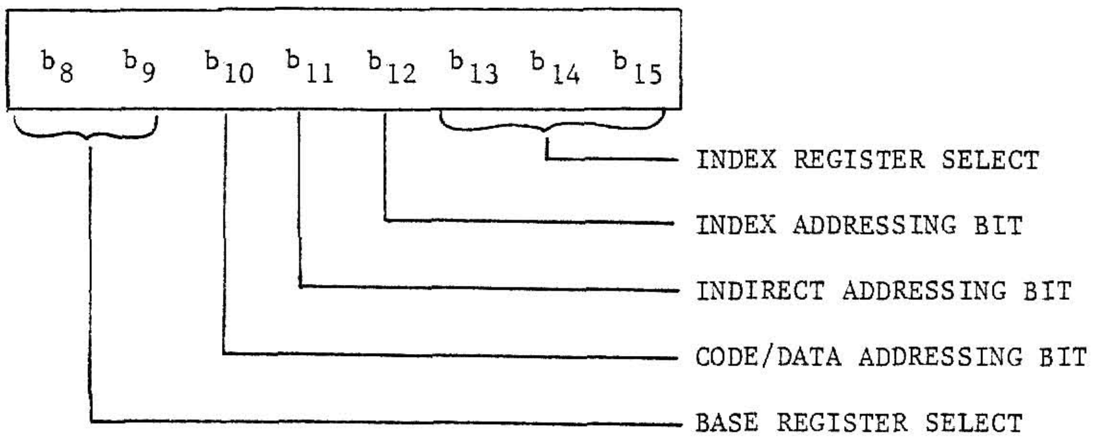
Effective Addresses for Memory Reference Instructions are typically computed using a variety of addressing techniques, such as Relative, Indexed, and Indirect. In addition, each of these techniques can be applied with respect to numerous base or pointer registers. The Program Counter and Stack Pointer are the two of the most commonly used pointer registers. Additional pointer registers, associated with both code and data portions of memory, are also found in typical Stack Machines. For example, the HP3000 has a total of 6 pointer registers [1]. Two of these, P and PB, are pointers to the code area of Memory. The remaining four, S, Q, DB, and DL, are pointers to various data areas in memory. Other registers, called Index registers, will also exist in a machine that has Indexed Addressing. These registers hold an index, or relative value, (not an address), used in computing the Effective Address when Indexed Addressing is specified. The HP3000 has one Index register.

In some Stack Machines, the various addressing techniques are applied in a slightly different fashion depending on the base register type. For example, the HP3000 makes a distinction between Code and Data addressing [1]. That is, Effective Addresses for Code areas are computed slightly differently than for Data areas. To complicate matters even more, relative addresses can be computed in only a positive direction with respect to some pointer registers, a negative direction for others, and both directions for the rest. Thus, it is apparent that computation of Effective Addresses in a Stack Machine may require a considerable number of algorithms.

In this machine, Effective Addresses are computed in microcode. Each of the different algorithms for computing these addresses requires a separate microinstruction routine. This machine provides up to 32 different routines for calculating the Effective Address. These routines are specified by a 5 bit field in the second byte of the instruction register. The format for this byte is the following.


REGISTER SELECT
b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}
ADDRESSING SCHEME

The first five bits, b_8 through b_{12}, specify the addressing scheme, and are directed to the Instruction logic in the Control Unit. The last four bits, b_{12} through b_{15}, select an ALU register to play a part in the computation of the Effective Address. The overlap of b_{12} between the two fields occurs here because the Register Select field is the same 4-bit field used to specify the source register for an ALU operation when functioning as a register machine. For Memory Reference Instructions this field will most likely be used as an Index Register Select field. It would be impractical, however, to suggest that all 16 ALU registers function as Index registers. Perhaps the best way to organize this field is to designate b_{12} as the Index Addressing bit. When b_{12} is 1, Index Addressing is specified, and the Register Select field can choose any one of the upper 8 ALU registers to play the part of an Index Register. When b_{12} is 0, the Register Select field is ignored. Specification of the various pointer registers is encompassed in the first 5 bits because Effective Addresses are often computed differently for each of them. The following example shows how the individual bits might be arranged to perform the various addressing schemes found in the HP3000 [1].



This configuration provides for up to 8 Index registers as opposed to the 1 found in the HP3000. The Effective Addresses are computed in accordance with the specified addressing scheme as summarized in Figure 6-2. Note, since there are only two registers used in Code Addressing, P and PB, 8 of the possible 32 schemes are duplicated in the table, resulting in 24 distinct ways of computing the Effective Address. The above organization of the second byte is just an example. The actual definition of bits b_8 through b_{15} is up to the microprogrammer. Examples of microcode written to compute Effective Addresses using the above scheme will be presented in detail later.

In summary, the Macroinstruction Formats for various types of instructions have been presented. The Op Code for all instructions always occupies the first 8-bits. Thus, this machine can perform up to 256 different instructions. In addition, the organization of the second byte in the Instruction Register provides multi-byte instructions with a wide range of capabilities. Instructions more than 2 bytes long are designated as Memory Reference Instructions. All information beyond the

Computation of the Effective Address using various Addressing Schemes

b_{11} IN	b_{12} IX	b_{10} T	b_8 R1	b_9 R2	EA, Effective Address
0	0	0	0	0	PB+n
0	0	0	0	1	P±n
0	0	0	1	0	PB+n
0	0	0	1	1	P±n
0	0	1	0	0	DB+n
0	0	1	0	1	Q±n
0	0	1	1	0	S-n
0	0	1	1	1	DL+n
0	1	0	0	0	PB+n+IX
0	1	0	0	1	P±n+IX
0	1	0	1	0	PB+n+IX
0	1	0	1	1	P±n+IX
0	1	1	0	0	DB+n+IX
0	1	1	0	1	Q±n+IX
0	1	1	1	0	S-n+IX
0	1	1	1	1	DL+n+IX
1	0	0	0	0	PB+n+(PB+n)
1	0	0	0	1	P±n+(P±n)
1	0	0	1	0	PB+n+(PB+n)
1	0	0	1	1	P±n+(P±n)
1	0	1	0	0	DB+(DB+n)
1	0	1	0	1	DB+(Q±n)
1	0	1	1	0	DB+(S-n)
1	0	1	1	1	DB+(DL+n)
1	1	0	0	0	PB+n+(PB+n)+IX
1	1	0	0	1	P±n+(P±n)+IX
1	1	0	1	0	PB+n+(PB+n)+IX
1	1	0	1	1	P±n+(P±n)+IX
1	1	1	0	0	DB+(DB+n)+IX
1	1	1	0	1	DB+(Q±n)+IX
1	1	1	1	0	DB+(S-n)+IX
1	1	1	1	1	DB+(DL+n)+IX

Figure 6-2

second byte is treated as an address field and is transferred from memory to the ALU during calculation of the Effective Address. The Effective Address can be computed using as many as 32 different schemes.

7 ALU Hardware Design

7.1 The 2901

The major ALU component is the 2901 LSI processor device, a block diagram for which is shown in Figure 7-1 [2]. The two most important features of this device are the 16 word by 4-bit dual ported RAM, and the high speed Arithmetic and Logic Function Unit (ALFU). The 2901 is a 4-bit slice which is easily cascaded to form a 16-bit ALU. The 16-words of RAM function as general purpose registers, any two of which can be accessed in parallel and sent to the ALFU as operands. These registers (RAM locations) are selected by two sets of address lines. A_3 through A_0 are the A register select lines. They select one of the 16 RAM locations to be latched at the Port A output of RAM. Likewise, B_3 through B_0 are the B register select lines which select the RAM location to be latched at the Port B output of the RAM. An additional register, separate from the RAM registers, called the Q-register, is used primarily for multiplication and division routines that require a double length operand.

The ALFU has two operand inputs, R and S, and a result output, F. The 2901 control inputs I_5, I_4 and I_3 select from 8 possible ALFU operations as shown in Figure 7-2 [2]. The ALFU operands, S and R, can be supplied from a variety of sources which include the Q-register, RAM Port A, RAM Port B, zero, and a set of direct data inputs whose values are externally supplied. The sources are referred to as Q, A, B, O, and D respectively and are selected via the I_2, I_1, and I_0 control inputs as shown in Figure 7-3 [2].

2901 Block Diagram

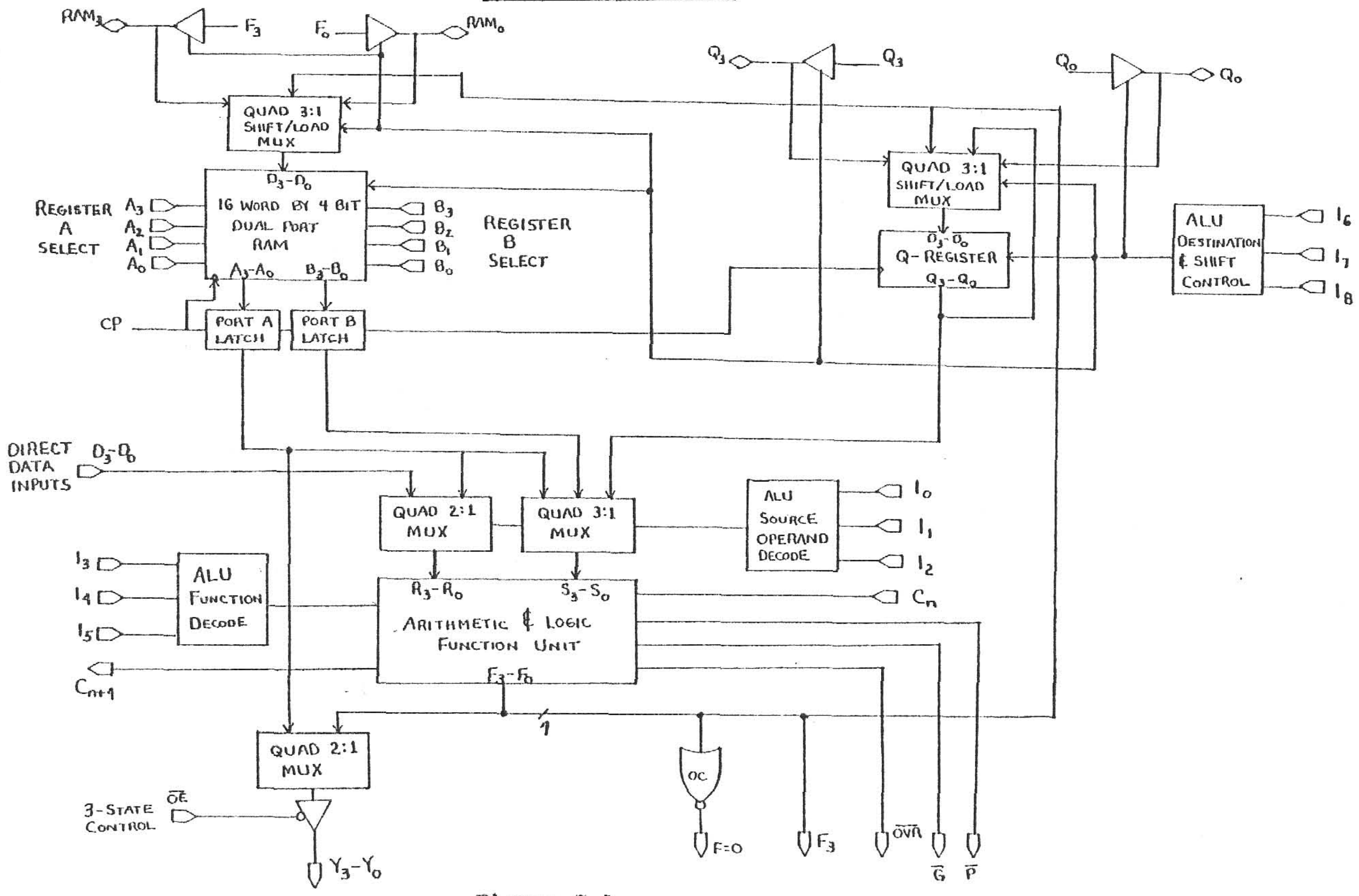


Figure 7-1

2901 ALFU FUNCTION CONTROLS

Control Inputs			
I5	I4	I3	Function
0	0	0	R plus S
0	0	1	S minus R
0	1	0	R minus S
0	1	1	R OR S
1	0	0	R AND S
1	0	1	$\overline{R}$ AND S
1	1	0	R EX-OR S
1	1	1	R EX-NOR S

Figure 7-2

2901 SOURCE OPERAND CONTROLS

Control Inputs			ALFU Source Operands	
I2	I1	I0	R	S
0	0	0	A	Q
0	0	1	A	B
0	1	0	0	Q
0	1	1	0	B
1	0	0	0	A
1	0	1	D	A
1	1	0	D	Q
1	1	1	D	0

Figure 7-3

The ALFU result, F, can be stored in 1 of the 16 RAM registers and/or the Q-register. The B register select lines always determine the location of the stored result in RAM. Before the result is stored it can be shifted either right or left by a set of multiplexers positioned at the RAM and Q-register inputs. F, as well as the selected A register value (RAM Port A), can be made available externally through the 2901 Y outputs. These outputs are 3-state to allow for direct connection to a system Bus. The I_8, I_7, and I_6 control inputs determine the shift operation, the destination of F, and the selected Y output as summarized in Figure 7-4 [2] .

In order to cascade a group of 2901s, various shift and carry information must be made available externally. In Figure 7-1, C_n and C_{n+4} are the "carry in" and "carry out" respectively. The ALFU features internal carry look-ahead to enhance the speed of arithmetic operations. The carry generate and propagate signals, $\overline{G}$ and $\overline{P}$, supply the information needed to extend the carry look-ahead operation over a group of 2901s. A 74S182 (carry look-ahead generator) can be used directly with a group of four 2901s to form a complete 16-bit ALU with carry look-ahead.

The RAM_3, RAM_0, Q_3, and Q_0 signals provide for the necessary shifting connections between cascaded 2901s. The interconnections for two successively ordered 2901s are shown in Figure 7-5. On a right shift, the RAM_0 and Q_0 outputs of device #n+1 are shifted into RAM_3 and Q_3 of device #n. On a left shift this operation is reversed. Thus, these lines are necessarily bi-directional. The RAM_3, RAM_0, Q_3, and Q_0 signals are 3-state internal to the 2901. A right shift opens the buffers for RAM_0 and Q_0, and a left shift opens them for RAM_3 and Q_3.

The 2901 also generates various ALU status information. OVR

ALFU Destination/Shift Control

Control Inputs			RAM Function		Q-reg Function		Y	RAM Shifter		Q Shifter	
I8	I7	I6	Shift	Load	Shift	Load	Output	RAM0	RAM3	Q0	Q3
0	0	0	X	none	none	$F \rightarrow Q$	F	X	X	X	X
0	0	1	X	none	X	none	F	X	X	X	X
0	1	0	none	$F \rightarrow B$	X	none	A	X	X	X	X
0	1	1	none	$F \rightarrow B$	X	none	F	X	X	X	X
1	0	0	down	$F/2 \rightarrow B$	down	$Q/2 \rightarrow Q$	F	F0	IN3	Q0	IN3
1	0	1	down	$F/2 \rightarrow B$	X	none	F	F0	IN3	Q0	X
1	1	0	up	$2F \rightarrow B$	up	$2Q \rightarrow Q$	F	IN0	F3	IN0	Q3
1	1	1	up	$2F \rightarrow B$	X	none	F	IN0	F3	X	Q3

Note:
X=don't care
down=shift right
up=shift left

Figure 7-4

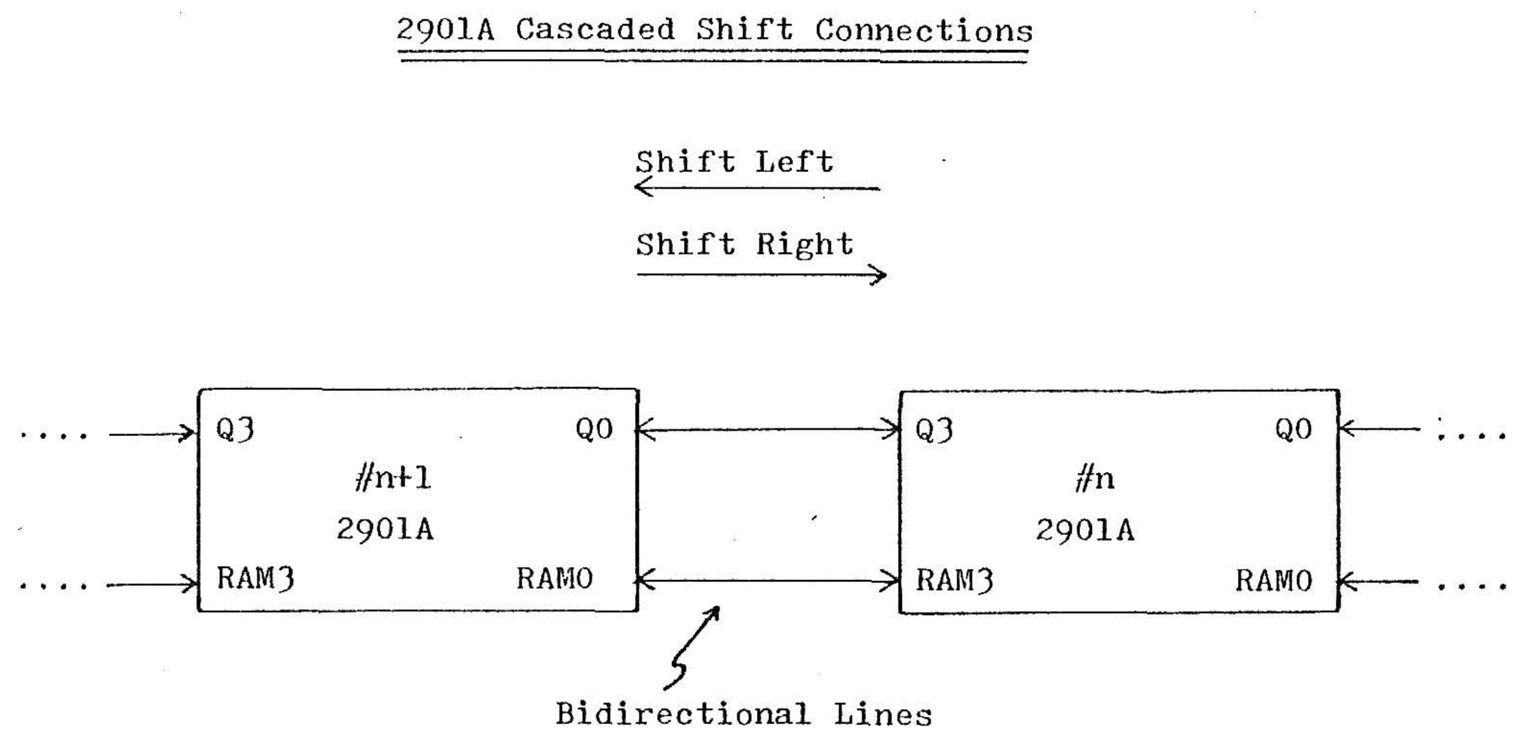

2901A Cascaded Shift Connections
Shift Left
Shift Right
Q3
Q0
#n+1
2901A
RAM3
RAM0
#n
2901A
Bidirectional Lines

Figure 7-5

(see Figure 7-1) is activated by an arithmetic operation resulting in 2's complement overflow. The msb of the ALFU result, F_3, is also brought out as status. This bit typically functions as a sign bit for 2's complement numbers. A zero result is detected and supplied at the F=0 output. This output is high when the ALFU result is zero. F=0 is an open collector output so it can be wire AND'ed with the corresponding output on each of the cascaded 2901s. In this way a zero result is signified only when the F=0 output is high for all 2901s.

Timing for the 2901s works in the following manner. When the clock input, CP, is high, the selected A and B register values are read from RAM. The Port A and Port B latches are transparent during this time. When CP goes low these latches will close and retain the previously entered value. If enabled, the ALFU result, F, is stored in the selected B register while CP is low. If F is to be stored in the Q-register it is loaded on the low-to-high transition of CP. CP is connected to the Execution Clock, $\overline{\emptyset}_3$, in this machine. Thus, during periods 1 and 2 the source operands (R and S) are selected and the ALFU operation is performed. The result, F, is either stored in the selected B register during period 3, or loaded into the Q-register at the end of this period.

7.2 The ALU Design

Figure 7-6 shows the 16-bit ALU in its entirety. As previously mentioned, the 74S182 carry lookahead generator is connected directly to the 2901s to provide full 16-bit carry lookahead. The A and B register select inputs are supplied by the Stack Tender/Register Select circuit, the details of which will be discussed in the next section.

The ALU Design

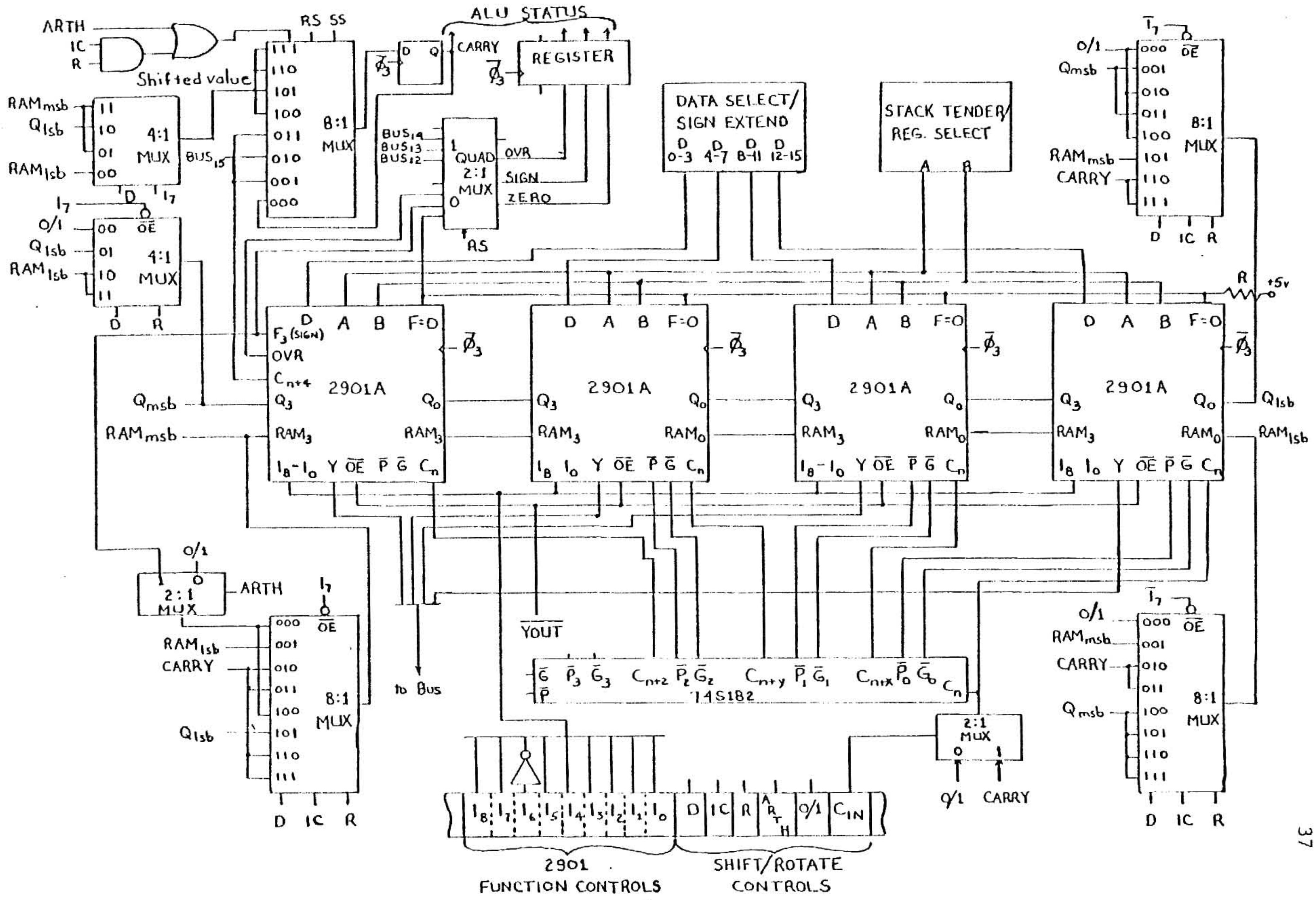


Figure 7-6

The Data Select/Sign Extend circuit, also presented in a subsequent section, supplies values for the 2901 direct data inputs. The Y outputs of the 2901 are connected directly to the system Bus. They are enabled onto the Bus by a control signal from the CPU Bus Interface circuit, $\overline{\text{YOUT}}$. The Execution Clock, $\overline{\phi}_3$, is used to clock the 2901s as described earlier.

The 2901 control inputs, I_8 through I_0, are supplied directly from the Pipeline Register. I_6, however, is inverted while the others are not. In this way the placement of zeros in these 9 bits for a particular microinstruction has the net effect of a No Op for the 2901s. This feature simplifies the microinstruction assembly process by requiring bits in this field to be set only for microinstructions that perform an operation with the 2901s.

The sign bit, F_3, and overflow, OVR, status signals for the ALU are taken from the leftmost 2901 (which works with the 4 most significant bits). These two signals, along with the wire AND'ed F=0 signal, are gated through a 2-to-1 multiplexer as inputs to a 3-bit portion of the machine status register. The outputs of this register are made available not only to the Control Unit (for decision making purposes), but also to the system Bus in anticipation of storing these values in memory when a change of context occurs (i.e. an interrupt or jump to subroutine). This register is easily restored to its original context since the same Bus lines that are used to store the register are also made available at the register input via the 2-to-1 multiplexer. This multiplexer is operated by the Return Status, RS, control bit in the Pipeline Register. When high, the RS bit selects the Bus lines as inputs to the register, and when low the current status values are selected. The register is clocked by $\overline{\phi}_3$, but will load only if the Set Status, SS, bit in the Pipeline Register is set.

The remaining circuitry in Figure 7-6 implements the desired shift, rotate, and carry functions. A complete set of the shift and rotate operations implemented in this design are shown in Figures 7-7 and 7-8. The control signals supplied by the Pipeline Register to perform these operations are defined as follows.

1. D - Double

   D = 0 Selects a single width (16 bit) shift or rotate.

   D = 1 Selects a double width (32 bit) shift or rotate.

2. IC - Include CARRY

   IC = 0 Will exclude the CARRY from the shift or rotate operation.

   IC = 1 Will include the CARRY in the shift or rotate operation.

3. R - Rotate

   R = 0 Selects a shift operation.

   R = 1 Selects a rotate operation.

4. ARTH - Arithmetic

   This control has an affect only when a shift operation is selected.

   ARTH = 0 Selects a logical shift.

   ARTH = 1 Selects an arithmetic shift.

5. 0/1

   This signal supplies a 0 or a 1 to be used directly in various shift operations.

Single Word Shift and Rotate Operations

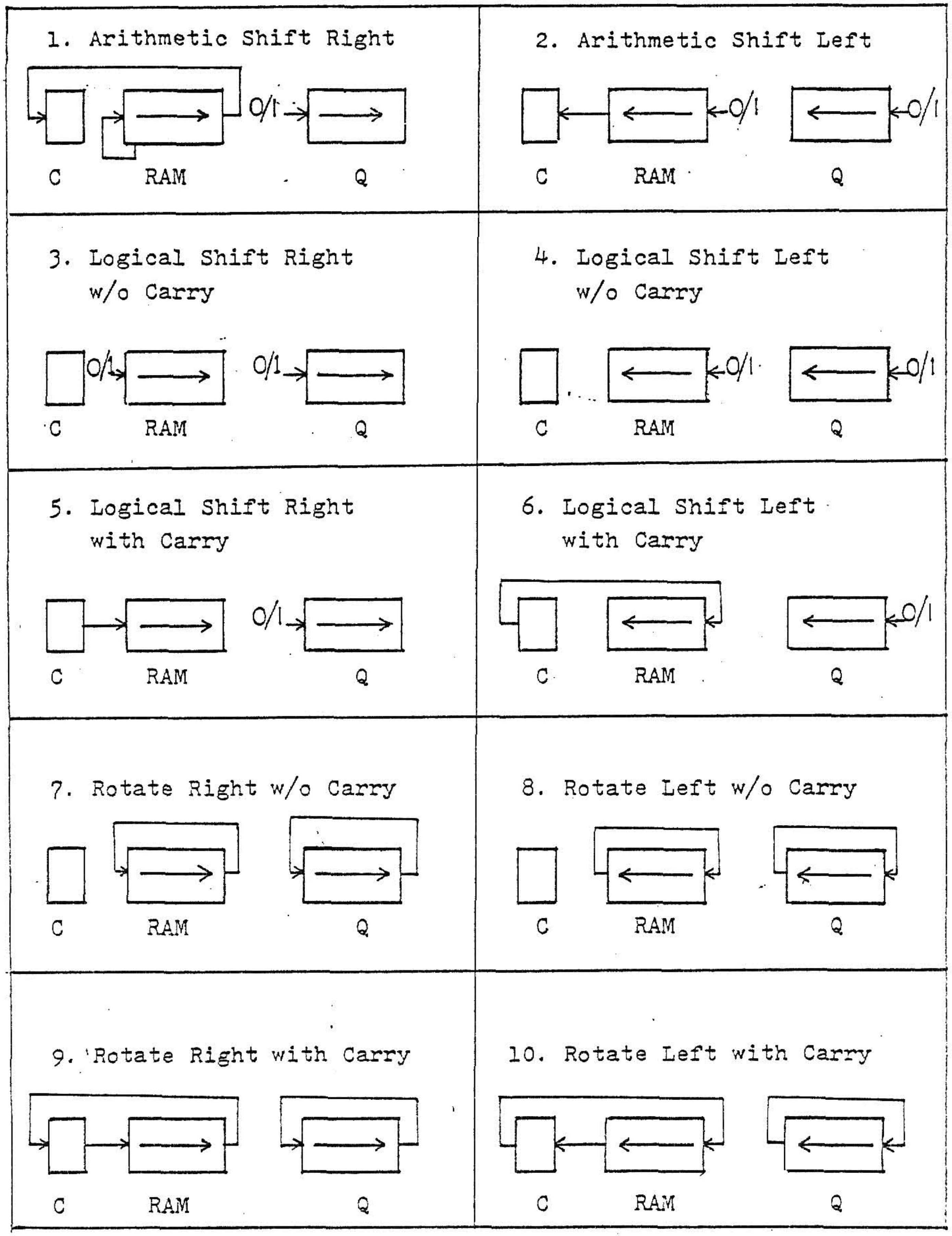
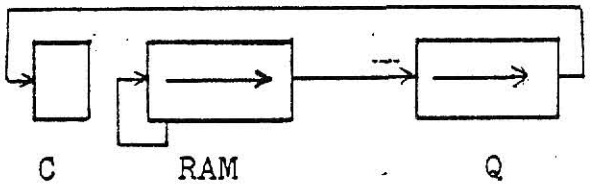

1. Arithmetic Shift Right
0/1
C
RAM
Q
2. Arithmetic Shift Left
0/1
0/1
C
RAM
Q
3. Logical Shift Right w/o Carry
0/1
0/1
C
RAM
Q
4. Logical Shift Left w/o Carry
0/1
0/1
C
RAM
Q
5. Logical Shift Right with Carry
0/1
C
RAM
Q
6. Logical Shift Left with Carry
0/1
C
RAM
Q
7. Rotate Right w/o Carry
C
RAM
Q
8. Rotate Left w/o Carry
C
RAM
Q
9. Rotate Right with Carry
C
RAM
Q
10. Rotate Left with Carry
C
RAM
Q

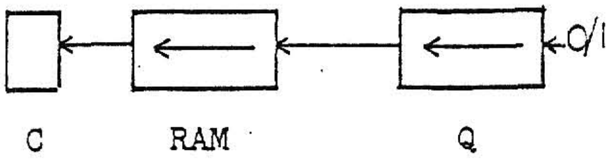
Figure 7-7

Double Word Shift and Rotate Operations

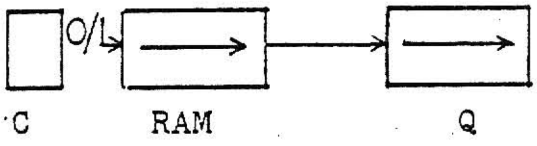
1. Arithmetic Shift Right



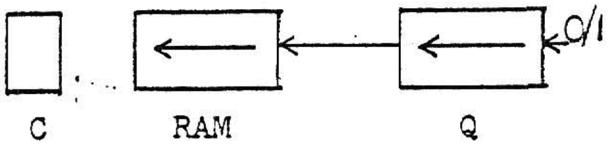
2. Arithmetic Shift Left



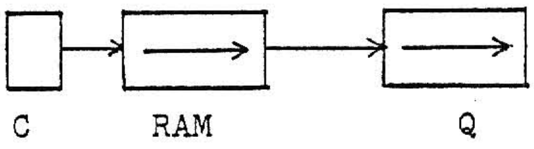
3. Logical Shift Right w/o Carry



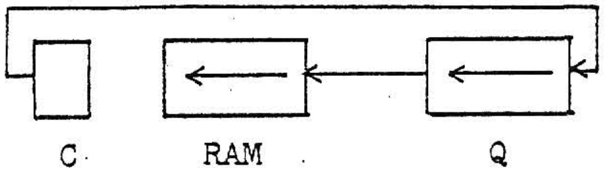
4. Logical Shift Left w/o Carry



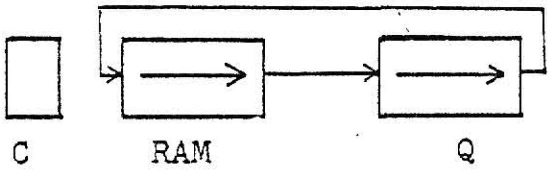
5. Logical Shift Right with Carry



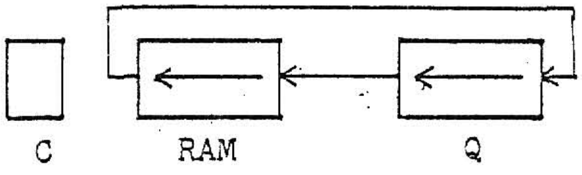
6. Logical Shift Left with Carry



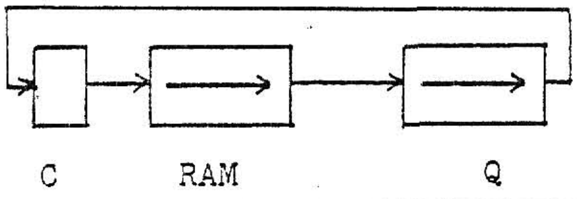
7. Rotate Right w/o Carry



8. Rotate Left w/o Carry



9. Rotate Right with Carry



10. Rotate Left with Carry

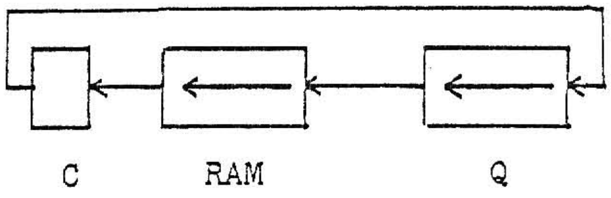


Figure 7-8

6. I_7

This is one of the 9 control bits supplied to the 2901s. When a shift (or rotate) operation is selected, this bit determines the direction of the shift.

$I_7 = 0$ Selects a shift right.

$I_7 = 1$ Selects a shift left.

As seen in Figure 7-6, these signals control a group of multiplexers positioned at the 2901 shift inputs (outputs). The various multiplexer operations are summarized in tabular form. Figure 7-9 defines the multiplexer operation for shifts into the msb of a selected RAM location. Figure 7-10 describes a similar operation for shifts into the lsb of a RAM location. Figures 7-11 and 7-12 are the Q-register equivalents of 7-9 and 7-10. It should be noted that all 4 multiplexer outputs are 3-stated , and each is enabled according to the shift direction specified by I_7.

