Signetics

8X305 Users Manual

8X305 Users Manual

June 1982

Signetics

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

This manual provides the system designer with a complete technical discussion of the Signetics 8X305 Bipolar MicroController. The first two chapters address the functional operation of the 8X305, Chapter 3 is a reference for the device instruction set, Chapter 4 discusses timing considerations, and the final two chapters deal with application of the device.

The 8X305 Data Sheet provides complementary data to this manual, including detailed timing and electrical characteristics. The 8X300 Family Product Capabilities Manual discusses the 8X305 in the context of the many compatible support devices available from Signetics. Together, the three documents provide the information necessary to design and implement a system that takes full advantage of the powerful features of the 8X305.

The Signetics 8X305 Bipolar MicroController provides a real alternative to the complexity of bit-slice designs and the relatively slow speed of MOS microprocessors in high performance, cost effective control systems.

1.1 DEVICE DESCRIPTION

The 8X305 MicroController (Figure 1-1) is a monolithic Central Processing Unit implemented in bipolar Schottky technology. It is designed to operate at a speed of 200 nsec for each 16-bit instruction, fetched on a dedicated bus for higher throughput. It controls a series of peripheral devices which are attached to it by means of a standard 8-bit bus known as the Interface Vector bus and its associated control signals. The 8X305 can be easily integrated into most support systems using 8X300 Family support devices.

The 8X305 is upward-compatible with its predecessor, the 8X300, allowing enhancement of existing MicroController systems. Software written for the 8X300 will function correctly on an 8X305, but the expansion of the instruction set and internal working storage allows more flexible manipulation of data, higher throughput, and simplification of code. Care should be taken, however, in analyzing signal and timing requirements for each application where the MicroController is to be updated.

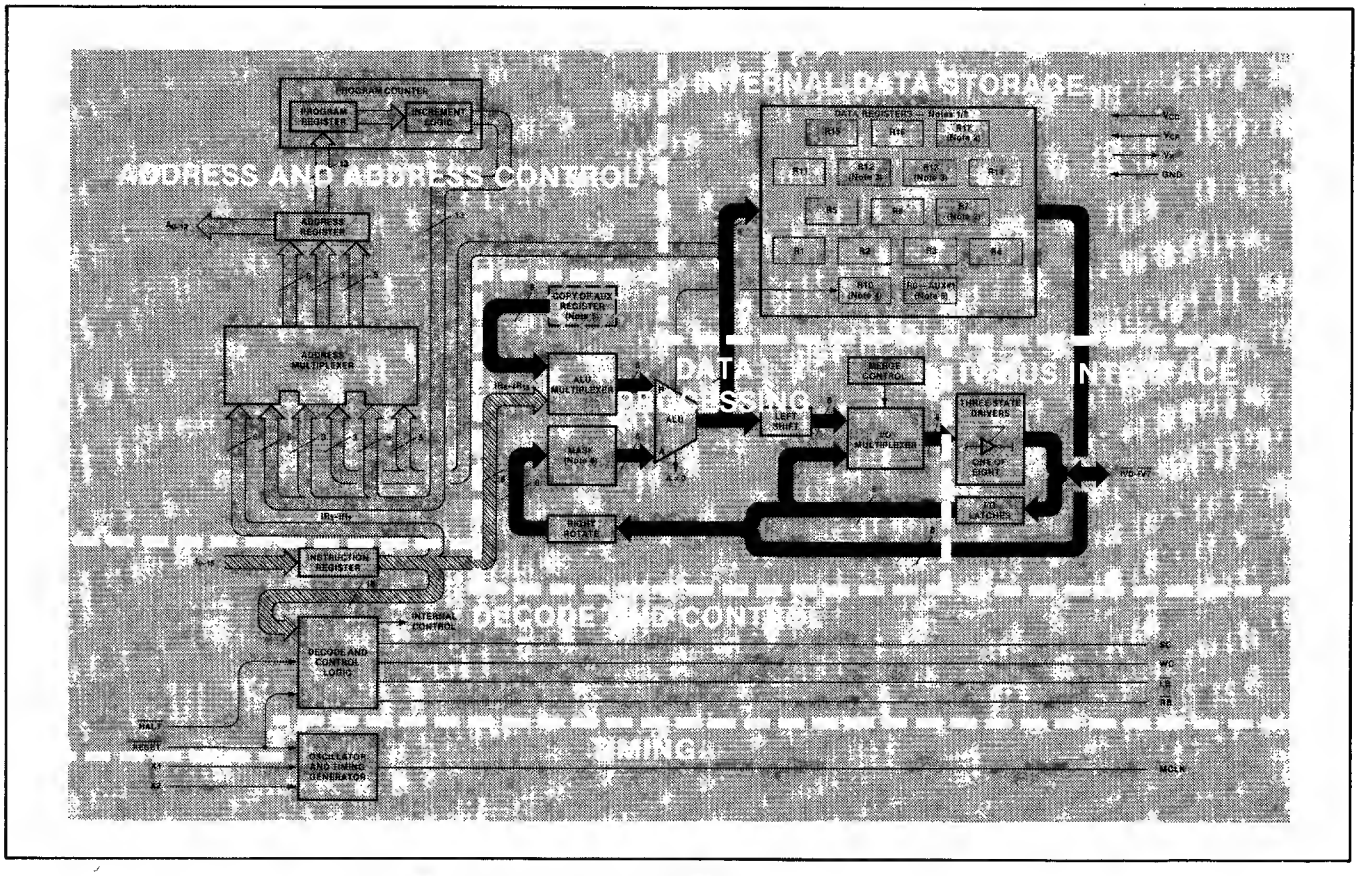


Figure 1-1. Architectural Overview of 8X305 MicroController

The 8X305 is designed to provide the optimum combination of features for controller design:

- Powerful, simple instruction set
- Eight instruction classes
- Single chip package
- Bipolar speed
- Family of compatible peripheral devices
- Flexible bit manipulation in a single instruction
- Single +5 volt supply
- TTL three-state bus operation

1.2 DEVICE ARCHITECTURE

An understanding of the internal architecture of the 8X305 is required to maximize the efficiency of a design. Figure 1-1 illustrates the logical structure, but does not necessarily represent exact physical connections within the device.

The instruction arrives at the Instruction Register from the Instruction Bus (I_0–I_{15}). It is interpreted on the basis of the Op Code which defines the significance of the other bits in the instruction. Data paths within the chip are set up by the Decode and Control logic. External control signals are also generated by this logic. At a later point in the cycle, the Program Counter, Increment Logic, and Address Multiplexer generate the address of the next instruction to be executed and place it in the Address Register. The address is then placed on the Instruction Address Bus (A_0–A_{12}) to fetch the next instruction.

All timing is generated by an on-chip oscillator running at twice the actual instruction cycle speed.

Source data can be accessed from three locations:

- 16 internal registers
- The IV bus
- Absolute or modified constant specifications from the current instruction word

Data from external sources can be manipulated by means of the Rotate and Mask Logic before becoming the first operand for an ALU operation. The implied second operand is the Auxiliary Register (AUX or R0). The result of the operation is stored in an Internal Register or transmitted to the IV bus.

The sixteen 8-bit registers contained in the 8X305 are used as temporary storage of data and pointers. Three of these registers have special applications for IV bus address transmission and flag storage, leaving thirteen available as general-purpose storage.

1.3 PIN DESCRIPTION

The 8X305 MicroController is housed in a 0.9-inch wide, 50-pin Dual In-Line (DIP) package, with pin assignments and designated functions as indicated in Figure 1-2.

1.4 THE INTERFACE VECTOR BUS

The 8X305 communicates with peripheral devices by means of a bidirectional, 8-bit TTL bus known as the Interface Vector, or IV bus. Five control signals generated by the 8X305 indicate the direction in which the bus is being driven and the configuration of the data or address on the bus.

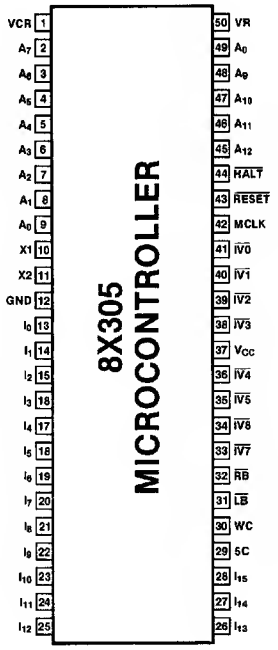
Devices connected to the IV bus are commonly referred to as I/O Ports. Any one of up to 256 such devices can be selected in a single cycle when the 8X305 places the I/O Port's unique address on the bus and asserts the Select Command (SC) signal. Once selected, an I/O Port normally remains selected until the SC signal is again asserted with a different address on the bus.

The direction of data flow is indicated by the Write Command (WC) signal. This signal is asserted when data is being placed on the bus by the 8X305, and is not asserted when data is being read from the bus. The data will normally be read from or written into whatever I/O Port was last selected.

To optimize the 8X305's capabilities in data manipulation and device selection, two control signals known as Left Bank ($\overline{LB}$) and Right Bank ($\overline{RB}$) are provided. These signals are asserted concurrently with other control signals based on the contents of the instruction word. They can be used in conjunction with the other signals to determine which I/O Port will be enabled at any given time. They can be used in a variety of ways, but the two most significant are as follows:

1. The optimum performance of the MicroController can be achieved by connecting the input I/O Port to one bank and the output port to the other. The two ports can then be selected concurrently and data can pass between them in a single cycle, resulting in a transfer rate of five megabytes per second.
2. Since the bank select signal ($\overline{LB}$ or $\overline{RB}$) must be asserted for a port to be enabled, two ports may share the same address provided that they are connected to opposite banks. This expands the number of I/O Ports that can be addressed in a single cycle to 512.

Timing on the bus is synchronized with the Master Clock (MCLK) signal. Together with the other control signals, it enables the I/O Ports to access the IV bus at the correct points in the MicroController's instruction cycle.