The CARRY bit in this machine consists of a D flip-flop clocked by $\overline{\emptyset}_3$ and loaded from 1 of 4 sources. The first of these is the CARRY bit itself, i.e. its value remains unchanged. The second is the system Bus which is used to restore the Carry when returning from a change of context. The third is the "carry out", C_{n+4}, of the leftmost 2901. The final source for the CARRY bit is the shifted or rotated value. This value is determined by a 4-to-1 multiplexer under the control of the D and I_7 bits from the Pipeline Register. A summary of the operation for this multiplexer along with that of the CARRY multiplexer is shown in Figure 7-13. Note, the shifted value of the CARRY is selected only for an arithmetic shift, or a rotate which includes the CARRY. The CARRY is

RAM_{msb} Shift Multiplexer Operation

$I_8, I_7, I_6 = 1, 0, X$

(Shift/Rotate Right into B Register)

Pipeline Register Controls				RAM3 shifted input to leftmost 2901
D	I	R	Operation	
0	0 (ARTH = 0)	0	Single Word, Logical Shift Right, w/o Carry, Figure 7-7(3)	0/1
0	0 (ARTH = 1)	0	Single Word, Arithmetic Shift Right, w/o Carry, Figure 7-7(1)	F_{msb}
0	0	1	Single Word, Rotate Right, w/o Carry, Figure 7-7(7)	RAM_{lsb}
0	1	0	Single Word, Shift Right, with Carry, Figure 7-7(5)	CARRY
0	1	1	Single Word, Rotate Right, with Carry, Figure 7-7(9)	CARRY
1	0 (ARTH = 0)	0	Double Word, Logical Shift Right, w/o Carry, Figure 7-8(3)	0/1
1	0 (ARTH = 1)	0	Double Word, Arithmetic Shift Right, w/o Carry, Figure 7-8(1)	F_{msb}
1	0	1	Double Word, Rotate Right, w/o Carry, Figure 7-8(7)	Q_{lsb}
1	1	0	Double Word, Shift Right, with Carry, Figure 7-8(5)	CARRY
1	1	1	Double Word, Rotate Right, with Carry, Figure 7-8(9)	CARRY

Figure 7-9

RAM_{lsb} Shift Multiplexer Operation

$I_8, I_7, I_6 = 1, 1, X$

(Shift/Rotate Left into the B Register)

Pipeline Register Controls			Operation	RAMO shifted input to rightmost 2901
D	IC	R		
0	0	0	Single Word, Shift Left, w/o Carry, Figure 7-7(2 or 4)	0/1
0	0	1	Single Word, Rotate Left, w/o Carry, Figure 7-7(8)	RAM_{msb}
0	1	0	Single Word, Shift Left, with Carry, Figure 7-7(6)	CARRY
0	1	1	Single Word, Rotate Left, with Carry, Figure 7-7(10)	CARRY
1	0	0	Double Word, Shift Left, w/o Carry, Figure 7-8(2 or 4)	Q_{msb}
1	0	1	Double Word, Rotate Left, w/o Carry, Figure 7-8(8)	Q_{msb}
1	1	0	Double Word, Shift Left, with Carry, Figure 7-8(6)	Q_{msb}
1	1	1	Double Word, Rotate Left, with Carry, Figure 7-8(10)	Q_{msb}

Figure 7-10

Q_{msb} Shift Multiplexer Operation

$I_8, I_7, I_6 = 1,0,0$

(Shift/Rotate Right into the Q Register)

Note: IC has no affect on this input

Pipeline Register Controls			Operation	Q3 shifted input to leftmost 2901
D	IC	R		
0	X	0	Single Word, Shift Right, Figure 7-7(1,3, or 5)	0/1
0	X	1	Single Word, Rotate Right, Figure 7-7(7 or 9)	Q_{1sb}
1	X	0	Double Word, Shift Right, Figure 7-8(1,3,or 5)	RAM_{1sb}
1	X	1	Double Word, Rotate Right, Figure 7-8(7 or 9)	RAM_{1sb}

Figure 7-11

Q_{lsb} Shift Multiplexer Operation

$I_8, I_7, I_6 = 1, 1, 0$
(Shift/Rotate Left into the Q Register)

Pipeline Register Controls				
D	IC	R	Operation	Q0 shifted input to rightmost 2901
0	0	0	Single Word, Shift Left, w/o Carry, Figure 7-7(2 or 4)	0/1
0	0	1	Single Word, Rotate Left, w/o Carry, Figure 7-7(8)	Q_{msb}
0	1	0	Single Word, Shift Left, with Carry, Figure 7-7(6)	0/1
0	1	1	Single Word, Rotate Left, with Carry, Figure 7-7(10)	Q_{msb}
1	0	0	Double Word, Shift Left, w/o Carry, Figure 7-8(2 or 4)	0/1
1	0	1	Double Word, Rotate Left, w/o Carry, Figure 7-8(8)	RAM_{msb}
1	1	0	Double Word, Shift Left, with Carry, Figure 7-8(6)	CARRY
1	1	1	Double Word, Rotate Left, with Carry, Figure 7-8(10)	CARRY

Figure 7-12

CARRY Mux Operations

PLR Controls			
$(R \wedge IC) \vee ARTH$	RS	SS	D input to CARRY
0	0	0	CARRY
0	0	1	C_{N+4} from ALU
0	1	0	BUS_{15}
0	1	1	C_{N+4} from ALU
1	0	0	Shifted Carry
1	0	1	Shifted Carry
1	1	0	Shifted Carry
1	1	1	Shifted Carry

"Shifted Carry" Mux

PLR Controls		
D	I_7	Shift Generated Carry
0	0	RAM_{lsb}
0	1	RAM_{msb}
1	0	Q_{lsb}
1	1	RAM_{msb}

Figure 7-13

unaffected by all other shift or rotate operations.

The "Carry in" to the ALU can be set under microprogram control via the C_{IN} control bit. This bit allows the "carry in" to be set to 0, 1, or the value of the CARRY bit. If $C_{IN} = 0$, the "carry in" is determined by the 0/1 bit in the Pipeline Register. $C_{IN} = 1$ selects the CARRY bit itself as the source of the "carry in".

In summary, the complete ALU is presented in Figure 7-6. Two circuits have yet to be defined , the Stack Tender/Register Select, and the Data Select/Sign Extend. The former is presented in the following section.

7.3 The Stack Tender/Register Select

The "Stack Tender/Register Select" circuit supplies the appropriate A and B register select addresses to the 2901s. The Stack Tender hardware has been designed to make the first four 2901 registers function as a Fast Stack. The remaining 12 play the roles of a Stack Pointer, Program Counter, Index Register, and various other registers required by the stack architecture. The Fast Stack is like a 4 location cache memory for the top four entries in the stack. It may hold anywhere from 0 to 4 valid entries at a time. At no time, however, are these entries duplicated in the Memory Stack. Thus, the top of the stack in this machine will often lie in the Fast Stack. Only when the Fast Stack is empty will the top of the stack actually lie in main memory. For this reason, a true Stack Pointer register is non-existant. Instead there is a Memory Stack Pointer (MSP), a Fast Stack Pointer (called TOS), and a counter that keeps tract of the number of valid entries in the Fast Stack (NIS).

The value of a logical Stack Pointer, SP, can be computed using the following equation.

$$SP = MSP + NIS * 2$$

The NIS value is multiplied by 2 here so that it represents the number of bytes in the Fast Stack rather than the number of words. Obviously, if the number of valid entries in the Fast Stack is 0, then the logical Stack Pointer is equal to the Memory Stack Pointer and the top of the stack lies in main memory.

TOS is a 2-bit counter which points to the 2901 register considered to be at the top of the Fast Stack. Hereafter, this register will be referred to as the TOS register. The four Fast Stack registers work as a circular buffer. Their mode of operation is illustrated in the following examples. Figure 7-14a shows the condition of the Fast Stack, Memory Stack, and associated registers upon initialization. The initial value of MSP has been arbitrarily chosen. The initial TOS register is 00, which is actually meaningless since the Fast Stack is empty (NIS = 000). Figure 7-14b shows the result of a push onto the stack. As indicated by the Figure, a push operation first increments TOS, then places the data value in the resulting TOS register. In this way, TOS always points to a valid entry in the Fast Stack rather than one ahead to an empty location.

Figure 7-14c shows the result of two additional pushes. TOS is now 11. Registers 10 and 01 are referred to as TOS - 1 and TOS - 2 respectively. On the next push, seen in Figure 7-14d, the TOS counter wraps around to 00, hence the notion of a circular buffer. As a result of

PUSH Operations with the Stack

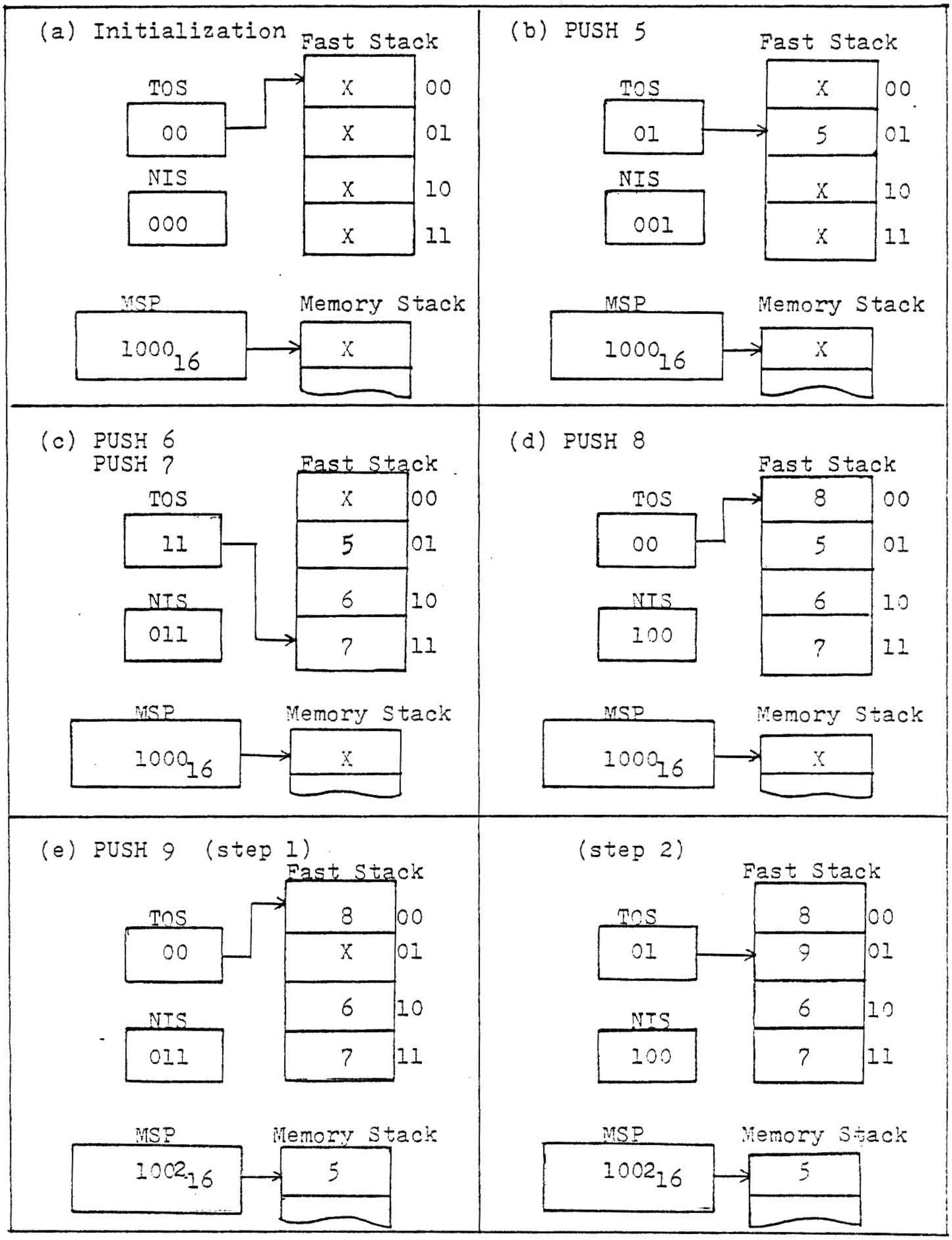


Figure 7-14

this push, the Fast Stack is now full (NIS=100). The next push will place data in Fast Stack register 01, but before this can happen the contents of this register must be transferred to the stack in main memory. Figure 7-14e summarizes the steps required to perform this operation. The effect of step 1, which transfers the contents of register 01 to the Memory Stack, is to free a location in the Fast Stack without changing TOS. This allows step 2 to carry out the push operation in a normal fashion, since the Fast Stack is no longer full (NIS=011).

The condition just described, where a push operation is attempted when the Fast Stack is full, is called a trap. This condition is detected in hardware and sent to the Control Unit (microcode) as an important status signal. In response, the Control Unit must free a location in the Fast Stack by transferring the TOS-3 register to the Memory Stack before proceeding with the push instruction.

Pop operations with the stack are performed in the following manner. As long as NIS is non-zero, pops are performed by first removing the data value from the TOS register, then decrementing TOS. This process is illustrated in Figure 7-15. Figure 7-15b shows the result of a pop from the stack in 7-15a. The effect of a pop operation is essentially that of decreasing the number of valid entries in the Fast Stack by one. Many of the common stack operations have the same effect. Figure 7-15c shows the result of an ADD instruction. Here the top two values on the stack are added, the result is placed back in the stack, and the two operands are deleted from the stack. This process can be summarized with the following set of equations.

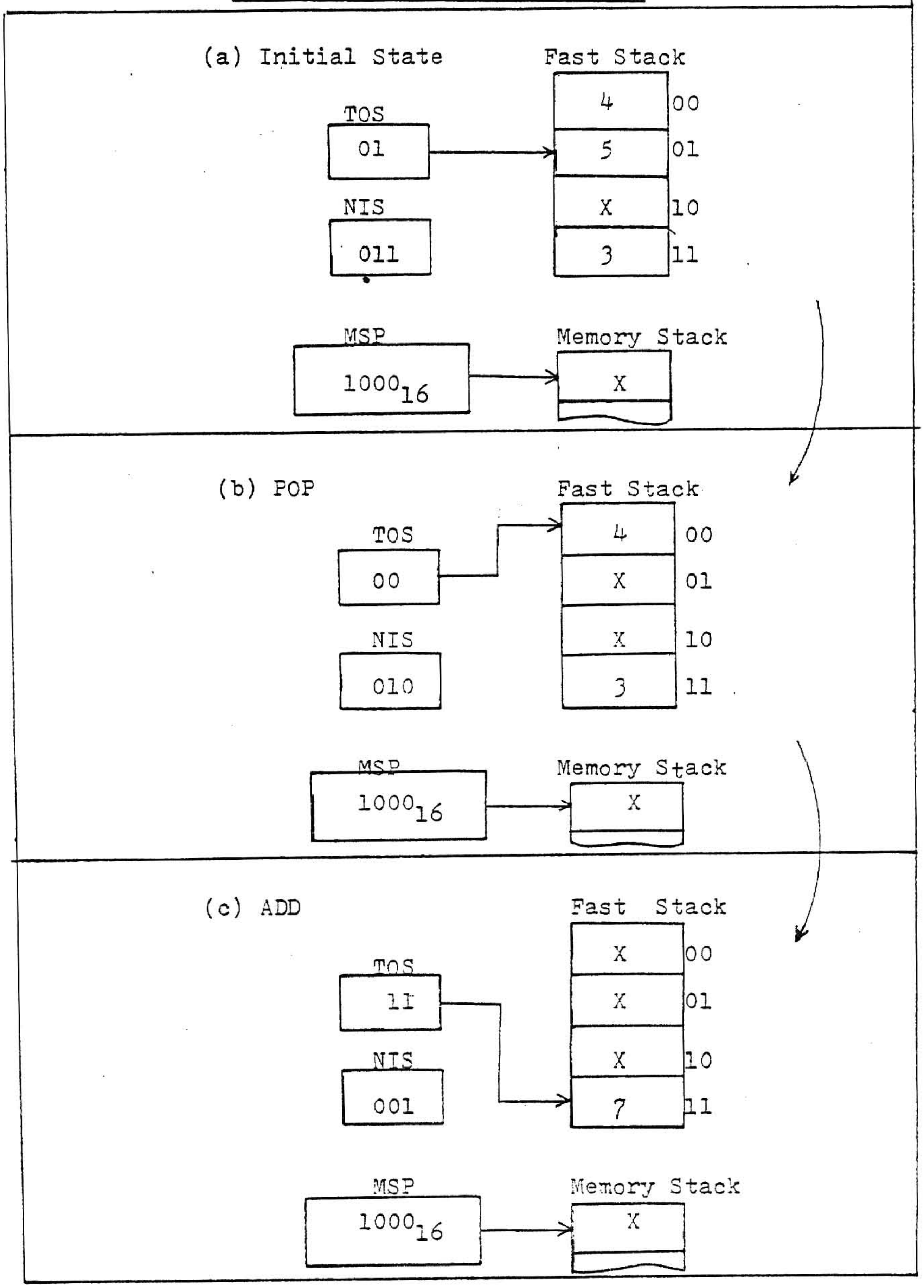

(a) Initial State
Fast Stack
TOS
01
4
00
5
01
NIS
011
X
10
3
11
MSP
1000_{16}
Memory Stack
X
(b) POP
Fast Stack
TOS
00
4
00
X
01
NIS
010
X
10
3
11
MSP
1000_{16}
Memory Stack
X
(c) ADD
Fast Stack
TOS
11
X
00
X
01
NIS
001
X
10
7
11
MSP
1000_{16}
Memory Stack
X

Figure 7-15

$$(TOS - 1) \leftarrow (TOS) + (TOS - 1)$$

$$TOS \leftarrow TOS - 1$$

$$NIS \leftarrow NIS - 1$$

Some instructions, such as a double precision add, operate on the top 4 values in the stack and actually reduce the number of stack entries by two.

A problem arises in all the operations above when the number of entries in the Fast Stack is insufficient to perform the desired instruction. Consider for example, a single precision add with only one entry in the Fast Stack. The second operand must be fetched from the top of the Memory Stack. Figure 7-16 summarizes the steps required to perform this operation. Step 1 transfers a value from the Memory Stack to the Fast Stack increasing the number of entries in the Fast Stack by one, and allowing step 2 to perform the add operation in normal fashion. An insufficient number of entries in the Fast Stack is the second of two trapping conditions associated with the stack hardware. When this signal is generated the Control Unit must respond by transferring a value from the Memroy Stack to the Fast Stack. It is apparent that the destination register for this transfer could be any one of the four Fast Stack locations (TOS through TOS - 3) depending on the situation. In each case, however, the data value should be placed in the first empty location at the bottom of the Fast Stack (called the BOS register). This location can always be computed with the following 2-bit subtraction.

$$BOS = TOS - NIS$$

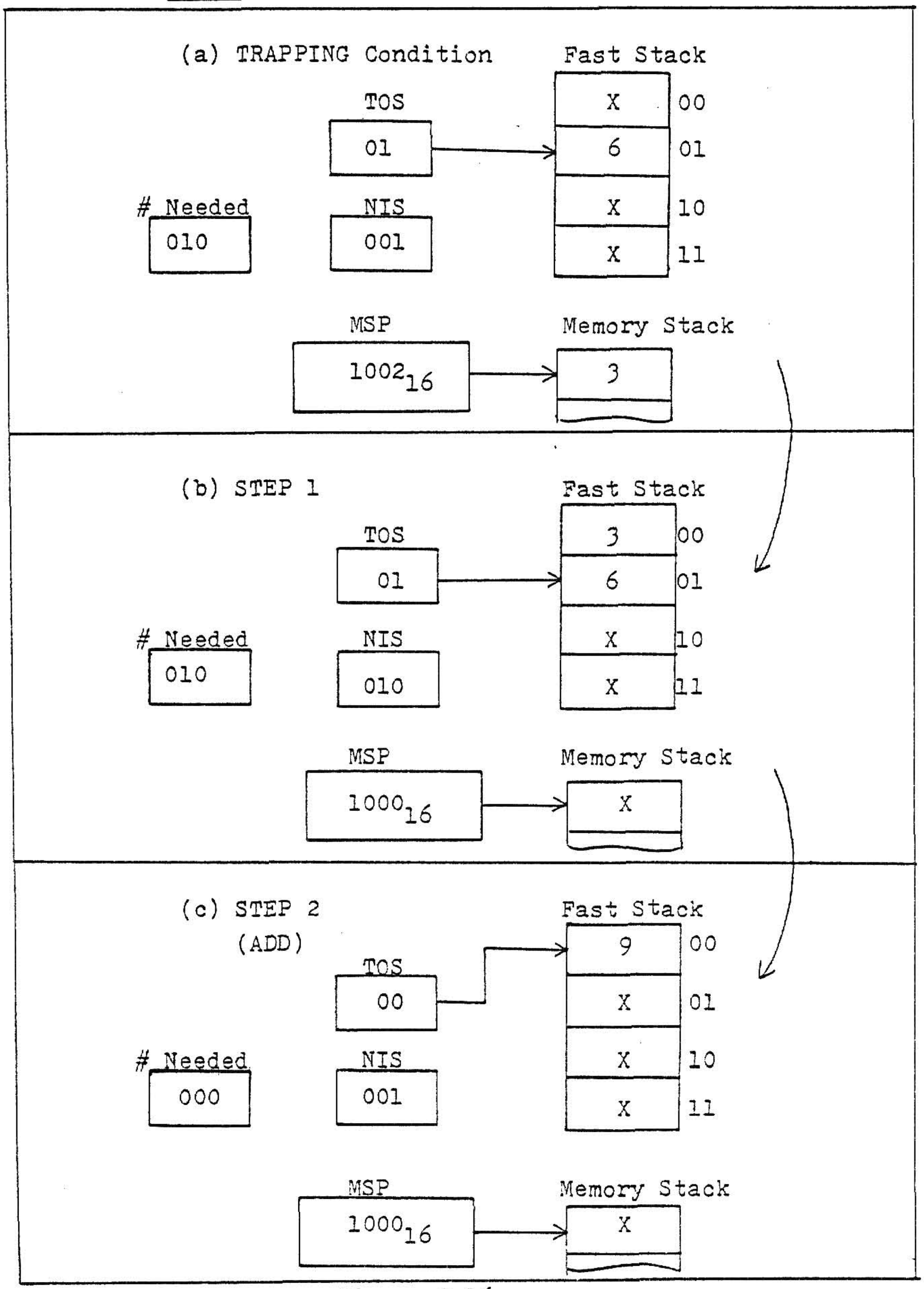

(a) TRAPPING Condition
Fast Stack
TOS
01
X
6
X
X
00
01
10
11
# Needed
010
NIS
001
MSP
1002_{16}
Memory Stack
3
(b) STEP 1
Fast Stack
TOS
01
3
6
X
X
00
01
10
11
# Needed
010
NIS
010
MSP
1000_{16}
Memory Stack
X
(c) STEP 2
(ADD)
Fast Stack
TOS
00
9
X
X
X
00
01
10
11
# Needed
000
NIS
001
MSP
1000_{16}
Memory Stack
X

Figure 7-16

For example, suppose a double precision add instruction is desired with the stack in Figure 7-17a. Obviously, two values are needed from the Memory Stack. Since the number of entries in the Fast Stack is insufficient, a trap is generated. A data value is fetched from the Memory Stack and placed in the BOS register.

$$\text{BOS} = \text{TOS} + \text{2's COMPLEMENT OF NIS}$$

$$\begin{array}{r} \text{TOS} = 10 \\ \underline{\text{2's of NIS} = 10} \\ \text{BOS} = 00 \end{array}$$

The result of this operation is shown in Figure 7-17b. A trap condition still exists however since the number of entries in the Fast Stack is still insufficient. Once again a data value is fetched from the Memory Stack and placed in the BOS register.

$$\begin{array}{r} \text{TOS} = 10 \\ \underline{\text{2's of NIS} = 01} \\ \text{BOS} = 11 \end{array}$$

The result is pictured in Figure 7-17c. The double precision add can now be performed. The final result is shown in Figure 7-17d.

In summary, the Fast Stack occupies the first four 2901 registers and operates as a circular buffer. A 2-bit counter register, TOS, always points to the register at the top of the Fast Stack. A 3-bit counter, NIS, keeps track of the number of valid entries in the Fast Stack. The Memory Stack Pointer, a 2901 register, always points to the

Illustration Pertaining to Computation of the BOS Register

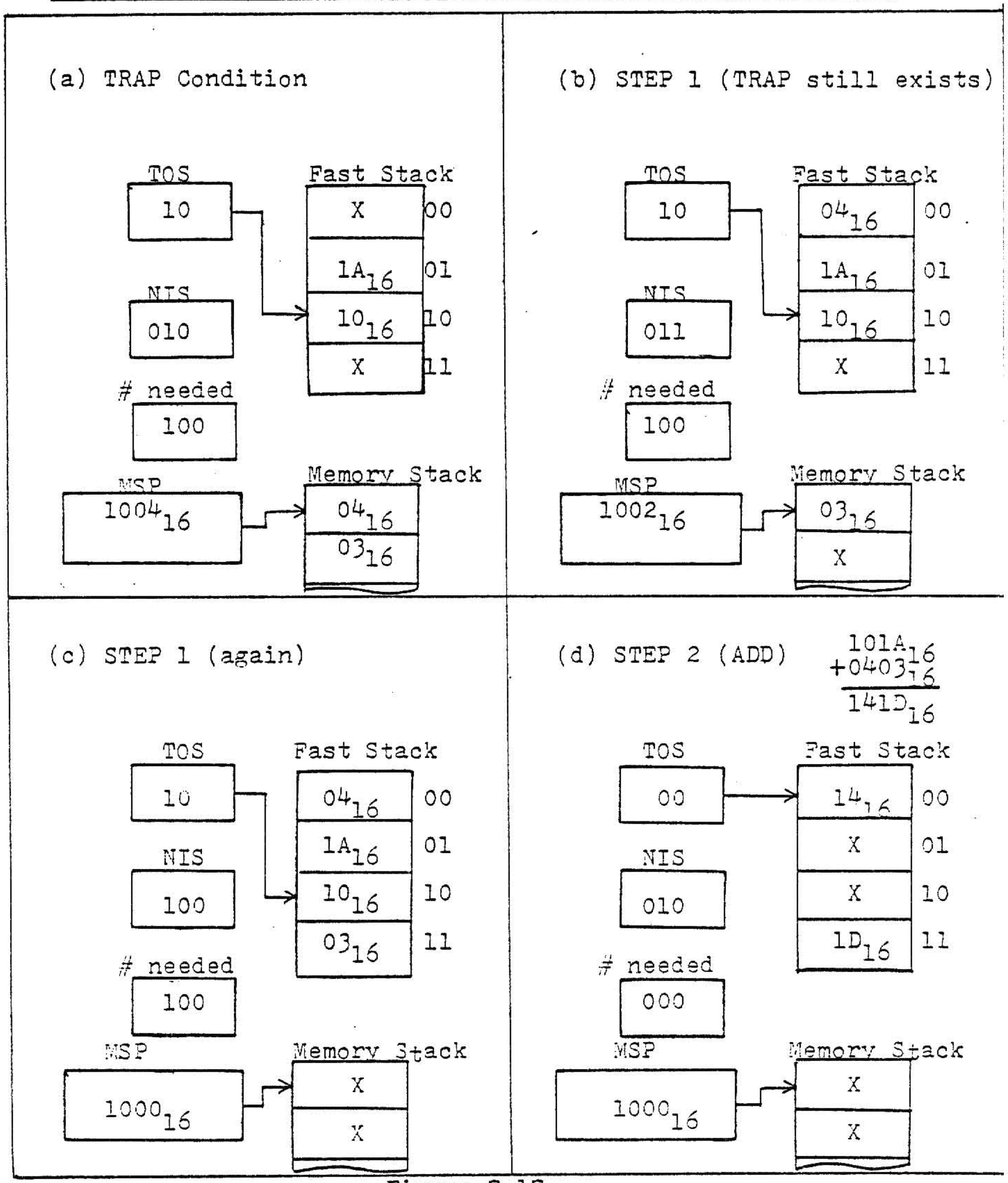


Figure 7-17

data value at the top of the stack in memory. Two hardware traps are generated as a result of Fast Stack conditions. The first is generated by a push instruction when the Fast Stack is full. The second condition occurs when the number of entries in the Fast Stack is insufficient to perform the present instruction. Both traps cause the Control Unit to respond with a transfer of data between the Fast Stack and the Memory Stack. This transfer will always modify the value of NIS, and leave TOS unchanged.

Figure 7-18 shows the hardware design for the Stack Tender/Register Select Unit. TOS and NIS both exist as 4 bit Up/Down counters, 74LS191's. Only the 2 low order bits are used for TOS, and the 3 low order bits for NIS. Both counters are clocked by the Execution Clock ($\overline{\emptyset_3}$), and are initialized to zero by the master reset signal at power up. The counters are controlled by 3 bits in the Pipeline Register. $\overline{U}/D$ determines the direction of the count. A 0 will count up and a 1 will count down. The counters can be enabled separately by placing a 1 in the appropriate EN TOS or EN NIS fields.

The four Fast Stack registers are accessed in the following manner. Since these registers work as a circular buffer, they are dynamic. That is, the actual register pointed to be TOS is changing with time. It is desirable, however, for the microcode to access the TOS register in a unique manner. The same is true of the other 3 Fast Stack registers, TOS - 1, TOS - 2, and TOS - 3. The Stack Tender has been designed so that these registers can be selected using the first four binary values of the A SEL and B SEL fields as defined in Figure 7-19. The TOS register can always be accessed in microcode by placing a 0000 in the appropriate SEL field. Similarly, the TOS - 1 register is always accessed with a 0001.

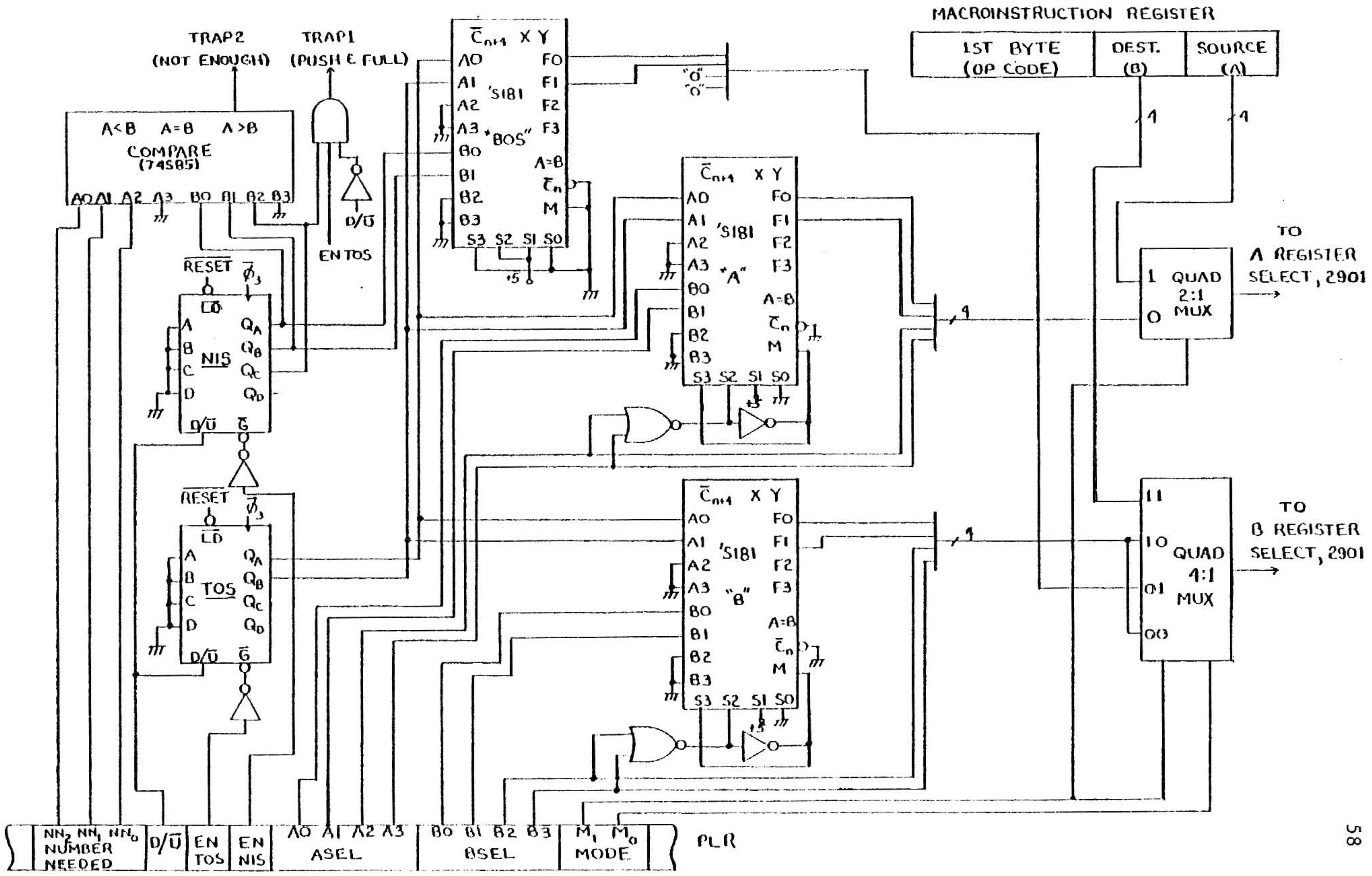


Figure 7-18 The Stack Tender/Register Select Circuit

Definition of the ASEL and BSEL Fields for Fast Stack Registers

ASEL or BSEL field				Selected FS Register
0	0	0	0	TOS
0	0	0	1	TOS-1
0	0	1	0	TOS-2
0	0	1	1	TOS-3

Figure 7-19

Fast Stack locations are accessed any time the upper two bits of the SEL field are zero. The actual Fast Stack register location in the 2901 is determined by the lower two bits using the following 2-bit equation.

FS REGISTER = TOS - SEL

The following is an illustrative example.

Assume the condition of the Fast Stack is that given in Figure 7-20. The TOS register is accessed in microcode by placing 0000 in the appropriate SEL field. The upper two bits of the resulting register select field are 00 (unchanged), while the lower two are computed using the above equation.

TOS + (2's complement of SEL)

$$\begin{array}{ll} \text{TOS} & = 10 \\ \underline{\text{2's of SEL}} & \underline{= 00} \\ \text{FS REGISTER} & = 10 \end{array}$$

Thus, the selected 2901 register is 0010 which is the desired TOS register. To select the TOS - 2 register a 0010 is placed in the SEL field. Again, the lower two bits are computed.

$$\begin{array}{ll} \text{TOS} & = 10 \\ \underline{\text{2's of SEL}} & \underline{= 10} \\ \text{FS REGISTER} & = 00 \end{array}$$

Stack Condition for Illustration of
Fast Stack Register Computation

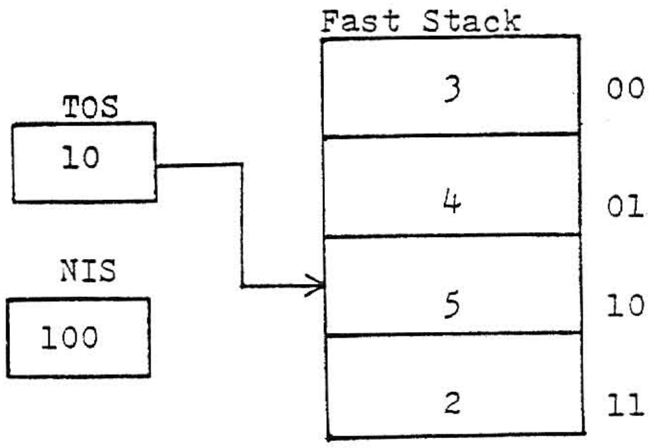


Figure 7-20

The select field actually sent to the 2901 in this case is 0000, the address of the TOS - 2 register. Further examples can be easily verified by the reader.

The above subtractions are carried out in hardware via a pair of 74S181 ALU's, one for the A SEL field, and the other for the B SEL field (see Figure 7-18). Both of these chips are designed to perform two operations. If the upper two bits of the appropriate SEL field are both zero (indicating the selection of a Fast Stack location), the 74S181's will carry out the 2-bit subtraction, TOS - SEL. If the upper two bits are non-zero, the 74S181s will pass the lower two SEL bits unmodified [3].

A composite 4-bit select field is then formed by concantinating the upper two bits of the SEL field with the 2-bit output of the 74S181. The A and B register select fields generated in this fashion will be referred to as the Pipeline Register Select Fields.

Recall that the trapping condition brought on by an insufficient number of entries in the Fast Stack requires knowledge of the BOS register address. This address is computed by the third 74S181 in Figure 7-18. It performs the 2-bit subtraction TOS - NIS. The full 4-bit BOS register address is formed by adding two high order zero bits to this 2-bit result.

Trap signals are generated in the following manner. TRAP1 results from an attempt to perform a push onto a full Fast Stack. This operation is detected by monitoring the TOS counter controls, and the number of entries in the Fast Stack. When the TOS counter is enabled to count up and the Fast Stack is full (NIS = 100), TRAP1 is generated.

$$\text{TRAP1} = (\text{NIS}_2) \wedge (\text{EN TOS}) \wedge (\overline{\overline{\text{U}}/\text{D}})$$

Similarly, TRAP2 is generated when the number of entries in the Fast Stack is insufficient to perform the present microinstruction. This signal is formed using a 74S85 magnitude comparator whose output is activated any time the NUMBER NEEDED field in the Pipeline Register is larger than the current value of NIS. The NUMBER NEEDED field should always hold a binary number from 000 to 100 indicating the number of Fast Stack entries needed by the current microinstruction. Both traps are sent to the Control Unit as status information.

The mode bits, M_1 and M_0, in Figure 7-18 determine the actual A and B register select inputs to the 2901s. They control a set of multiplexers whose inputs include the Pipeline Register Select Fields, the BOS select field, and two 4-bit select fields supplied directly from the second byte of the Instruction Register. The four modes of operation determined by M_1 and M_0 are summarized in Figure 7-21.

The first mode, 00, gives the microcode exclusive access to the ALU registers. Here, both the A and B registers are selected by the Pipeline Register Select Fields. Simple Stack operations, where both operands come from the top 4 entries in the Stack, are performed in this mode. For example, the operands for an instruction involving the top two values in the Stack are selected by placing 0000 and 0001 in the A SEL and B SEL fields respectively. Operations on the remaining 12 ALU registers are also performed in this mode. In addition, these registers can be transferred directly to and from the Stack by mixing Fast Stack and register addresses in the A SEL and B SEL fields. This type of transfer is useful during a change of context when register

Functional Definition of the Mode Bits M_1 & M_0

M_1	M_0	MODE	A Register Select	B Register Select
0	0	Normal	Pipeline Register Select Field determined by ASEL	Pipeline Register Select Field determined by BSEL
0	1	TRAP 2	Pipeline Register Select Field determined by ASEL	BOS Register Select Field
1	0	Index Addressing	IR_A , Instruction Register Select Field A	IR_B , Instruction Register Select Field B
1	1	Register Machine	IR_A , Instruction Register Select Field A (source register)	IR_B , Instruction Register Select Field B (destination reg.)

Figure 7-21

values must be saved or restored.

The second mode, 01, is used primarily when TRAP2 occurs. Recall, this condition requires a transfer of data from the top of the Memory Stack to the BOS register in the Fast Stack. In this mode, the destination register, B, is always the selected BOS register. The other register, A, is chosen by the corresponding Pipeline Register Select Field.

Mode 10, the third mode, is designed for use primarily with Memory Reference Instructions. Here the A register is selected by a 4-bit field from the second byte of the Instruction Register. In Memory Reference Instructions this field is used to select an Index register for use in computing the Effective Address as described earlier. The B register is selected by its corresponding Pipeline Register Select Field. In this manner the Index Register can be added to any one of the 2901 registers chosen by the microcode to form the Effective Address.

The last mode, 11, allows this computer to operate as a register machine. Both the A and B registers are selected by two 4-bit fields in the second byte of the IR. In this mode, the SEL fields in the Pipeline Register are ignored, and the Stack Tender hardware is essentially disabled.

In summary, the Stack Tender/Register Select Unit supplies the A and B register select addresses to the 2901s from a variety of sources. There are a total of 16 control bits in the Pipeline Register associated with this unit. Most of the hardware in this unit is dedicated to performing the various functions required by the Fast Stack. A set of multiplexers controlled by M_1 and M_0 provide four possible modes of operation. In one of these modes, the Fast Stack hardware is disabled, and the computer is allowed to function as a register machine.

7.4 The Data Select/Sign Extend Logic

The purpose of the Data Select/Sign Extend circuit is 3-fold. Its primary task is that of selecting a value for the direct data inputs (D inputs) to the 2901s. Possible sources for these inputs include;

The Bus Information Lines

The Instruction Register

The NIS Counter

The Pipeline Register

The Bus can supply both 8 and 16 bit values. 8 bit values always travel the high byte of the Bus. The Instruction Register supplies an 8-bit value from its second byte. This byte often carries information that must be added or subtracted from various ALU registers. The NIS counter is made available to the ALU for the purpose of computing a value for the logical Stack Pointer. Recall,

$$SP = MSP + NIS * 2$$

The NIS counter value is doubled here so that it represents the number of bytes in the Fast Stack, rather than the number of words. This doubling is accomplished by adding a least significant bit of 0 to the 3-bit counter output. Four additional 0's are used as upper bits to form an 8-bit source for the D inputs. The fourth possible source for the D inputs is an 8-bit register. This is an extremely useful feature because it allows the microcode to supply values directly to the ALU. Thus, such tasks as adding 2, 4, or 8 to various ALU registers can be performed in a single microinstruction rather than using a series of increments.

Any of the 8-bit sources mentioned above can be sign extended to 16 bits. This is the second task performed by this circuit. If these sources are not sign extended, the upper 8 bits are set to zero.

The third task performed by this circuit is quite specialized. It provides the option of forcing the D inputs to zero based on the value of a selected status signal. This feature is particularly useful when implementing conditional branch instructions in microcode. The normal method of implementing a conditional branch would require two microinstructions, one to test the status, and the other to add the relative value to the PC if the test is true. However, this feature allows the equivalent operation to be performed in a single micro-instruction. The microinstruction always adds a value to the PC as if the test were true. If the test is false however, 0's are forced into the D inputs, leaving the PC unchanged.

The hardware design for the Data Select/Sign Extend circuit is shown in Figure 7-22. Controls from the Pipeline Register are defined as follows.

1. BUS - Select Bus Value

   When high, the D input source is taken form the Bus.

2. DS_1, DS_0 - Data Select 1 and Data Select 0

   These two controls determine the source for the D inputs when

The Data Select/Sign Extend Circuit

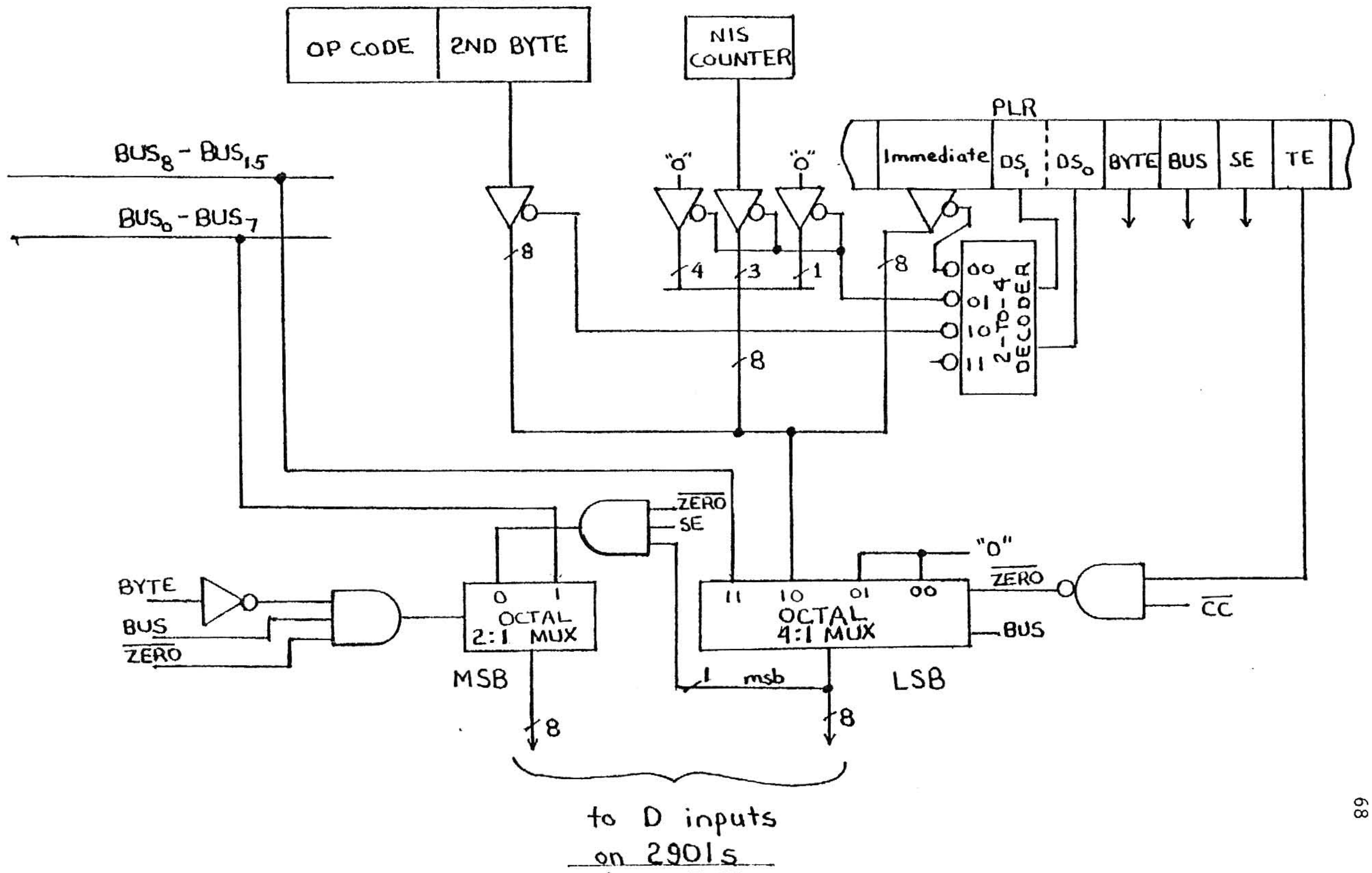


Figure 7-22

the BUS signal is low. They select from the NIS counter, the second byte of the IR, and the Immediate Field in the Pipeline Register.

3. BYTE - Byte Select

   When this bit is high, and BUS is high, only the high byte of the BUS is propagated to the D inputs. The other byte can be sign extended or set to 0's. This bit has no further meaning here when BUS is low. It does, however, perform an additional function in the CPU Bus Interface circuit to be described later.

4. SE - Sign Extend

   When high, the 8 bit value at the high byte of the D inputs will be sign extended to 16 bits. That is, the msb of the high byte will be propagated through the upper 8 bits. If an 8-bit source is selected while this bit is low, the upper 8 bits will be set to 0.

5. TE - Test Enable

   When high, the selected status from the Control Unit, CC, is allowed to force the D inputs to zero. If CC is low, the test is false and the D inputs are forced to zero. If CC is high, the test is true and the D inputs are determined in the normal fashion. When TE is low, the above action on the part of CC is disabled.

The hardware design in Figure 7-22 is relatively straightforward. When TE is high and the selected status signal is false, (CC is low), then a $\overline{\text{ZERO}}$ signal is generated (active low). Otherwise $\overline{\text{ZERO}}$ is high. The Data Select controls, DS_1, and DS_0, select one of three 8-bit sources via a 2-to-4 decoder. The selected source serves as one of three inputs to the LSB multiplexer. The other two inputs are the high byte of the Bus and zeros. When $\overline{\text{ZERO}}$ is high the output of the LSB multiplexer comes from either the Bus or the selected 8-bit source depending on the value of the BUS signal. When $\overline{\text{ZERO}}$ is low, the output is always zero, independent of the BUS signal. The MSB multiplexer selects between the low byte of the Bus and the Sign Extended value. As long as a full 16-bit value is chosen from the Bus and $\overline{\text{ZERO}}$ is high, the low byte of the BUS is propagated through. Otherwise the Sign Extended value, which is forced to 0 if SE is low or $\overline{\text{ZERO}}$ = low, is propagated through all 8 bits of the MSB multiplexer.

8 The CPU Bus Interface

The CPU Bus Interface is shown in Figure 8-1. All 9 of the Bus control signals,

$\overline{\text{MEMSEL}}$,

LDMAR,

R/$\overline{\text{W}}$,

BYTE,

$\overline{\text{I/O SEL}}$,

I/O STAT/$\overline{\text{DATA}}$,

$\overline{\text{INT ACK}}$,

$\overline{\text{MSKEN}}$, and

$\overline{\text{BRST}}$

come directly from the Pipeline Register. Of these signals, four are associated with main memory, and five with I/O. The function of each has been defined in Section 5 on The System Bus. Many of these signals are active low. Inverter buffers for those that are have been placed between the Pipeline Register and the corresponding Bus control line so that each can be activated by setting the control bit to 1. Note, all 9 signals can be disabled by the Console. When the CONSOLEON signal is activated, the control signals from the Pipeline Register are 3-stated. This effectively transfers Bus control to the Console. In normal operating mode the CONSOLEON signal is low.

The CPU places information on the Bus with the Bus Select control bits, BS_1 and BS_0. Through the use of a 2-to-4 decoder these two bits

CPU Bus Interface

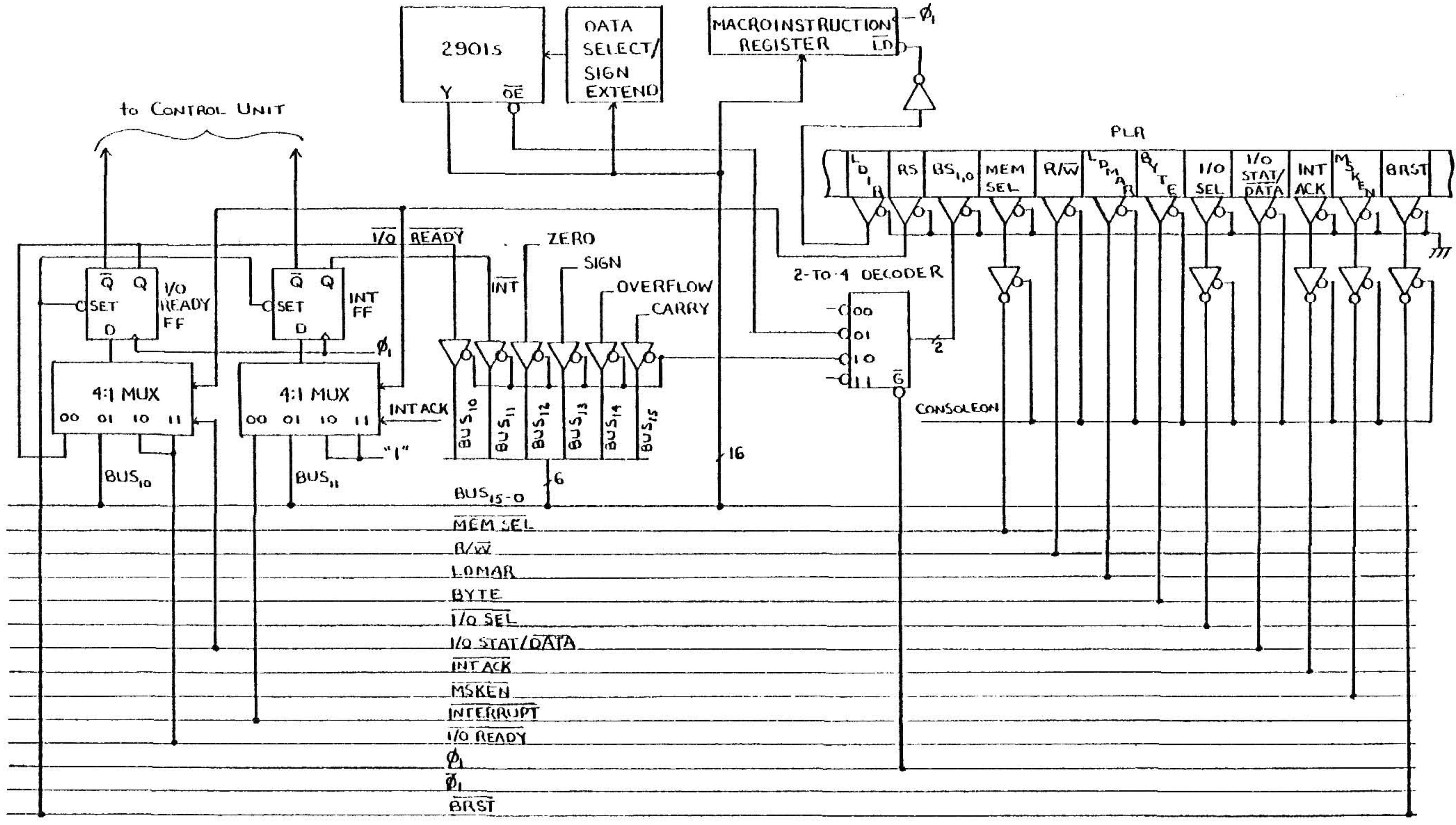


Figure 8-1

select one of three sources for Bus transfer. A 01 selects the Y outputs of the 2901 and a 10 the 6-bit status register. A 11 is reserved for selection of a third source which as yet is undetermined. Note, the decoder outputs are enabled by $\emptyset_1$ to ensure that information is placed on the Bus during periods 2 and 3 of the microinstruction cycle. When the Bus Select bits are 00, no source from the CPU is enabled onto the Bus.

The LDIR (Load Instruction Register) control bit is used to load the Instruction Register from the Bus. The Instruction Register will be loaded on the rising edge of $\emptyset_1$ when this bit is set to 1.

Four of the six status bits shown in Figure 8-1 come from the ALU. These include the CARRY, OVERFLOW, SIGN, and ZERO bits. The other two, INTERRUPT and I/O READY come from the Bus. A D flip-flop is associated with each. Both can be saved in memory along with the other 4 status bits. All status is restored from memory when the RS (Return Status) control bit is set to 1.

The mode of operation for the interrupt flip-flop is as follows. It is set to 0 at the end of the first cycle in which the $\overline{\text{INTERRUPT}}$ line is pulled low. It remains in this state until the CPU acknowledges the interrupt by activating the INT ACK control bit. This will reset the flip-flop to 1. The 4-to-1 multiplexer at the input to this flip-flop insures this mode of operation. When INT ACK is high, the D input to this flip-flop is a hard-wired 1. When INT ACK is low, the interrupt flip-flop will be set according to the $\overline{\text{INTERRUPT}}$ Bus line unless the RS bit is set, in which case the input to this flip-flop comes from a Bus information line. The $\overline{Q}$ output of this flip-flop is sent to the Control Unit as an active high status signal.

The I/O READY flip-flop functions somewhat differently. It is set according to the $\overline{\text{I/O READY}}$ Bus line only when such action is requested by the CPU. Otherwise it simply retains its past value. The operation of this flip-flop is also governed by a 4-to-1 multiplexer. When the I/O STAT/$\overline{\text{DATA}}$ control bit is high, this flip-flop is set according to the $\overline{\text{I/O READY}}$ Bus line. When I/O STAT/$\overline{\text{DATA}}$ is low it simply retains its past value, unless the RS control bit is set in which case the D input to this flip-flop comes from a Bus information line. The $\overline{Q}$ output of this flip-flop is sent to the Control Unit as an active high status signal just like that of the interrupt flip-flop.

9 Main Memory Design

Main Memory contains 64 K-bytes of storage arranged as 32 K-words of 16 bits each. It has been designed using the 6116 which is a 2Kx8 bit static CMOS RAM. A fully populated memory requires 32 of these RAMs arranged in two 8-bit banks of 16 blocks each (see Figure 9-1). The low and high bytes of a word in memory are stored in the same location of the left and right banks respectively. Individual bytes can be accessed by chosing a value from either the left or right bank of a selected word. The 16-bit Memory Address Register, MAR, for this system holds a byte address. The upper 15 bits address a word location in the RAM while the least significant bit, lsb, selects between the left and right bank location. During word transfers the lsb is ignored and the location in both banks is used.

The complete hardware design for Main Memory is shown in Figure 9-2. The 16 bit MAR works as a latch. It is loaded from the Bus information lines during periods 2 and 3 ($\overline{\emptyset}_1$ is high) when the LDMAR line is activated (high). The upper 4 bits of the MAR select one of sixteen 2K RAMs in each bank while the next ten bits are used to address a location within the selected RAM. The lsb of the MAR is combined with the BYTE control line from the Bus to generate two signals, $\overline{\text{LEFT}}$ and $\overline{\text{RIGHT}}$. $\overline{\text{LEFT}}$ will be low when byte information is to be transferred to/from the left bank of memory. Similarly, $\overline{\text{RIGHT}}$ will be low when byte information is to be transferred to/from the right bank. During word transfers both $\overline{\text{LEFT}}$ and $\overline{\text{RIGHT}}$ are high.

Information is transferred between Memory and the Bus via 3 groups of byte wide tranceivers labelled A, B, and C. Transceiver groups A and B

Main Memory Block Diagram

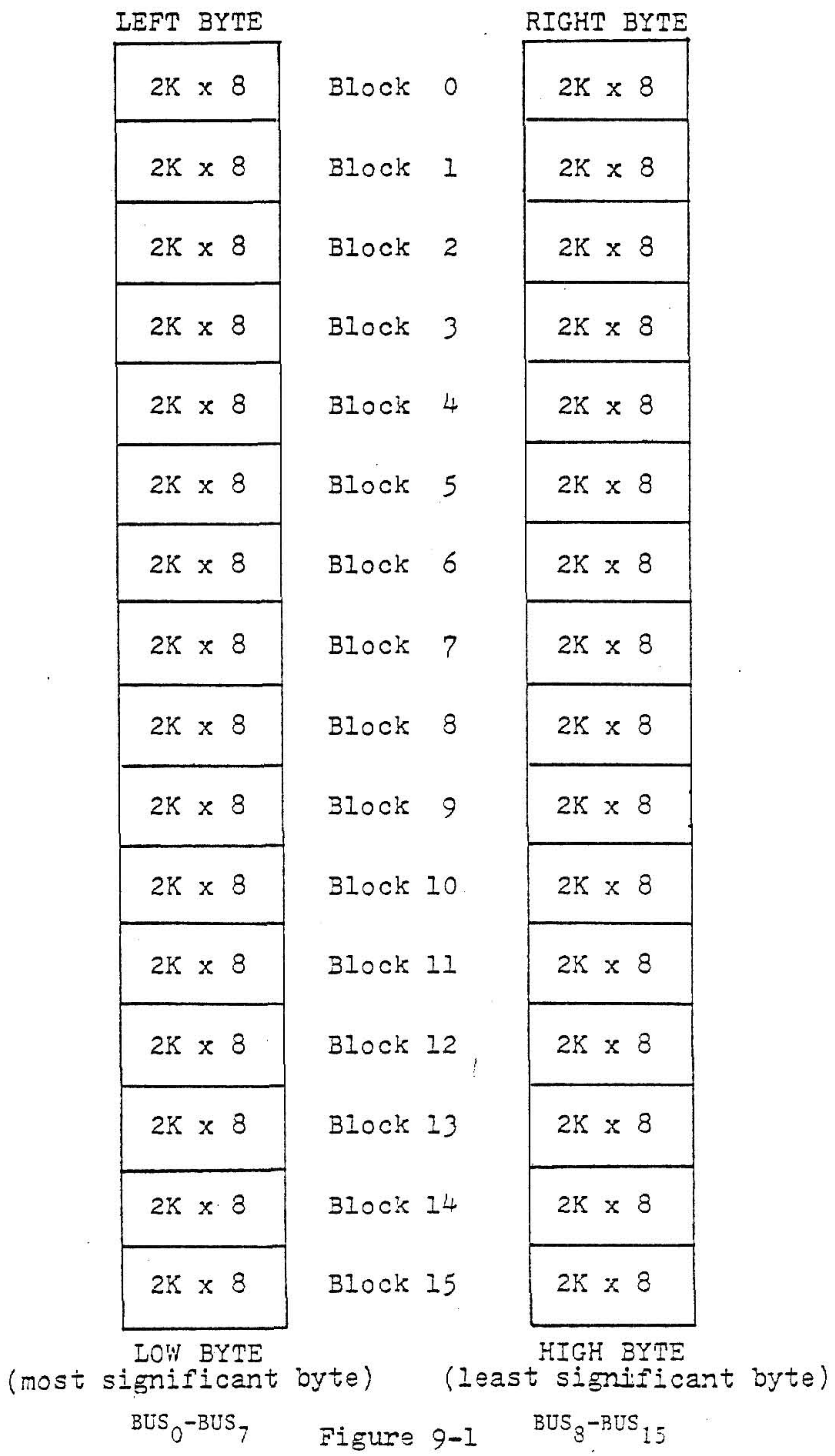


Figure 9-1

Memory Design

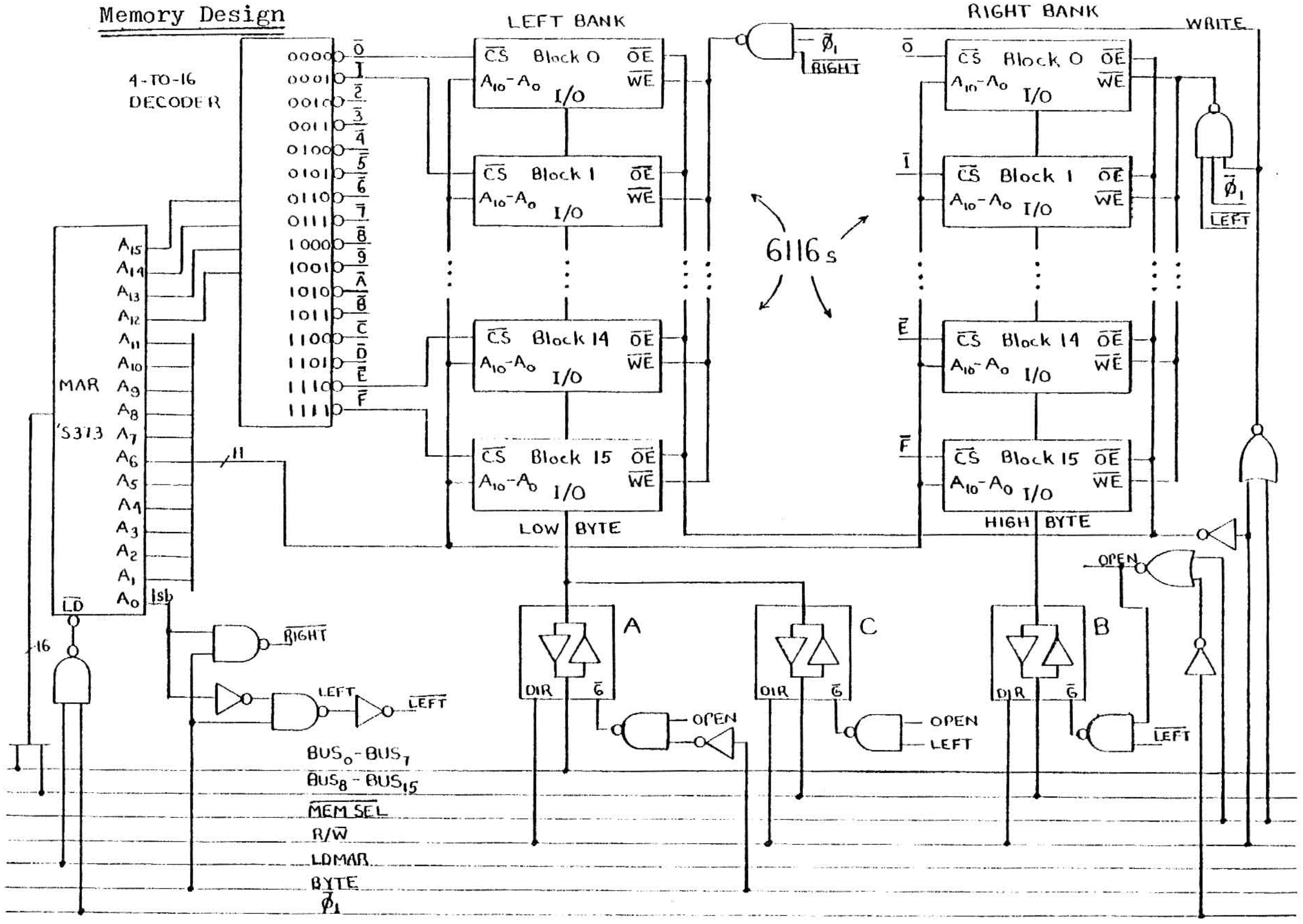


Figure 9-2

are used during word transfers, and B and C are used during byte transfers. Byte information is always transferred on the high byte of the Bus. The C transceiver group is used to move bytes between the left bank of Memory and the high byte of the Bus when $\overline{\text{LEFT}}$ is activated. The A transceiver group is used only in the word mode (i.e. when BYTE is low), and the B transceiver group is used for all transfers except when $\overline{\text{LEFT}}$ is activated. The directional control on all the transceivers is determined by the R/$\overline{\text{W}}$ control line. When the $\overline{\text{MEM SEL}}$ signal is activated, the appropriate transceiver groups are enabled to pass data during periods 2 and 3 of the microinstruction cycle.

A read operation with memory requires two consecutive microinstruction cycles, one to send the address, and the other to receive the data. In the 6116s, data access begins as soon as the MAR is loaded and the chip select signals have propagated through the 4-to-16 decoder. Since the MAR works as a latch, data access actually begins during the latter part of the first microinstruction cycle. The Output Enable, $\overline{\text{OE}}$, control on the 6116s comes directly from an inverted R/$\overline{\text{W}}$ signal, and is therefore activated very early in the first period of the data transfer cycle. This makes the data available for Bus transfer very early in the second period of this cycle. As a result, read data is available from the 6116s just as it's being enabled onto the Bus during the second period of the data transfer cycle.

A write operation also requires two consecutive microinstruction cycles, one to send the address and the other to send the data. The first of these two is identical to that for a read operation. Hence, the 150ns write operation in the 6116 also begins in the first microinstruction cycle. In the second cycle, the Write Enable, $\overline{\text{WE}}$, control on the 6116s is pulled low during periods 2 and 3 concurrent with the transceiver enable signals which supply data at the 6116 inputs. During byte transfers, the $\overline{\text{WE}}$ control will be enabled for one bank only. If a byte is to be written into the right bank, the $\overline{\text{WE}}$ control for the left bank is held high. Similarly, if a byte is to be written into the left bank, the $\overline{\text{WE}}$ control for the right bank is held high.

A more detailed analysis of both read and write cycle timing for memory is presented in Appendix A.

In summary, the hardware design for main memory has been presented in detail. Its major components are the 16-bit MAR and the 32K word by 16-bit storage composed of 6116s. A memory buffer register is not needed in this system because no handshaking is required for memory transfers. Both reads and writes are performed in two microinstruction cycles, one to send the address, and the other for the data transfer. Memory can be accessed in the byte mode by activating the BYTE Bus control line. This signal controls only the routing of data, and therefore need be activated only during the data transfer cycle. Byte data always traverses the high byte of the Bus. During a word transfer with memory, the lsb of the MAR is ignored. Thus, words are always read from and written to even byte addresses.

10 Direct I/O

Data is transferred between the CPU and Direct I/O devices via the Bus information lines. Each I/O device requires a Bus Interface Circuit to synchronize its interaction with the CPU. A simple interface contains a data buffer register, Device Code decoding logic, and a device Ready Flag. The Ready Flag is used to implement a full handshake between the CPU and the I/O device.

The handshake for an Input device functions in the following manner. The interface enters the ready state when the Input device places data in the Input Buffer Register, IBR. It is returned to the not ready state by the CPU when it reads this value. To the CPU, the Ready Flag indicates the validity of data in the IBR. The same Ready Flag tells the Input device when the IBR has been emptied.

The handshake for an Output device functions in a similar manner. The interface will enter the not ready state when the CPU places data in the Output Buffer Register, OBR. It is returned to the ready state by the Output device when it empties the OBR. Thus, the Ready Flag tells the CPU when the OBR can receive more data. To the Output device, it signifies the validity of the data in the OBR.

The various control, data, and status signals between the interface and the Bus include

$\overline{\text{I/O SEL}}$,

I/O STAT/$\overline{\text{DATA}}$,

$\overline{\text{I/O BUSY}}$,

$\emptyset_1$,

$\overline{\text{BRST}}$,

and the Bus information lines.

The function of each of these signals has been defined earlier in section 5. Recall, the Bus information lines carry Device Codes as well as data. The various signals supplied by the actual I/O devices will differ slightly between Input and Output interfaces (see Figure 10-1).

An Input device must provide the interface with the data and two control signals. In return it receives a status flag, $\overline{\text{EMPTY}}$. When $\overline{\text{EMPTY}}$ is low, the IBR is empty. To place a value in the IBR the Input device must perform the following three operations;

1. Supply data to the IDATA lines.

2. Enable the IBR by pulling the $\overline{\text{LDIBR}}$ line low.

3. Load the data into the IBR by supplying a low-to-high transition at the ICLOCK input.

As soon as the data is loaded by the clock transition the interface is placed in the ready state and the $\overline{\text{EMPTY}}$ flag is returned high. It will remain in this state until the IBR is read by the CPU.

An Output device must also supply the interface with two control signals. In return it receives a status flag, $\overline{\text{DAV}}$, and data from the OBR. A low state of $\overline{\text{DAV}}$ signifies the presence of data in the OBR. To take this data the Output device must;

Input and Output Interface Diagram

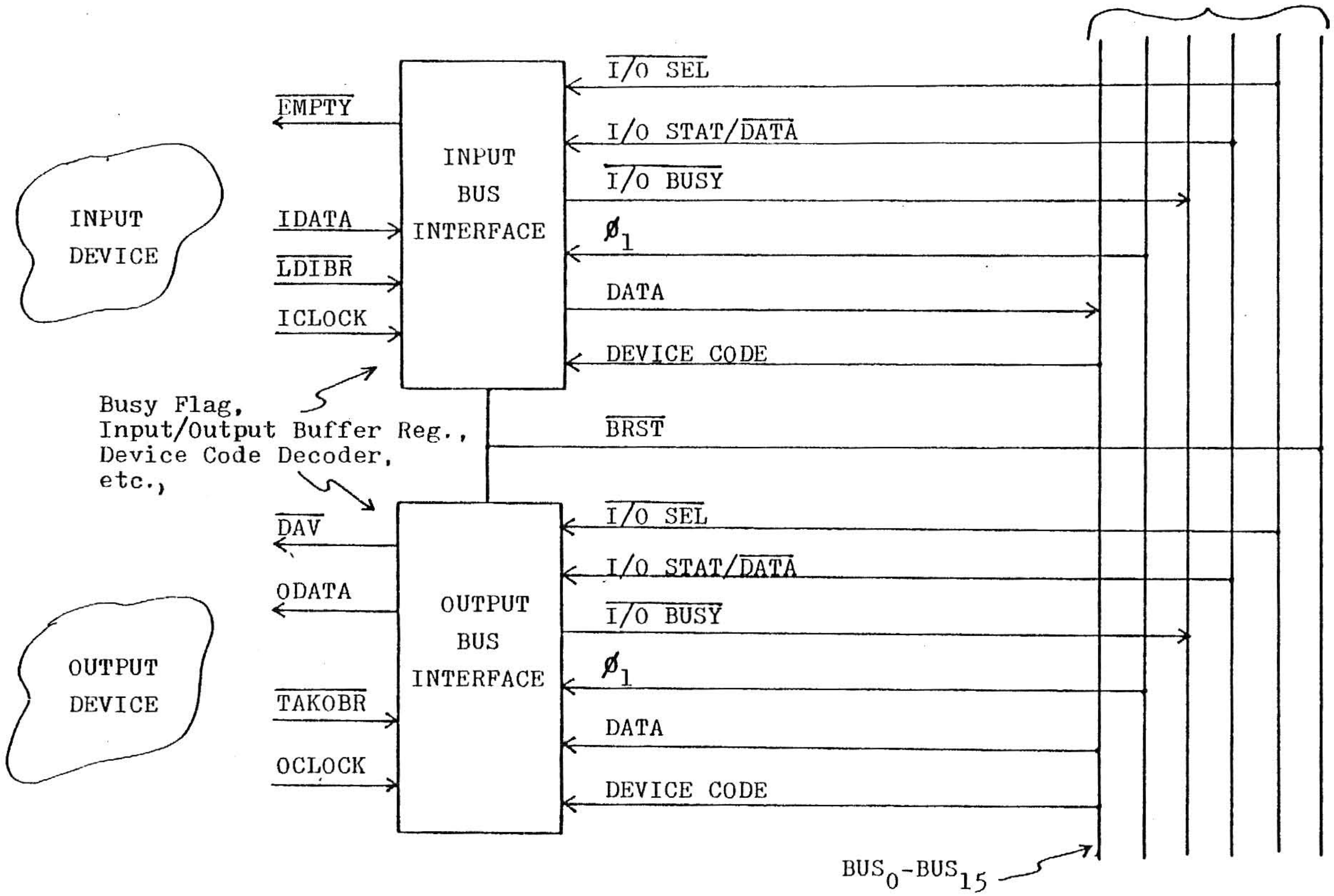

BUS Lines
$\overline{\text{I/O SEL}}$
I/O STAT/$\overline{\text{DATA}}$
$\overline{\text{I/O BUSY}}$
ϕ_1
DATA
DEVICE CODE
$\overline{\text{BRST}}$
I/O SEL
I/O STAT/$\overline{\text{DATA}}$
$\overline{\text{I/O BUSY}}$
ϕ_1
DATA
DEVICE CODE
BUS_0-BUS_{15}
INPUT BUS INTERFACE
OUTPUT BUS INTERFACE
INPUT DEVICE
OUTPUT DEVICE
$\overline{\text{EMPTY}}$
IDATA
$\overline{\text{LDIBR}}$
ICLOCK
$\overline{\text{DAV}}$
ODATA
$\overline{\text{TAKOBR}}$
OCLOCK
Busy Flag,
Input/Output Buffer Reg.,
Device Code Decoder,
etc.,

Figure 10-1

1. Place the OBR value on the ODATA lines by pulling the $\overline{\text{TAKOBR}}$ line low.

2. Signify the completion of the transfer by supplying a low-to-high transition on the OCLOCK line. (Note: $\overline{\text{TAKOBR}}$ must remain low during the OCLOCK transition.)

On the low-to-high transition of OCLOCK, the Output Interface is placed in the ready state and the $\overline{\text{DAV}}$ signal is returned high. A CPU write to the OBR will again place this interface in the not ready state.

10.1 Input Interface Design

The hardware design for an Input Bus Interface is shown in Figure 10-2. The Device Code for this interface is 1, as determined by the 8 input NAND gate at the left of the figure. The input of this gate will go low when Device Code 1 is placed on the Bus. This interface contains an 8-bit IBR (which could just as easily be 16-bits). The $\overline{\text{LDIBR}}$ signal from the Input device controls the load enable on the IBR. It also serves, indirectly, as an input to the Asynchronous Ready flip-flop. Thus when the ICLOCK transition occurs, the IBR is loaded from the IDATA lines and the Asynchronous Ready flip-flop is set to 0. A second Ready flip-flop is used to synchronize the setting of the Ready Flag with the Bus timing. This flip-flop is set to 1 on the first low-to-high transition of $\emptyset_1$ after the IBR has been loaded with data. When Device 1 is selected, and the I/O STAT/$\overline{\text{DATA}}$ line is high, the value of this flip-flop will be transferred to the CPU over the I/O READY Bus line. (Note, the $\overline{Q}$ output of the Data Select flip-flop is normally 0.)

Two microinstruction cycles are required for a CPU read from the IBR. In the first cycle the Device Code is placed on the Bus and the $\overline{\text{I/O SEL}}$ line pulled low, much the way an address is sent to memory. In the same cycle the I/O STAT/$\overline{\text{DATA}}$ line is pulled low to distinguish this operation from one that simply reads the Ready Flag. As a result the Data Select flip-flop in the interface circuit will be set to 0 at the end of the cycle. In the second cycle data is transferred to the CPU. Once again the $\overline{\text{I/O SEL}}$ and I/O STAT/$\overline{\text{DATA}}$ lines must both be low. The content of the IBR is then placed on the Bus for transfer to the CPU during the second and third periods of this cycle. During the data transfer, the Asynchronous and Synchronous Ready flip-flops are set and cleared respectively, thus placing the interface in the not ready state. Note, the Data Select flip-flop will automatically return to 1 at the beginning of the next (3rd) cycle, insuring that the transfer occurs only during the second cycle. The application of a Master Bus Reset, $\overline{\text{BRST}}$, has the effect of clearing the Synchronous Ready flip-flop and setting the Asynchronous Ready flip-flop, hence placing the interface in the not ready state.