PIN NO.	IDENTIFIER	FUNCTION
1	VCR	Regulated voltage input from series-pass transistor (2N5320 or equivalent).
2-9, 45-49	A_0-A_{12}	**Program Address Lines:** These active-high outputs permit direct addressing of up to 8192 words of program storage; A_{12} is least significant bit.
10, 11	X1, X2	Timing generator connections for a capacitor, a series resonant crystal, or an external clock source with complementary outputs.
12	GND	Ground.
13-28	I_0-I_{15}	**Instruction Lines:** These active-high input lines receive 16-bit instructions from program storage; I_{15} is least significant bit.
29	SC	**Select Command:** When high (binary 1), an address is being output on pins $\overline{IV0}$ through $\overline{IV7}$.
30	WC	**Write Command:** When high (binary 1), data is being output on pins $\overline{IV0}$ through $\overline{IV7}$.
31	$\overline{LB}$	**Left Bank Control:** When low (binary 0), devices connected to the Left Bank are accessed. (Note. *Typically, the $\overline{LB}$ signal is tied to the $\overline{ME}$ input pin of I/O peripherals.)*
32	$\overline{RB}$	**Right Bank Control:** When low (binary 0), devices connected to the Right Bank are accessed (Note. *Typically, the $\overline{RB}$ signal is tied to the $\overline{ME}$ input pin of I/O peripherals.)*
33-36, 38-41	$\overline{IV0}$-$\overline{IV7}$	Interface Vector (Input/Output Bus) — these bidirectional active-low three-state lines communicate data and/or addresses to I/O devices and memory locations. A low voltage level equals a binary "1"; $\overline{IV7}$ is Least Significant Bit.
37	V_{CC}	+5V power supply.
42	MCLK	**Master Clock:** This active-high output signal is used for clocking I/O devices and/or synchronization of external logic.
43	$\overline{RESET}$	When $\overline{RESET}$ input is low (binary 0), the 8X305 is initialized — sets Program Counter/Address Register to zero and inhibits MCLK. For the period of time $\overline{RESET}$ is low, the Left Bank/Right Bank ($\overline{LB}$/$\overline{RB}$) signals are forced high asynchronously.
44	$\overline{HALT}$	When $\overline{HALT}$ input is low (binary 0), internal operation of the 8X305 stops at the start of next instruction; MCLK is not inhibited nor is any internal register affected; however, both the Left Bank/ Right Bank ($\overline{LB}$/$\overline{RB}$) signals are synchronously driven high during the first quarter of the instruction cycle time and remain high during the time $\overline{HALT}$ is low.
50	VR	Internally-generated reference output voltage for external series-pass regulator transistor.

Figure 1-2. 8X305 Pin Designations and Functions

The I/O Ports must be capable of performing the address select operation, determining the direction of the bus, analyzing the bank control signals, and presenting or receiving data. Signetics offers numerous circuits that perform these functions for specific applications. Refer to the 8X300 Family Product Capabilities Manual for information on these devices.

1.5 SYSTEM ENVIRONMENT

A generalized system configuration consisting of program storage (ROM/PROM), working storage (RAM), I/O devices, and the IV bus interface ($\overline{IV0}$-$\overline{IV7}$) is shown in Figure 1-3. Except for the off-chip timing crystal, the regulator transistor, and any user-generated logic associated with the $\overline{HALT}$ and $\overline{RESET}$ lines, no external parts are required for implementation. The TTL-compatible bus, simple interface logic, and bit-manipulation features optimize the 8X305 MicroController for almost any system where high-speed operation, flexibility, and minimum board space are required.

The program storage can consist of any ROM or PROM with sufficient access speed. It can be configured to the size of program required by the application, up to a maximum of 8,192 instruction words.

An Interrupt Control Coprocessor, the 8X310, is available. This device connects to the Instruction Bus and Instruction Address Bus, providing a single-chip interrupt handler and an enhanced ability for subroutine processing.

Ports on the IV bus must be tailored to the application. Some additional high-speed working storage may also be required. This is implemented by assigning contiguous memory locations in a small byte-wide RAM to a series of contiguous port addresses. It is good practice to pair peripherals which transfer a good deal of data between each other on opposite banks, which enables single-instruction processing and transfer of data.

Since they have been designed specifically for the IV bus, the 8X300 Family of parts provide the simplest solution to most requirements encountered in an 8X305 system. The family includes an array of I/O Ports, working storage RAM, and single-chip solutions to problems such as floppy disk control and computer bus interfacing.

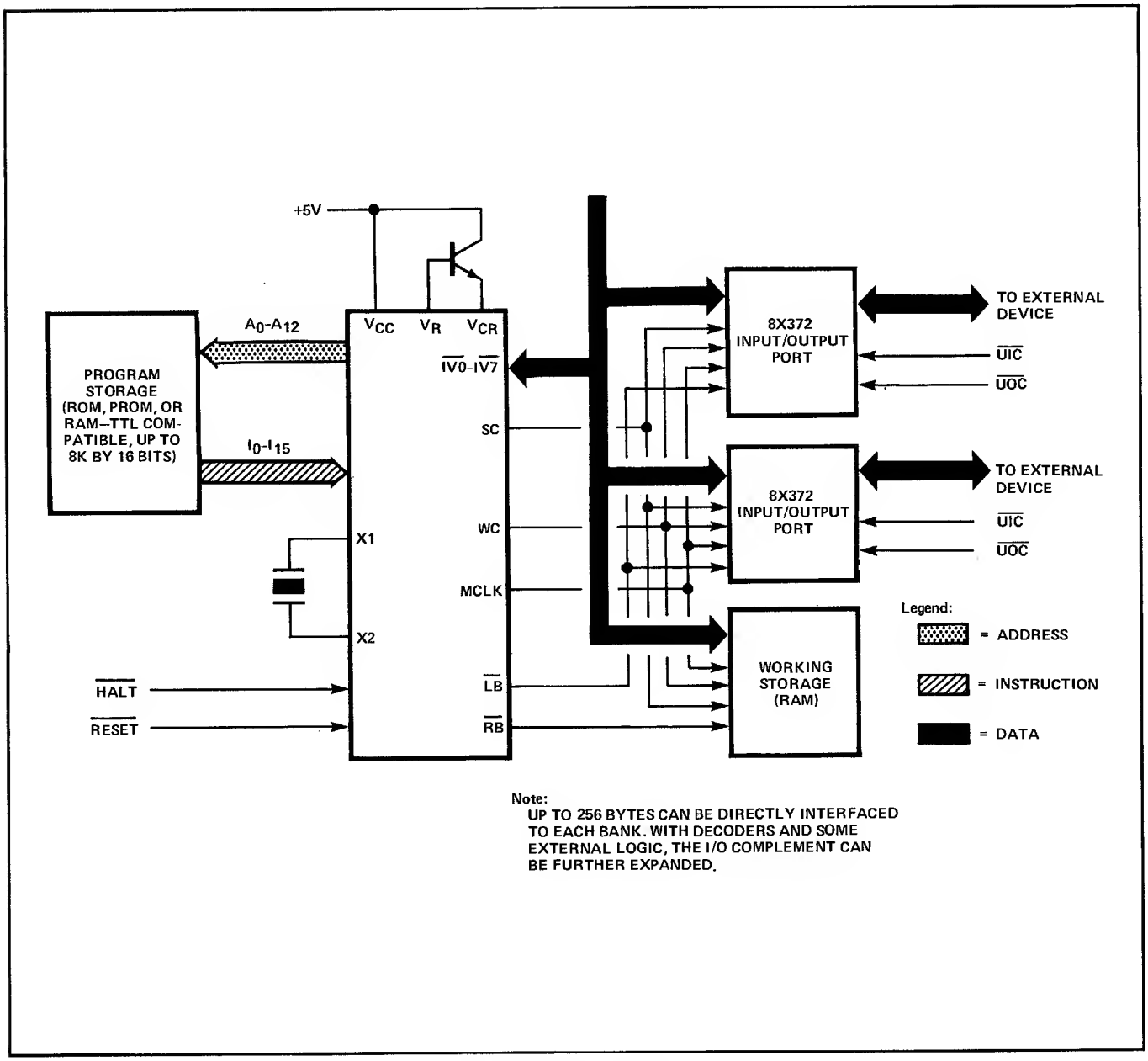
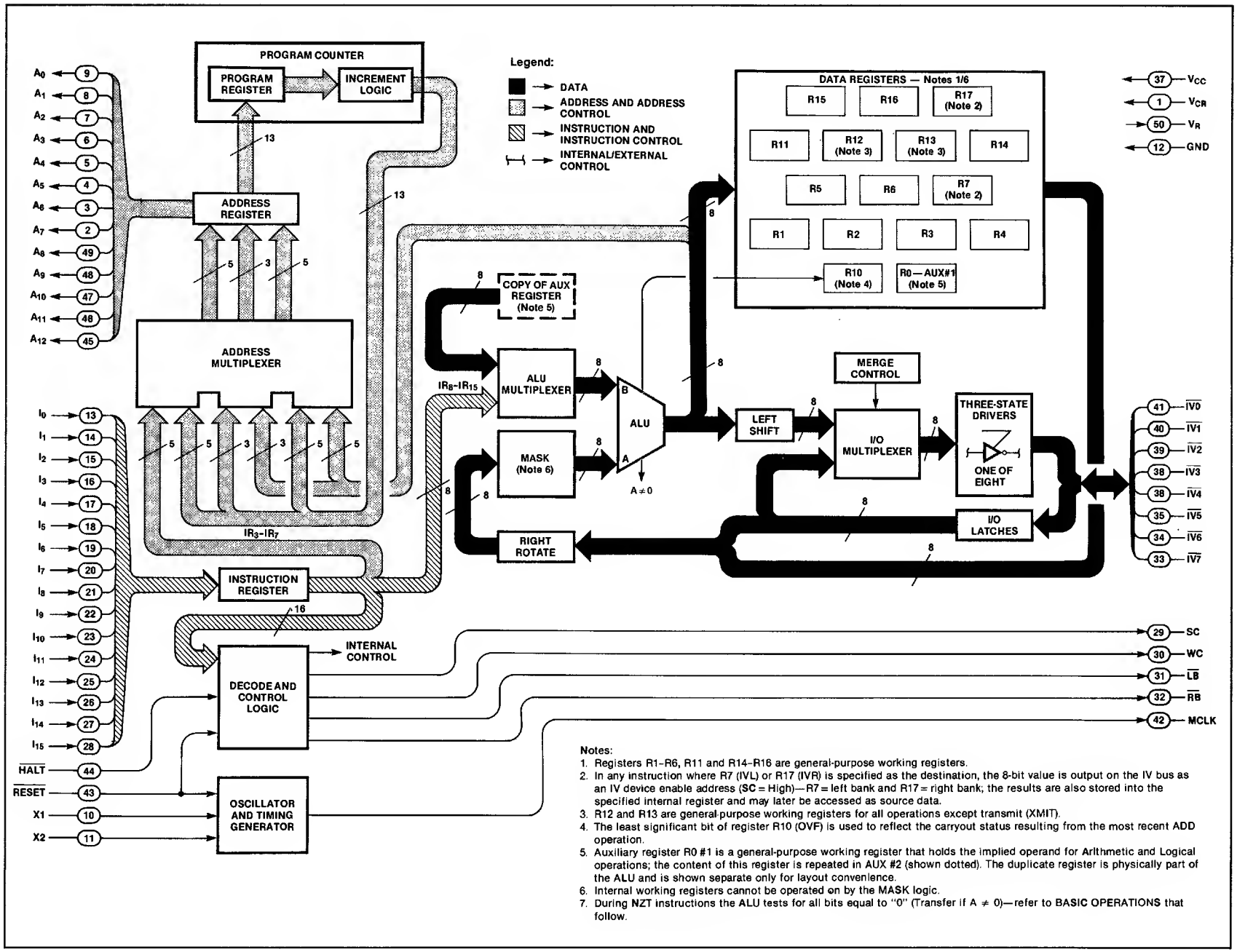


Figure 1-3. General System Example



Notes:
1. Registers R1–R6, R11 and R14–R16 are general-purpose working registers.
2. In any instruction where R7 (IVL) or R17 (IVR) is specified as the destination, the 8-bit value is output on the IV bus as an IV device enable address (SC = High)—R7 = left bank and R17 = right bank; the results are also stored into the specified internal register and may later be accessed as source data.
3. R12 and R13 are general-purpose working registers for all operations except transmit (XMIT).
4. The least significant bit of register R10 (OVF) is used to reflect the carryout status resulting from the most recent ADD operation.
5. Auxiliary register R0 #1 is a general-purpose working register that holds the implied operand for Arithmetic and Logical operations; the content of this register is repeated in AUX #2 (shown dotted). The duplicate register is physically part of the ALU and is shown separate only for layout convenience.
6. Internal working registers cannot be operated on by the MASK logic.
7. During NZT instructions the ALU tests for all bits equal to "0" (Transfer if A ≠ 0)—refer to BASIC OPERATIONS that follow.

Figure 2-1. Functional Block Diagram

Chapter 2

FUNCTIONAL OPERATION

The purpose of this chapter is to provide the user of the 8X305 MicroController with a detailed description of the functional blocks used to process data under program control. Understanding of these mechanisms will assist the user in making full use of the power and flexibility of the MicroController within the context of a particular application.

The 8X305 will be discussed in several stages; from the hardware components of which it is composed, through the basic steps of each instruction cycle and a discussion of the instruction format, to the flow of data in the device during instruction execution. Figure 2-1 shows a detailed functional block diagram of the 8X305, which will be used as the basis of discussion for this chapter.

2.1 HARDWARE COMPONENT BLOCKS

The functional blocks of the 8X305 may be categorized as:

- Oscillator and Timing
- Program Addressing
- Decode and Control
- Arithmetic and Logic Unit
- Bit Manipulation
- Internal Registers
- IV Bus Control

2.1.1 Oscillator and Timing Generator

Using an external frequency-determining element, such as a crystal (or capacitor), the on-chip oscillator provides reference pulses to the Timing Generator which, in turn, produces the Master CLocK (MCLK) and four quarter-cycle pulses. MCLK is used for external control and system synchronization whereas the 1st, 2nd, 3rd, and 4th quarter-cycle pulses are used for internal gate control during the instruction cycle.

2.1.2 Program Address Logic

Functional detail of the program address logic is shown in Figure 2-2. The program address logic revolves around two basic registers—the Address Register (AR) and the Program Register (PR). The Address Register holds the address of the current instruction for program memory and the Program Register holds the address of the current (or next) instruction. Normally, these two registers contain the same address; however, for branch operations, the content of the two registers may differ. Except for XEC (Execute), NET (non-zero transfer), and JMP (Jump) instructions, the output of the Program Counter is incremented during the third quarter cycle. The incremented data is passed through the lower ports of multiplexers A (bits 8-12), B (bits 5-7), and C (bits 0-4) and the composite is loaded into the Address Register. During the fourth quarter cycle, the incremented address is looped back to update the Program Counter.

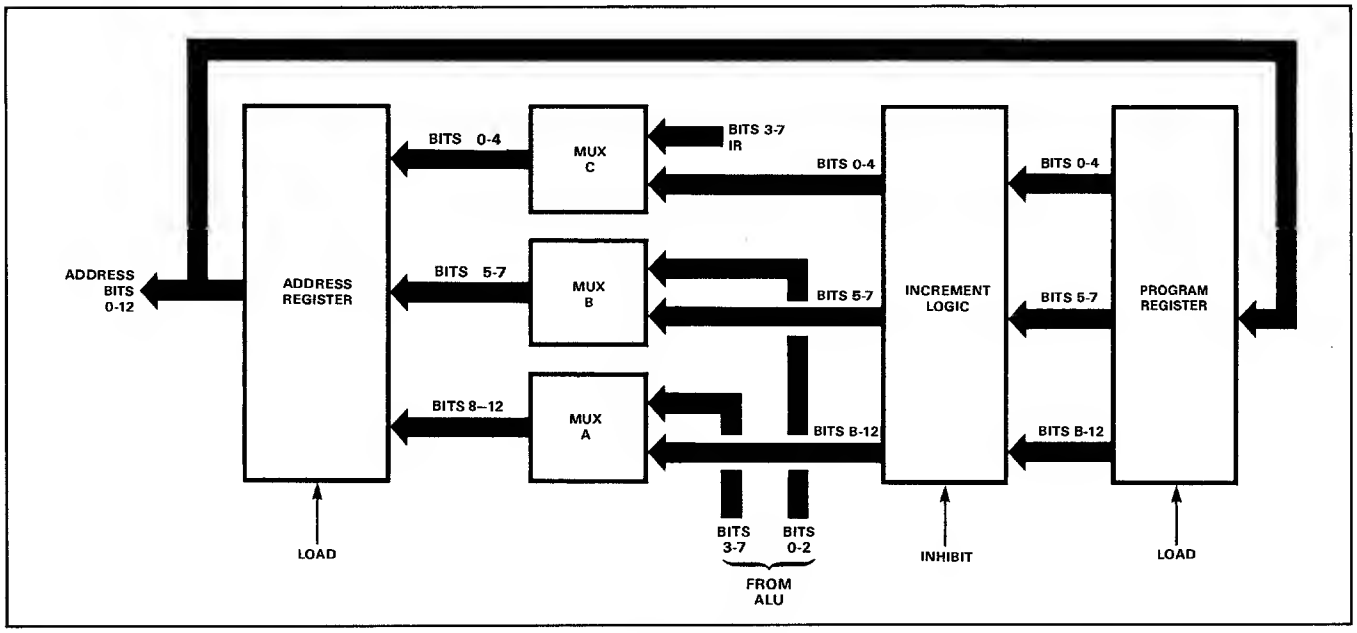


Figure 2-2. Program Address Data Flow

For branch operations, the generation of the next address is modified by internal logic to achieve ease of programming and operating flexibility. Multiplexers A, B, and C allow the Address Register to be loaded from two sources—8 bits from the ALU via multiplexers A and B and 5 bits from the instruction register via multiplexer C. For XEC and NZT instructions, these loading options provide the programmer with 256-byte pages for register operations and 32-byte pages for IV-bus operations. If the source for an execute operation is an internal register, the eight least significant bits of the Address Register is updated via the ALU; if the source is the IV bus, the five least significant bits are changed. In either case, incrementing of the Program Register is inhibited, that is, the contents of this register do not change. For non-zero transfer operations, the source data (register or bus) is tested for a "zero" or "not-zero" condition. If the result is zero, the Program Register is incremented in normal fashion; if the result is not zero, the 8 LSB (source = internal register) or the 5 LSB (source = IV bus) of both the Address Register and Program Register are updated via the ALU. In the case of the JMP instruction, all thirteen bits of the Address Register and the Program Register are changed to identify the new address.

2.1.3 Decode and Control Logic

An instruction from program memory is latched into the Instruction Register at the beginning of each cycle, and is interpreted by the Decode and Control Logic on the basis of the high-order three bits, which specify the Operation, or Op Code. There are eight (8) instruction classes and they can be separated into two control areas—*data control* and *program control*; general functions within these areas are:

- Data Control —
  - ADD, AND, XOR } Arithmetic and Logic Operations
  - MOVE, XMIT } Movement of Data and Constants
- Program Control —
  - XEC, NZT, JMP } Branch or Test

The operation and function of each instruction class are defined in Table 2-1.

Table 2-1. Instruction Classes

Mnemonic	Op Code	Operation	Function
Data Control:			
MOVE	0	Move	Data Transfer
ADD	1	Add A and B	Arithmetic
AND	2	Logic AND	Logic
XOR	3	Exclusive OR	Logic
XMIT	6	Transmit	Data Transfer
Program Control:			
XEC	4	Execute	Indexed Temporary Branch
NZT	5	Non-Zero Transfer	Conditional Branch
JMP	7	Jump	Non-Conditional Branch

As shown in Figure 2-3, the rest of the instruction word is divided into three Operand Fields whose significance is a function of the Op Code. The Decode and Control Logic generates the internal enable signals for data flow and manipulation and also provides the external bus control signals, using timing provided by the timing generator.