10.2 Output Interface Design

The hardware design for an Output Bus Interface is shown in Figure 10-3. It strongly resembles that of the Input interface and works in a very similar manner. The Device Code for this interface is 0. The 8-bit OBR is easily extended to 16 bits. The output of the OBR is 3-stated so it can be connected directly to a 3-state Bus. The $\overline{\text{TAKOBR}}$ signal, supplied by the Output device, is used to place the OBR output on the

Input Bus Interface Design

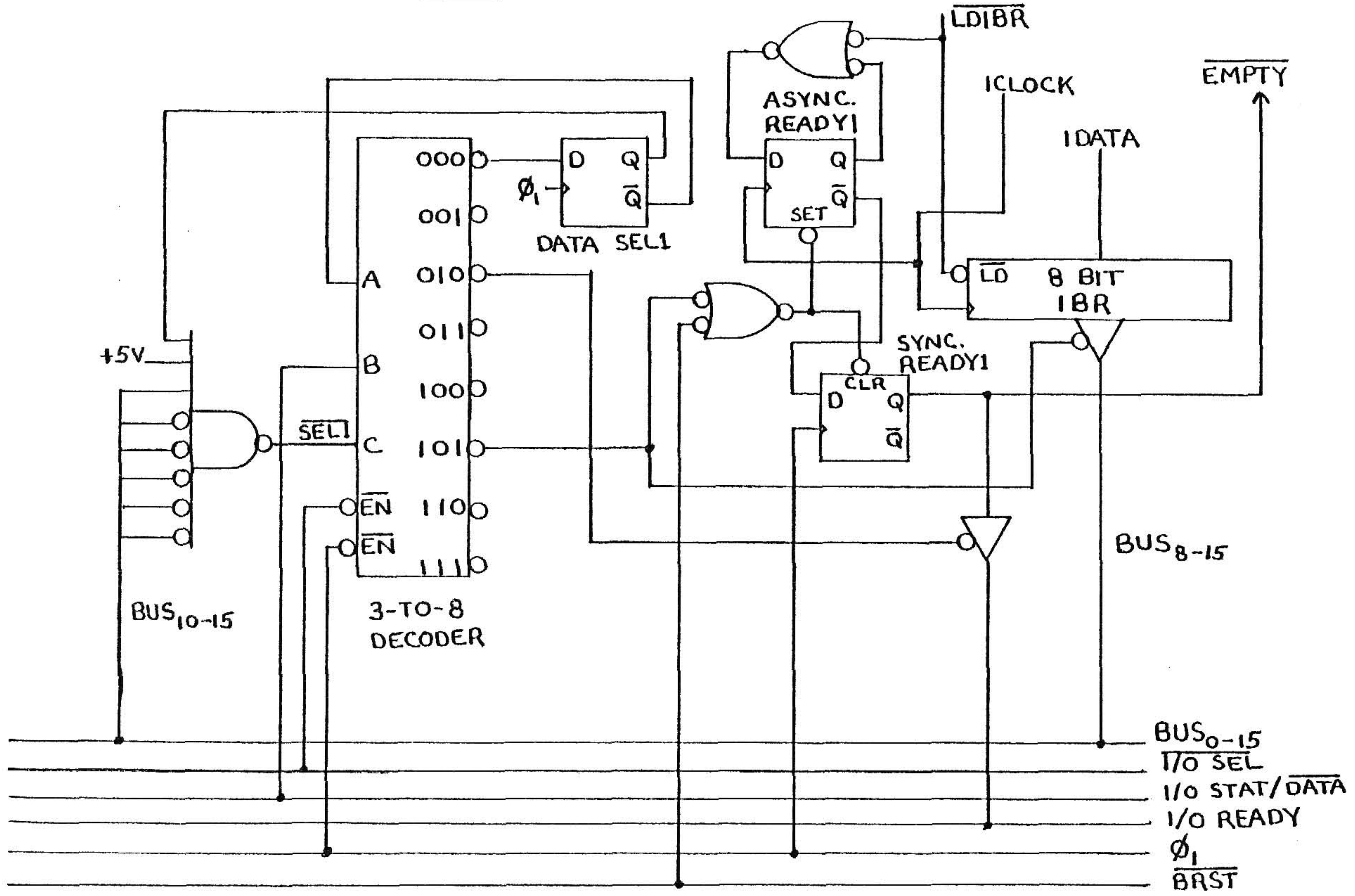


Figure 10-2

Output Bus Interface Design

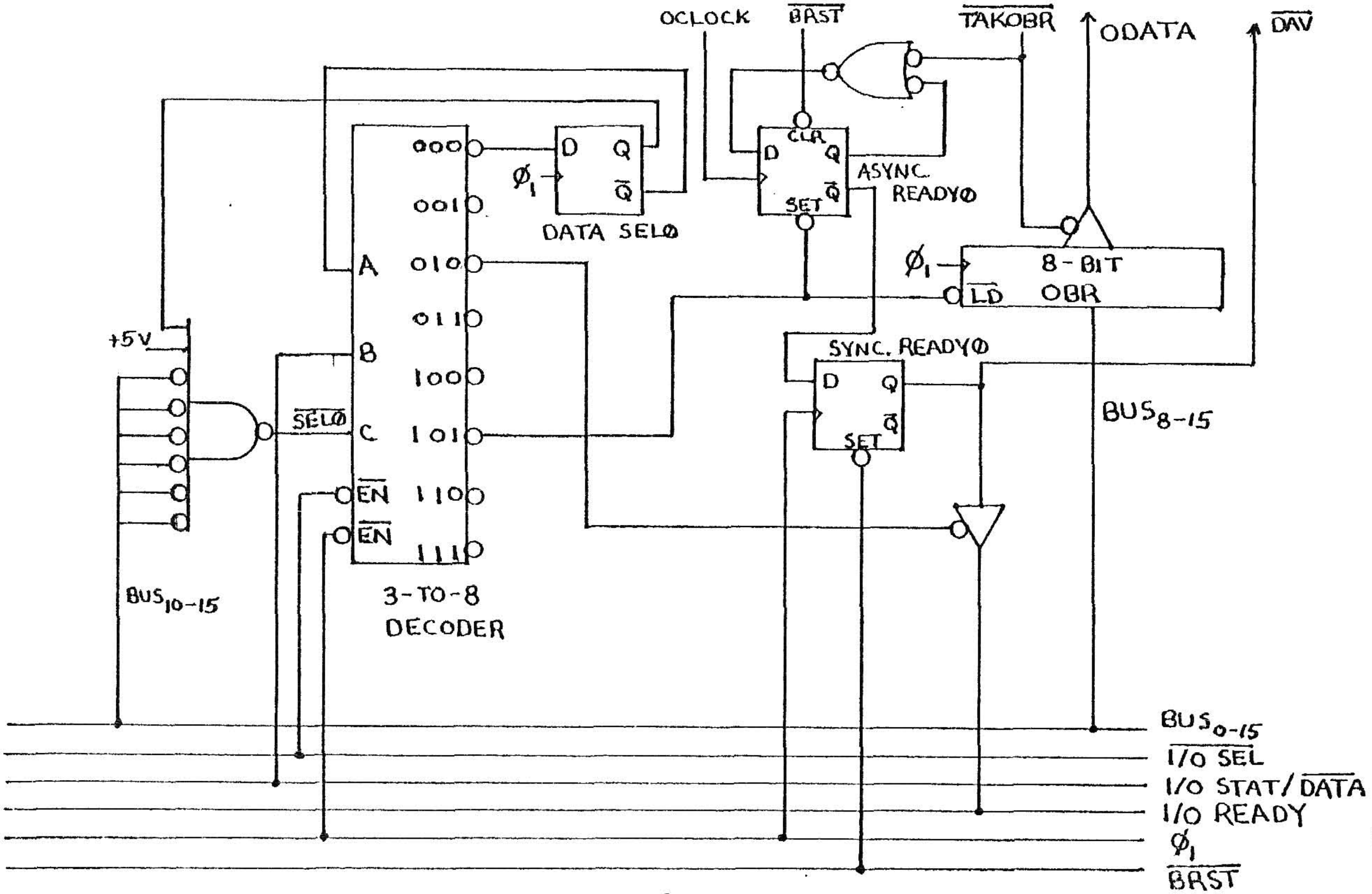


Figure 10-3

ODATA lines. It will also, on the low-to-high transition of the OCLOCK signal, activate the Asynchronous Ready flip-flop. Again, a second Ready flip-flop is used to synchronize the setting of the Ready Flag with the Bus timing. The value of this flag is transferred to the CPU when Device 0 is selected and the I/O STAT/$\overline{\text{DATA}}$ line is high.

A CPU write to the OBR requires two microinstruction cycles. During the first, the Device Code is placed on the Bus, and the $\overline{\text{I/O SEL}}$ and I/O STAT/$\overline{\text{DATA}}$ lines pulled low. This causes the Data Select flip-flop to be set to 0 at the end of this cycle. In the second cycle, data is transferred from the CPU to the OBR. Both $\overline{\text{I/O SEL}}$ and I/O STAT/$\overline{\text{DATA}}$ are again pulled low and work with the Data Select flip-flop to activate the load enable on the OBR. Thus, at the end of this cycle, the OBR is loaded from the Bus. The Data Select flip-flop also sets the Asynchronous Ready flip-flop during this cycle. As a result the Synchronous Ready flip-flop is cleared at the end of this cycle by the same clock pulse that loads the OBR. Thus, the interface is placed in the not ready state, and the $\overline{\text{DAV}}$ signal indicates the presence of data in the OBR. At the end of the second cycle the Data Select flip-flop is returned to 1, thus enabling the data transfer for the second cycle only. A Master Bus Reset will place this interface in the ready state.

10.3 Interrupt Level Interface Design

Devices that use the interfaces described above are passive in nature. That is, they are unable to initiate a transaction with the CPU. To do this, additional circuitry must be added to provide interrupting capabilities. This system supplies up to 16 different interrupt levels for I/O devices. Numerous devices can share the same level. In this case, however, some sort of priority system must be arranged between these devices to handle multiple interrupt conditions. If no more than 16 I/O devices are to be connected to the system, each device can reside at a different interrupt level eliminating the need for an additional priority system.

Interrupts to the CPU are generated and handled in the following manner. An interrupt is initiated at one of the 16 levels by pulling the $\overline{\text{INTERRUPT}}$ Bus line low in synchronism with the rising edge of $\emptyset_1$. When it's ready, (perhaps a few microinstructions later), the CPU responds by pulling the $\overline{\text{INT ACK}}$ line low. This signal propagates through a daisy chain network to the highest priority interrupting interface. The interface at this level must break the chain and place the Device Code of the interrupting device on the Bus so that the CPU can determine the source of the interrupt. The CPU has the capability of masking interrupts from any of the 16 levels. Associated with each level is an Interrupt Mask Bit which when set to 1 will disable all interrupts from that level. The CPU sets the state of these bits by placing a 16-bit mask word (1 bit for each level) on the Bus Information lines and pulling the $\overline{\text{MASKEN}}$ line low. In this way the interrupt mask bits for all levels can be set/cleared in one operation.

The hardware design for the interface at interrupt level 0 is shown in Figure 10-4. An interrupt is initiated in this interface by setting the Asynchronous Interrupt flip flop to 1. This is accomplished through proper application of the $\overline{\text{INTRPT}}$ and I/O CLOCK inputs. Next, if the Mask flip-flop is not set ($\overline{Q} = 1$), the Synchronous Interrupt flip-flop will be activated ($Q = 1$) on the next low-to-high transition of $\emptyset_1$. In this way, the $\overline{\text{INTERRUPT}}$ Bus line is pulled low at the beginning

Interrupt Level 0 Bus Interface Design

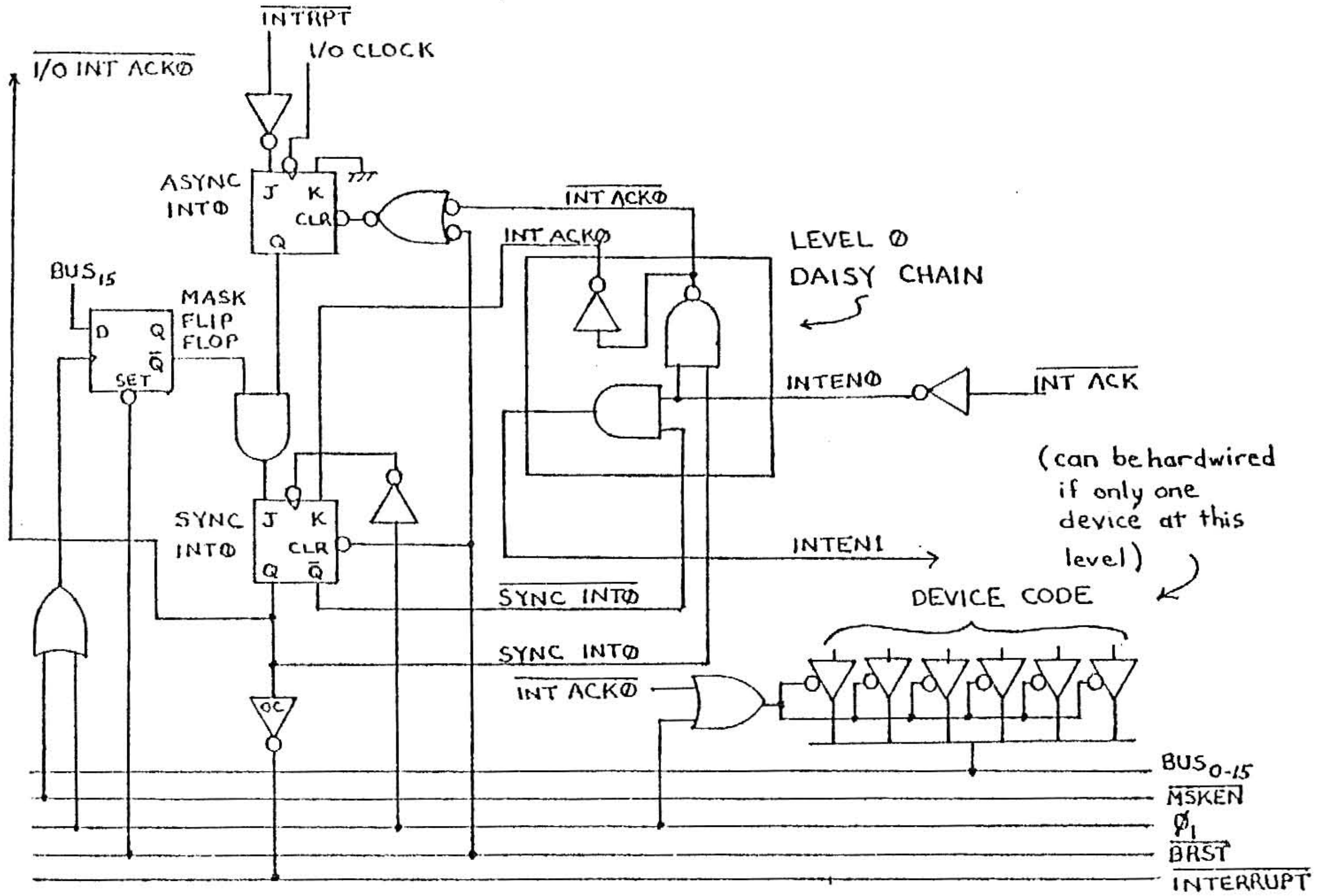

$\overline{\text{INTRPT}}$
I/O CLOCK
$\overline{\text{I/O INT ACK}\emptyset}$
ASYNC INTØ
J
K
CLR
Q
$\overline{\text{INT ACK}\emptyset}$
INT ACKØ
LEVEL Ø DAISY CHAIN
BUS$_{15}$
D
Q
$\overline{Q}$
SET
MASK FLIP FLOP
INTENØ
$\overline{\text{INT ACK}}$
(can be hardwired if only one device at this level)
SYNC INTØ
INTEN1
$\overline{\text{SYNC INT}\emptyset}$
DEVICE CODE
SYNC INTØ
$\overline{\text{INT ACK}\emptyset}$
OC
BUS$_{0-15}$
$\overline{\text{MSKEN}}$
$\emptyset_1$
$\overline{\text{BRST}}$
$\overline{\text{INTERRUPT}}$

Figure 10-4

of a microinstruction cycle. It will remain low until all unmasked interrupts have been serviced.

To service an interrupt, the CPU first pulls the $\overline{\text{INT ACK}}$ line low, expecting in return to receive the Device Code of highest priority interrupting device. The $\overline{\text{INT ACK}}$ signal functions as the input to a daisy chain network. The daisy chain connections for this interrupt level are shown in Figure 10-4.

In general, INTENi is a signal supplied by the i - 1 interface level. If high, the Interrupt Acknowledge from the CPU has propagated through all the higher levels, (0 to i - 1). If this interface is not interrupting ($\overline{\text{SYNCH INTi}}$ = 1), then the signal is propagated forward to the next interface level, INTENi+1 . However if this interface is interrupting, a 0 is propagated to INTENi+1 , which in turn is propagated through the remaining i + 2 to 16 levels. In addition, an INT ACKi signal and its complement are generated for use by the local interface circuit.

In Figure 10-4 it is apparent that the $\overline{\text{INT ACK0}}$ signal, along with $\emptyset_1$, is used to place a Device Code on the Bus. If there is only one device at this interrupt level then the Device Code can be a hard-wired value, otherwise the Device Code must be that of the I/O device allowed to interrupt at this level. In the latter case, the Device Code will most likely be supplied by some additional priority logic designed solely for devices at this interrupt level. The $\overline{\text{INT ACK0}}$ signal is also used to clear the Asynchronous Interrupt flip-flop, which in turn clears the Synchronous Interrupt flip-flop on the rising edge of $\emptyset_1$. This is an indication to the interrupting device that its request has been acknowledged.

The Interrupt Mask flip-flop at level 0 is set via the Bus_{15} information line by $\emptyset_1$ when the $\overline{\text{MASKEN}}$ line is pulled low. Masks for the remaining 15 levels are set in a similar fashion according to the states of the other 15 Bus information lines. A Master Bus Reset will set the Mask bit in all devices to a 1, thus inhibiting interrupts at any level. It will also clear both interrupt flip-flops, nullifying interrupts that may have been pending.

10.4 General I/O Design

The interfaces presented above are quite general. Figure 10-5 illustrates a simple way to make an intelligent I/O device for this system. Here the I/O Ports from a typical microprocessor system are used to communicate with the Input, Output, and Interrupt interfaces. The actual Input and Output devices are part of the microprocessor system. Microprocessor RAM can be used to buffer both Input and Output data. Both devices can function at the same interrupt level. Software in the microprocessor can be used to prioritize between the two, and Device Codes for each can be supplied to the Interrupt interface via the microprocessor I/O Port. This system can be easily expanded to incorporate even more I/O devices by simple replication of existing circuitry.

Intelligent I/O Device Block Diagram

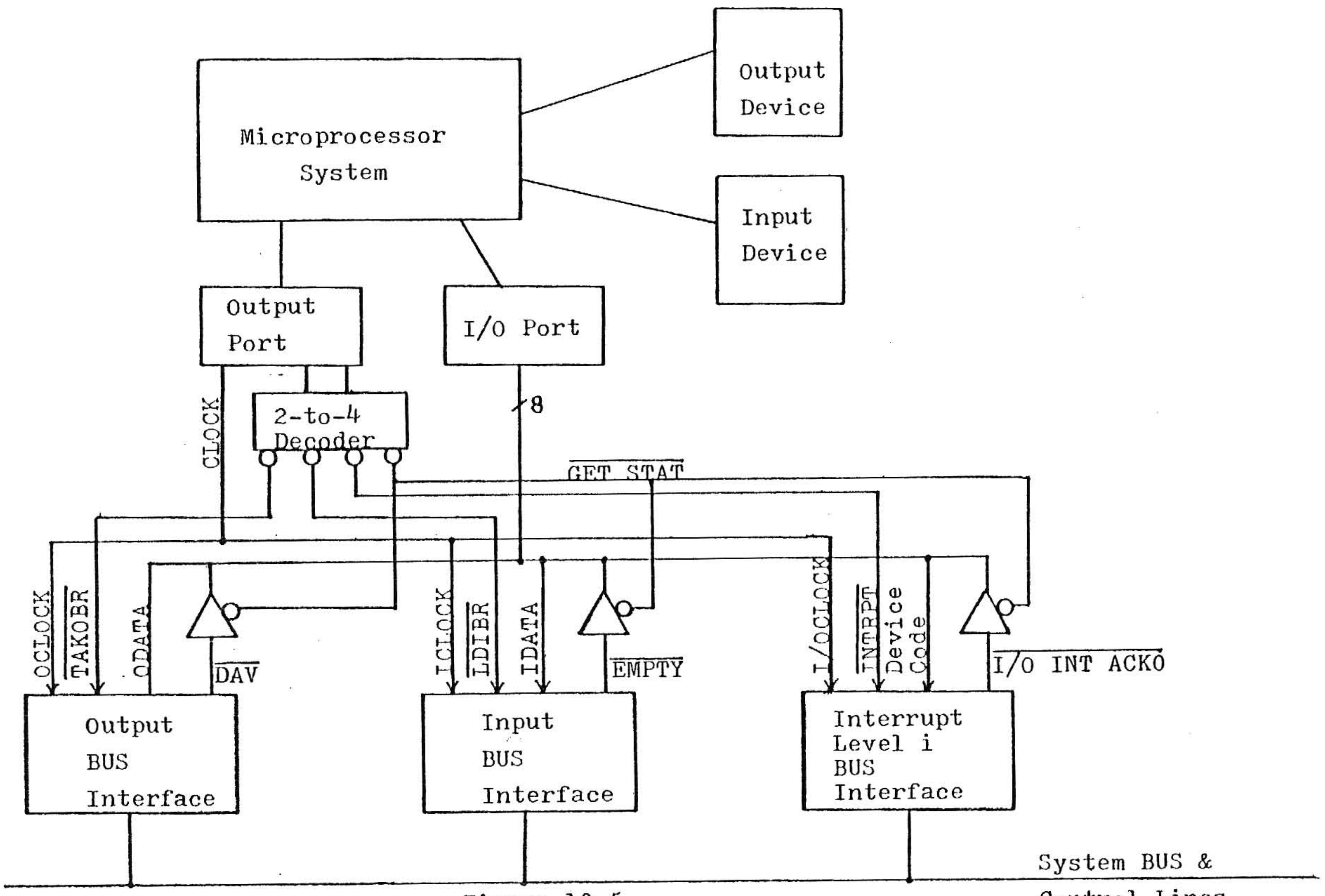


Figure 10-5

11 The Control Unit

The Control Unit for this machine is microprogrammed (see Figure 11-1). The sequence of steps required to carry out the fetch and execution of macroinstructions is performed by a series of microinstructions. Microinstructions are 88 bit control words that reside in the Control Store, CS. Microinstructions to be executed are read into the Pipeline Register, PLR. Individual bits of this register supply controls needed by various units throughout the computer. Microinstructions are usually divided into two major fields. The first field provides controls needed to execute an operation with the ALU, memory, or I/O. The second field provides controls which determine the next microinstruction. Both sets of controls work in parallel. That is, the fetch of the next microinstruction is overlapped with the execution of the current one. The longer of these two operations determines the length of a microinstruction cycle.

Instrumental in the fetch of the next microinstruction is the microprogram sequencer (also called the Next Address Generator). This unit supplies the CS with the address of the next microinstruction (the Next Address). This address can come from 5 possible sources, two of which are shown in Figure 11-1, the Next Address field from the PLR and the Macroinstruction Input. Determination of the Next Address is a function of both the Fetch Controls in the current microinstruction, and the Conditional Control, CC, obtained from the status register. The Fetch Controls generally narrow the Next Address selection from 5 sources to 2, while the Conditional Control chooses from the remaining two. Selection of the Next Address based on the Conditional Control input is

The Control Structure

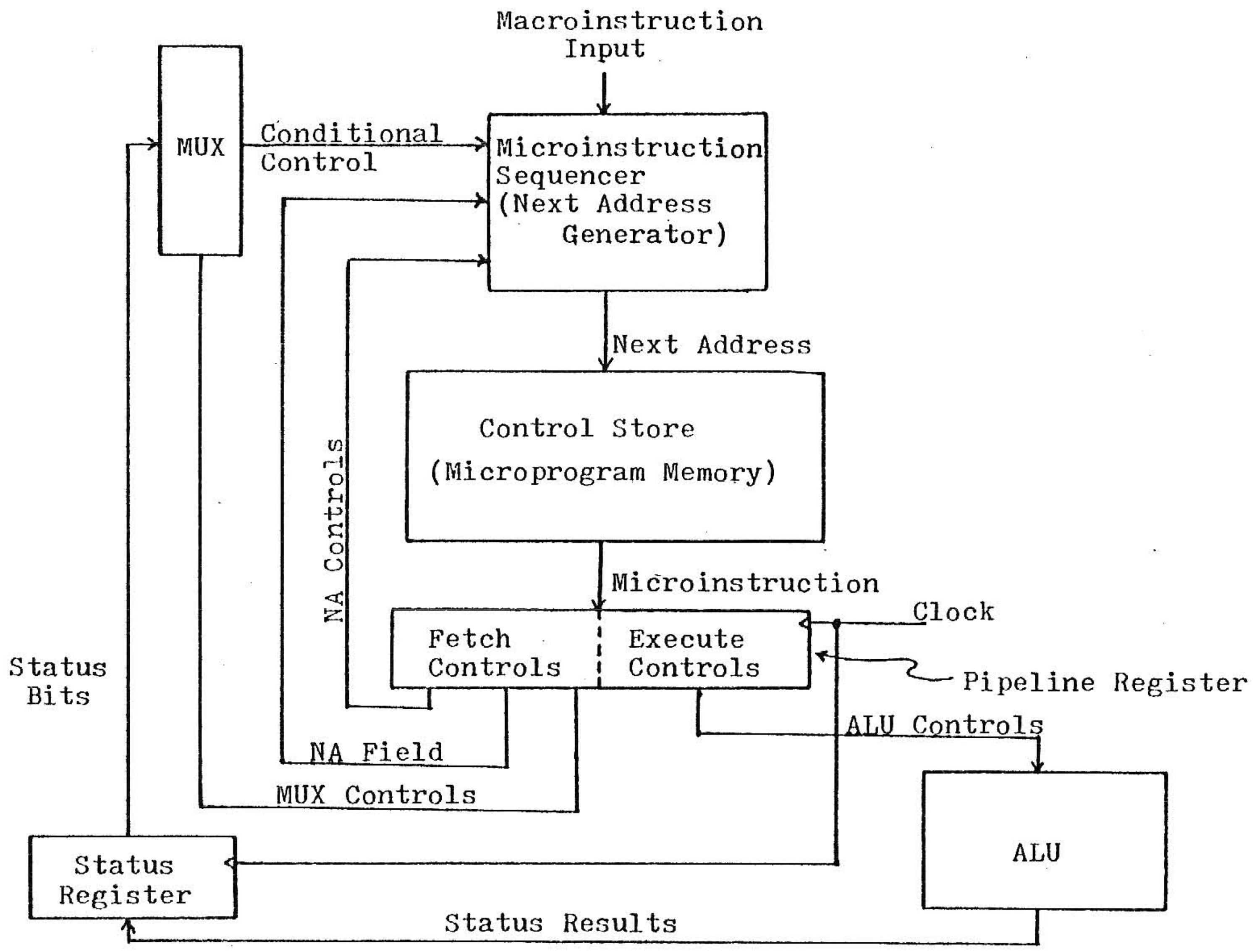


Figure 11-1

the root of all decision making in this machine.

All pipelined structures experience a degradation in performance when an attempt is made to alter their normal flow of operation. Such degradation is experienced here as a result of conditional jumps at the microinstruction level. Consider the following sequence of operations.

$$\begin{tikzcd}
 & A \arrow[d] & \\
 & B \arrow[dl] \arrow[dr] & \\
D & & G
\end{tikzcd}$$

Here B follows directly after A. The operation to follow B, however, may be D or G depending on the result of B.

Figure 11-2 shows the sequence of microinstructions required to carry out the above sequence. During Cycle 1, microinstruction B is fetched while A is executed. In Cycle 2, B is executed. Since the status results of B are not available until the end of Cycle 2, B itself is unable to make the decision between D and G. Instead, a microinstruction C is introduced whose sole purpose is to fetch either D or G based on the status results of B. The introduction of C here results in a degradation of performance since it performs no useful execution, it simply fetches the next microinstruction.

11.1 The 2910

The microprogram sequencer in this machine consists primarily of the 2910, a member of the 2900 family of microcomputer devices. A

Pipelining Illustration for Microprogram Conditional Jump

	Cycle 1	Cycle 2	Cycle 3	Cycle 4	.....	Cycle 4
Microinstruction Execute	A	B	C	D	.....	G
Status Register Contents	Results from Prevoius Micro-inst.	Results from A	Results from B	Results from C	.....	Results from C
Microinstruction Fetch	B	C	Condi-tional • D • G	?	.....	?

Figure 11-2

structural block diagram of the 2910 is shown in Figure 11-3. The Next Address is supplied to the Control Store via the Y outputs of this device. These outputs are 3-state to allow the address to be supplied from a different source if so desired. Within the 2910 are four sources for the Next Address; the Microprogram Counter (μPC), the Stack (F), the register/counter (R), and the externally supplied input (D). The 2910 performs 16 instructions (operations) determined by the four instruction inputs, $I_3 - I_0$. Note that all registers in this device (PC, R, F, and the Stack Pointer) are loaded on the rising edge of the clock input, CP.

The μPC is used to sequence <u>micro</u>instructions in a manner similar to which the Program Counter sequences <u>macro</u>instructions. The μPC register is preceded by an incrementer which is controlled by the CI input. If CI is high, the μPC will be loaded with Y + 1, i.e. the address of the next sequential microinstruction. If CI is low, the μPC is loaded with Y, the address of the same microinstruction currently being fetched. Through proper control of the CI input, a microinstruction can be executed repeatedly until a chosen status condition occurs.

The function of the register/counter in the 2910 is two-fold. It can be used to hold a Next Address value supplied by an external source or it can be used as a counter, to determine the number of times a series of microinstructions are performed. The second function is quite useful when implementing fixed point multiplication and division routines in microcode. To execute a routine N times, the register/counter must be loaded with N - 1. Independent of its function, this register is loaded from the D inputs. The $\overline{RLD}$ input controls this operation. When low, the register/counter is loaded on the rising edge of the clock.

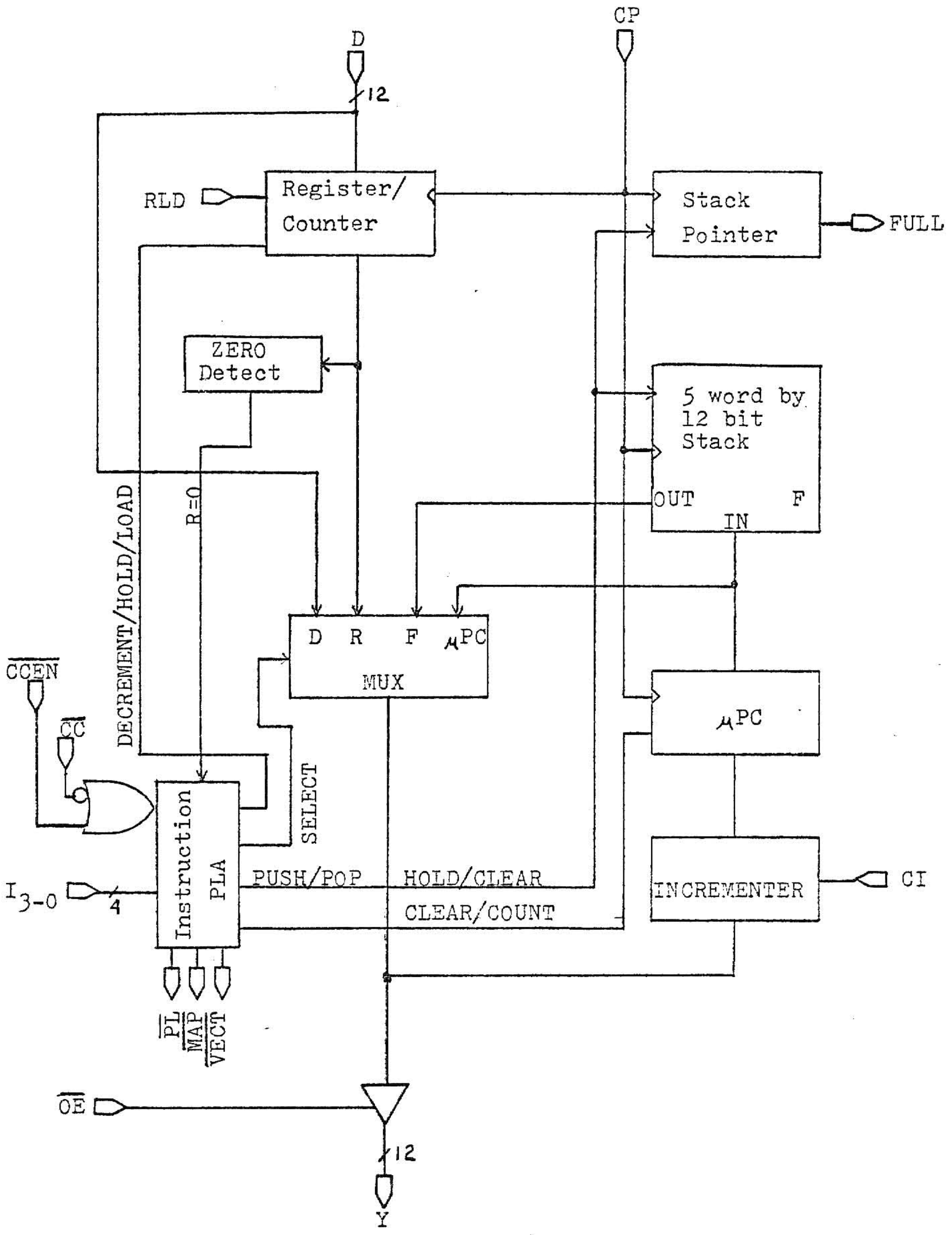


Figure 11-3 The 2910 Block Diagram

This register is also loaded by two of the 2910 operations independent of the $\overline{\text{RLD}}$ control.

The 2910 contains a 5 word stack. Values are pushed onto the Stack from the μPC. The stack is used primarily for microinstruction subroutining, which can be up to 5 levels deep. The Stack Pointer always points to the last word written in the stack. A FULL signal is generated when the stack is full. Pushing a value onto a full stack simply overwrites the top entry. Poping from an empty stack provides a meaningless value, but otherwise has no effect. In addition to its use with subroutines, the stack also plays a special role in 3 other 2910 operations. Details of these operations will be presented in tabular form.

The D inputs to the 2910 serve as the fourth possible source of the Next Address. It is through these inputs that the Next Address Field from the Pipeline Register, and the Macroinstruction Input are made available to the Control Store.

A summary of the 16 operations performed by the 2910 is presented in Figure 11-4 [3]. Eleven of these operations perform a conditional jump. Ten perform 2-way jumps based on the result of a single testing condition. The other uses two test conditions to select from three sources for the Next Address. The remaining 5 operations always supply the Next Address from the same source, independent of the test conditions.

The primary test condition is the Conditional Control input, $\overline{\text{CC}}$. This input affects instructions 1, 3, 5, 6, 7, A, B, D, and F. When this input is low, the test is said to pass. In this case, the Y output and stack operation is read from the PASS column in the table. When $\overline{\text{CC}}$ is high, the test is said to fail, and the operations are read from

The 2910 Instruction Set

HEX I_3-I_0	Mnemonic	Name	Reg/ Cntr	FAIL $\overline{CCEN}=0$ & $\overline{CC}=1$		PASS $\overline{CCEN}=1$ or $\overline{CC}=0$		Reg/ Cntr	Enable
				Y	Stack	Y	Stack		
0	JZ	Jump Zero	X	0	Clear	0	Clear	Hold	PL
1	CJS	Cond JSB PL	X	PC	Hold	D	Push	Hold	PL
2	JMAP	Jump MAP	X	D	Hold	D	Hold	Hold	MAP
3	CJP	Cond Jump PL	X	PC	Hold	D	Hold	Hold	PL
4	PUSH	PUSH/Cond LD Cntr	X	PC	PUSH	PC	PUSH	Note1	PL
5	JSRP	Cond JSB R/PL	X	R	PUSH	D	PUSH	Hold	PL
6	CJV	Cond Jump Vector	X	PC	Hold	D	Hold	Hold	VECT
7	JRP	Cond Jump R/PL	X	R	Hold	D	Hold	Hold	PL
8	RFCT	RPT Loop, Cntr 0	≠0	F	Hold	F	Hold	Dec	PL
			=0	PC	POP	PC	POP	Hold	PL
9	RPCT	RPT PL, Cntr 0	≠0	D	Hold	D	Hold	Dec	PL
			=0	PC	Hold	PC	Hold	Hold	PL
A	CRTN	Cond RTN	X	PC	Hold	F	POP	Hold	PL
B	CJPP	Cond Jump PL & POP	X	PC	Hold	D	POP	Hold	PL
C	LDCT	LD Cntr & Continue	X	PC	Hold	PC	Hold	Load	PL
D	LOOP	Test End Loop	X	F	Hold	PC	POP	Hold	PL
E	CONT	Continue	X	PC	Hold	PC	Hold	Hold	PL
F	TWB	Three Way Branch	≠0	F	Hold	PC	POP	Dec	PL
			=0	D	POP	PC	POP	Hold	PL

Note1: If $\overline{CCEN}=0$ and $\overline{CC}=1$, Hold, else Load

* taken from The Complete Motorola Microcomputer Data Library , 1978

Figure 11-4

the FAIL column in the table. The $\overline{\text{CC}}$ input can be disabled via the $\overline{\text{CCEN}}$ control. When $\overline{\text{CCEN}}$ is high, $\overline{\text{CC}}$ is disabled and all tests are forced to pass. When $\overline{\text{CCEN}}$ is low, the $\overline{\text{CC}}$ input determines the condition of the test in the manner described above.

The register/counter provides a second possible test condition used by three of the 2910 operations. The contents of this register are tested for zero by instructions 8, 9, and F. Both 8 and 9 are used to terminate a microprogram loop. Instruction F uses both test conditions, $\overline{\text{CC}}$ and the counter value, to perform a 3-way jump.

The $\overline{\text{PL}}$, $\overline{\text{MAP}}$, and $\overline{\text{VECT}}$ outputs of the 2910 are designed to provide 3-state enables for possible D input sources. Only one of the three is activated at a time. $\overline{\text{PL}}$ is designed to enable the Next Address field from the Pipeline Register. $\overline{\text{MAP}}$ is meant to be used with the Macroinstruction Input, and the $\overline{\text{VECT}}$ signal with an interrupt of some kind. The right hand column in Figure 11-4 shows which signals are activated for the various 2910 operations.