2.1.4 Arithmetic Logic Unit

Referring to Figure 2-4, the Arithmetic Logic Unit (ALU) receives the "A" inputs from either internal registers or I/O (through Rotate/Mask logic) and the "B" inputs from either the Instruction Register (IR8-IR15) or from a register containing an exact duplicate of the information contained in the Auxiliary Register R0 via a multiplexer. The ALU performs one of four functions on the data. It may ADD, AND or XOR the A and B input data, or it may pass it straight through with no modification (MOVE,XMIT,JMP). The ALU output data is then transferred to either the internal registers, the I/O Bus through Shift and Merge logic, or the Program Address logic. The ALU also indicates overflow condition during ADD functions by means of a discrete output to the LSB of the Overflow Register, R10. This bit is set if the 8th bit overflows. During NZT instructions, the ALU tests for all bits "0".

2.1.5 Bit Manipulation Logic

As shown by the block diagram in Figure 2-1, the "A" data path to and from the ALU contains three operating functions—*Right Rotate, Mask,* and *Left Shift.* Coupled with the *Merge* control logic, these three functions provide extensive bit manipulation capability and lend themselves to programming techniques that are efficient and powerful.

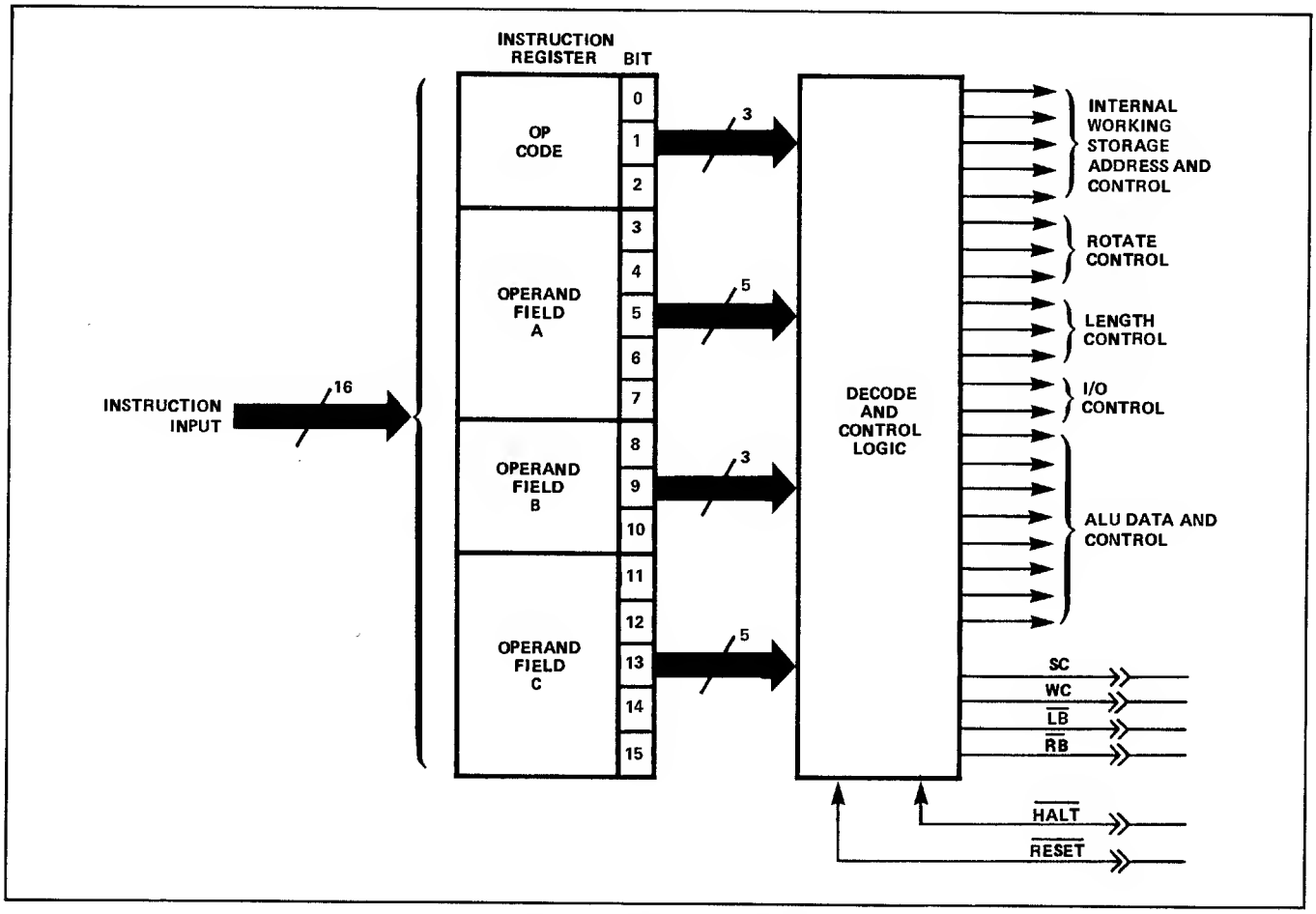


Figure 2-3. Instruction Decode

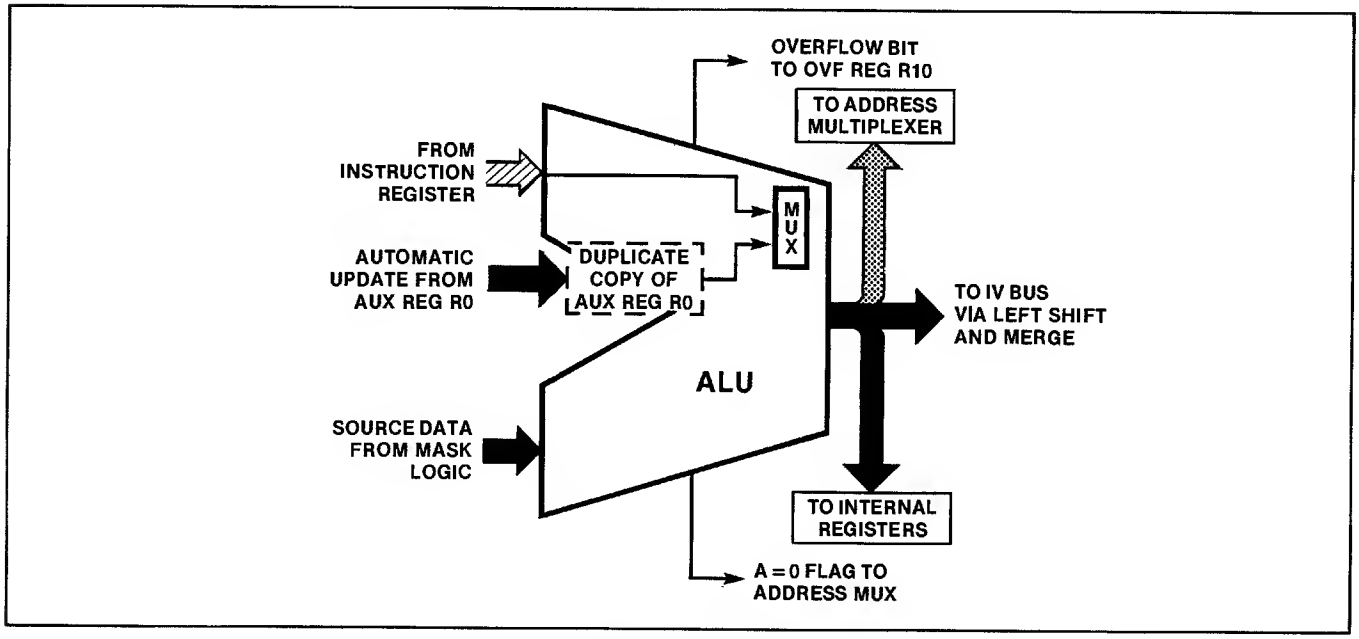
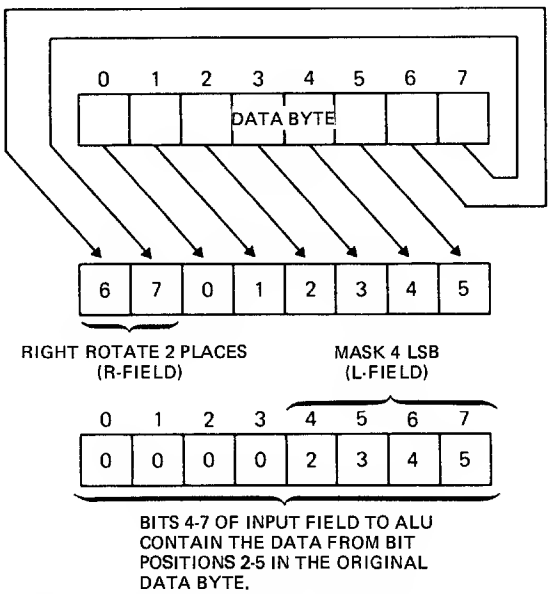


Figure 2-4. ALU Data Flow

The combination of the *Right Rotate* and *Mask* functions allows selection of one or more bits from a source data field. For instructions where both the source and destination are internal registers, only the *Right Rotate* function can be used—the data being a fixed length of 8 bits. The right rotate function provides an end-around-shift of one to seven places of the 8-bit source field. In this manner, the least significant bit of the required bit string can be positioned in the least significant position of the data byte and readied for further processing. Refer to Figure 2-5.



It is assumed that the data of interest in the input byte are bits 2 through 5, inclusive.

A rotate of two is used to align bit position 5 with the LSB, end, after masking the four LSBs, bit positions 0, 1, 6, and 7 of the original data ere overwritten with zeroes.