For a more detailed discussion on the 2910 and its operation see [3].

11.2 The Control Unit Design

The hardware design of the complete Control Unit is shown in Figure 11-5. Both the Instruction Register and the 2910 are clocked by $\emptyset_1$. The Pipeline Register is clocked by $\emptyset_{1(PLR)}$ which in normal operating mode is identical to $\emptyset_1$. In fact, $\emptyset_{1(PLR)}$ is taken from the same clock signal as $\emptyset_1$. However, they are supplied to the computer via separate gates. When the CPU is halted, and the Console Device

The Control Unit Design

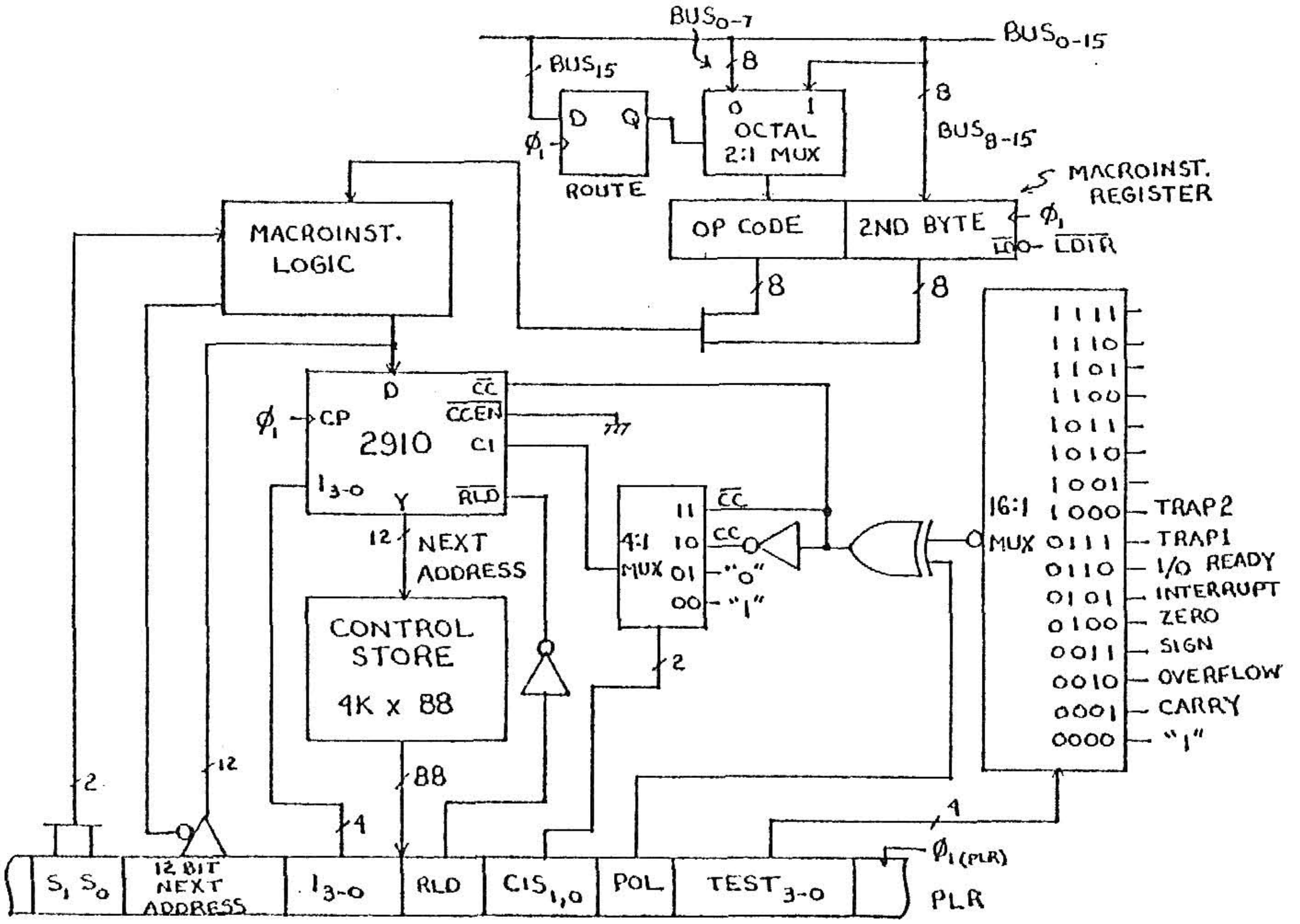


Figure 11-5

takes control, $\emptyset_{1(PLR)}$ from the Clock Circuit is halted so that the Console can supply this signal. In this way the Console can load the Pipeline Register without affecting the rest of the computer. This is a particularly useful feature, especially during initialization.

The $\overline{RLD}$ and I_3 - I_0 controls on the 2910 are supplied directly from the Pipeline Register. The pass or fail condition of the $\overline{CC}$ input can be determined by testing either the TRUE or the FALSE state of a selected machine status bit. A 16-to-1 multiplexer, controlled by a 4 bit TEST field in the Pipeline Register, is used to select this status bit. A status of 1 is interpreted as a TRUE condition and 0 as FALSE. A test can be made on either polarity of a selected status signal via the POL bit in the Pipeline Register. To test for a TRUE condition, the POL bit is set to 0. Setting the POL bit to 1 will test for a FALSE condition. Note, one of the status inputs is hard-wired to a 1. Testing this input for a TRUE condition will always PASS. Likewise, testing this input for a FALSE condition will always FAIL. This allows the microprogrammer to force a PASS or FAIL condition for each of the sixteen 2910 operations. As a result, there is never a need to disable the Conditional Control input, $\overline{CC}$, so the $\overline{CCEN}$ input is permanently tied low.

The Incrementer control input, CI, can be set to 0, 1, CC, or $\overline{CC}$ by a two bit select field in the Pipeline Register. This control provides the microprogrammer with a very powerful tool. A better understanding of the exact use of this input is given later in some example routines.

The following signals in the 2910 are unused; FULL, $\overline{PL}$, $\overline{MAP}$, and $\overline{VECT}$. It is up to the microprogrammer to make sure the stack bounds are not exceeded. This is not thought to be a big restriction. Routines that require up to 5 levels of subroutining are probably far more complex than most of those found in the microprogramming environment.

The $\overline{PL}$, $\overline{MAP}$, and $\overline{VECT}$ signals are replaced by a 2-bit field, S_1 and S_0, in the Pipeline Register. This field selects from 4 possible sources for the 2910 D inputs. One of these sources is the Next Address Field in the Pipeline Register. The other 3 come from the Instruction Register.

As previously mentioned, the Control Store is made of 6116's, the same type of RAM found in Main Memory, but a slightly faster version (120ns verses 150ns). Each of the 4K locations is 88 bits wide. As of yet, 4 of these bits are undefined and can be used for future expansion. A total of 22 RAMs are required to fully populate the Control Store. When the computer is operating in normal mode this memory remains in the read state. The Console however, can both read and write directly with this memory once the CPU is halted. A detailed diagram of the Control Store will be presented along with the Console later.

11.3 The Macroinstruction Logic Design

The Macroinstruction Logic circuit is shown in Figure 11-6. As illustrated, the Instruction Register is loaded by $\emptyset_1$ when the $\overline{LDIR}$ signal (from the PLR) is activated. The first byte of the IR always holds an Op Code. It is therefore referred to as the Op Code register. Note, while the second byte of the IR is loaded from only the high

Macroinstruction Logic Design

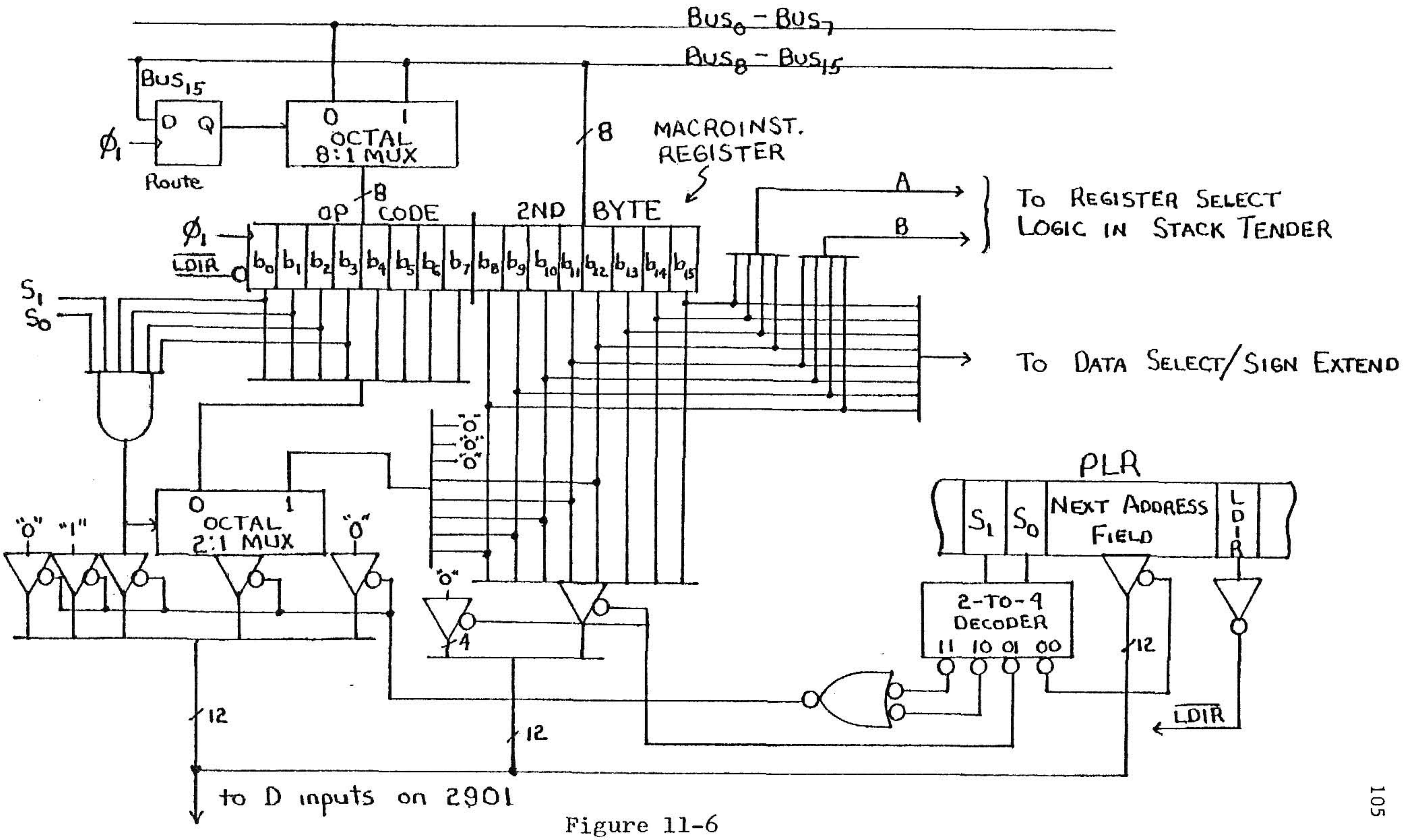


Figure 11-6

byte of the Bus, the Op Code register can be loaded from either the high or low byte. This is a direct result of allowing 1-byte instructions (Op Codes) to reside in both even and odd byte locations (refer to Section 6). It's quite possible for the Macroinstruction Fetch Routine to detect a fetch from an odd byte location and provide the controls needed to shift the high byte of the Bus into the Op Code register. This however is costly in terms of both time and CS space. It is more advantageous to have the fetch routine simply read a word from memory and let the Macroinstruction Logic direct the proper byte to the Op Code register. This operation is implemented via use of a Route flip-flop. This flip-flop controls a multiplexer at the input to the Op Code Register. It is loaded from the BUS_{15} line at the end of every microinstruction cycle, but serves a purpose only when a macroinstruction is being read from memory. Recall, to read from memory requires two microinstruction cycles, the first to send the address, and the second to receive the data. During a macroinstruction fetch, the Route flip-flop is loaded with the least significant bit of the address at the end of the first cycle. Thus, in the second cycle, when the actual macroinstruction is placed on the Bus, the output of Route is used to gate the appropriate byte to the Op Code register.

The Instruction Register supplies three sources for input to the 2910, each serves a slightly different purpose. These sources will be referred to as the Op Code Address, the Effective Address Routine's Address (EAR's Address), and the Count value.

The Op Code Address has the following format,

$$\boxed{0\quad 1\quad 0\quad \overbrace{X\quad X\quad X\quad X\quad X\quad X\quad X\quad X}^{\text{Op Code}}\quad 0}$$

where the X's come directly from the Op Code Register. Thus, each of the 256 possible Op Codes will address a distinct location in the Control Store. At this location a microinstruction routine, designed to perform the operation specified by the Op Code, begins. Note, the beginning of routines for successive Op Codes are separated by one location. That is, space for two microinstructions has been allotted to carry out the execution phase of each macroinstruction. Obviously some will require more space, but many macroinstructions can be performed in as few as one or two microinstructions. Routines that require more space must jump to another part of the Control Store to complete their task.

Memory Reference Instructions are handled in a special way. Recall, the first 5 bits of the second byte in a Memory Reference Instruction are used to specify up to 32 different schemes for calculating effective addresses. An EAR's Address source to the 2910 incorporates these 5 bits into the following format,

$$\boxed{0\quad 1\quad 1\quad 0\quad 0\quad 0\quad X\quad X\quad X\quad X\quad X\quad 0}$$

where the X's here come from IR_8 - IR_{12}. Up to 32 different Control Store locations can be selected by these 5 bits. Microinstruction routines that calculate Effective Addresses are expected to begin at these locations. Again, the first microinstruction of each routine is separated from the next sequential one by 1 microinstruction.

In the process of carrying out Memory Reference Instructions the first thing that is always done, independent of the operation specified by the Op Code, is the calculation of the Effective Address. For this reason, the hardware has been designed to detect the Op Code of a Memory Reference Instruction and gate the EAR's Address (rather than the Op Code Address) to the 2910 to begin execution of the instruction. This requires a priori knowledge of the Op Code format for a Memory Reference Instruction. The Op Code for all Memory Reference Instructions in this machine must have 1's in the first 4 bits. Thus, up to 16 Memory Reference Instructions can be defined by the last 4 bits.

In light of the above discussion, the proper operation for the Macroinstruction Fetch Routine is the following. Once the Instruction Register has been loaded, the Next Address should be retrieved from the Macroinstruction Logic by setting the S_1 S_0 field in the Pipeline Register to 11. This will send the Op Code Address to the Control Store, unless the first 4 bits of the Op Code Register are 1's, in which case the EAR's Address is sent. Once the Effective Address has been computed, the operation performed by the routine at the Op Code Address can be carried out. The address of this routine is sent to the Control Store by setting the S_1 S_0 field to 10. This will send the Op Code Address to the 2910 independent of the Op Code value. In general, the above operation saves both time and Control Store space.

The third source for the 2910 from the Macroinstruction Logic is called the Count Value. The format for this source is,

0	0	0	0	X	X	X	X	X	X	X	X

where the X's come directly from the second byte of the Instruction Register. The primary purpose of supplying this source to the 2910 is to load the register/counter. In this way, the second byte of the Instruction Register can be used to specify the number of times an operation is performed. For example, rather than fetch and execute n identical shift instructions, a single shift instruction can be made to shift n times, where n is a value specified by the second byte of the instruction. If desired, the Count Value can also be used as a Next Address source, although it is not intended for that purpose. Placing a 01 in the S_1 S_0 field will make the Count Value available at the D inputs to the 2910.

The Next Address Field from the Pipeline Register is sent to the 2910 when S_1 S_0 is 00. The function of this field is determined solely by the microprogrammer. Most likely it will provide a Next Address value, but can also be used to supply a count value to the register/ counter.

A Memory Map for the Control Store is shown in Figure 11-7. The portion of the Control Store Accessed by the various Op Code Addresses occupies 1/8 of the total storage (locations 2000_8 to 2777_8). Herein lie the first two microinstructions for each of the 256 Op Codes. EAR's Addresses access locations 3000_8 through 3077_8 where the first two microinstructions of the 32 different addressing schemes reside.

Control Store "Memory Map"

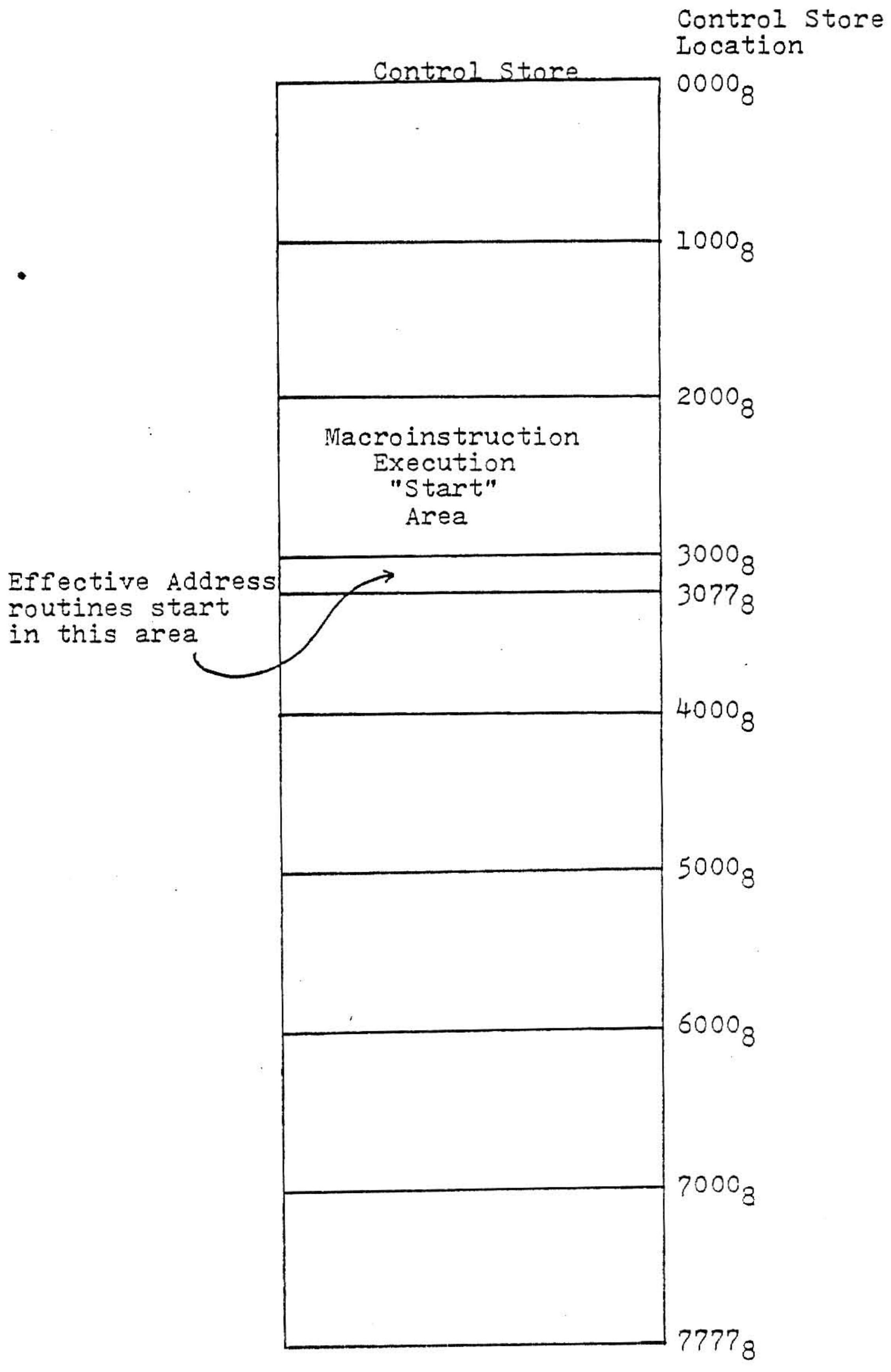


Figure 11-7

The remainder of the Control Store is left to be defined by the micro-programmer. Much of it will contain microcode that simply completes routines started in the two pre-defined areas. Parts of it will also contain Macroinstruction Fetch routines and initialization routines. The Power Up initialization routine will most likely begin at 0000_8. A description of this operation is provided along with the Console in subsequent material.

12 The Clock Circuit Design

The clock circuit generates the $\emptyset_1$, $\emptyset_2$, and $\emptyset_3$ signals along with their complements. Of these signals, only $\emptyset_1$, $\overline{\emptyset}_1$, and $\overline{\emptyset}_3$ are used for timing throughout the system. A summary of their use is given in Figure 12-1. Note, $\overline{\emptyset}_3$ is used exclusively by the ALU, and $\overline{\emptyset}_1$ by Main Memory. $\emptyset_1$ provides timing for the rest of the computer.

Operation of the entire computer can be halted by "stopping the clock", i.e. holding $\emptyset_1$, $\overline{\emptyset}_1$, and $\overline{\emptyset}_3$ at a constant level. The computer can be halted only after the completion of a microinstruction cycle, that is, at the beginning of a new one. Thus, when halted, $\overline{\emptyset}_1$ is held low while $\emptyset_1$ and $\overline{\emptyset}_3$ are held high. Once halted, a Single Step can be performed at the microinstruction level by allowing the clocks to proceed for one cycle, at the end of which they are again halted.

The Execution Clock, $\overline{\emptyset}_3$, must be handled in a special way when a trap occurs. Recall, the Stack Tender generates two traps, one when a push is attempted with a full Fast Stack, and the other when the number of entries in the Fast Stack is insufficient. These signals are generated as a result of controls placed in the NUMBER NEEDED and COUNTER CONTROL fields in the Pipeline Register. Not only are they an indication that corrective action must be taken with the Fast Stack, but also that the ALU operation specified by the current microinstruction must be aborted. It's quite possible that this operation, if carried out, would overwrite a 2901 register incorrectly and/or affect the NIS and TOS counters in an undesirable manner. The easiest way to prevent this from happening is to inhibit the Execution Clock for one cycle. This can be accomplished by holding $\overline{\emptyset}_3$ high in the manner shown in Figure 12-2. Since $\overline{\emptyset}_3$ is used

Use of $\emptyset_1$, $\overline{\emptyset}_1$, and $\overline{\emptyset}_3$ throughout the System

$\emptyset_1$

Serves as the clock input to:

* Pipeline Register ($\emptyset_{1(PLR)}$)
* 2910
* Macroinstruction Register
* I/O Ready and Interrupt Status flip-flops in the CPU Bus Interface
* Route flip-flop
* Data Select flip-flops in I/O interface
* Synchronous I/O Ready flip-flops in I/O interface
* OBR in Output Interface
* Synchronous Interrupt flip-flop and Mask flip-flop in Interrupt Level interface

Enables the following onto the Bus lines:

* Y outputs of the 2901s
* Status bits
* IBR from an Input Interface
* Device Code from an Interrupt Level interface
* Ready flags from I/O interface

$\overline{\emptyset}_1$

Provides timing for the following functions:

* Load the MAR
* the write operation with memory
* memory data on/off the Bus

$\overline{\emptyset}_3$

Serves as a clock input to:

* 2901s
* CARRY,OVERFLOW,SIGN, and ZERO status bits in ALU
* NIS and TOS counters

Figure 12-1

Inhibiting $\overline{\phi}_3$ on a TRAP

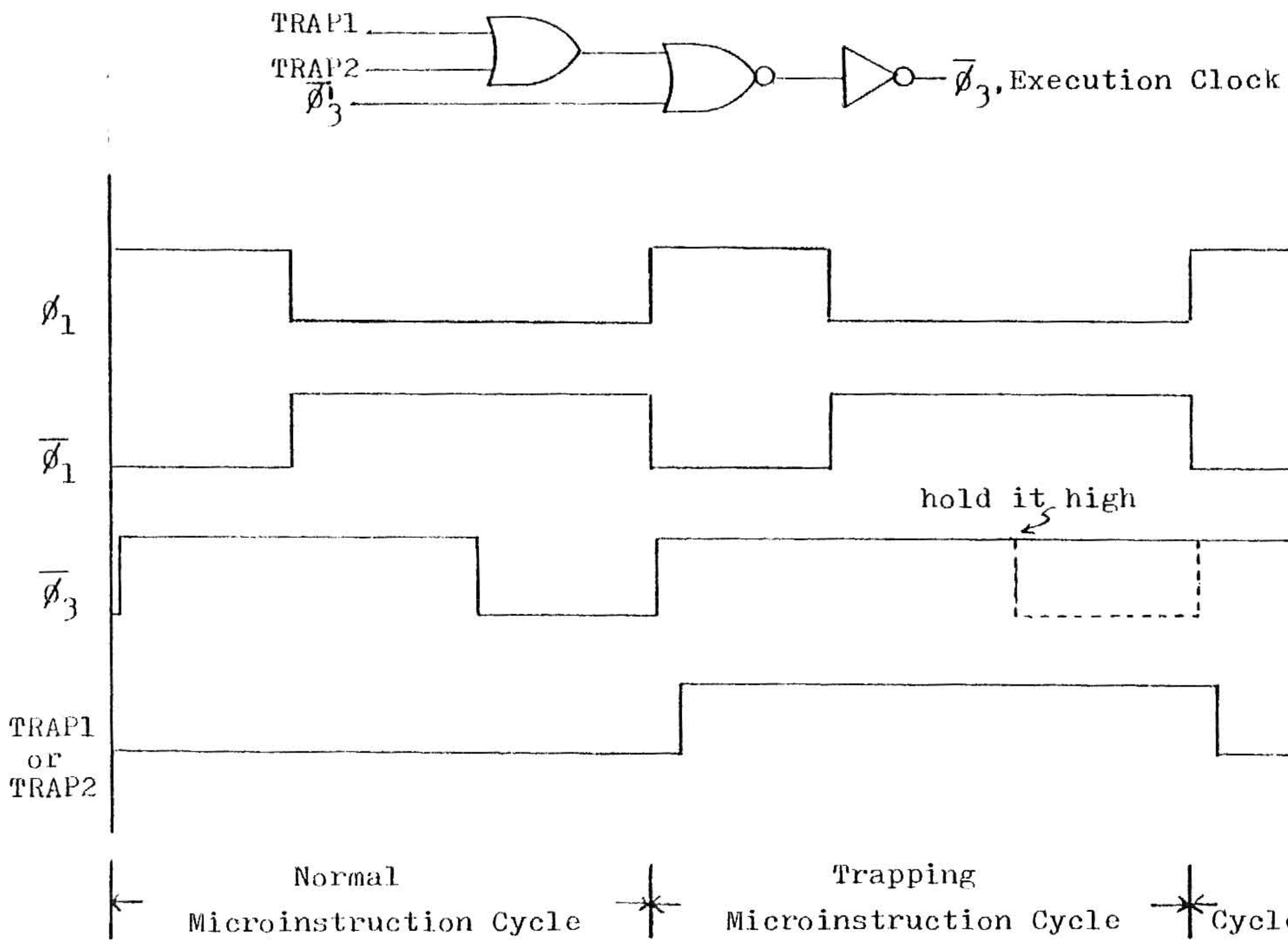

TRAP1
TRAP2
$\overline{\phi}'_3$
$\overline{\phi}_3$, Execution Clock
ϕ_1
$\overline{\phi}_1$
$\overline{\phi}_3$
TRAP1 or TRAP2
hold it high
Normal Microinstruction Cycle
Trapping Microinstruction Cycle
1st Microinstruction Cycle of "fix it" routine

Figure 12-2

exclusively by the ALU this action will not affect the rest of the computer, only the ALU. One is inclined to believe that a trapping condition might require the prevention of additional operations outside the ALU from being carried out undesirably. This, however, is not the case. Verification for this statement is provided in subsequent sections that develop microinstruction routines to deal with traps.

Routines designed to correct a trapping condition will be referred to as "fix it" routines. The first microinstruction in these routines should follow directly after the microinstruction in which the trap occurs. This requires that the trapping microinstruction perform a conditional jump based on the presence or absence of a trap condition. This is true of all microinstructions that perform operations with Fast Stack locations. The TRAP signal which inhibits ALU operation will not be present in the microinstruction of the "fix it" routine because the NUMBER NEEDED and COUNTER CONTROL fields will be different. Thus $\overline{\emptyset}_3$ is inhibited only for the microinstruction cycle in which the trap occurs.

The hardware design of the Clock Circuit is shown in Figure 12-3. In normal mode the $\emptyset_1$, $\emptyset_2$, and $\emptyset_3$ flip-flops function as a 3-bit shift register. A 1 is shifted through in a circular fashion to create the $\emptyset_1$, $\emptyset_2$, and $\emptyset_3$ clock signals. Since these signals are made available to the computer through inverted buffers, $\emptyset_1$ and $\emptyset_{1(PLR)}$ are taken from $\overline{Q}$ while $\overline{\emptyset}_1$ and $\overline{\emptyset}_3$ are supplied from the Q outputs. The $\overline{\text{Reset}}$ signal initializes $\emptyset_1$, $\emptyset_2$, and $\emptyset_3$ to 1, 0, 0 at power up. A shift occurs every 100ns as determined by the 10 MHz Master Clock. Shifting a 1 through all 3 flip-flops constitutes a complete microinstruction cycle (300ns).

All three of the flip-flops are clocked by the same signal so that

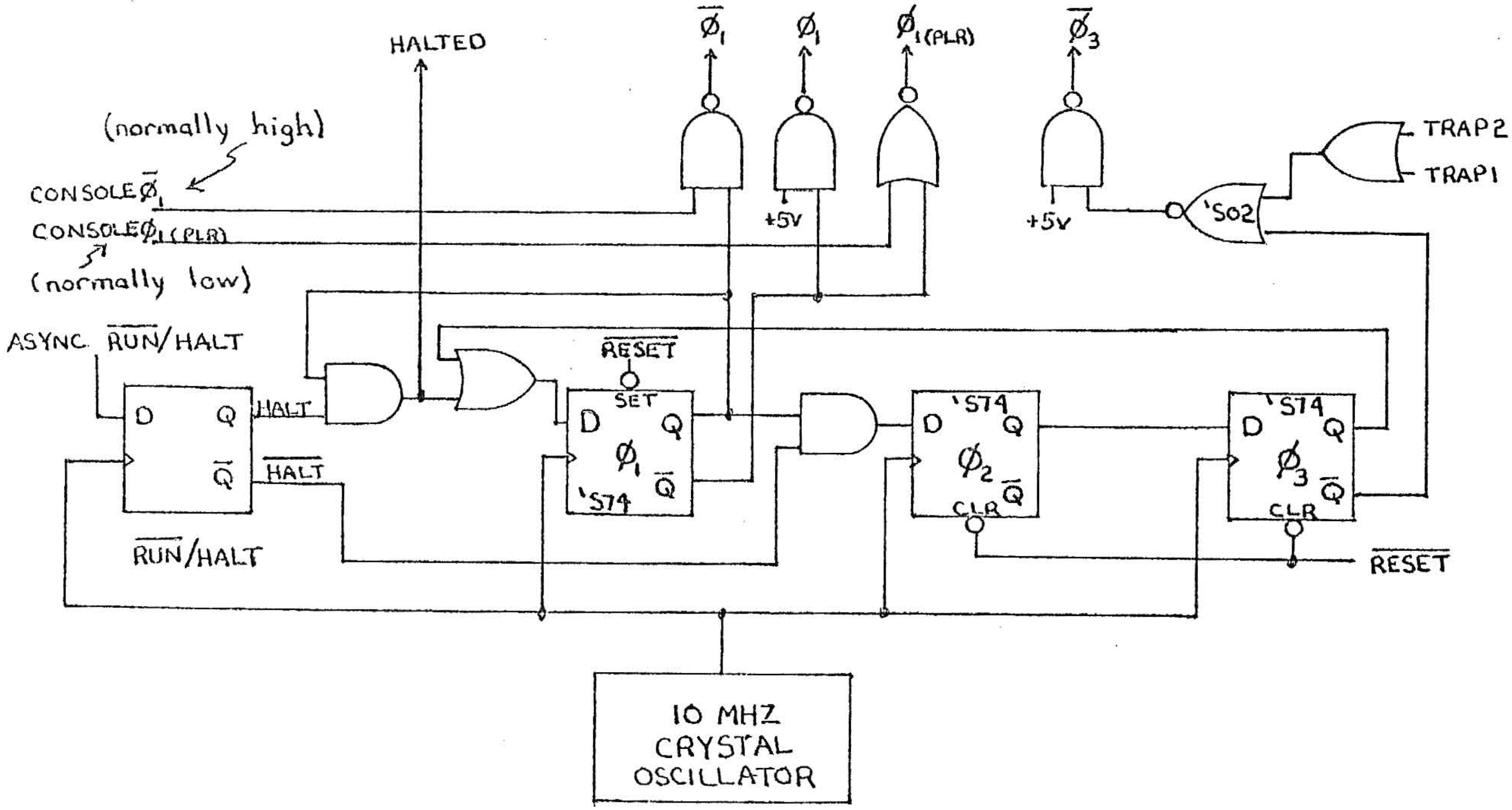


Figure 12-3 The Clock Circuit Design

their outputs will change in synchronism. It is well known, however, that the clock-to-output delay time can be slightly different for the Q and $\overline{Q}$ outputs, and may vary even more from one flip-flop to another. This effect can be minimized by using Schottky parts which have a maximum clock-to-output delay of 9ns for both Q and $\overline{Q}$ outputs [4].

The clock signals are made available to the system via a set of buffer/drivers. The purpose of these buffers is two-fold. They function primarily as drivers to provide sufficient current sinking capabilities. They also allow the Console Device to control $\overline{\emptyset}_1$ and $\emptyset_{1(PLR)}$ through proper application of CONSOLE$\overline{\emptyset}_1$ and CONSOLE$\emptyset_{1(PLR)}$. Note, during normal operation the Console must hold CONSOLE$\overline{\emptyset}_1$ high and CONSOLE$\emptyset_{1(PLR)}$ low. The Console uses $\emptyset_{1(PLR)}$ to load the Pipeline Register and $\overline{\emptyset}_1$ to carry out transactions with Main Memory. Again it is suggested that Schottky buffers be used for the drivers to minimize the relative time shifts between the different clock signals. Maximum propagation delays as low as 9ns can be achieved through proper part selection [4]. An additional delay is incurred by the $\overline{\emptyset}_3$ signal not seen by the others due to the NOR gate used to inhibit this signal on a trapping condition as described earlier. If Schottky is used, the maximum delay of this gate can be reduced to 7ns [4].

A timing diagram for $\emptyset_1$,$\overline{\emptyset}_1$, and $\overline{\emptyset}_3$ is given in Figure 12-4. In this figure, typical delay times were used for all circuit components. Note, $\overline{\emptyset}_3$ is slightly out of phase (5ns) compared to the other two due to the extra NOR gate. The shift of these signals with respect to the Master Clock is irrelevant. Their shift with respect to each other is however very important. If a minimum delay of 2ns is assumed for each circuit component, a worst case analysis will show $\overline{\emptyset}_3$ to be a maximum

Clock Timing Diagram

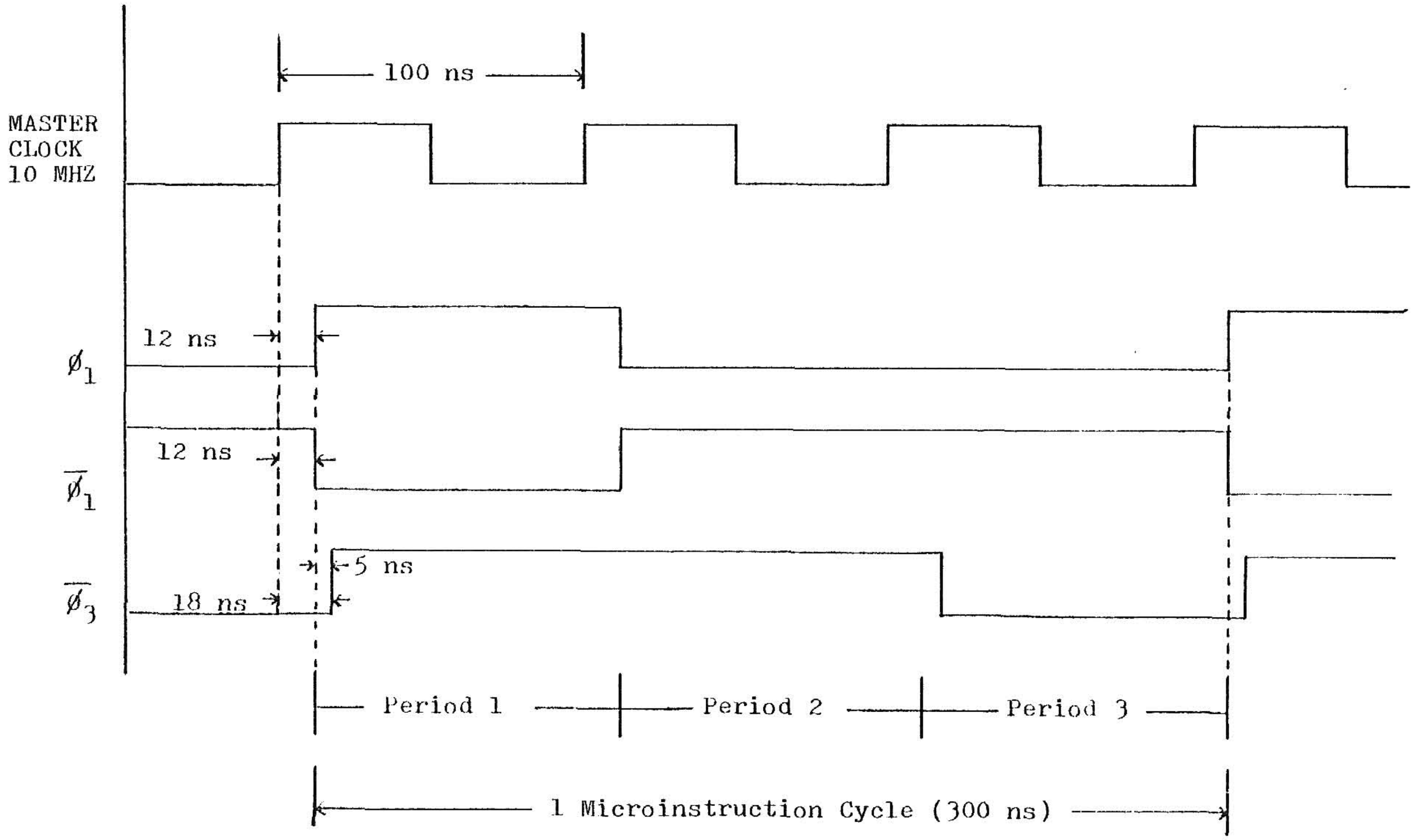


Figure 12-4

of 21ns out of phase with either of the other two signals. This condition is highly unlikely, however, since it implies that $\overline{\emptyset}_3$ incurs a maximum delay at each component while the other signal experiences only a minimum delay through similar components. Conditions like this can be avoided by using components from the same package as much as possible.