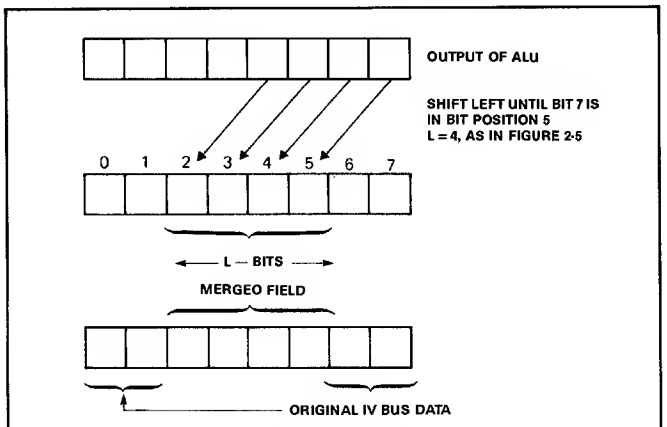
The operand fields controlling these operations depend on the instruction format — see later discussions for more detail.

Figure 2-5. Rotate and Mask Operations

The number of places that the data is to be rotated is specified by the R-field operand (when present) and by the S_0-field when the source is the IV bus. The R-field specifies the number of places the data is to be rotated; the S_0-field specifies the bit of the source data field which will be rotated to bit position seven before masking.

The *Mask* function allows selection of the least significant "L" bits of the rotated IV bus source data for subsequent processing. The L-field operand is specified by the "L" field of the instruction. After masking, the least significant bits specified by "L" are output to the ALU. All remaining bits of the byte are set to zero.

The *Left Shift* and *Merge* control logic provide the wherewithal to alter a bit string within an IV bus data byte. The preceding *Right Rotate* and *Mask* functions ensure that the required processed data is in the least significant bits of the ALU output; the *Left Shift* operation then aligns the data with the proper bit positions prior to merging. Refer to Figure 2-6. Because the process is not an end-around-shift, data shifted from the MSB (position 0) is lost. The number of positions to be shifted is specified by the value of D_0 and the data is left shifted until the LSB reaches this value.



In this exemple, the result of figure 2-5 is to be merged with the original IV Bus data. Note that shift and mask cannot be performed on data whose destination is an internel register.

A left shift of two aligns the LSB of the processed data with bit position 5. A merge of four results in a byte output to the IV Bus where Bits 2 through 5 are processed data end Bits 0, 1, 6 and 7 is original input data from a port on the IV Bus.

Figure 2-6. Shift and Merge Operations

The user can update 1-to-7 bits of existing IV bus data without affecting other bits via the *Merge* control logic. The length of the bit string to be merged with existing data is specified by the L-field operand; the LSB of the bit string is specified by D_0 (after shifting).

Regardless of what bit-manipulation functions are invoked, the ALU performs all arithmetic and logic operations. For this purpose, the ALU has a direct input from the AUXiliary register for the implied operand in AND, ADD, and XOR instructions. The output of the ALU may go directly to the address or data registers or, via the shift and merge circuits, to the IV bus.

2.1.6 Registers (Internal Working Storage)

The 8X305 contains sixteen 8-bit registers, thirteen of which are used as general-purpose storage of data during program execution. Any of the internal registers can be specified as a *source* of data and, if so specified, the data passes through the Rotate and Mask operators on the way to the ALU. In any operation which requires a second operand, such as an ADD instruction, the implied second operand is always the AUXiliary Register (R_0).

Operational parameters of the preceding registers are given in Table 2-2. Note that the Overflow (OVF) Register R_{10} is loaded automatically during each ADD operation, and consists effectively of only one bit.

Table 2-2. Internal Register Assignments of 8X305

Reg — Octal Desig	Function	Note
R0	AUXiliary Register	1
R1	General Purpose Operation	
R2	General Purpose Operation	
R3	General Purpose Operation	
R4	General Purpose Operation	
R5	General Purpose Operation	
R6	General Purpose Operation	
R7	Left Bank Register	2
R10	Overflow (OVF) Register	3
R11	General Purpose Operation	
R12	Special Purpose Register used for Transmit Operation	4
R13	Special Purpose Register used for Transmit Operation	4
R14	General Purpose Operation	
R15	General Purpose Operation	
R16	General Purpose Operation	
R17	Right Bank Register	2

Notes:
1. Contains the second or Implied operand. Physically consists of two registers with identical data—the second register (Accumulator) is located in the ALU.
2. When R7 is specified as the *destination*, the 8-bit value is also output to the selected port on Left Bank of IV bus; the operation is exactly the same for R17, except the Right Bank is affected. Either register can also be used as a *source* of data.
3. Least Significant Bit used as OVF flag storage for the most recent ADD operation. Only the LSB is used and this register can only be specified as a *source*.
4. For all instructions except transmit (XMIT), R12 and R13 operate as general-purpose registers; when either register is specified as a *destination* during a transmit instruction, data is transmitted to the Left Bank of IV bus (R12 specified) or to the Right Bank (R13 specified). Whichever the case, the contents of the register is unchanged.

2.1.7 IV Bus Control

Timing of the IV bus is synchronous with the Master Clock (MCLK) signal—an output of the Timing Generator. MCLK is high during the last quarter of each instruction cycle.

IV bus control is provided by four signals generated by the Decode and Control Logic. These control signals and their functions are as follows:

*S*elect *C*ommand (SC)—a high (binary 1) on this line indicates that an address is being output on the IV bus to enable an I/O port or device.

*W*rite *C*ommand (WC)—a high (binary 1) on this line indicates that data is being output on the IV bus, to be stored in a previously-enabled register/port. When both SC and WC are low (binary 0) the MicroController expects data from the selected device. Both SC and WC high (binary 1) is a condition not generated by the 8X305.

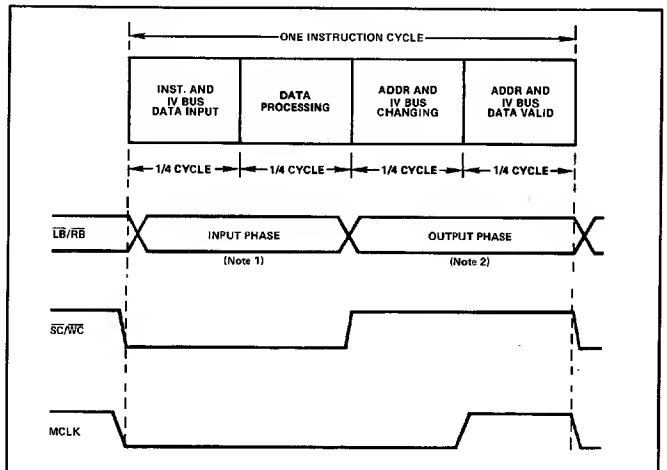
*L*eft *B*ank Enable ($\overline{LB}$)—a low (binary 0) on this line enables one of two groups of I/O devices (or memory locations). In all following text, this group of devices is referred to as the Left Bank.

*R*ight *B*ank Enable ($\overline{RB}$)—a low (binary 0) on this line enables the second of the previously mentioned two groups of I/O devices (or memory locations). In all following text, this group of devices is referred to as the Right Bank.

Note: When neither bank-select signal is true (both high) this signifies that the 8X305 has released the bus by putting $\overline{IV0}$-$\overline{IV7}$ in a three-state condition; the bus may then be used for other activities.

2.2 BASIC INSTRUCTION CYCLE

The instruction cycle of the 8X305 equals one-half of the external clock cycle. The instruction cycle is subdivided into four quarter-cycles, which are used to gate data flow and processing in the correct sequence. The cycle is outlined here for the purpose of understanding its effect on data processing. Refer to Figure 2-7 during the following timing discussion of the four quarter-cycles. Note that the quarter-cycles are internal to the device and cannot be seen as external signals.



Notes:
1. I/O Port drivers active and 8X305 drivers three-stated at this time.
2. I/O Port drivers three-stated and 8X305 drivers active at this time.

Figure 2-7. Instruction Cycle Subdivisions

The exact operations performed depend on the instruction to be executed. An instruction which performs an IV bus-to-IV bus transfer is described in the rest of this subsection.

During the first quarter-cycle, a new instruction is input to the 8X305 from program storage, via the I_0-I_{15} lines. This instruction is interpreted by the Decode and Control Logic. Data from the user system is accessed via the $\overline{IV0}$-$\overline{IV7}$ lines under control of the bus control signals. At the end of this quarter-cycle, the instruction is latched. The input phase, when data is input to the 8X305 from a point in the IV bus consists of this and the following quarter-cycle.

During the second quarter-cycle, instruction processing is initiated. The bus data is latched into the IV bus and ALU latches at the end of the quarter-cycle. Since there are a number of gate delays between the IV bus and ALU latches, the bus data must be stable early in this quarter-cycle to allow time for correct ALU latching of data.

In the third quarter-cycle, while the address for the next instruction fetch is output to the A_0-A_{12} lines, control signals for execution of the output phase of the present instruction are generated, and output data for the bus is stabilized. During this time, the address latch is open.

During the fourth quarter-cycle, a master clock signal (MCLK) generated by the 8X305 is used to latch valid address or data into peripheral devices connected to the IV bus; MCLK is also used to synchronize any external logic with timing circuits of the 8X305. This concludes the output phase and the instruction cycle.

To summarize the action, the first half of the instruction cycle deals primarily with input functions, and the second half is mostly concerned with output functions. Some instructions, such as register-to-register, make no use of the IV bus, and others, such as a branch, manipulate address information before it is output to program memory. All instructions, however, conform with this basic sequence.

2.3 INSTRUCTION FORMATS

The 16-bit instruction word is fetched from program storage and input to the Decode and Control Logic.

The twenty-four variations of 8X305 instructions are grouped in eight classes, each distinguished by a unique Op Code. These are classified by seven formats of the instruction, as follows:

- Manipulation of register data (Format RR)
- Manipulation of register and IV bus data (Format RB)
- Branch or Test on register data (Format JR)
- Transmission of data to a register (Format XR)
- Branch or Test on IV bus data (Format JB)
- Transmission of data to the IV bus (Format XB)
- Unconditional branch (Format JA)

Within each class, operands of the instruction word provide powerful variations for the manipulation of data down to the bit level. This section is a detailed machine-level discussion of the operation of the various instruction fields and helps the reader understand internal workings of the 8X305 which are masked by the simplicity of the MCCAP CrossAssembler. For further information on MCCAP, refer to the MCCAP Manual.