The Clock Circuit will be halted when the Q output of the $\overline{\text{Run}}$/Halt flip-flop is set to 1. Since this flip-flop is clocked by the Master Clock in synchronism with the rest of the circuit the Console must maintain a 1 at the Asynchronous $\overline{\text{Run}}$/Halt input to halt the Clock. Once the $\overline{\text{Run}}$/Halt flip-flop is set, the clock will be halted by stopping the circular shift operation when the 1 reaches the $\emptyset_1$ flip-flop. The 1 is essentially trapped in this flip-flop by the HALTED signal, generated when both the HALT and $\emptyset_1$ signals are high. In addition, the $\overline{\text{HALT}}$ signal will inhibit the 1 from propagating onward through $\emptyset_2$ and $\emptyset_3$. In this way the Clock is guaranteed to stop at the beginning of a microinstruction cycle.

The run mode can be re-entered by returning the Asynchronous $\overline{\text{Run}}$/Halt input to the low state. This will clear the $\overline{\text{Run}}$/Halt flip-flop and allow the circular shift operation to proceed in a normal fashion. A Single Step is performed by simply re-entering the run mode for a single cycle. This can be accomplished by pulling the Asynchronous $\overline{\text{Run}}$/Halt input low until the first shift occurs. In other words, it is pulled low until $\emptyset_1$ goes low, then it's returned high. In this way, the clock is again halted as soon as the 1 has circulated back to the $\emptyset_1$ flip-flop.

In summary, the Clock Circuit consists of a 3-bit shift register clocked by a 10 MHz Master Clock. The microinstruction cycle time can be changed by simply modifying the Master Clock frequency. The Execution

Clock, $\overline{\emptyset}_3$, is inhibited every time a trap occurs. $\emptyset_{1(PLR)}$ and $\overline{\emptyset}_1$ can be controlled by the Console through proper application of the CONSOLE$\overline{\emptyset}_1$ and CONSOLE$\emptyset_{1(PLR)}$ signals. In this way the Console can supply the timing needed to load the Pipeline Register and communicate with main memory. The Console can also control halt and single step operations via the Asynchronous $\overline{\text{Run}}$/Halt input .

13 The Control Store Design

A detailed diagram of the Control Store is presented in Figure 13-1. The two rows of 6116's provide 4K locations, 2K per row. Eleven RAMs are required in each row to achieve an 88 bit word length. The 2910 supplies a 12-bit address, the msb of which enables one of the two rows. The remaining 11 bits select an 88 bit wide location within the chosen row. The value in this location is loaded into the Pipeline Register on the rising edge of $\emptyset_{1(PLR)}$.

The 11 RAMs in each row are grouped into pairs to form five 16-bit banks and one 8-bit bank for purposes of communication with the Console. Each bank is linked to the System Bus via a group of Bus Transceivers. To access the Control Store, the Console must first halt the CPU. This will place the 2910 output in the high impedance state. Then by activating the $\overline{\text{CONSOLEON}}$ signal, the 15-bit CSAR value will be enabled. The upper 12 bits of this register select an 88 bit Control Store location in exactly the same manner as the Next Address output of the 2910. The lower 3 bits work with the Bank Select Decoder to enable a transceiver group for 1 of the 6 banks. This decoder is activated only when both $\overline{\text{CONSOLEON}}$ and $\overline{\text{CSRWEN}}$ are generated by the Console Device. Directional control on the transceivers is a function of the Write Decoder outputs. These outputs also supply the Write Enable, $\overline{\text{WE}}$, controls for each of the 6 banks. During a read operation, when the CSR/$\overline{\text{W}}$ signal is high, the Write Decoder outputs are disabled, and the $\overline{\text{OE}}$ controls on the 6116s are activated. Thus all banks of memory are placed in the read state and a full 88-bit word is read. Only the portion of this word that lies in the selected bank, however, will be transferred to the Console.

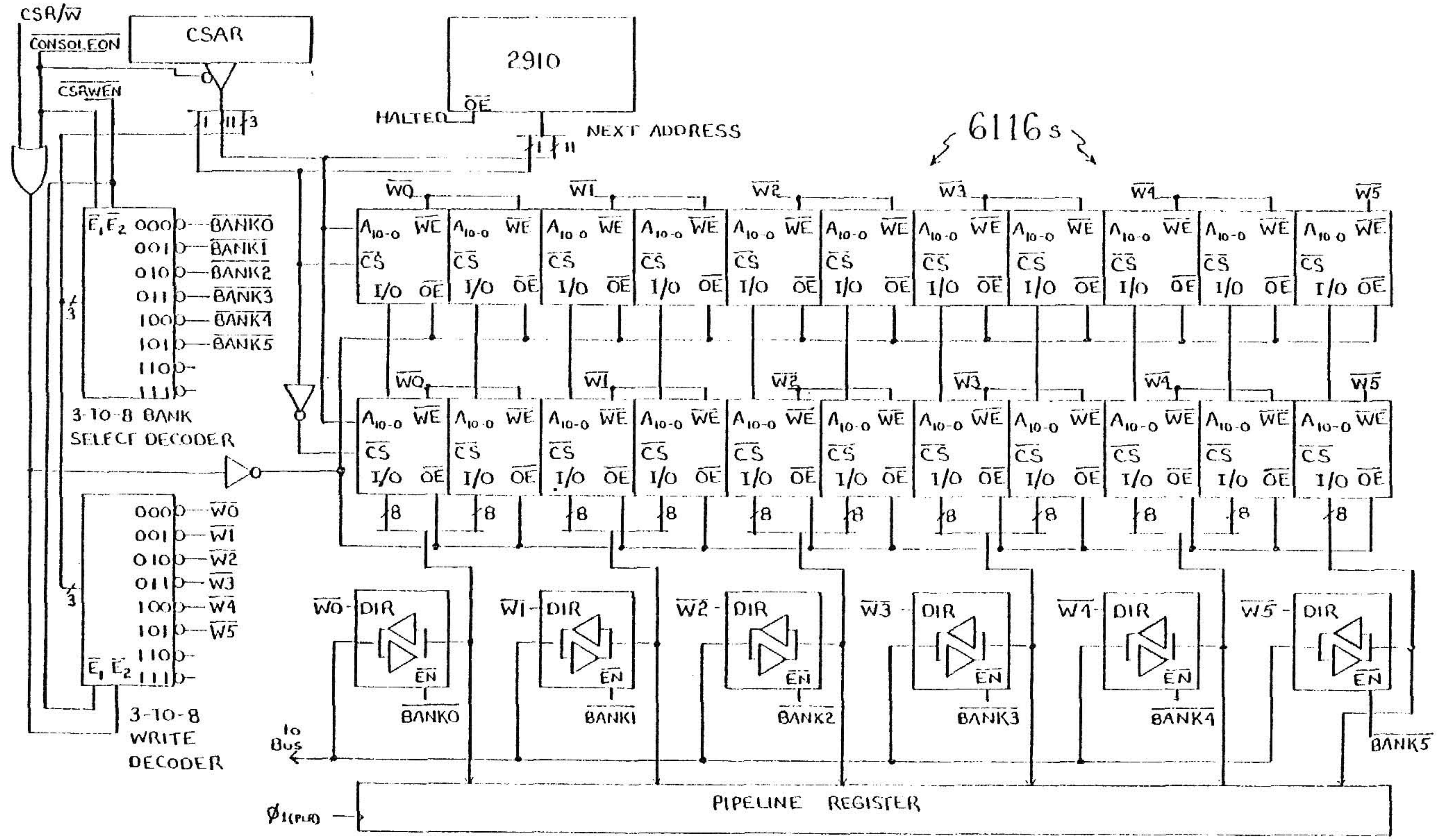


Figure 13-1 The Control Store Design

The write mode is entered when both the $\overline{\text{CONSOLEON}}$ and $\overline{\text{CSRWEN}}$ signals are activated and the CSR/$\overline{\text{W}}$ signal is pulled low. This will force the $\overline{\text{OE}}$ control on the 6116s to the high state. In this mode the directional control on the selected transceiver group will be reversed. In addition, the $\overline{\text{WE}}$ control for the selected bank of memory is enabled. In this way, data can be transferred from the Console to the Control Store. Note, the non-selected banks in this mode remain unchanged. It is important to note that during a write operation, the CSR/$\overline{\text{W}}$ signal should be returned high before the $\overline{\text{CSRWEN}}$ signal is disabled. This will guarantee that the $\overline{\text{WE}}$ control on the 6116's is returned high before their data inputs change.

It is a simple matter for the Console Device to load the Pipeline Register from a Control Store location. This requires only the placement of the desired address in the CSAR, the setting of CSR/$\overline{\text{W}}$ to 1, and application of a low-to-high transition on the $\emptyset_{(PLR)}$ line.

When returning to the normal mode of operation the Console Device must first de-activate the $\overline{\text{CONSOLEON}}$ signal so that the CSAR output is disabled. Then it's safe to re-enable the Next Address output of the 2910 by placing the Clock Circuit back in the run mode.

14 The Console

14.1 Console Operation

The Console Device can communicate directly with both Main Memory and the Control Store. Transactions with Main Memory are carried out via the Bus in exactly the same way that they are with the CPU, except at a much slower rate. Information transfers with the Control Store are also carried out over the Bus, 16 bits at a time. To read or write a full 88-bit microinstruction requires 6 data transfers. Addresses to 16 bit locations in the Control Store are supplied via a 15-bit Control Store Address Register, CSAR. The upper 12-bits of the CSAR address an 88-bit word in exactly the same way it is done with the 12-bits from the 2910. The lower 3 bits are used to select from 1 of 6 fields within this word, 5 of which are 16-bits wide while the last one is only 8. Before the Console can communicate with either the Control Store or Main Memory it must first halt the CPU and take over various control signals normally supplied by the Pipeline Register.

A halt signal generated by the Console will initiate the following sequence of events. First, the microinstruction currently being executed is allowed to finish. Then the machine is effectively halted by holding all the clocks at a fixed level. Once this has been accomplished the Clock Curcuit will generate an active high signal, HALTED, which in turn will tri-state the Next Address output of the 2910. The only useful operation that can be performed by the Console at this stage is a micro-instruction Single Step which is carried out by returning the Clock Circuit to the run mode for one complete cycle.

Before the Console can communicate with Main Memory or the Control Store it must activate the CONSOLEON signal. This signal disables the following controls from the Pipeline Register by placing them in the high impedance state.

$\overline{\text{MEMSEL}}$

LDMAR

$\text{R}/\overline{\text{W}}$

BYTE

$\overline{\text{I/O SEL}}$

$\text{I/O STAT}/\overline{\text{DATA}}$

$\overline{\text{INT ACK}}$

$\overline{\text{MSKEN}}$

$\overline{\text{BRST}}$

In addition, once CONSOLEON is activated, the Console can control $\overline{\emptyset}_1$ and $\emptyset_{1(\text{PLR})}$. The CONSOLEON signal also enables the CSAR and sets up the Control Store to be accessed in the 16-bit mode. In effect the Console Device is given complete authority over the Bus Controls. To perform transactions with Main Memory, it needs only to supply the appropriate control and timing ($\overline{\emptyset}_1$) signals. To communicate with the Control Store it must supply two additional control signals.

1. $\text{CSR}/\overline{\text{W}}$ - Control Store Read/Write

   This signal is set high to signify a read and low for a write.

2. $\overline{\text{CSRWEN}}$ - Control Store Read/Write Enable

   This is an active low signal which places the Control Store data on the Bus during a read and takes it off the Bus during a write.

Using these signals the Console can read from the Control Store by carrying out the following sequence of operations.

1. Load the CSAR with the address of the desired data.
2. Set the CSR/$\overline{\text{W}}$ signal high to signify a read operation.
3. Activate the $\overline{\text{CSRWEN}}$ signal to place the 16-bit data value on the Bus.
4. Read the value from the Bus into the Console Input Register.
5. Deactivate the $\overline{\text{CSRWEN}}$ signal to free the Bus.

Note, the Control Store is left in the read mode when not being used.

A write operation requires execution of the following steps.

1. Load the CSAR with the desired address.
2. Set the CSR/$\overline{\text{W}}$ signal low to signify a write operation.
3. Place data from the Console Output Register on the Bus.
4. Activate the $\overline{\text{CSRWEN}}$ signal to transfer this value from the Bus to the selected Control Store location.
5. Return the CSR/$\overline{\text{W}}$ signal high to terminate the write operation.
6. Deactivate the $\overline{\text{CSRWEN}}$ signal to free the Bus.

Both the read and write operations outlined above transfer only 16-bits of information. Each must be performed 6 times, with a different address each time, in order to transfer a full 88-bit microinstruction.

If need be, the Console Device can also place a value in the Pipeline Register. The microinstruction placed in this register will be the first one performed upon return to the normal operating mode. Any previous value in the Pipeline Register is overwritten, which means that the microinstruction placed in this register when the CPU is halted is destroyed. This, however, is a desirable feature during initialization where the previous value in the Pipeline Register is meaningless. Since the Console Device cannot supply a full 88-bit value itself, it must load the Pipeline Register from a Control Store location. All reads from the Control Store access a full 88-bit word even though the Console can only accept 16 bits of it at a time. Thus, the Pipeline Register can be loaded in the following manner.

1. Place the address of the desired Control Store location in the upper 12 bits of the CSAR. (The value of the lower 3 bits is irrelevant in this case.)
2. Set the $CSR/\overline{W}$ signal high to signify a read operation.
3. Load the Pipeline Reigster by supplying a low-to-high transition on the $\emptyset_{1(PLR)}$ line.

Upon return to the normal operating mode, the Console Device must first deactivate the CONSOLEON signal. This will re-enable the control signals from the Pipeline Register and the Clock Circuit. Once this is done, the normal operating mode can be re-entered by returning the Clock Circuit to the run mode.

14.2 Console Design

Most of the functions performed by the Console Device are actually quite complex. Since all of its operations are carried out while the CPU is halted it must provide both control and timing signals along with data. For this reason a microprocessor has been chosen to carry out these operations. In this way the complexity of the Console can be embedded in the microprocessor software. The Console is connected to the rest of the computer via a Console Interface circuit. The microprocessor communicates with the computer through this interface. A Block Diagram of the Console Device is shown in Figure 14-1.

The microprocessor is linked to the Console Interface via two 8-bit ports and two clock bits. Port A holds a control word which determines the current interface operation. Port B is used to transfer data between the microprocessor and the interface. All timing for the interface is performed by Clock B. Clock A provides the timing for operations carried out by this interface with the rest of the computer (i.e. $\emptyset_{1(PLR)}$ and $\overline{\emptyset}_1$).

The interface itself consists primarily of 4 registers, a 16-bit Console Input Register (CIR), a 16-bit Console Output Register (COR), a 4-bit Console Control Register (CCR), and the 15 bit Control Store Address Register (CSAR) discussed earlier. The Console sends and receives information over the system Bus through the CIR and COR. The CCR supplies control to the Clock and the CS. Bus control signals, required to carry out transactions with Main Memory, are supplied from Port B through a set of 3-state buffers. Clock A is logically combined with various interface controls to generate both the CONSOLE$\overline{\emptyset}_1$ and CONSOLE$\emptyset_{1(PLR)}$ clock signals. Since Clock A is used for both signals, only one can be

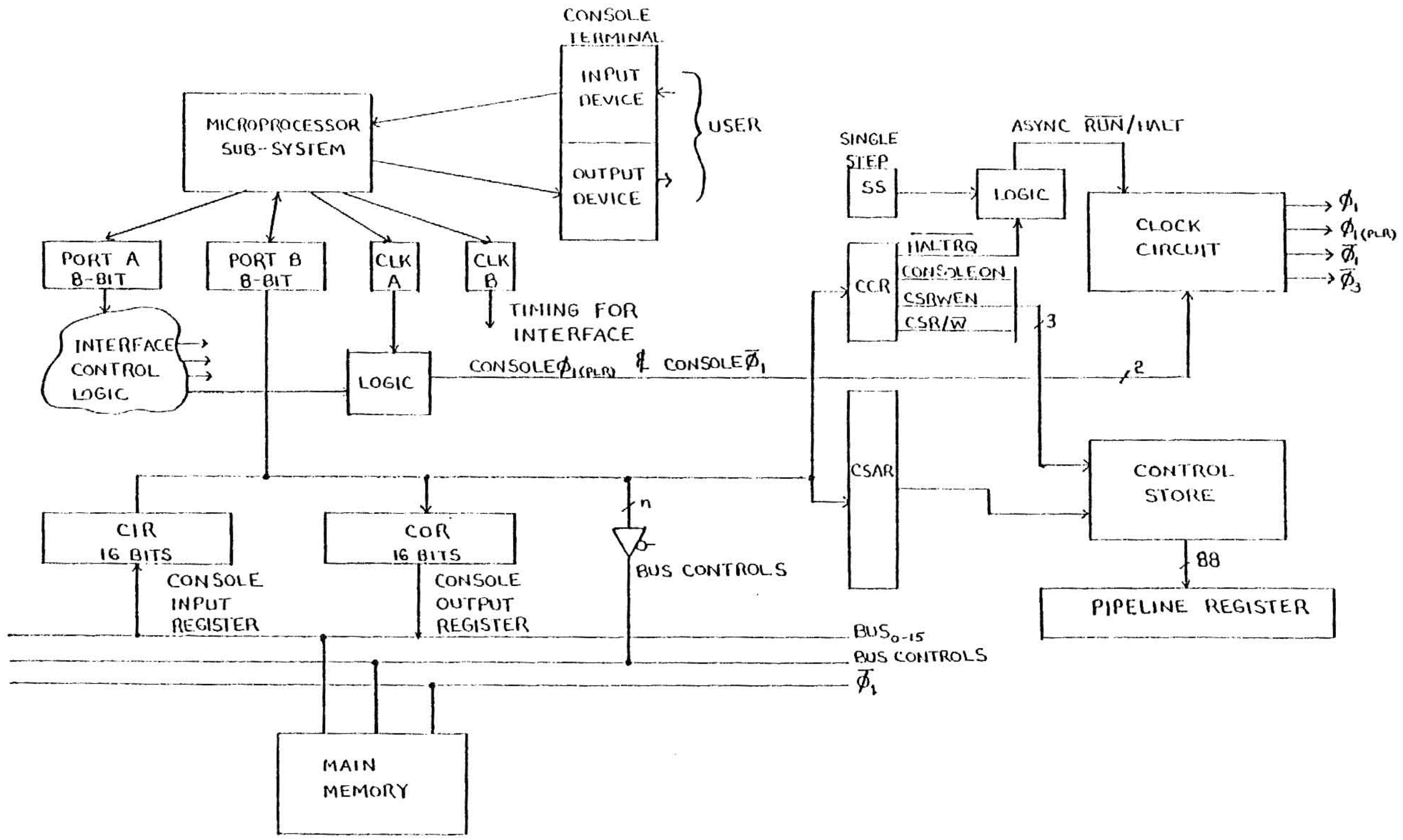


Figure 14-1 Console Block Diagram

activated at a time.

The Console Interface also contains a Single Step (SS) flip-flop used in coordination with the Clock Circuit to carry out a microinstruction single step. The output of this flip-flop is logically combined with the Halt Request bit from the CCR to supply the Asynchronous $\overline{\text{Run}}$/Halt input to the Clock Circuit. The SS flip-flop is activated through proper application of the control word in Port A.

A detailed design of the Console Interface is shown in Fiugre 14-2. A 6821 Peripheral Interface Adapter, PIA, is used to link the microprocessor to the interface circuitry. This device is designed to function with any of the 6800 family of microprocessors. It contains two bidirectional 8-bit ports, A and B. Each port has two handshake lines. The CA1 and CB1 lines are not used in this design. CA2 and CB2 are used, however, to supply the Clock A and Clock B signals. Both CA2 and CB2 can be toggled in software as an extra 1-bit register, independent of the port activity. When enabled, all registers and flip-flops in this interface are loaded on the rising edge of CB2. CA2 is used to generate the CONSOLE$\overline{\emptyset}_1$ and CONSOLE$\emptyset_{1(PLR)}$ signals.

All operations in this interface are enabled by a 6-bit control word in Port A. This word is divided into two 3-bit fields, each of which can specify up to 8 operations via a 3-to-8 decoder. The decoder outputs are used primarily as enables for the various interface operations. To actually perform most operations requires the application of a clock signal (CB2), or perhaps a transaction with Port B. A 000 value in either 3-bit field of the control word will activate the "safe state" output of the corresponding decoder. In this state, no operation is enabled. Two operations can be performed concurrently by specifying a non-zero value for both 3-bit

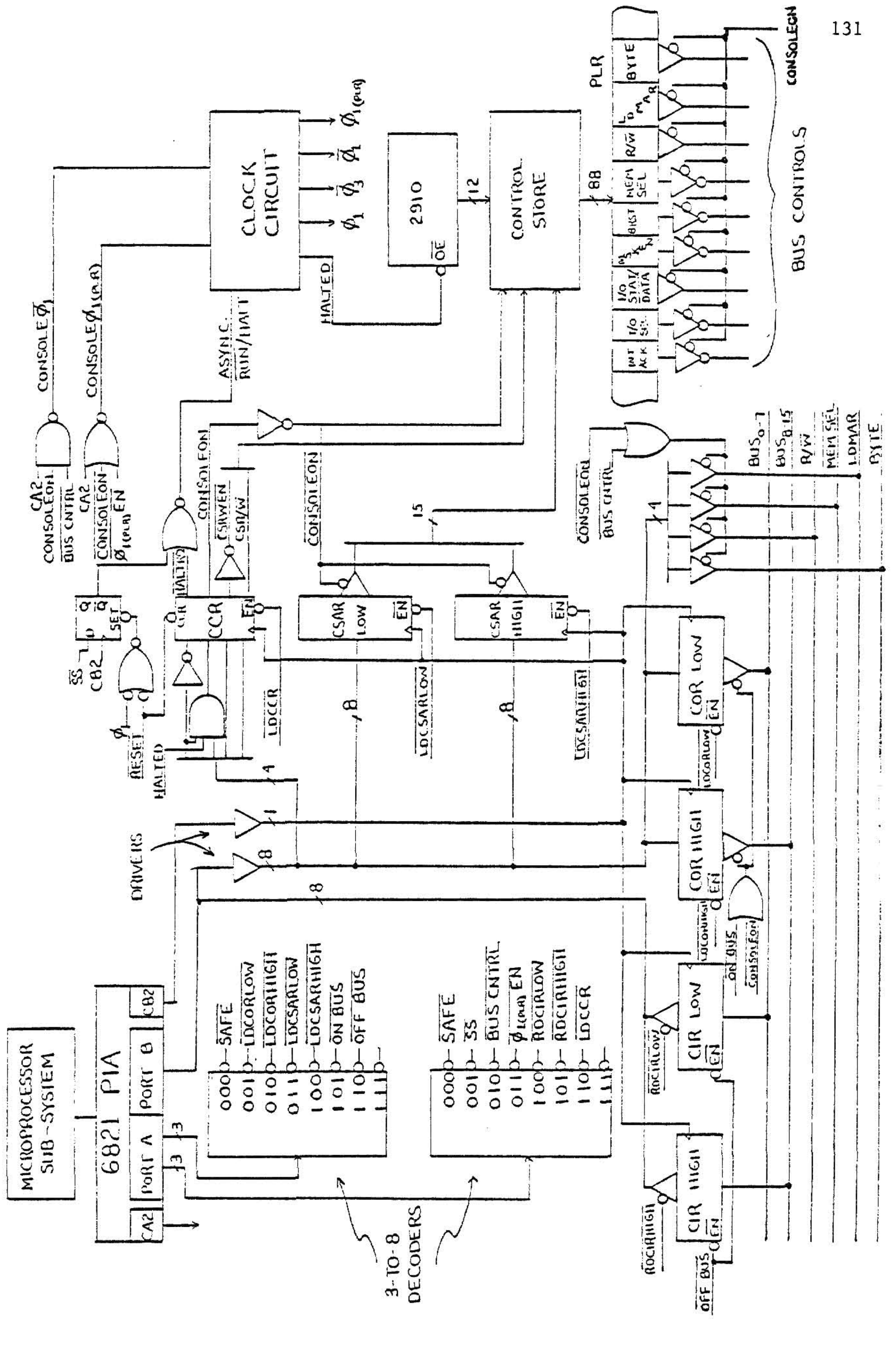


Figure 14-2 The Console Interface Design

fields. For some operations, however, this is undesirable. These will be apparent once the interface design is more fully understood. A summary of the operations that can be performed by this interface is given in Figure 14-3.

Interface registers are loaded from the microprocessor with the following steps.

1. Set up Port B as an output port.
2. Send a control word to Port A that will activate the load enable on the desired register.
3. Send the data to Port B.
4. Toggle CB2.

Note, COR and CSAR must be loaded a byte at a time. Thus, steps 2 through 4 are repeated for these registers.

To transfer a value from the CIR to the microprocessor requires a similar sequence of operations.

1. Set up Port B as an input port.
2. Send a control word to Port A that will enable the 3-state output of the desired register.
3. Read the data into Port B.
4. Transfer the data from Port B to the microprocesser.

Again, steps 2 through 4 are repeated to get both bytes. Note, both the CB2 and Port B outputs are supplied with a set of buffer/drivers, since alone they can drive only 2 TTL loads.

Port A Control Word Definitions

Port A Control Word (octal)	CB2	Operation
00_8	X	No operation
01_8	↑	$\overline{\text{LDCORLOW}}$ is activated; The low byte of the COR is loaded from Port B.
02_8	↑	$\overline{\text{LDCORHIGH}}$ is activated; The high byte of the COR is loaded from Port B.
03_8	↑	$\overline{\text{LDCSARLOW}}$ is activated; The low byte of the CSAR is loaded from Port B.
04_8	↑	$\overline{\text{LDCSARHIGH}}$ is activated; The high byte of the CSAR is loaded from Port B.
05_8	X	$\overline{\text{ONBUS}}$ is activated; The contents of the COR are enabled onto the Bus Information lines.*
06_8	↑	$\overline{\text{OFFBUS}}$ is activated; The CIR is loaded from the Bus Information lines.
07_8	X	No operation
10_8	↑	$\overline{\text{SS}}$ is activated; The SS flip-flop is cleared, hence activating a microinstruction single step if the clock is halted.
20_8	X	$\overline{\text{BUSCNTRL}}$ is activated; The $\overline{\text{MEMSEL}}$, LDMAR, R/$\overline{\text{W}}$, and BYTE controls along with $\overline{\varnothing}_1$ are applied by this interface via Port B and CA2.*

Figure 14-3

Port A Control Word (octal)	CB2	Operation
(continued)		
30_8	X	$\overline{\not{0}_{1(PLR)}EN}$ is activated; CA2 plays the part of $\not{0}_{1(PLR)}$, i.e. it supplies the clock to the Pipeline Register.*
40_8	X	$\overline{RDCIRLOW}$ is activated; The low byte of the CIR is enabled to be read at Port B.
50_8	X	$\overline{RDCIRHIGH}$ is activated; The high byte of the CIR is enabled to be read at Port B.
60_8	↑	$\overline{LDCCR}$ is activated; The CCR is loaded from the low 4 bits of Port B.
70_8	X	No operation

*This action is enabled only if the CONSOLEON signal has been activated.

Figure 14-3
(cont.)

The Single Step flip-flop works in the following manner. In normal operating mode it is set to 1 ($\overline{Q} = 0$). This allows the $\overline{\text{HALTRQ}}$ bit of the CCR to determine the Run/Halt mode of the computer. Once halted ($\overline{\text{HALTRQ}} = 0$), a single step is performed by clearing the SS flip-flop ($\overline{Q} = 1$), (i.e. placing a 10_8 in the control word and applying the proper transition on CB2). This forces a 0 on the Asynchronous $\overline{\text{Run}}$/Halt line. Once the Clock shifts into the second period, the SS flip-flop is returned high ($\overline{Q} = 0$) by $\emptyset_1$. Thus, the Clock will again be halted at the beginning of the next cycle. Clearing the SS flip-flop has no effect when the $\overline{\text{HALTRQ}}$ bit is high (i.e. normal operating mode).

When enabled, the CCR is loaded from the lower 4 bits of Port B. Note, the CONSOLEON bit can be set only if the following two conditions hold;

1. The Clock is already halted (HALTED = 1).
2. The Clock will continue to be halted by maintaining the low level of $\overline{\text{HALTRQ}}$.

This prevents this signal from being activated illegally. In addition, CONSOLEON is automatically deactivated when an attempt is made to re-enter the run mode by setting $\overline{\text{HALTRQ}}$ to 1. In this way the interface is forced to enter and exit the Console Control mode in the proper manner.

Since the halt request ($\overline{\text{HALTRQ}}$) bit of the CCR has an inverted input, it is activated by sending a 1 from Port B. The CONSOLEON and CSRWEN bits are also activated with 1s. Complemented forms of these two signals are generated outside the CCR. The computer is placed in normal operating mode by sending all zeros to the CCR. The $\overline{\text{RESET}}$ signal forces the computer into the halt state at power up. Since $\overline{\text{RESET}}$ clears all 4 bits of the CCR, the CONSOLEON and CSRWEN signals are deactivated.

To load the PLR and communicate with main memory this interface must control the $\emptyset_{1(\text{PLR})}$ and $\overline{\emptyset}_1$ clocks. This is done through the CONSOLE$\emptyset_{1(\text{PLR})}$ and CONSOLE$\overline{\emptyset}_1$ signals. In normal operating mode, CONSOLE$\emptyset_{1(\text{PLR})}$ is low and CONSOLE$\overline{\emptyset}_1$ is high. When CONSOLEON is activated, however, these signals, if enabled, can be driven by CA2. CONSOLE$\emptyset_{1(\text{PLR})}$ is enabled by the $\overline{\emptyset_{1(\text{PLR})}\text{EN}}$ control, and CONSOLE$\overline{\emptyset}_1$ by the $\overline{\text{BUS CNTRL}}$ control. Only one of these two can be enabled at a time.

To perform transactions with main memory this interface provides the

$\overline{\text{MEMSEL}}$,

R/$\overline{\text{W}}$,

LDMAR, and

BYTE

Bus control signals. These are supplied from Port B and enabled onto the Bus control lines by placing a 20_8 in the control word (as long as $\overline{\text{CONSOLEON}}$ is activated).

Appendix B includes a detailed presentation of the steps required by the microprocessor to perform typical Console operations with the Control Store and Main Memory.

15 Control Bit Summary

The following is a summary of the 88 control bits in the Pipeline Register. Each bit has been classified into 1 of 7 major fields. (4 of the bits are unused in this design.)

1. Microprogram Sequencer (26 bits)
2. ALU Function (9 bits)
3. Stack Tender/Register Select (16 bits)
4. Data Select/Sign Extend (13 bits)
5. Shift and Rotate (6 bits)
6. Status Control (2 bits)
7. Bus Interface (12 bits)

A brief functional description is supplied for each bit (or group of bits).

15.1 Microprogram Sequencer

$TEST_{3,2,1,0}$ - Test Field

This 4-bit field selects a machine status value to be tested. The selected status determines the Conditional Control input to the 2910.

POL - Polarity

This bit allows the test to be performed on the TRUE or FALSE state of the selected status value.

0 - test for the TRUE state of the status

1 - test for the FALSE state of the status

$CIS_{1,0}$ - CI Select Bits

These bits select from 4 possible inputs to the CI control on the 2910.

CIS_1	CIS_0	CI Inputs
0	0	1
0	1	0
1	0	CC
1	1	$\overline{CC}$

RLD - Register/Counter Load Enable

A 1 enables the register/counter in the 2910 to be loaded on the rising edge of $\emptyset_1$.

$I_{3,2,1,0}$ - 2910 Instruction Inputs

These controls specify 16 operations for the 2910 (see Figure 11-4).

NEXT ADDRESS FIELD

This 12-bit field supplies a potential Next Address value to the D inputs of the 2910.

$S_{1,0}$ - Next Address Select

These bits select 1 of 4 sources for the D inputs to the 2910.

15.2 ALU Function

$I_{2,1,0}$ - Source Operand Controls

These bits determine 8 combinations of source operands for the ALFU in the 2901 (see Figure 7-3).

$I_{5,4,3}$ - ALFU Function Controls

These bits select 1 of 8 functions to be performed by the ALFU, (see Figure 7-2).

$I_{8,7,6}$ - ALFU Destination/Shift Control

These bits determine the destination/shift operation for the ALFU result. They also select the source of the Y outputs for the 2901. (see Figure 7-4).

15.3 Stack Tender/Register Select

$NN_{2,1,0}$ - Number Needed Field

This field holds a 3-bit binary number between 000 and 100 (inclusive) which indicates the number of Fast Stack entries need to perform the current microinstruction. It should be set to 000 for microinstructions to which it does not apply.

$\overline{U}/D$ - $\overline{Up}$/Down Counter Control

This bit determines the direction of count for the NIS and TOS counters.

0 = count up

1 = count down

EN TOS - Enable TOS Counter

A 1 in this bit will enable the TOS counter to count in the direction specified by $\overline{U}/D$.

EN NIS - Enable NIS Counter

A 1 in this bit will enable the NIS counter to count in the direction specified by $\overline{U}/D$.

$ASEL_{3,2,1,0}$ - Register A Select Field

This field is combined with the Stack Tender logic to supply a 4-bit A register select field to the 2901s.

$BSEL_{3,2,1,0}$ - Register B Select Field

This field is combined with the Stack Tender logic to supply a 4-bit B register select field to the 2901s.

$M_{1,0}$ - Mode Bits

These bits determine 4 combinations of sources for the A and B register select inputs to the 2901s (see Figure 7-21).

15.4 Data Select/Sign Extend

BUS - Bus Select

When 1, the D inputs to the 2901 will come from the Bus. Otherwise these inputs are determined by the Data Select controls.

$DS_{1,0}$ - Data Select Controls

These bits select a source for the D inputs to the 2901s when the BUS control bit is 0.

DS_1	DS_0	Selected D inputs source
0	0	Immediate Field from PLR
0	1	NIS Counter
1	0	Second Byte of IR
1	1	Not Used

SE - Sign Extend

When a byte source has been selected for the D inputs to the 2901 it can be sign extended to 16 bits by setting this bit to 1.

TE - Test Enable

This control enables the test result to force the D inputs of the 2901 to zero. If the test is FALSE, the D inputs are set to zero, otherwise they are determined in normal fashion. TE must be set to 1 to enable this operation.

15.5 Shift and Rotate

D - Double

0 = Single width (16 bit) shift or rotate

1 = Double width (32 bit) shift or rotate

IC - Include Carry

0 = Don't include CARRY

1 = Include CARRY

R - Rotate

0 = Shift

1 = Rotate

ARTH - Arithmetic Mode

This bit affects only left and right shift operations.

0 = non-arithmetic shift

1 = arithmetic shift

0/1 - Zero/One

This bit is used to supply 0 or 1 on various shift inputs. It can also function as the "carry in" value to the ALU.

C_{IN} - Carry In Control

0 = "Carry in" comes from 0/1

1 = "Carry in" comes from CARRY bit

15.6 Status Control

SS - Set Status

A 1 will cause the ALFU status bits to be set according to the results of the current microinstruction, otherwise these bits are unchanged.

RS - Return Status

A 1 enables all 6 status bits to be loaded from memory via the Bus.

15.7 Bus Interface

BRST - Bus Reset

A 1 will activate the $\overline{\text{BRST}}$ signal causing a Bus reset.

$BS_{1,0}$ - Bus Select Controls

These bits enable the following CPU devices onto the Bus.

BS_1	BS_0	On Bus
0	0	Nothing
0	1	Y outputs of 2901
1	0	6-bit status word
1	1	Not Used

LDIR - Load Instruction Register

A 1 will enable the IR to be loaded from the Bus.

LDMAR - Load Memory Address Register

When set to 1, the MAR will be loaded from the Bus.

MEMSEL - Memory Select

A 1 will activate the $\overline{\text{MEMSEL}}$ Bus control line. This signifies a data transfer with Main Memory.

$R/\overline{W}$ - Read/Write

This bit specifies the direction of data transfer with Main Memory.

0 = write

1 = read

BYTE

A 1 signifies that a byte is to be transferred to/from Main Memory, otherwise transfers with memory are performed on a word basis.

I/O SEL - I/O Select

A 1 activates the $\overline{\text{I/O SEL}}$ Bus control line. This indicates a transaction with I/O.

I/O STAT/$\overline{\text{DATA}}$ - I/O Status/$\overline{\text{Data}}$

When the I/O SEL bit is set, this bit specifies one of two operations.

0 = Two steps are required to transfer data with I/O. This bit must be low for both of these steps.

1 = The Ready Flag of the selected I/O device is placed on the I/O READY line and transferred to the I/O READY status bit.

INT ACK - Interrupt Acknowledge

A 1 activates the $\overline{\text{INT ACK}}$ Bus control line which in turn causes the Device Code of the highest priority interrupting device to be placed on the Bus.

MSKEN - Interrupt Mask Enable

A 1 activates the $\overline{\text{MSKEN}}$ Bus control line which causes the interrupt mask bits of all 16 levels to be set according to the mask word placed on the Bus information lines.

16 On the Assembly of Microinstructions

The functions performed by each bit (or group of bits) in a microinstruction can be placed into 1 of 4 catagories.

Single Operation

Either - Or

Multiple Choice

Data Fields

Single Operation determine the on/off condition of a specific control signal. Examples include the RLD, ARTH, and MEM SEL bits. All Single Operation bits in this machine are set to 1 to activate the corresponding control signal.

Either - Or bits are those that specify either one condition or another. The $R/\overline{W}$ bit is a good example. These bits are usually meaningless unless a related operation has been actiavted by a Single Operation bit or a Multiple Choice group of bits.

Groups of bits that specify a class of functions belong to the Multiple Choice catagory. The instruction bits, I_8 - I_0, for the 2901 are a good example.

Fields such as the Next Address and Immediate fields are designated as Data Fields. These fields supply binary numbers rather than control signals.

With the exception of one 4-bit field, setting all 84 bits of a microinstruction to zero will have the effect of a No Op. Thus, microinstructions can be assembled by setting bits and filling fields to perform only those operations desired. The only field that must be specified for every microinstruction is the 2910 instruction field, $I_3 - I_0$.

To repeatedly assemble these 84 bits to useful microinstructions can be a very tedious task in the absence of some sort of structured approach. Obviously there is a hierarchial relationship between various bits and/or groups of bits. For example, it would be useless to assign values for R, D, IC, and ARTH unless a shift operation has been declared by the I_8, I_7, I_6 field for the 2901. Similarly, it would be pointless to designate a value for $R/\overline{W}$ unless MEM SEL has been set. In addition, some bits, or groups of bits, are restrained because their operations cannot be performed in parallel with others. For example, a read from memory cannot be performed concurrently with an input from I/O because both require use of the Bus. In light of the above discussion it seems necessary to develop an organized approach to assembling microinstructions that accounts for their hierarchial structure and at the same time enforces their various restrictions. The development of such an approach is left for those who plan to microprogram this machine.

17 Microinstruction Routines

In hopes of tying the various parts of the computer together, a number of microinstruction routines are developed here. Routines that deal with the following operations will be presented.

Macroinstruction Fetch

Memory Reference Instructions

TRAP "fix it" Routines

I/O Instructions

Conditional Jump Macroinstruction

Shift/Rotate Instruction

17.1 Approach to the TRAP Condition

Since a large number of routines deal with the Fast Stack, it is important that the proceedure for handling a TRAP condition is fully understood. Microinstructions that have the potential of generating a TRAP will be called critical microinstructions. TRAPs are generated as a result of an inadequate condition of the Fast Stack. Most routines can be written so that no time is wasted checking for this condition. The best approach for microcode development is to write the routines assumming that the Fast Stack is sufficient, then return to the critical microinstructions and modify them to test for the TRAP. When a TRAP occurs, control is passed to a "fix it" routine via a microinstruction jump. This routine simply modifies the condition of the Fast Stack and returns to the microinstruction from which the trap was generated.

Because of the manner in which traps are generated, it is impossible to implement "fix it" routines in the form of microprogram subroutines. This is due largely to the fact that control must be returned to the microinstruction that generates the TRAP, not the succeeding one. Another attempt must be made to execute this microinstruction since its operation is inhibited by the trapping condition. Thus, a method must be devised for returning from the "fix it" routines. The register/counter in the 2910 plays an important role here. Both "fix it" routines use the address in the register/counter as the address of the return microinstruction. This implies that all microinstructions preceding a critical microinstruction <u>must load the register/counter with the address of this microinstruction</u>. This address must therefore be supplied via the D inputs of the 2910.

In summary, the return address for the "fix it" routine is loaded into the register/counter by the microinstruction directly preceding the critical one. The critical microinstruction then tests for the TRAP via a conditional jump operation. If the test is positive, a jump is made to the "fix it" routine, otherwise the next sequential microinstruction is performed. Once the Fast Stack condition is appropriately altered, the "fix it" routine can return to the critical microinstruction via the address in the register/counter. Examples of "fix it" routines are given in the section on Memory Reference Instructions.

17.2 General Approach for the Presentation of Microinstruction Routines.

Each of the microinstruction routines that follow are presented in three sections.

1. Algorithm
2. Word Description
3. Assembly

The Algorithm gives a general idea of the data transfers and operations to be performed. Many details are left out at this stage. The Word Description describes, in words, the operations performed by each microinstruction in the routine designed to carry out the Algorithm. Finally, the Assembly section defines the 1s and 0s for each microinstruction in the Word Description. Figures 17-1 and 17-2 are used to aid in the development of these routines.

None of the routines that follow are unique. Each can be written in a slightly different manner. The intent is primarily to provide the reader with a reasonable understanding of microcode development in this machine.

Entry points for various routines have been arbitrarily chosen as shown below.

Routine	CS Location
FETCH1	1000_8
FETCH2	1002_8
"Interrupt Service Routine"	0400_8
TRAP1 "fix it"	5000_8
TRAP2 "fix it"	5002_8

Summary of Control Bits with Computer Block Diagram

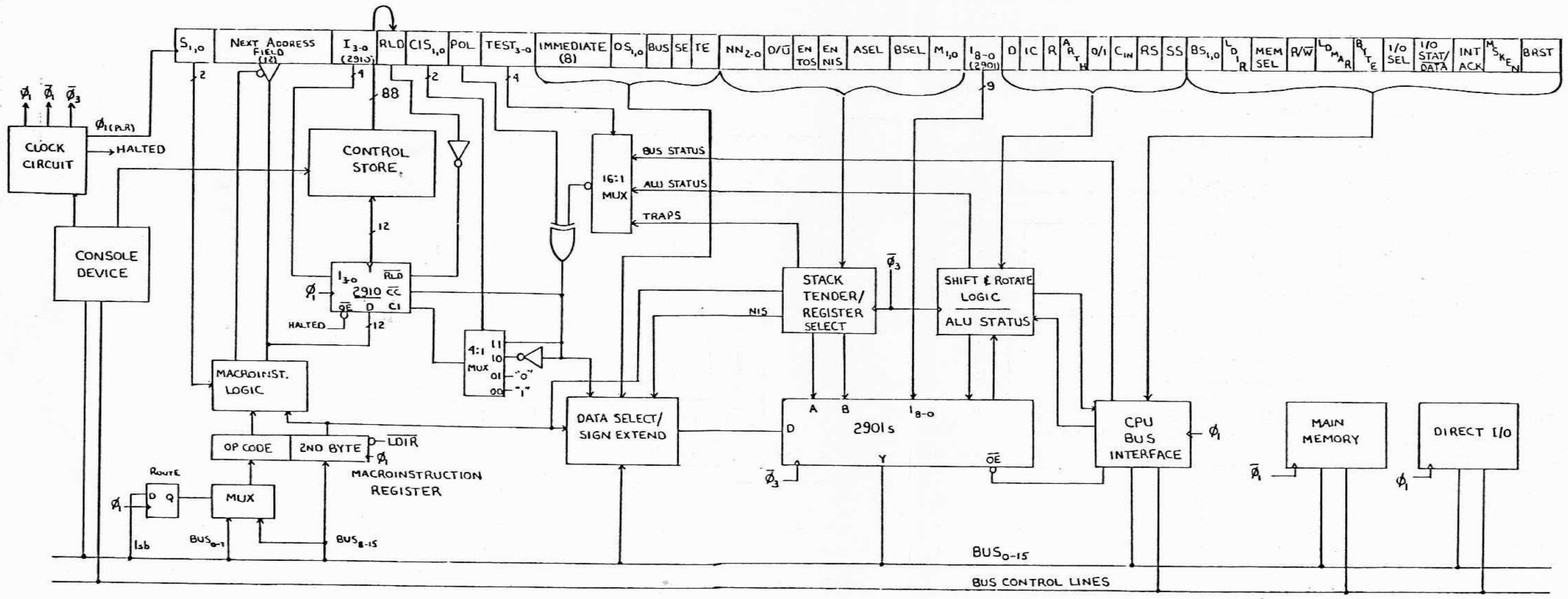


Figure 17-1

2910 Instruction Set

HEX I_3-I_0	Mnemonic	Name	Reg/ Cntr	FAIL $\overline{CCEN}=0$ & $\overline{CC}=1$ Y	Stack	PASS $\overline{CCEN}=1$ or $\overline{CC}=0$ Y	Stack	Reg/ Cntr	Enable
0	JZ	Jump Zero	X	0	Clear	0	Clear	Hold	PL
1	CJS	Cond JSB PL	X	PC	Hold	D	Push	Hold	PL
2	JMAP	Jump MAP	X	D	Hold	D	Hold	Hold	MAP
3	CJP	Cond Jump PL	X	PC	Hold	D	Hold	Hold	PL
4	PUSH	PUSH/Cond LD Cntr	X	PC	PUSH	PC	PUSH	Note1	PL
5	JSRP	Cond JSB R/PL	X	R	PUSH	D	PUSH	Hold	PL
6	CJV	Cond Jump Vector	X	PC	Hold	D	Hold	Hold	VECT
7	JRP	Cond Jump R/PL	X	R	Hold	D	Hold	Hold	PL
8	RFCT	RPT Loop, Cntr 0	≠0	F	Hold	F	Hold	Dec	PL
			=0	PC	POP	PC	POP	Hold	PL
9	RPCT	RPT PL, Cntr 0	≠0	D	Hold	D	Hold	Dec	PL
			=0	PC	Hold	PC	Hold	Hold	PL
A	CRTN	Cond RTN	X	PC	Hold	F	POP	Hold	PL
B	CJPP	Cond Jump PL & POP	X	PC	Hold	D	POP	Hold	PL
C	LDCT	LD Cntr & Continue	X	PC	Hold	PC	Hold	Load	PL
D	LOOP	Test End Loop	X	F	Hold	PC	POP	Hold	PL
E	CONT	Continue	X	PC	Hold	PC	Hold	Hold	PL
F	TWB	Three Way Branch	≠0	F	Hold	PC	POP	Dec	PL
			=0	D	POP	PC	POP	Hold	PL

Note1: If $\overline{CCEN}=0$ and $\overline{CC}=1$, Hold, else Load

Source Operand and ALFU Function Matrix

Octal $I_{5,4,3}$	ALFU function	Octal $I_{2,1,0}$ ALFU Source: 0 A,Q	1 A,B	2 0,Q	3 0,B	4 0,A	5 D,A	6 D,Q	7 D,0
0	$C_n=0$ R plus S	A + Q	A + B	Q	B	A	D + A	D + Q	D
	$C_n=1$	A+Q+1	A+B+1	Q + 1	B + 1	A + 1	D+A+1	D+Q+1	D + 1
1	$C_n=0$ S minus R	Q-A-1	B-A-1	Q-1	B-1	A-1	A-D-1	Q-d-1	-D-1
	$C_n=1$	Q-A	B-A	Q	B	A	A-D	Q-D	-D
2	$C_n=0$ R minus S	A-Q-1	A-B-1	-Q-1	-B-1	-A-1	D-A-1	D-Q-1	D-1
	$C_n=1$	A-Q	A-B	-Q	-B	-A	D-A	D-Q	D
3	R OR S	$A \vee Q$	$A \vee B$	Q	B	A	$D \vee A$	$D \vee Q$	D
4	R AND S	$A \wedge Q$	$A \wedge B$	0	0	0	$D \wedge A$	$D \wedge Q$	0
5	$\overline{R}$ AND S	$\overline{A} \wedge Q$	$\overline{A} \wedge B$	Q	B	A	$\overline{D} \wedge A$	$\overline{D} \wedge Q$	0
6	R EX-OR S	$A \veebar Q$	$A \veebar B$	Q	B	A	$D \veebar A$	$D \veebar Q$	D
7	R EX-NOR S	$\overline{A \veebar Q}$	$\overline{A \veebar B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \veebar A}$	$\overline{D \veebar Q}$	$\overline{D}$

$\veebar$ = EX-OR $\vee$ = OR $\wedge$ = AND + = Plus − = Minus

TEST Select

Test Field T_3	T_2	T_1	T_0	Selected Status
0	0	0	0	1
0	0	0	1	CARRY
0	0	1	0	OVERFLOW
0	0	1	1	SIGN
0	1	0	0	ZERO
0	1	0	1	INTERRUPT
0	1	1	0	I/O READY
0	1	1	1	TRAP 1
1	0	0	0	TRAP 2
1	0	0	1	N/A
1	0	1	0	.
1	0	1	1	.
1	1	0	0	.
1	1	0	1	.
1	1	1	0	.
1	1	1	1	N/A

Polarity

POL	Test for:
0	TRUE (1)
1	FALSE (0)

2910 D Inputs

S_1	S_0	2910 D Inputs
0	0	Next Address Field
0	1	IR 2nd Byte
1	0	IR Op Code
1	1	IR EA/Op Code

CI Control

CIS_1	CIS_0	CI
0	0	1
0	1	0
1	0	CC
1	1	$\overline{CC}$

ALU Registers

ASEL or BSEL	2901 Register
0 0 0 0	Fast Stack
0 0 0 1	.
0 0 1 0	.
0 0 1 1	Fast Stack
0 1 0 0	TEMP 1
0 1 0 1	TEMP 2
0 1 1 0	Z, Stack Limit Ptr.
0 1 1 1	PL, Code Limit Ptr.
1 0 0 0	PB, Code Base Ptr.
1 0 0 1	PC, Program Counter
1 0 1 0	INDEX1
1 0 1 1	INDEX2
1 1 0 0	DB, Data Base Ptr.
1 1 0 1	Q, Stack Marker
1 1 1 0	MSP, Memory Stack Ptr.
1 1 1 1	DL, Data Limit Ptr.

ALU "Carry In"

C_{IN}	"Carry In" to ALU
0	0/1
1	CARRY

A and B Register Select

MODE Bits M_1	M_0	Reg. A Select	Reg. B Select
0	0	ASEL	BSEL
0	1	ASEL	BDS
1	0	A_{IR}	BSEL
1	1	A_{IR}	B_{IR}

ALFU Destination/Shift Control

Control Inputs I8	I7	I6	RAM Function Shift	Load	Q-reg Function Shift	Load	Y Output	RAM Shifter RAM0	RAM3	Q Shifter Q0	Q3
0	0	0	X	none	none	F→Q	F	X	X	X	X
0	0	1	X	none	X	none	F	X	X	X	X
0	1	0	none	F→B	X	none	A	X	X	X	X
0	1	1	none	F→B	X	none	F	X	X	X	X
1	0	0	down	F/2→B	down	Q/2→Q	F	F0	IN3	Q0	IN3
1	0	1	down	F/2→B	X	none	F	F0	IN3	Q0	X
1	1	0	up	2F→B	up	2Q→Q	F	IN0	F3	IN0	Q3
1	1	1	up	2F→B	X	none	F	IN0	F3	X	Q3

Note;
X=don't care
down=shift right
up=shift left

2901 Direct Data Inputs

(BUS = 0)

DS_1	DS_2	D Inputs to 2901
0	0	IMMEDIATE
0	1	NIS
1	0	IR 2nd Byte
1	1	N/A

CPU ON the Bus

BS_1	BS_0	ON the Bus
0	0	Nothing (Safe state)
0	1	Y's from 2901
1	0	CPU STATUS bits
1	1	N/A

Figure 17-2 Summary of Various Control Fields.

17.3 Macroinstruction Fetch Routine

The Macroinstruction Fetch routine has numerous responsibilities. It must first test for interrupts since they are honored between macroinstructions only. If an interrupt is pending, the Fetch routine is aborted and control passed to an interrupt service routine. The Fetch routine is also responsible for proper adjustment of the PC. For this reason it has two entry points, one that increments the PC by 1, and one that increments it by 2.

The first entry point, FETCH1, increments the PC by 1, and the second entry point, FETCH2, increments it by 2. FETCH1 begins at Control Store location 1000_8 and FETCH2 at 1002_8. The purpose of each of these routines is to fetch the next macroinstruction from main memory and jump to the microcode routine determined by the Op Code of that instruction. For Memory Reference Instructions it will jump instead to the EA routine determined by the addressing scheme in the second byte of the IR. Before the macroinstruction is fetched, however, a test is made to see if an interrupt is pending. If so, control is passed to an interrupt service routine which is chosen to begin at CS location 0400_8. Note, the register/counter is always loaded by the last microinstruction in the fetch routine, i.e. the one that jumps to the execution code for the fetched instruction. Thus, the register/counter will hold the address of the first microinstruction in the next routine in the event that a TRAP is generated.

ALGORITHM for FETCH1 (or FETCH2)

$$\begin{tikzcd}
\text{Interrupt ?} \arrow[r, "\text{YES}"] \arrow[d, "\text{NO}"] & \text{Interrupt Service Routine} \\
\text{PC} \leftarrow \text{PC} + 1 \text{ (or PC + 2)} \arrow[d] & \\
\text{IR} \leftarrow [\text{PC}] \arrow[d] & \\
\text{Jump to the microcode routine determined by the Op Code (or EA scheme)} &
\end{tikzcd}$$

WORD DESCRIPTION

LOCATION	DESCRIPTION
1000_8	* TEST for INTERRUPT a) PASS - Jump to Interrupt Service routine at 0400_8 via NA field b) FAIL - Fetch the next sequential microinstruction * Add 1 to the PC, but don't store the result * Send the result to the MAR
1001_8	* Read from Main Memory into the IR * Add 1 to the PC and store the result into the PC * Fetch the next microinstruction from 1004_8 via the NA field

LOCATION	DESCRIPTION
1002_8	* TEST for an INTERRUPT a) PASS - Jump to Interrupt Service routine at 0400_8 via NA field b) FAIL - Fetch next sequential microinstruction * Add 2 to the PC but don't store the result * Send the result to MAR
1003_8	* Read from Memroy into the IR * Add 2 to the PC and store the result back in the PC * Fetch the next sequential microinstruction
1004_8	* Fetch the next microinstruction from the IR EA/OPCODE source in the Macroinstruction Logic circuit * Load the Register/Counter

ASSEMBLY

LOCATION	MICROINSTRUCTION	COMMENTS
1000_8	2910	
	$I_{3-0} \leftarrow 0011$	CJP
	$TEST_{3-0} \leftarrow 0101$	TEST for INTERRUPT
	$POL \leftarrow 0$	
	$S_{1,0} \leftarrow 00$	D inputs are 0400_8
	$NA \leftarrow 0400_8$	(Interrupt Service Routine)
	ALU	
	$I_{5-0} \leftarrow 000101$	ADD, A plus D
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_1M_0 \leftarrow 00$	Normal Mode
	$ASEL \leftarrow 1001$	PC
	$BUS \leftarrow 0$	
	$DS_{1,0} \leftarrow 00$	1 to D inputs
	Immediate $\leftarrow 00000001$	
	$I_{8-6} \leftarrow 000$	Result to Y outputs Don't store result
	BUS	
	$BUS_{1,0} \leftarrow 01$	Result on Bus
	$LDMAR \leftarrow 1$	Load MAR

LOCATION	MICROINSTRUCTION	COMMENTS
1001_8	<u>2910</u>	
	$I_{3-0} \leftarrow 0010$	JMAP
	$S_{1,0} \leftarrow 00$	Next Address is 1004_8
	$NA \leftarrow 1004_8$	
	<u>ALU</u>	
	$I_{5-0} \leftarrow 000101$	ADD, A plus D
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_{1,0} \leftarrow 00$	Normal Mode
	ASEL $\leftarrow$ 1001	PC
	BUS $\leftarrow$ 0	
	$DS_{1,0} \leftarrow 00$	1 to D inputs
	Immediate $\leftarrow$ 00000001	
	$I_{8-6} \leftarrow 010$	Result to PC
	BSEL $\leftarrow$ 1001	
	<u>BUS</u>	
	MEMSEL $\leftarrow$ 1	
	$R/\overline{W} \leftarrow 1$	Read from Memory into IR
	LDIR $\leftarrow$ 1	

LOCATION	MICROINSTRUCTION	COMMENTS
1002_8	<u>2910</u>	
	I_{3-0} ← 0011	CJP
	$TEST_{3-0}$ ← 0101	TEST for INTERRUPT
	POL ← 0	
	$S_{1,0}$ ← 00	D inputs are 0400_8
	NA ← 0400_8	(Interrupt Service routine)
	<u>ALU</u>	
	$I_{5,0}$ ← 000101	ADD, A plus D
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1001	PC
	BUS ← 0	
	$DS_{1,0}$ ← 00	2 to D inputs
	Immediate ← 00000010	
	I_{8-6} ← 000	Result to Y outputs Don't store result
	<u>BUS</u>	
	$BS_{1,0}$ ← 01	Result on Bus
	LDMAR ← 1	Load MAR

LOCATION	MICROINSTRUCTION	COMMENTS
1003_8	<u>2910</u>	
	I_{3-0} ← 1110	CONTINUE
	<u>ALU</u>	
	I_{5-0} ← 000101	ADD, D plus A
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1001	PC
	BUS ← 0	
	$DS_{1,0}$ ← 00	2 to D inputs
	Immediate ← 00000010	
	I_{8-6} ← 010	Result to PC
	BSEL ← 1001	
	<u>BUS</u>	
	MEMSEL ← 1	
	$R/\overline{W}$ ← 1	Read value from Memory to IR
	LDIR ← 1	
1004_8	<u>2910</u>	
	I_{3-0} ← 0010	JMAP
	$S_{1,0}$ ← 11	IR EA/OPCODE is source of Next Address
	RLD ← 1	Load Register/Counter with this value

17.4 Microcode for Memory Reference Instructions

As described earlier, this machine can perform up to 16 different Memory Reference Instructions, each with as many as 32 addressing schemes. Each of these instructions is carried out via two separate routines; one to compute the Effective Address, and the other to perform the operation specified by the Op Code. The EA routine is completely independent from that of the Op Code. It is determined solely by the addressing scheme specified in the second byte of the Instruction Register. Once the EA routine is completed, the EA is passed to the Op Code routine via a temporary ALU register. Here it is used as a possible source and/or destination memory address.

The format used for Memory Reference Instructions in the following material is the 3-byte version presented in Section 6. The first byte holds the Op Code, the second byte determines the addressing scheme, and the third byte supplies the relative address. The first 5 bits of the second byte determine the actual addressing mode and are used to form the EAR's Address as defined in Section 11.3.

17.4a Computation of the Effective Address with respect to the Program Counter.

This routine calculates the Effective Address relative to the Program Counter. This type of addressing (PC$\pm$n) is selected by placing 01000XXX in the second byte of the instruction (the lower 3 bits are irrelevant). Thus, the CS address of this routine is 3020_8.

Algorithm

$PC \leftarrow PC + 2$

$TEMP1 \leftarrow PC + [PC]_{BYTE,\ SE}$

Word Description

Location	Description
3020_8	*Add 2 to the PC and store the result back in the PC *Send result to the MAR *Fetch Next Sequential microinstruction
3021_8	*Read a byte from memory, sign extend this value to 16 bits and add it to the PC. Place the result in TEMP1. *Fetch the next microinstruction from the IR OP CODE address

Microinstruction Assembly

Location	Microinstruction	Comments
3020_8	2910	
	$I_{3-0} \leftarrow 1110$	CONTINUE
	ALU	
	$I_{5-0} \leftarrow 000101$	D plus A
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_{1,0} \leftarrow 00$	Normal Mode
	ASEL ← 1001	PC
	$DS_{1,0} \leftarrow 00$	2 to D inputs
	Immediate ← 00000010	
	$I_{8-6} \leftarrow 010$	Result to PC
	BSEL ← 1001	
	BUS	
	$BS_{1,0} \leftarrow 01$	Result to Bus
	LDMAR ← 1	Load MAR
3021_8	2910	
	$I_{3-0} \leftarrow 0010$	JMAP
	$S_{1,0} \leftarrow 10$	Next Address = IR OP CODE
	ALU	
	$I_{5,0} \leftarrow 000111$	D plus 0
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	BUS ← 1	Sign Extended
	SE ← 1	byte from the Bus
	BYTE ← 1	to D inputs
	$I_{8-6} \leftarrow 010$	Result to TEMP1
	BSEL ← 0100	
	BUS	
	MEM SEL ← 1	Read from memory
	$R/\overline{W} \leftarrow 1$	

17.4b Computation of the Effective Address with respect to the Stack Pointer.

Computation of the EA for most addressing modes requires a straight forward sequence of microinstructions. Computing the EA relative to the Stack Pointer, however, is not so straight forward. Since no real Stack Pointer exists, a value for the logical Stack Pointer, SP, must first be determined. Recall,

$$SP = MSP + NIS * 2.$$

The EA is then computed using,

$$EA = SP - n$$

i.e.,

$$EA = MSP + NIS * 2 - n$$

where n is the relative address in the third byte of the instruction. It is apparent from the above equation that the EA could turn out to be larger than the MSP value. This implies that the operand lies in the Fast Stack rather than Main Memory. This condition poses two problems.

1. Even though the operand is known to lie in the Fast Stack, the exact Fast Stack location is not easily determined.
2. Also, the Op Code routine to follow expects to receive an address to the operand in Main Memory.

Perhaps the simplest solution here is to transfer values from the Fast Stack to the memory stack until the operand actually does lie in Main Memory. Although this would seem to defeat the purpose of the Fast Stack, it is probably the most efficient solution in terms of both execution time and Control Store space. It should be noted that this condition is an exceptional one. Most operations involving operands at the top of the stack should be performed with Stack Ops rather than Memory Reference Instructions.

The microcode to implement the above routine is presented below. The addressing mode (S-n) is specified by placing 10100XXX in the second byte of the instruction. This results in a CS address of 3050_8 for this routine.

Algorithm

```
          TEMP2  ←  MSP + NIS*2
          PC   ←  PC + 2
          TEMP1  ←  TEMP2 - [PC]
          TEMP2  ←  TEMP1
          TEMP2  ←  MSP - TEMP2
  ┌────→  ? NON-NEGATIVE ?
  │              /     \
  │           N /       \ Y
  │            /         \
  │     MSP ← MSP + 2      \
  │     [MSP] ← BOS REGISTER   RETURN TO ROUTINE SPECIFIED BY OP CODE
  │     NIS  ← NIS - 1          WITH EA IN TEMP1.
  │     TEMP2 ← TEMP2 - 2
  │           │
  └───────────┘
```

Word Description

Location	Description
3050_8	*Add NIS*2 to the MSP and store result in TEMP2. *Fetch next sequential microinstruction
3051_8	*Add 2 to the PC and store result back in PC *Send result to the MAR *Fetch next microinstruction from 4000_8 via the Next Address field.
4000_8	*Read a byte from memory, subtract it from TEMP2, store result in TEMP1. *Fetch next sequential microinstruction
4001_8	*Transfer TEMP1 to TEMP2 *Fetch next sequential microinstruction
4002_8	*Subtract TEMP2 from MSP, store result back in TEMP2 *Set ALU status bits *Fetch next sequential microinstruction
4003_8	*Test for non-negative result by testing the TRUE state of the CARRY a) PASS - fetch the next microinstruction from the IR OP CODE address. b) FAIL - fetch the next sequential microinstruction. *Load the register/counter with the IR OP CODE address (In case of a TRAP in succeeding routine).
4004_8	*Add 2 to the MSP, store result back in the MSP *Send result to MAR *Fetch the next sequential microinstruction

Location	Description
4005_8	*Write the BOS register to memory *Decrement the NIS counter *Fetch the next sequential microinstruction
4006_8	*Subtract 2 from TEMP2, store result back in TEMP2 *Set the ALU status bits *Fetch the next microinstruction from 4003_8 via the Next Address field

Assembly

Location	Microinstruction	Comments
3050_8	2910	
	I_{3-0} ← 1110	CONTINUE
	ALU	
	I_{5-0} ← 000101	D plus A
	C_{IN} ← 0	"Carry in " = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1110	MSP
	BUS ← 0	NIS*2 to D inputs
	$DS_{1,0}$ ← 01	
	I_{8-6} ← 010	Result to TEMP2
	BSEL ← 0101	
3051_8	2910	
	I_{3-0} ← 0010	JMAP
	$S_{1,0}$ ← 00	Next Address is 4000_8
	NA ← 4000_8	
	ALU	
	I_{5-0} ← 000101	D plus A
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1001	PC
	BUS ← 0	2 to D inputs
	$DS_{1,0}$ ← 00	
	Immediate ← 00000010	
	I_{8-6} ← 010	Result to PC
	BSEL ← 1001	
	BUS	
	$BS_{1,0}$ ← 01	Result to MAR
	LDMAR ← 1	

Location	Microinstruction	Comments
4000_8	<u>2910</u>	
	$I_{3,0}$ ← 1110	CONTINUE
	<u>ALU</u>	
	$I_{5,0}$ ← 001101	A minus D
	C_{IN} ← 0	"Carry in" = 1
	0/1 ← 1	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 0101	TEMP2
	BUS ← 1	Bus to D inputs
	I_{8-6} ← 010	Result to TEMP1
	BSEL ← 0100	
	<u>BUS</u>	
	MEM SEL ← 1	Read a byte from memory
	R/$\overline{W}$ ← 1	
	BYTE ← 1	
4001_8	<u>2910</u>	
	I_{3-0} ← 1110	CONTINUE
	<u>ALU</u>	
	$I_{5,0}$ ← 000100	A plus 0
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 0100	TEMP1
	I_{8-6} ← 010	Result to TEMP2
	BSEL ← 0101	

Location	Microinstruction	Comments
4002_8	<u>2910</u>	
	I_{3-0} ← 1110	CONTINUE
	<u>ALU</u>	
	$I_{5,0}$ ← 010001	A minus B
	C_{IN} ← 0	"Carry in" = 1
	0/1 ← 1	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1110	MSP
	BSEL ← 0101	TEMP2
	I_{8-6} ← 010	Result to TEMP2
	SS ← 1	Set Status
4003_8	<u>2910</u>	
	I_{3-0} ← 0011	CJP
	$TEST_{3-0}$ ← 0001	Test for CARRY = 1
	POL ← 0	
	$S_{1,0}$ ← 10	IR OP CODE to D inputs
	RLD ← 1	Load Register/Counter
4004_8	<u>2910</u>	
	$I_{3,0}$ ← 1110	CONTINUE
	<u>ALU</u>	
	I_{5-0} ← 000101	A plus D
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1110	MSP
	BUS ← 0	2 to D inputs
	$DS_{1,0}$ ← 00	
	Immediate ← 00000010	
	I_{8-6} ← 010	Result to MSP
	BSEL ← 1110	

Location	Microinstruction	Comments
4004_8 (cont'd)	BUS	
	$BS_{1,0} \leftarrow 01$	Result to MAR
	LDMAR $\leftarrow$ 1	
4005_8	2910	
	$I_{3-0} \leftarrow 1110$	CONTINUE
	ALU	
	$I_{5-0} \leftarrow 000011$	B plus 0
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_{1,0} \leftarrow 01$	B = BOS register
	$D/\overline{U} \leftarrow 1$	Decrement NIS
	EN NIS $\leftarrow$ 1	
	$I_{8-6} \leftarrow 000$	Don't store result Result to Y outputs
	BUS	
	$BS_{1,0} \leftarrow 01$	Write result to memory
	MEM SEL $\leftarrow$ 1	
	$R/\overline{W} \leftarrow 0$	
4006_8	2910	
	$I_{3-0} \leftarrow 0010$	JMAP
	$S_{1,0} \leftarrow 00$	Next Address is 4003_8
	NA $\leftarrow 4003_8$	
	ALU	
	$I_{5,0} \leftarrow 001101$	A minus D
	$C_{IN} \leftarrow 0$	"Carry in" = 1
	$0/1 \leftarrow 1$	
	$M_{1,0} \leftarrow 00$	Normal Mode
	ASEL $\leftarrow$ 0101	TEMP2

Location	Microinstruction	Comments

4006_8 (cont'd)	BUS ← 0
	$DS_{1,0}$ ← 00
	Immediate ← 00000010
		2 to D inputs

	I_{8-6} ← 010 BSEL ← 0101	Result to TEMP2
	SS ← 1	Set Status

Once the EA is computed, control is passed to the Op Code routine. The EA is passed to this routine in TEMP1. Two separate Memory Reference Instructions have been chosen for illustration here, a PUSH and a POP. Each has the potential of generating a different TRAP. Thus, microinstruction routines to handle both TRAP1 and TRAP2 are also presented.

17.4c The PUSH Macroinstruction

Assumptions:

1. The EA lies in the TEMP1 register.
2. The Op Code for this instruction is 11110000. This implies that the CS address of this routine is 2740_8.

Algorithm

$TOS \leftarrow TOS + 1$

$NIS \leftarrow NIS + 1$

$[TOS] \leftarrow [EA]$

Word Description

Location	Description
2740_8	*Send TEMP1 to the MAR *Increment TOS and NIS *Test for TRAP1 a) PASS - jump to the TRAP1 "fix it" routine at location 5000_8 via NA field b) FAIL - Fetch next sequential micro-instruction
2741_8	*Read from Memory into the TOS register *Jump to the FETCH1 routine at 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2740_8	2910	
	I_{3-0} ← 0011	CJP
	$TEST_{3-0}$ ← 0111	Test for TRAP1
	POL ← 0	
	$S_{1,0}$ ← 00	D inputs = 5000_8
	NA ← 5000_8	
	ALU	
	I_{5-0} ← 000011	B plus 0
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	M_1M_0 ← 00	Normal Mode
	BSEL ← 0100	TEMP1
	I_{8-6} ← 000	Result to Y Don't store result
	BUS	
	$BS_{1,0}$ ← 01	Result to MAR
	LDMAR ← 1	
2741_8	2910	
	$I_{3,0}$ ← 0010	JMAP
	$S_{1,0}$ ← 00	Next Address is 1000_8
	NA ← 1000_8	
	ALU	
	I_{5-0} ← 000111	D plus 0
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	BUS ← 1	D inputs from Bus
	$M_{1,0}$ ← 00	Normal Mode
	I_{8-6} ← 010	Result to TOS
	BSEL ← 0000	
	BUS	
	MEMSEL ← 1	Read from memory
	$R/\overline{W}$ ← 1	

17.4d The TRAP1 Microinstruction Routine

Algorithm

$MSP \leftarrow MSP + 2$

$[MSP] \leftarrow [BOS]$

$NIS \leftarrow NIS - 1$

Word Description

Location	Description
5000_8	*Add 2 to the MSP, store the result back in the MSP *Send the result to the MAR *Fetch next sequential microinstruction
5001_8	*Write BOS register to memory *Decrement NIS *Fetch next microinstruction from the address in the register/counter

Assembly

Location	Microinstruction	Comments
5000_8	2910	
	$I_{3-0} \leftarrow 1110$	CONTINUE
	ALU	
	$I_{5-0} \leftarrow 000101$	A plus D
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_1M_0 \leftarrow 00$	Normal Mode
	ASEL $\leftarrow$ 1110	MSP
	BUS $\leftarrow$ 0	2 to D inputs
	$DS_{1,0} \leftarrow 00$	
	Immediate $\leftarrow$ 00000010	
	$I_{8-6} \leftarrow 010$	Result to Y
	BSEL $\leftarrow$ 1110	Result to MSP
	BUS	
	$BS_{1,0} \leftarrow 01$	Result to MAR
	LDMAR $\leftarrow$ 1	

Assembly

Location	Microinstruction	Comments
5001_8	2901	
	$I_{3-0} \leftarrow 0111$	-JRP
	$TEST_{3-0} \leftarrow 0000$	Test will FAIL, Next Address determined by register/counter
	$POL \leftarrow 1$	
	ALU	
	$I_{5,0} \leftarrow 000011$	B plus 0
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_1M_0 \leftarrow 01$	B is the BOS register
	$D/\overline{U} \leftarrow 1$	Decrement NIS
	EN NIS $\leftarrow 1$	
	$I_{8,6} \leftarrow 000$	Result to Y Don't store result
	BUS	
	$BS_{1,0} \leftarrow 01$	Write result to memory
	MEM SEL $\leftarrow 1$	
	$R/\overline{W} \leftarrow 0$	

17.4e The POP Macroinstruction

Assumptions:

1. The EA lies in the TEMP1 register
2. The Op Code for this instruction is 11110001. This implies that the CS location of this routine is 2742_8.