2.3.1 General Formats

Each instruction class has its individual members defined by the six Operand Fields that comprise bits I_3 through I_{15}. Refer to Figure 2-8 for the various configurations of these fields which result in the seven general formats.

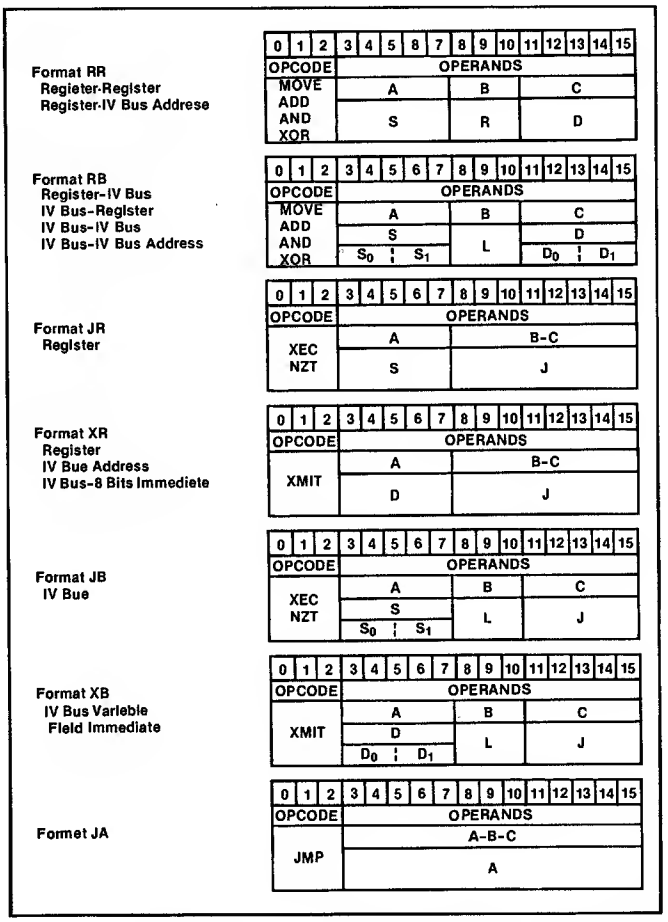


Figure 2-8. General Formats of the Eight Instruction Classes

All instructions in the 8X305 can be characterized by one of these formats, and reference will be made to Figure 2-8 while explaining specific instruction operations discussed in the rest of this section.

2.3.2 Instruction Fields

The exact contents of each instruction field is a function of the specific instruction, but the following fields may be distinguished:

- *Op Code*, which defines the class of instruction
- *Source* (S), which defines the origin of the data
- *Destination* (D), which defines the final location of the data
- *Rotate* (R), which defines the amount of source data rotation
- *Length* (L), which defines the number of bits of the source or destination data
- *Immediate* (J), which defines a Literal Constant
- *Address* (A), which points to the next instruction to be executed

Each of these fields specifies a functional component, the net effect of which is to identify a particular instruction. Since the *Source, Destination, Rotate* and *Length* fields are somewhat complex in their definition, they are explained in separate subsections.

The Op Code identifies the class of instruction, which includes a definition of the ALU operation to be performed on the data, if any. Table 2-3 gives a brief overview of the 8X305 instruction classes.

The J-field specifies a literal as a source for the Instruction, which may be modified in accordance with the particular instruction type. This is most often used to introduce a constant into the program flow, such as presetting a register to a particular value, or providing an offset address for conditional branches.

The A-field is only used in a JMP (Jump) instruction, and is a special case of a Literal, which is used to branch to a specific address, or location in program memory for the next instruction.

Table 2-3. Description of Instruction Classes

Op Code	Mnemonic	Operation
0	MOVE	Move data in location specified by S-field to location specified by D-field.
1	ADD	Add data in location specified by S-field to contents of accumulator (AUX Register R0) and store result in location specified by D-field.
2	AND	Logically AND data in location specified by S-field with contents of accumulator and store result in D-field location
3	XOR	Exclusively-OR data in location specified by S-field with contents of accumulator and move result to location specified by D-field.
4	XEC	Execute out of sequence, the instruction within the current address page pointed to by a combination of the S-field location data and the J-field data within current page
5	NZT	If the data specified by the S-field is non-zero, a branch is performed. If the data is zero, increment the program counter by one.
6	XMIT	Transfer the data of the J-field to the D-field location
7	JMP	Branch to the instruction pointed to by the A-field

2.3.3 Source and Destination Fields

The S and D fields define where the data of the first operand of the instruction is to come from, and where the result of the operation is to be stored. In the case of Arithmetic/Logic operations, a third field is implied, although not specifically defined in the instruction. This is the second operand of the ALU function. Since this is effectively an automatic source in that AUX register R_0 is always used, this need not be explicitly stated in instruction coding. Note that the desired value of the second operand must have been stored in R_0 during a previous instruction.

The source of data in the 8X305 can be any one of the internal registers or it can be the IV bus. The S-field has two distinct formats, depending on which of these is being used as the source.

When an internal register is the data source, the 4 low-order bits of the S-field are used to address the register, and the high-order bit is always zero.

When the IV bus is the data source, this is flagged by the high-order bit being one, and the next-to-high-order bit selecting either the Left Bank (bit is zero) or Right Bank (bit is one). These two bits form the S_1-field. The remaining three low-order bits form the S_0-field, and define the amount of rotation that will be performed on the incoming data. Refer to Figure 2-9.

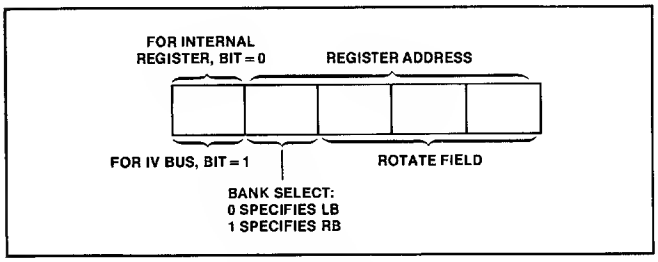


Figure 2-9. S-Field Formats

Note that it is not possible to rotate data from an internal register source under control of the S-field, although internal register rotate is possible using the R-field.

The destination of the result of an instruction is defined similarly to the above, but a few additional effects should be noted.

Any instruction specifying register R_7 or R_{17} as the destination will, in addition to placing the result in that register, also output it to the IV bus as an I/O device address when SC is high.

A XMIT instruction that specifies R_{12} or R_{13} as a destination will output the result on the IV bus as data, and the register contents remain unchanged.

Any internal register except R_{10} can be selected as a destination by having the high-order bit of the D-field set to zero, and placing the register address in the remaining four bits, similar to register source selection above.

To select the IV bus as the destination, the high-order bit of the D-field is set to one, and the next-to-high-order field is used for bank select, as in IV bus sources. This is designated the D_1-field, with the other three bits being the D_0-field and specifying the amount of left shift to be applied to the result from ALU.

Note that is it not possible to shift data destined for an internal register under D-field control.

2.3.4 Rotate and Length Field

Rotation of IV bus data is achieved by the proper coding of the S_0-Field. In the case of a register-to-register (Format RR) operation, where both the S-field and D-field specify Internal Registers, then the R/L-field is interpreted as a rotation to be performed on the source data before it is presented to the ALU. In this case, no masking of data is possible.

In the case of a Format RB, JB, or XB instruction, the R/L-field is used as a Length field to control the number of bits of the I/O data field (data source or destination on the IV bus). Refer to Figures 2-5 and 2-6 for the details of the bits allowed to pass the Mask or Merge logic.

For an IV Bus-to-Register instruction, the L-field specifies the length of source data allowed to pass the Mask logic and enter the ALU with the high order bits being set to zero. Note that the source data could have been right-rotated after input from the IV bus in accordance with the S-field.

For a Register-to-IV Bus instruction, the L-field specifies the length of the destination field whose least significant bit is specified by D_0.

For an IV Bus-to-IV Bus instruction, the L-field specifies the length of both the source data passing the Mask logic, and the destination data to be replaced by the Merge logic.

2.3.5 Instruction Sequence

Sequential execution of instructions in the program memory can be deviated from by use of the XEC, NZT and JMP instructions.

The Address Register and Program Counter are used to generate addresses for accessing an instruction from program storage. The instruction address is formed in any one of four ways:

- For all except the JMP, XEC, and a "satisfied" NZT instruction, the Program Counter is incremented by one and placed in the Address Register and the Program Counter.
- For the JMP instruction, the 13-bit A-field contained in the JMP instruction word replaces the contents of the Address Register and the Program Counter.
- For the XEC instruction, the Address Register is loaded with the high-order bits of the Program Counter and the low-order bits modified as follows:

  XEC using IV Bus Data: low order 5-bits of ALU output replaces counterpart bits in Address Register. The Program Counter is not modified.

  XEC using Data from Internal Register: low order 8-bits of ALU output replaces counterpart bits in Address Register. The Program Counter is not modified.
- For a "satisfied" NZT instruction, the high order bits of the Address Register and the Program Counter are loaded from the Program Counter and the low order 5-bits (NZT source is IV Bus Data) or low order 8-bits (NZT source is an Internal Register) are loaded with the literal value specified by the J-field of the instruction word.

Refer to Chapter 3 for a more complete discussion of individual instructions.

2.4 DATA FLOW DURING INSTRUCTION EXECUTION

This section illustrates the major paths of data flow within the device during execution of instructions from each of the general formats. This is illustrated diagrammatically in each case by using a block diagram on which the path has been outlined. This is not an exhaustive collection of all possible data paths. Refer to Figure 2-8 for the format being used in each case.

The details illustrated serve as aids in understanding internal functions of the 8X305. They are not required for understanding normal programming of the 8X305.

Throughout this section, the numeric examples are given in terms of octal values within each operand field, and not in pure octal.

2.4.1 Operations That Use Format RR

Operations that use Format RR are Register-to-Register and Register-to-IV Bus Address; either operation can be invoked by a MOVE, AND, ADD, or XOR instruction. Paths for the address, the instruction, the data flow, and the required control lines for a Register-to-Register operation are shown in Figure 2-10a. A summary of operational parameters pertaining to Figure 2-10a are given below.