Algorithm

$\boxed{EA} \leftarrow \boxed{TOS}$

TOS $\leftarrow$ TOS - 1

NIS $\leftarrow$ NIS - 1

Word Description

Location	Description
2742_8	*Send TEMP1 to the MAR *Set Number Needed field to 1 *Test for TRAP2 a) PASS - jump to TRAP2 "fix it" routine at 5002_8 via the Next Address field b) FAIL - fetch the next sequential microinstruction
2743_8	*Write TOS register to memory *Decrement NIS and TOS *Fetch next microinstruction from FETCH1 routine at 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2742_8	2910	
	$I_{3-0} \leftarrow 0011$	CJP
	$TEST_{3-0} \leftarrow 1000$	Test for TRAP2
	$POL \leftarrow 0$	
	$S_{1,0} \leftarrow 00$	D inputs are 5002_8
	$NA \leftarrow 5002_8$	(TRAP2 "fix it" routine)
	ALU	
	$I_{5-0} \leftarrow 000011$	B plus 0
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_{1,0} \leftarrow 00$	Normal Mode
	$BSEL \leftarrow 0100$	TEMP1
	$NN_{2-0} \leftarrow 001$	NUMBER NEEDED = 1
	$I_{8-6} \leftarrow 000$	Result to Y outputs Don't store result
	BUS	
	$BS_{1,0} \leftarrow 01$	Result to MAR
	$LDMAR \leftarrow 1$	

Assembly

Location	Microinstruction	Comments
2743_8	2910	
	$I_{3-0} \leftarrow 0010$	JMAP
	$S_{1,0} \leftarrow 00$	Next Address is 1000_8
	$NA \leftarrow 1000_8$	
	ALU	
	$I_{5-0} \leftarrow 000011$	B plus 0
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_{1,0} \leftarrow 00$	Normal Mode
	$BSEL \leftarrow 0000$	TOS
	$D/\overline{U} \leftarrow 1$	Decrement TOS and NIS
	$EN\ TOS \leftarrow 1$	
	$EN\ NIS \leftarrow 1$	
	$I_{8-6} \leftarrow 000$	Result to Y Don't store result
	BUS	
	$BS_{1,0} \leftarrow 01$	Write result to memory
	$MEM\ SEL \leftarrow 1$	
	$R/\overline{W} \leftarrow 0$	

17.4f TRAP2 "fix it" Routine

Algorithm

$$
\begin{aligned}
\boxed{\text{BOS}} &\leftarrow \boxed{\text{MSP}} \\
\text{MSP} &\leftarrow \text{MSP} - 2 \\
\text{NIS} &\leftarrow \text{NIS} + 1
\end{aligned}
$$

Word Description

Location	Description
5002_8	*Send MSP to MAR *Subtract 2 from the MSP and store the result back in the MSP *Fetch the next sequential microinstruction
5003_8	*Read a value from Memory into the BOS register *Increment NIS *Fetch the next microinstruction from the address in the register/counter

Assembly

Location	Microinstruction	Comments
5002_8	2910	
	I_{3-0} ← 1110	CONTINUE
	ALU	
	I_{5-0} ← 001101	A minus D
	C_{IN} ← 0	"Carry in" = 1
	0/1 ← 1	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1110	MSP
	BUS ← 0	2 to D inputs
	$DS_{1,0}$ ← 00	
	Immediate ← 00000010	
	I_{8-6} ← 011	Result to MSP
	BSEL ← 1110	MSP to Y outputs
	BUS	
	$BS_{1,0}$ ← 01	MSP to MAR
	LDMAR ← 1	

Assembly

Location	Microinstruction	Comments
5003_8	2910	
	I_{3-0} ← 0111	JRP
	TEST ← 0000	TEST will FAIL
	POL ← 1	Next Address from register/counter
	ALU	
	I_{5-0} ← 000111	D plus 0
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	BUS ← 1	D inputs are from BUS
	I_{8-6} ← 010	Result to BOS
	$M_{1,0}$ ← 01	
	$D/\overline{U}$ ← 0	Increment NIS
	EN NIS ← 1	
	BUS	
	MEM SEL ← 1	Read from memory
	$R/\overline{W}$ ← 1	

17.5 Microcode for I/O instructions

17.5a The OUTPUT Macroinstruction

The following macroinstruction routine should be used to send data to an Output Device.

1. Push the Device Code onto the Stack
2. Check to see if the Output Device is busy
3. Busy ?

$$\begin{tikzcd} \text{2.} \\ \text{3. Busy ?} \arrow[u, bend left=60, "\text{Yes}"] \arrow[d, "\text{No}"] \\ \text{4.} \end{tikzcd}$$

4. Push the data value onto the Stack
5. Send the data to the Output Device

The fifth macroinstruction above is called the OUTPUT instruction. In this instruction, the data at the top of the Stack is sent to the device whose Device Code is one deep in the Stack. The data is then deleted from the Stack leaving the Device Code at the top of the Stack. The Op Code for this instruction is chosen to be 00001000.

Algorithm

```
OUTPUT DEVICE  ←  [TOS - 1]     ; Send Device Code
OUTPUT DEVICE  ←  [TOS]         ; Send Data
TOS  ←  TOS - 1
NIS  ←  NIS - 1
```

Word Description

Location	Description
2020_8	*Place TOS - 1 register on the BUS as a Device Code *Number Needed in the Fast Stack is 2 *TEST for TRAP2 a) PASS - Jump to the TRAP2 "fix it" routine via the NA field b) FAIL - fetch the next sequential micro-instruction
2021_8	*Place the TOS register on the BUS as data *Decrement the TOS and NIS counters *Fetch the next microinstruction from 1000_8 via the NA field (FETCH1 routine)

Assembly

Location	Microinstruction	Comments
2020_8	2910	
	I_{3-0} ← 0011	CJP
	TEST ← 1000	TEST for TRAP2
	POL ← 0	
	$S_{1,0}$ ← 00	D inputs = 5000_8
	NA ← 5002_8	(TRAP2 "fix it" routine)
	ALU	
	I_{5-0} ← 000011	B plus 0
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	BSEL ← 0001	TOS - 1
	NN_{2-0} ← 001	Number Needed = 1
	I_{8-6} ← 000	Result to Y outputs Don't store result
	BUS	
	$BS_{1,0}$ ← 01	Result to I/O as a Device Code
	I/O SEL ← 1	
	I/O STAT/$\overline{\text{DATA}}$ ← 0	

Assembly

Location	Microinstruction	Comments
2021_8	2910	
	I_{3-0} ← 0010	JMAP
	$S_{1,0}$ ← 00	Next address is 1000_8 (FETCH1 routine)
	NA ← 1000_8	
	ALU	
	I_{5-0} ← 000011	B plus 0
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	BSEL ← 0000	TOS register
	$D/\overline{U}$ ← 1	Decrement TOS and NIS
	EN TOS ← 1	
	EN NIS ← 1	
	I_{8-6} ← 000	Result to Y outputs, Don't store result
	BUS	
	$BS_{1,0}$ ← 01	Result to I/0 as data
	I/0 SEL ← 1	
	I/0 STAT/$\overline{\text{DATA}}$ ← 0	

17.5b The INPUT Macroinstruction

The following sequence of macroinstructions are required to read from an Input Device.

1. Push the Device Code onto the Stack.

$$\begin{tikzcd} \text{2. Check to see if the Input Device is Busy.} \\ \text{3. Busy ?} \arrow[u, bend left, "\text{Yes}"] \arrow[d, "\text{No}"] \\ \text{4. Read the data and Push it onto the Stack.} \end{tikzcd}$$

The fourth macroinstruction above is called the INPUT instruction. This instruction always assumes the Device Code to be at the top of the Stack. The value it reads from the Input device is PUSHed onto the top of the Stack leaving the Device Code one deep in the Stack. The Op Code for this instruction has been arbitrarily assigned a value of 00001001.

Algorithm

```
        TOS  ←  TOS + 1
        NIS  ←  NIS + 1
        INPUT DEVICE  ← [TOS - 1]   ; Send Device Code
       [TOS] ←  INPUT DEVICE        ; Receive Data
```

Word Description

Location	Description
2022_8	*Place TOS register on the Bus as a Device Code *Number Needed in Fast Stack = 1 *Test for TRAP2 a) PASS - Jump to 5002_8 via Next Address field b) FAIL - Fetch next sequential microinstruction
2023_8	*Read data from Input device into TEMP1 *Fetch next microinstruction from 4100_8 via the Next Address field *Load the register/counter in case of TRAP on next microinstruction
4100_8	*Transfer TEMP1 to the TOS + 1 register *Increment TOS and NIS *Test for NOT TRAP1 a) PASS - jump to 1000_8 via Next Address field (FETCH1 routine) b) FAIL - fetch next sequential microinstruction
4101_8	*Jump to 5000_8 via the Next Address field (TRAP1 "fix it" routine)

Assembly

Location	Microinstruction	Comments
2022_8	2910	
	$I_{3-0} \leftarrow 0011$	CJP
	$TEST_{3-0} \leftarrow 1000$	Test for TRAP2
	$POL \leftarrow 0$	
	$S_{1,0} \leftarrow 00$	5002_8 to D inputs
	$NA \leftarrow 5002_8$	(TRAP2 "fix it" routine)
	ALU	
	$I_{5-0} \leftarrow 000100$	A plus 0
	$C_{IN} \leftarrow 0$	"Carry in" = 0
	$0/1 \leftarrow 0$	
	$M_{1,0} \leftarrow 00$	Normal Mode
	$ASEL \leftarrow 0000$	TOS
	$NN_{2-0} \leftarrow 001$	Number Needed = 1
	$I_{8-6} \leftarrow 000$	Result to Y Don't store result
	BUS	
	$BS_{1,0} \leftarrow 01$	Result to I/O
	$I/O\ SEL \leftarrow 1$	device as a
	$I/O\ STAT/\overline{DATA} \leftarrow 0$	Device Code

Location	Microinstruction	Comments
2023_8	2910	
	I_{3-0} $\leftarrow$ 0010	JMAP
	$S_{1,0}$ $\leftarrow$ 00	Next Address is 4100_8
	NA $\leftarrow$ 4100_8	
	RLD $\leftarrow$ 1	Load register/counter
	ALU	
	I_{3-0} $\leftarrow$ 000111	D plus 0
	C_{IN} $\leftarrow$ 0	"Carry in" = 0
	I/O $\leftarrow$ 0	
	BUS $\leftarrow$ 1	D inputs from Bus
	I_{8-6} $\leftarrow$ 010	Result to TEMP1
	$M_{1,0}$ $\leftarrow$ 00	
	BSEL $\leftarrow$ 0100	
	BUS	
	I/O SEL $\leftarrow$ 1	Data from Input
	I/O STAT/$\overline{\text{DATA}}$ $\leftarrow$ 0	device onto Bus

Location	Microinstruction			Comments
4100_8	<u>2910</u>			
	I_{3-0}	←	0011	CJP
	$TEST_{3-0}$	←	0111	Test for NOT TRAP1
	POL	←	1	
	$S_{1,0}$	←	00	1000_8 to D inputs
	NA	←	1000_8	(FETCH1 routine)
	<u>ALU</u>			
	I_{5-0}	←	000100	A plus 0
	C_{IN}	←	0	"Carry in" = 0
	0/1	←	0	
	$M_{1,0}$	←	00	Normal Mode
	ASEL	←	0100	TEMP1
	I_{8-6}	←	010	Result to TOS + 1
	BSEL	←	0011	(i.e. TOS - 3)
	$\overline{U}$/D	←	0	Increment NIS
	EN NIS	←	1	and TOS
	EN TOS	←	1	
4101_8	<u>2910</u>			
	I_{3-0}	←	0010	JMAP
	$S_{1,0}$	←	00	Next Address is 5000_8
	NA	←	5000_8	(TRAP1 "fix it" routine)

17.6 The Conditional Jump Macroinstruction

The following microinstruction is designed to perform a "Jump on Zero Result" Macroinstruction. The format for this macroinstruction is,

OP CODE	n

where n is a relative value in the second byte of the IR. n can be either positive or negative. If it's negative, it must be represented in 2's complement form. The Op Code for this instruction has been arbitrarily chosen as 00000100.

Algorithm

Test the ZERO status bit

PASS - PC $\leftarrow$ PC + n
FAIL - PC $\leftarrow$ PC + 0

Word Description

Location	Description
2010_8	*Test the ZERO status bit *Sign Extend the second byte of the IR and add it to the PC; if the test fails add zero to the PC instead *Fetch the next microinstruction from 1002_8 via the Next Address field (FETCH 2 routine)

Assembly

Location	Microinstruction	Comments
2010_8	2910	
	I_{3-0} ← 0010	JMAP
	$S_{1,0}$ ← 00	Next Address is 1002_8
	NA ← 1002_8	(FETCH2 routine)
	$TEST_{3-0}$ ← 0100	Test ZERO status bit
	POL ← 0	
	ALU	
	I_{5-0} ← 000101	D plus A
	C_{IN} ← 0	"Carry in" = 0
	0/1 ← 0	
	$M_{1,0}$ ← 00	Normal Mode
	ASEL ← 1001	PC
	BUS ← 0	Second byte of IR to
	$DS_{1,0}$ ← 10	D inputs, sign extended
	SE ← 1	
	TE ← 1	Enable Test to force zeros at D inputs
	I_{8-6} ← 010	Result to PC
	BSEL ← 1001	

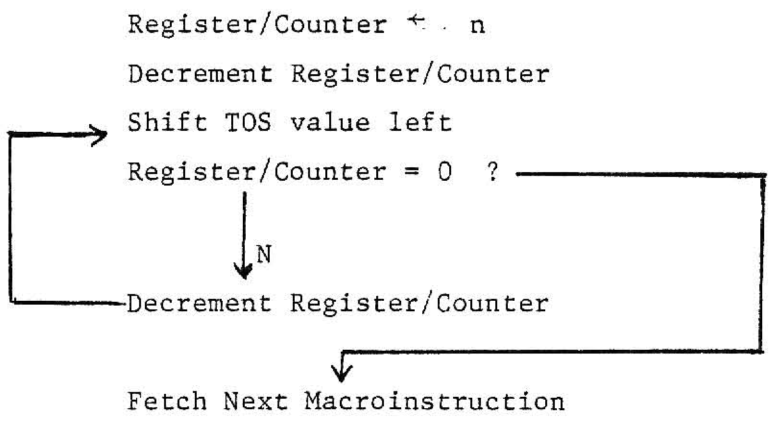
17.7 The Shift/Rotate Macroinstruction

The sequence of microinstructions developed below illustrate the procedure required to perform a left shift operation on the TOS register a total on n times. The format for this macroinstruction is as follows,

OP CODE	n

where n is a positive binary number in the second byte which specifies the number of shifts to be performed. The Op Code for this macroinstruction has been arbitrarily chosen as 00000010.

Algorithm


Register/Counter ← n
Decrement Register/Counter
Shift TOS value left
Register/Counter = 0 ?
N
Decrement Register/Counter
Fetch Next Macroinstruction

Word Description

Location	Description
2004_8	*Number Needed in Fast Stack = 1 *Test for NOT TRAP2 a) PASS - Jump to 6000_8 via the Next Address field b) FAIL - Fetch the next sequential microinstruction
2005_8	*Jump to 5002_8 via Next Address field (TRAP2 "fix it" routine)

Location	Description
6000_8	*Load register/counter from the second byte of the IR. *Fetch the next sequential microinstruction
6001_8	*Decrement the register/counter *Fetch the next sequential microinstruction
6002_8	*Shift the TOS register to the left *TEST for zero value of register/counter a) PASS - Fetch next sequential microinstruction b) FAIL - Repeat this microinstruction via Next Address field *Decrement the register/counter
6003_8	*Fetch the next microinstruction from 1002_8 via the Next Address field (FETCH2 routine)

Assembly

Location	Microinstruction	Comments
2004_8	2910	
	$I_{3-0} \leftarrow 0011$	CJP
	$TEST_{3-0} \leftarrow 1000$	Test for NOT TRAP2
	$POL \leftarrow 1$	
	$S_{1,0} \leftarrow 00$	6000_8 to D inputs
	$NA \leftarrow 6000_8$	
	ALU	
	$NN_{2-0} \leftarrow 001$	Number Needed = 1
2005_8	2910	
	$I_{3-0} \leftarrow 0010$	JMAP
	$S_{1,0} \leftarrow 00$	Next Address is 5002_8
	$NA \leftarrow 5002_8$	(TRAP2 "fix it" routine)
6000_8	2910	
	$I_{3-0} \leftarrow 1110$	COTNINUE
	$S_{1,0} \leftarrow 01$	Load register/counter from second byte of IR
	$RLD \leftarrow 1$	
6001_8	2910	
	$I_{3-0} \leftarrow 1001$	RPCT, (decrement register/counter)
	$S_{1,0} \leftarrow 00$	
	$NA \leftarrow 6002_8$	Next Address is always 6002_8

Location	Microinstruction			Comments
6002_8	<u>2910</u>			
	I_{3-0}	←	1001	RPCT
	$S_{1,0}$	←	00	6002_8 to D inputs
	NA	←	6002_8	
	<u>ALU</u>			
	I_{5-0}	←	000011	B plus 0
	C_{IN}	←	0	"Carry in" = 0
	0/1	←	0	
	$M_{1,0}$	←	00	Normal Mode
	BSEL	←	0000	TOS register
	I_{8-6}	←	110	Shift result left, store it in TOS register
	R	←	0	
	D	←	0	
6003_8	<u>2910</u>			
	I_{3-0}	←	0010	JMAP
	$S_{1,0}$	←	00	Next Address is 1002_8
	NA	←	1002_8	(FETCH2 routine)

REFERENCES

1 The HP3000 Systems Reference Manual.

2 Daniel P. Siewiorek, C. G. Bell, and Allen Newell, Computer Structures: Principles and Examples, McGraw-Hill Book Company Inc., 1982.

3 The Complete Motorola Microcomputer Data Library, 1978.

4 The TTL Data Book for Design Engineers, Second Edition, Texas Instruments Inc., 1976.

5 Hatachi IC Memories, Semiconductor Data Book, 1980.

6 Ivon Catt, David Walton, and Malcolm Davidson, Digital Hardware Design, The Macmillan Press Ltd., London and Basingstoke, 1979.

APPENDIX A

READ and WRITE Cycle Timing for Main Memory

Recall, both read and write operations are carried out by two micro-instructions, one to load the MAR, and the other for data transfer. The primary purpose of this appendix is to prove that these operations can be performed in the allotted time. To this end, a worst case analysis is developed for the following 3 operations.

1. Send an address to the MAR
2. Read data from memory
3. Write data to memory

Associated with each is a worst case microinstruction whose control and data signals encounter the longest path of delay possible to carry out the operation. This path has been carefully sought out for each operation and is presented in the following material.

Timing specifications for the various circuit components are provided in the references. Logic delays encountered by the 7400 series of chips have been taken from [4] . Timing specifications for the 2901A can be found in [3] , and timing diagrams for the 6116 in [5]. Delays associated with component interconnections and the Bus have been excluded from this analysis. These delays depend a great deal on the physical lay-out of the system as well as the various impedance matching techniques employed. A detailed presentation of this material is beyond the scope of this thesis. However, an excellent discussion of the same can be found in [6].

A.1 Send an address to the MAR

The longest path encountered when sending a value to the MAR occurs when this value is the result of an arthmetic operation in the ALU involving at least one of the sixteen 2901 registers. Initially, an operand register must be selected by signals propagated through the Stack Tender/Register Select logic. This is followed by the arithmetic operation, whose speed is enhanced somewhat by the 74S182 Carry Look-ahead Generator. Once the address is available at the Y outputs it is transferred to the MAR. Recall, the MAR works as a latch. Once loaded, the MAR supplies the A_{10} - A_0 address inputs to the 6116s. Shortly after this the Chip Select signal for the selected pair of 6116s comes true. Data access time for the 6116s actually begins once the Chip Select comes true. The following analysis shows that in the worst case the Chip Select will be valid 201.5ns (plus the delays associated with the BUS) into the microinstruction.

Path	Maximum Delay (ns)	Figure Reference
1. FROM: Rising edge of $\emptyset_1$, TO: ASEL & BSEL outputs of the PLR		
PLR - 74S374	18	7-18
2. FROM: ASEL and/or BSEL outputs of the PLR TO: S3, S2, and M controls on 74S181 ALU chip in the Stack Tender circuit		
NOR - 74S02	5.5	
INVERTER - 74S04	5	7-18

Path	Maximum Delay (ns)	Figure Reference
3. FROM: S3, S2, & M inputs on the 74S181 TO: F outputs of 74S181 (i.e. the Mode Multiplexer inputs)	22	7-18
4. FROM: Mode Multiplexer inputs TO: Mode Multiplexer outputs (i.e. A and/or B register select inputs) 2:1 MUX - 74S157 (7.5ns) 4:1 MUX - 74S153 (9ns)	9	7-18
5. FROM: A and/or B register select inputs on 2901 TO: $\bar{G}$, $\bar{P}$ outputs of 2901 (i.e. $\bar{G}$, $\bar{P}$ inputs to 74S182)	59	7-6
6. FROM: $\bar{G}$, $\bar{P}$ inputs to 74S182 TO: C_{n+x}, C_{n+y}, & C_{n+z} outputs of 74S182, (i.e. the C_n inputs to the 2901's)	7	7-6
7. FROM: C_n inputs to 2901's TO: Y outputs of 2901's, (i.e. MAR inputs via the BUS)	27	7-6
8. FROM: MAR inputs TO: MAR outputs, (i.e. decoder inputs) MAR - 74S373	13	9-2
9. FROM: Decoder inputs TO: Decoder outputs, (i.e. CS on appropriate 6116 pair)	36	9-2
Total delay to CS on selected pair of 6116s (access time begins)	201.5ns	

A.2 Read Data from Memory

To complete a read operation, the second microinstruction transfers data from memory to the ALU during periods 2 and 3. From the analysis in the previous section it is apparent that the address and chip select signals to the 6116s come true long before this microinstruction begins ($\approx$ 100ns). The Output Enable, $\overline{OE}$, control on these chips is a function of the R/$\overline{W}$ line. Not knowing the state of the R/$\overline{W}$ line during the first microinstruction we will assume it to have been low so that $\overline{OE}$ is not activated until this microinstruction. Proper part selection (Schottky) will ensure that $\overline{OE}$ goes low a maximum of 23ns into this microinstruction. Thus, the memory data is available 123ns into the cycle (100ns later). Due to the delays associated with the A, B, and C transceivers, the data need not be available until approximately 33ns into the second period (133ns into the cycle). Thus, the data from RAM is available slightly ahead of time. Therefore, the transceivers play the most important role in determination of the longest path for placing read data on the Bus.

The worst case microinstruction for a read operation occurs when the read data serves as an ALU operand for an arithmetic function whose result is first shifted and then stored into the selected B register. The following analysis shows that in the worst case this operation takes 194.5ns from the beginning of the second period. This leaves only 5.5ns for Bus delays, etc.

Path	Maximum Delay (ns)	Figure Reference
1. FROM: The rising edge of $\overline{\phi}_1$ TO: $\overline{G}$ enable on transceiver in main memory		
INVERTER - 74S04	5	
2 INPUT NOR - 74S02	5.5	
2 INPUT NAND - 74S00	5	9-2
2. FROM: $\overline{G}$ enable on transceivers TO: Transceiver output onto BUS (i.e. the Multiplexer inputs in the Data Select/Sign Extend circuit)		
TRANSCEIVERS - 74LS245	40	9-2
3. FROM: High byte multiplexer input TO: High byte multiplexer output		
4:1 MUX - 74S153	9	7-22
4. FROM: msb output of high byte Multiplexer TO: Input of low byte multiplexer for sign extension		
3 INPUT AND - 74S11	7.5	7-22
5. FROM: Low Byte Multiplexer input TO: Low Byte Multiplexer output, (i.e. D inputs to 2901)		
2:1 MUX - 74S157	7.5	7-22
6. FROM: D inputs on 2901 TO: $\overline{G}$, $\overline{P}$ outputs on 2901 (i.e. $\overline{G}$, $\overline{P}$ inputs to 74S182)	31	7-6

Path	Maximum Delay (ns)	Figure Reference
7. FROM: $\bar{G}$, $\bar{P}$ inputs on 74S182 TO: C_{n+x}, C_{n+y}, C_{n+z} outputs on 74S187, (i.e. C_n inputs on 2901s)	7	7-6
8. FROM: C_n inputs on 2901 TO: RAM_0 or RAM_3 shift outputs, (i.e. inputs to the shift multiplexers)	45	7-6
9. FROM: Inputs to shift multiplexers TO: Outputs of shift multiplexers, (i.e. RAM_0 or RAM_3 shift inputs) 8:1 MUX - 74S251	12	7-6
10. FROM: RAM_0 or RAM_3 inputs TO: Result loaded in A or B register (taken from $RAM_{0,3}$ set up time)	20	7-6
TOTAL DELAY	194.5	

A.3 Write Data to Memory

The completion of the write operation by the second microinstruction involves the following control sequence for the 6116s.

1. The $\overline{OE}$ input is pulled high shortly after the beginning of the first period to place the 6116 data lines in the high impedance state.
2. The $\overline{WE}$ line is pulled low at the beginning of the second period to enable the write operation. Data is also supplied to the 6116 inputs at this time via the A, B, and C transceiver groups.
3. The $\overline{WE}$ signal is returned high at the end of the third period to complete the write operation. The transceivers are also disabled at this time. $\overline{OE}$ will probably remain high (unless the next microinstruction sets R/$\overline{W}$ to 1, which is possible but unlikely.

Timing specifications for the 6116 require the $\overline{OE}$ signal to remain high for at least 10ns after $\overline{WE}$ returns high. From Figure 9-2, the $\overline{WE}$ will return high a maximum of 5ns after $\overline{\emptyset}_1$ goes low (assuming Schottky parts are used). The $\overline{OE}$ signal experiences delays through the Pipeline Register, over the Bus, and through an additional level of logic which will most likely add up to at least 15ns. If this is not the case, it does not appear to be a big restriction to require that the following microinstruction maintain the R/$\overline{W}$ line low. It is unacceptable to have a read data transfer follow directly after a microinstruction performing a write data transfer.

The 6116 also requires the data to remain valid at least 10ns after $\overline{WE}$ is returned high. Due to propagation delays encountered by the control signals in the CPU, the data will remain valid on the Bus for at least this long. In addition, the A, B, and C transceivers remain enabled for approximately two gate delays after WE is returned high. This, coupled with the "turn off" time for these devices, will maintain valid data at the 6116 inputs for the required length of time.

Now that verification of the 6116 operation has been established, it remains to be shown that the data can be supplied to the 6116 inputs in time to complete the write operation. The worst case write microinstruction occurs when the write data is a result of an arithmetic operation performed on a selected A and/or B register. The following analysis verifies that this operation is easily completed within the allotted time. In fact, the write is actually complete approximately 67.5ns before the end of the cycle.

Path	Maximum Delay (ns)	Figure Reference
1. FROM: Rising edge of ϕ_1		
TO: ASEL and/or BSEL outputs of PLR		
PLR - 74S374	18	7-18
2. FROM: ASEL and/or BSEL outputs of PLR		
TO: S3, S2, and M controls on 74S181 ALU chip in the Stack Tender circuit		
NOR - 74S02	5.5	7-18
INVERTER - 74S04	5	
3. FROM: S3, S2, and M inputs on the 74S181		
TO: F outputs of 74S181, (i.e. Mode Multiplexer inputs)	22	7-18
4. FROM: Mode Multiplexer inputs		
TO: Mode Multiplexer outputs, (i.e. A and/or B register select inputs to 2901's)		
2:1 MUX - 74S157 (7.5ns)		
4:1 MUX - 74S153 (9ns)	9	7-18

Path	Maximum Delay (ns)	Figure Reference
5. FROM: A and/or B register select inputs on 2901s TO: $\bar{G}$, $\bar{P}$ outputs of 2901s, (i.e. $\bar{G}$, $\bar{P}$ inputs to 74S182)	59	7-6
6. FROM: $\bar{G}$, $\bar{P}$ inputs to 74S182 TO: C_{n+x}, C_{n+y}, C_{n+z} outputs of 74S182, (i.e. C_n inputs to 2901s)	7	7-6
7. FORM: C_n inputs on 2901s TO: Y outputs of 2901s, (i.e. transceiver inputs in main memory via the Bus)	27	7-6
8. FROM: Transceiver inputs TO: Transceiver outputs, (i.e. inputs to 6116s) TRANSCEIVERS - 74LS245	40	9-2
9. FROM: Inputs to 6116s TO: Stored data in 6116s t_{DW} on 6116 (minimum)	40	9-2
TOTAL DELAY	232.5ns	

APPENDIX B

Console Operation with the CS and Main Memory

The Console Device communicates with the rest of the computer via the Console Interface circuit presented in section 14.2. This interface is controlled by microprocessor software through two 8-bit ports, Port A (the control port), and Port B (the data port). The purpose of this appendix is to provide word descriptions of the microprocessor routines required to carry out desired Console operations. To this end, the following routines are described.

1. Initialization (at power up)
2. Halt the CPU
3. Single Step
4. Activate the Console Device
5. Load the CSAR
6. Read a value from the CS
7. Write a value to the CS
8. Load the PLR
9. Load the MAR
10. Read a value from Main Memory
11. Write a value to Main Memory.

Note, the following descriptions are independent of the type of microprocessor used.

B.1 Initialization (at power up)

Operation	Comments
1. Set up Port A as an Output Port and send 0 to it.	Make sure no operation is enabled by the decoders.
2. Set up CA2 and CB2 in the programmed control mode.	
3. Initialize Port B as desired.	

B.2 Halt the CPU

Operation	Comments
1. Set up Port B as Output Port.	
2. Send 010_8 to Port B	Prepare to set the $\overline{\text{HALTRQ}}$ bit in CCR
3. Send 060_8 to Port A	Activate load enable on CCR
4. Toggle CB2	Activate the $\overline{\text{HALTRQ}}$ bit
5. Send 000_8 to Port A	Put interface in the idle state

B.3 Single Step

Assumption:

CCR = 0000

Operation	Comments
1. Send 010_8 to Port A	Prepare to activate SS flip-flop
2. Toggle CB2	Activate SS flip-flop
3. Send 000_8 to Port A	Put interface in idle state

B.4 Activate CONSOLEON

Assumption: CCR = 0000

Operation	Comments
1. Set up Port B as an Output Port	
2. Send 014_8 to Port B	Prepare to set CONSOLEON bit in CCR (also maintains the $\overline{\text{HALTRQ}}$ bit at 0)
3. Send 060_8 to Port A	Activate load enable on CCR
4. Toggle CB2	Load CCR, set the CONSOLEON bit (& maintain $\overline{\text{HALRQ}}$ at 0)

B.5 Load the CSAR

Operation	Comments
1. Set up Port B as an Output Port.	
2. Send the low byte of the desired address to Port B	Prepare to load the low byte of the CSAR
3. Send 003_8 to Port A	Activate load enable on the low byte CSAR
4. Toggle CB2	Load the low byte CSAR
5. Send the high byte of the desired address to Port B	Prepare to load the high byte CSAR
6. Send 004_8 to Port A	Activate load enable on the high byte CSAR
7. Toggle CB2	Load the high byte CSAR
8. Send 000_8 to Port A	Place the interface in the idle state

B.6 Read a 16-bit value from the CS

Assumptions:

CSAR contains desired address

CCR = 0100

Operation	Comments
1. Set up Port B as an Output Port	
2. Send 015_8 to Port B (00001101_2)	Prepare to set the CSR/$\overline{W}$ bit to 1
3. Send 060_8 to Port A	Activate load enable on CCR
4. Toggle CB2	Load CCR, set the CSR/$\overline{W}$ bit indicating a Read operation
5. Send 017_8 to Port B (00001111_2)	Prepare to set the CSRWEN bit
6. Toggle CB2	Set the CSRWEN bit to enable data from CS onto the Bus
7. Send 006_8 to Port A	Activate load enable on CIR
8. Toggle CB2	The CIR is loaded with the 16-bit data value from the CS
9. Send 015_8 to Port B (00001101)	Prepare to clear the CSRWEN bit in the CCR
10. Send 060_8 to Port A	Activate Load enable on CCR

Operation	Comments
11. Toggle CB2	Clear CSRWEN bit in CCR, this takes the CS data off the Bus
12. Make Port B an Input Port	
13. Send 040_8 to Port A	Enable the low byte output of the CIR
14. Input the value from the low byte of the CIR to Port B	
15. Transfer value at Port B to microprocessor memory for storage	
16. Send 050_8 to Port A	Enable high byte output of CIR
17. Input the value from the low byte of the CIR to Port B	
18. Transfer the value at Port B to microprocessor memory for storage	
19. Send 000_8 to Port A	Place interface in the idle state

B.7 Write a 16-bit value to the CS

Assumptions:

CCR = 0100

CSAR has desired address

Operation	Comments
1. Set up Port B as an Output Port	
2. Send low byte of data to Port B	Prepare to load the low byte of the COR
3. Send 001_8 to Port A	Activate load enable on low byte COR
4. Toggle CB2	Load low byte COR with data from Port B
5. Send high byte of data to Port B	Prepare to load the high byte of the COR
6. Send 002_8 to Port A	Activate load enable on high byte COR
7. Toggle CB2	Load high byte COR with data from Port B
8. Send 016_8 to Port B (00001110_2)	Prepare to set CSRWEN bit in CCR for the transfer of data for a write operation
9. Send 065_8 to Port A	Activate load enable on CCR, and place contents of COR on the Bus
10. Toggle CB2	Load CCR, set CSRWEN bit for a write data transfer

Operation	Comments
11. Send 014_8 to Port B (00001100)	Prepare to clear CSRWEN to complete the write operation
12. Toggle CB2	Clear the CSRWEN bit in the CCR (write operation complete)
13. Send 000_8 to Port A	Place interface in idle state

B.8 Load the PLR

Assumptions:

CCR = 0100

CSAR holds desired address

Operation	Comments
1. Set up Port B as an Output Port	
2. Send 015_8 to Port B (00001101)	Prepare to set the CSR/$\overline{W}$ bit to 1 to indicate a read from the CS
3. Send 060_8 to Port A	Activate load enable on the CCR
4. Toggle CB2	Load the CCR, set CSR/$\overline{W}$ bit to 1
5. Set CA2 low	CA2 functions as $\phi_{1(PLR)}$ in the following operation
6. Send 030_8 to Port A	This allows $\phi_{1(PLR)}$ to be applied from CA2
7. Toggle CA2	Load the PLR with the value read from the CS
8. Send 000_8 to Port A	Put interface in idle state

B.9 Send a value to the MAR

Assumptions:

CCR = 0100

Operation	Comments
1. Make Port B an Output Port	
2. Send the low byte of the desired address to Port B	Prepare to load the low byte of the COR
3. Send 001 to Port A	Activate load enable on the low byte COR
4. Toggle CB2	Load the low byte COR
5. Send the high byte of the desired address to Port B	Prepare to load the high byte of the COR
6. Send 002 to Port A	Activate load enable on the high byte COR
7. Toggle CB2	Load the high byte COR
8. Send XXXXXX11 to Port B	Prepare to apply the memory Bus controls LDMAR $\leftarrow$ 1 ($\overline{\text{MEM SEL}} \leftarrow$ 1)
9. Make Sure CA2 is low	CA2 will act as $\emptyset_1$ in this operation
10. Send 025_8 to Port A	Place COR on the Bus, and apply memory controls to the Bus

Operation	Comments
11. Toggle CA2	This will load the MAR
12. Send 000 to Port A	Put interface in the idle state

B.10 Read from Main Memory

Assumptions:

CCR = 0100

MAR holds desired address

Operation	Comments
1. Set up B as an Output Port	
2. Send XXXX0100 to Port B	Prepare to apply memory Bus controls $\overline{\text{MEM SEL}} \leftarrow 0$ $R/\overline{W} \leftarrow 1$ $\text{BYTE} \leftarrow 0$ $(\text{LDMAR} \leftarrow 0)$
3. Make sure CA2 is low	CA2 functions as $\overline{\phi}_1$ in this operation
4. Send 026_8 to Port A	Apply Memory Controls, and Activate the load enable on the CIR
5. Set CA2 to the high state	This will place the data from Memory on the Bus
6. Toggle CB2	Load the CIR from the Bus
7. Return CA2 to the low state	Takes data off the Bus
8. Send 000 to Port A	Safe State
9. Make Port B an Input Port	

Operation	Comments
10. Send 040_8 to Port A	Enable the low byte CIR output to be transferred to Port B
11. Read a value from the low byte CIR to Port B	
12. Transfer the data in Port B to the microprocessor memory	
13. Send 050_8 to Port A	Enable the high byte CIR output to be transferred to Port B
14. Read a value from the high byte CIR to Port B	
15. Transfer the data from Port B to the microprocessor memory	
16. Send 000 to Port A	Put interface in the idle state

B.11 Write a value to Main Memory

Operation	Comments
1. Set up Port B as an Output Port	
2. Send the low byte of the write data to Port B	Prepare to load the low byte COR
3. Send 001_8 to Port A	Activate the load enable on the low byte COR
4. Toggle CB2	Load the low byte COR with the low byte write data
5. Send the high byte of the write data to Port B	Prepare to load the high byte COR
6. Send 002_8 to Port A	Activate load enable on the high byte COR
7. Toggle CB2	Load the high byte COR with the high byte write data
8. Send 000_8 to Port B	Prepare to apply memory Bus Controls $\overline{\text{MEM SEL}} \leftarrow 0$ $\text{R/W} \leftarrow 0$ $\text{BYTE} \leftarrow 0$ $(\text{LDMAR} \leftarrow 0)$
9. Make sure CA2 is low	CA2 acts as $\emptyset_1$ in this operation
10. Send 025_8 to Port A	Enable the COR outputs onto the Bus, and apply the Memory Control signals

Operation	Comments
11. Toggle CA1	Write the data into Memory
12. Send 000_2 to Port A	Put interface in the idle state

THE DESIGN PROPOSAL OF A 16-BIT MICROPROGRAMMED STACK MACHINE

by

DON RHEA HUSH

B. S., Kansas State University, 1980

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

ABSTRACT

This paper presents the complete design of a 16-bit computer system. All parts of the computer are designed with general purpose hardware which can be readily obtained. Since the design requires no special hardware, the computer can be assembled and tested by most anyone with a basic understanding of computer architecture and digital design. Once built, it can provide an excellent educational tool in such areas as microprogramming, machine instructions sets, and computer design.

The Control Unit for this machine is microprogrammed. It features a writable Control Store with 4K locations, each 88 bits wide. The heart of the Control Unit is the 2910, a microprogram sequencer from the 2900 family of microcomputer logic. The microinstruction cycle time for this computer is 300ns.

The ALU is built around the 2900, a 4-bit slice with 16 general purpose registers and a multifunction ALU. Four of these registers function as a Fast Stack which is like a cache memory for the top 4 entries in the Stack. This operation can be disabled allowing the computer to function as a register machine.

Main Memory contains 64K bytes (32K words) of storage. Memory is accessed via a 16 bit Bus which carries both address and data. Data can be accessed in both word (16 bit) and byte modes.

I/O is implemented as Direct or Programmed I/O. All I/O operations are non-memory mapped and are carried out under CPU control. I/O devices are capable of interrupting the CPU from one of sixteen interrupt levels. Interrupts are serviced on a priority basis determined by a hardware Daisy Chain.

The Console Device has ultimate control of the computer. It can halt and single step the CPU at the microinstruction level. It can also communicate directly with the Control Store and Main Memory.

While this computer contains few, if any, special features, it does have all the parts required for a complete system. This, coupled with its simplicity in design, gives the system strong potential as an educational tool.