Format: See Format RR, Figure 2-8.

Description: Contents of the register specified by S are right-rotated as specified by R and placed in the destination register specified by D. The contents of the source register remain unchanged; original contents of the destination register are lost. Overflow (OVF) Register R10 is updated if Op Code specifies an ADD operation; this register cannot be used as a destination. Also, the Copy of AUX Register is used only for AND, ADD, and XOR instructions—*not* for MOVE. (Note: As defined by the RR format, registers R7 and R17 can only be used as a *Source* for a Register-to-Register operation; this is not, however, a hardware limitation—see Figure 2-10b and discussion below.)

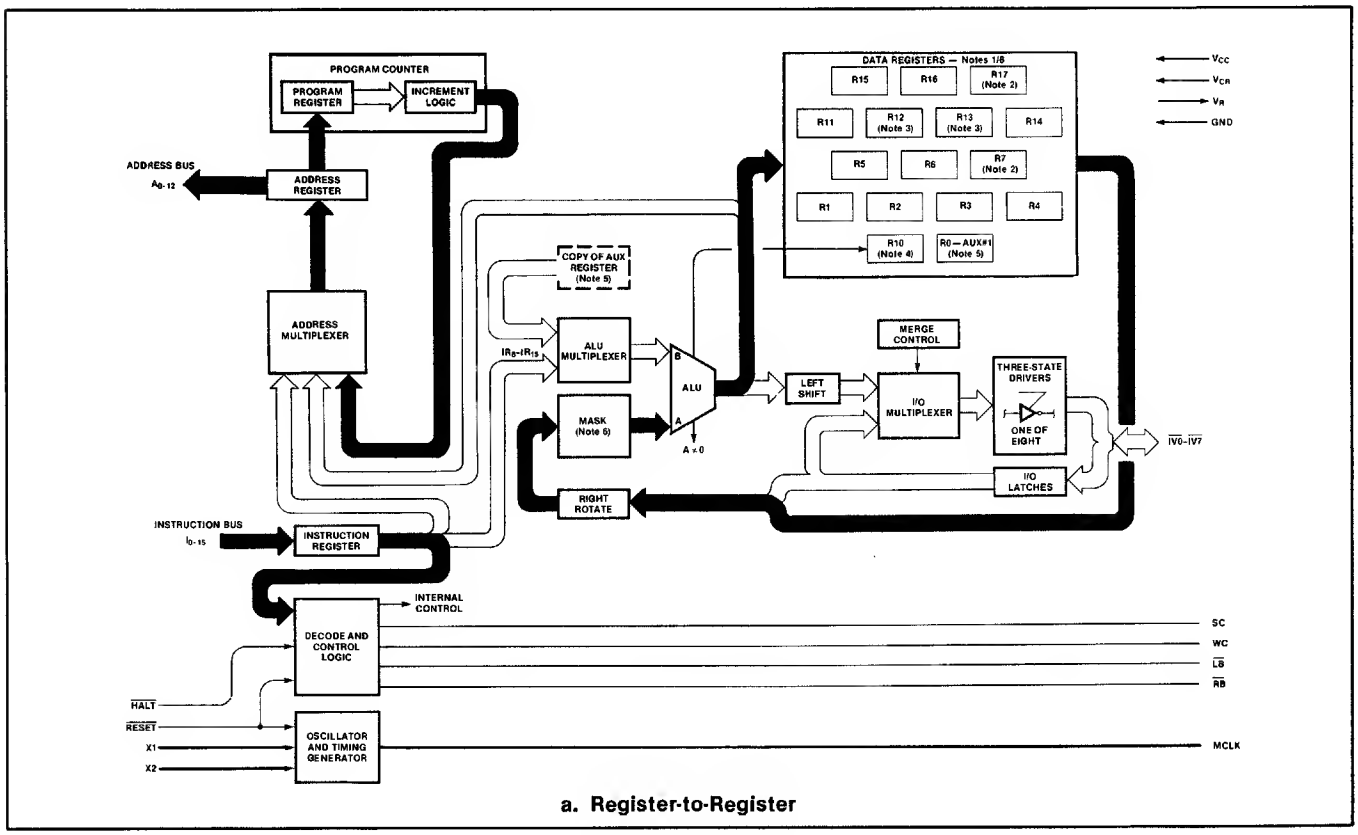
Permitted Operand Values:

S = R0-R17

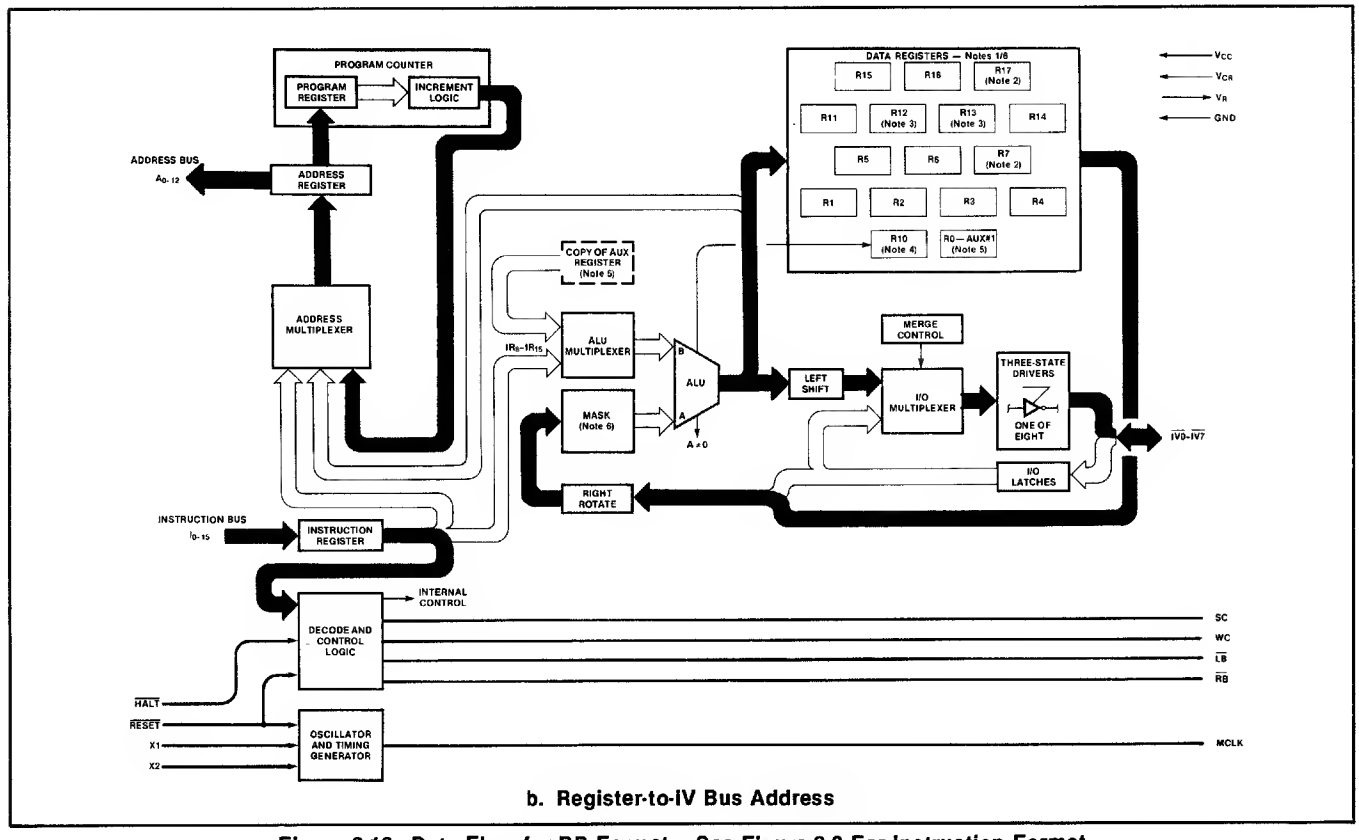
R = 0-7

D = R0-R6, R11-R16, R7/R17 (refer to Note in Description)

Paths for the address, the instruction, the data flow, and the required control lines for a Register-to-IV Bus Address operation are shown in Figure 2-10b. Operational parameters for this operation are identical to those for a Register-to-Register operation, except that one or the other of Registers R7 (Left Bank) or R17 (Right Bank) must always be the destination. In any case, the final result is output on the IV bus as an enabling address and stored in the destination register as well. The last step in the operating sequence puts the contents of the Address Register into the Program Counter.



a. Register-to-Register



b. Register-to-IV Bus Address

Figure 2-10. Data Flow for RR Format—See Figure 2-8 For Instruction Format

2.4.2 Operations That Use Format RB

Operations that use Format RB are Register-to-IV Bus, IV Bus-to-Register, IV Bus-to-IV Bus, and IV Bus-to-IV Bus Address; any one of these operations can be invoked by a MOVE, AND, ADD, or XOR instruction.

Register-to-IV Bus

Paths for the address, the instruction, the data flow, and the required control lines for a Register-to-IV Bus operation are shown in Figure 2-11a. A summary of operational parameters pertaining to Figure 2-11a is given below.

Format: See Format RB, Figure 2-8
Description: Move the least significant L bits of the register specified by S to a variable length field of the IV bus as specified by D_0 and D_1, where,
S = the source register
L = number of bits in masked data field to be merged with existing byte of IV data.
D_0 = bit position of IV data with which LSB of processed data field is to be aligned; this implies that the data field is left-shifted until bit 7 is aligned with bit D_0 of the IV bus.
D_1 = specifies bank of IV bus; 2 specifies Left Bank and 3 specifies Right Bank.
The order of operation is:
- Copy contents of selected IV data into I/O latches.
- Read contents of source register.
- Left shift source data field as specified by D_0.
- Merge least significant L bits with data in I/O latches.
- Output modified 8-bit data field to selected IV device as specified by D_1.

Observe that the original IV data outside the merged L-bit field remains unaltered; also, the contents of the source register remain unchanged. Overflow (OVF) Register R10 is updated if Op Code specifies an ADD operation; the Copy of AUX Register is used only for AND, ADD, and XOR instructions—*not* for MOVE.

Permitted Operand Values:
S = R0-R17
L = 0-7 (L = 0 selects an 8-bit field)
D_0 = 0-7
D_1 = 2 or 3

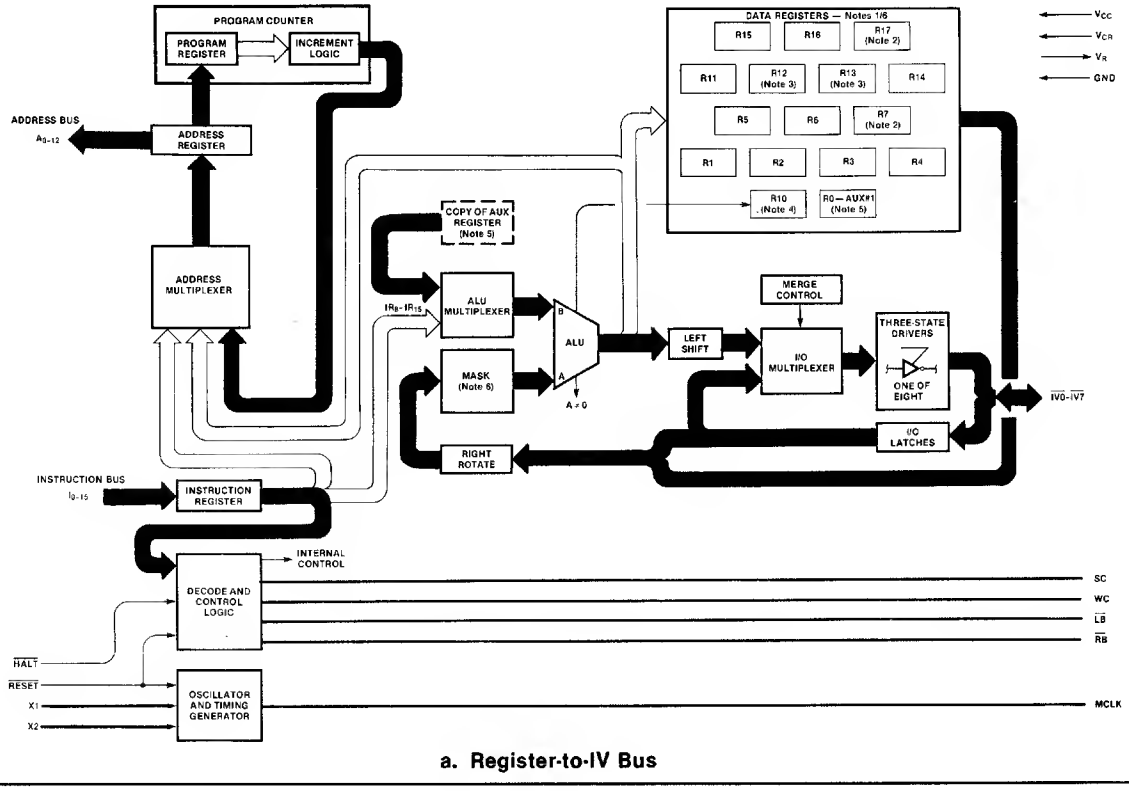
IV Bus-to-Register

Paths for the address, the instruction, the data flow, and the required control lines for an IV Bus-to-Register operation are shown in Figure 2-11b; a summary of operational parameters pertaining to this figure is given below.

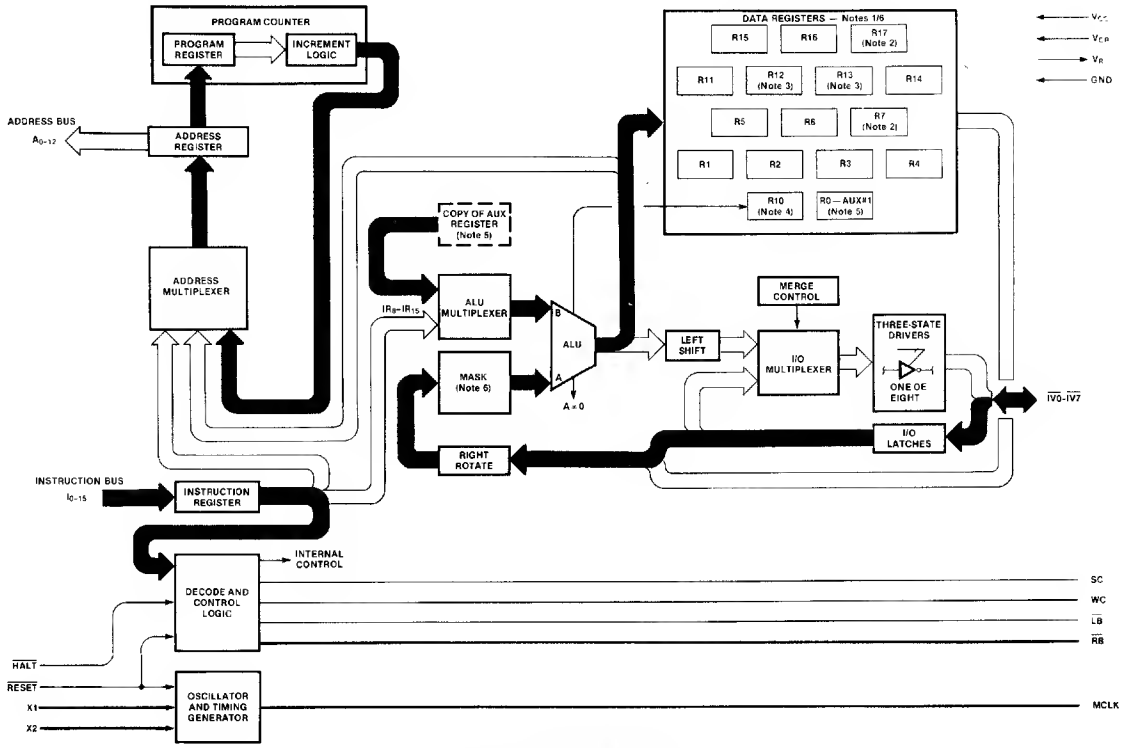
Format: See Format RB, Figure 2-8
Description: Move the least significant L bits of the IV data specified by S_0 and S_1 to the least significant L bits of the destination register specified by D, where,
S_0 = least significant bit of IV data field after rotation.
S_1 = specifies bank of IV bus; 2 specifies Left Bank and 3 specifies Right Bank.
L = number of bits in masked data field.
D = the destination register.
The order of operation is:
- Read contents of selected IV data as specified by S_1 into I/O latches.
- Right rotate the input data field as specified by S_0.
- Mask the least significant L bits of the rotated data.
- Move masked field to the least significant L bits of destination register with zeroes in the unmasked positions.

Overflow (OVF) Register R10 is updated if Op Code specifies an ADD operation; this register cannot be used as a destination. Also, the Copy of AUX Register is used only for AND, ADD, and XOR instructions—*not* for MOVE.

Permitted Operand Values:
S_0 = 0-7
S_1 = 2 or 3
L = 0-7 (L = 0 selects an 8-bit field)
D = R0-R6, R11-R16



a. Register-to-IV Bus



b. IV Bus-to-Register

Figure 2-11. Data Flow for RB Format—See Figure 2-8 For Instruction Format

IV Bus-to-IV Bus

Paths for the address, the instruction, the data flow, and the required control lines for an IV Bus-to-IV Bus operation are shown in Figure 2-11c; a summary of operational parameters pertaining to this figure is given below.

Format: See Format RB, Figure 2-8

Description: Move the variable length field specified by S_0, S_1, and L to the field and bank specified by D_0, D_1, where,

S_0 = the LSB of the rotated IV input data field.

S_1 = specifies input bank of IV bus; 2 specifies Left Bank and 3 specifies Right Bank.

L = number of bits in masked data field to be merged with existing byte of IV data.

D_0 = bit position of IV output data with which LSB of processed data field is to be aligned; this implies that the data field is left-shifted until bit 7 is aligned with bit D_0 of the IV bus.

D_1 = specifies output bank of IV bus; 2 specifies Left Bank and 3 specifies Right Bank.

The order of operation is:

- Read contents of selected IV data as specified by S_1 into I/O latches.
- Right rotate the input data field as specified by S_0.
- Mask the least significant L bits of the rotated data.
- Left shift source data field as specified by D_0.
- Merge least significant L bits with data in I/O latches.
- Output modified 8-bit data field to selected IV device as specified by D_1.

Overflow (OVF) Register R10 is updated if Op Code specifies an ADD operation; the Copy of AUX register is used only for AND, ADD, and XOR instructions—*not* for MOVE.

Permitted Operand Values:

S_0 = 0-7

S_1 = 2 or 3

L = 0-7 (L = 0 selects an 8-bit field)

D_0 = 0-7

D_1 = 2 or 3

IV Bus-to-IV Bus Address

Paths for the address, the instruction, the data flow, and the required control lines for an IV Bus-to-IV Bus Address operation are shown in Figure 2-11d. A summary of operational parameters pertaining to Figure 2-11d is given below.

Format: See Format RB, Figure 2-8.

Description: Copy data from IV Bus as specified by S_0 and S_1

L = number of bits in masked field.

D = destination bank for processed data; R7 specifies Left Bank and R17 specifies Right Bank.

Note: The output address is also stored in the selected register, R7 or R17.

The order of operation is:

- Read input data from the IV bus.
- Right rotate input data as specified by S_0.
- Mask the least significant L bits. (Note: Bits outside the mask in the output field are set to zero.)
- Output result to designated bank of the IV bus as an enabling address.

Observe that the output address data is written into Register R7 (Left Bank) or R17 (Right Bank) and can subsequently be accessed as a source. Overflow (OVF) Register R10 is updated if Op Code specifies an ADD operation; the Copy of AUX Register is used only for AND, ADD, and XOR instructions—*not* for MOVE.

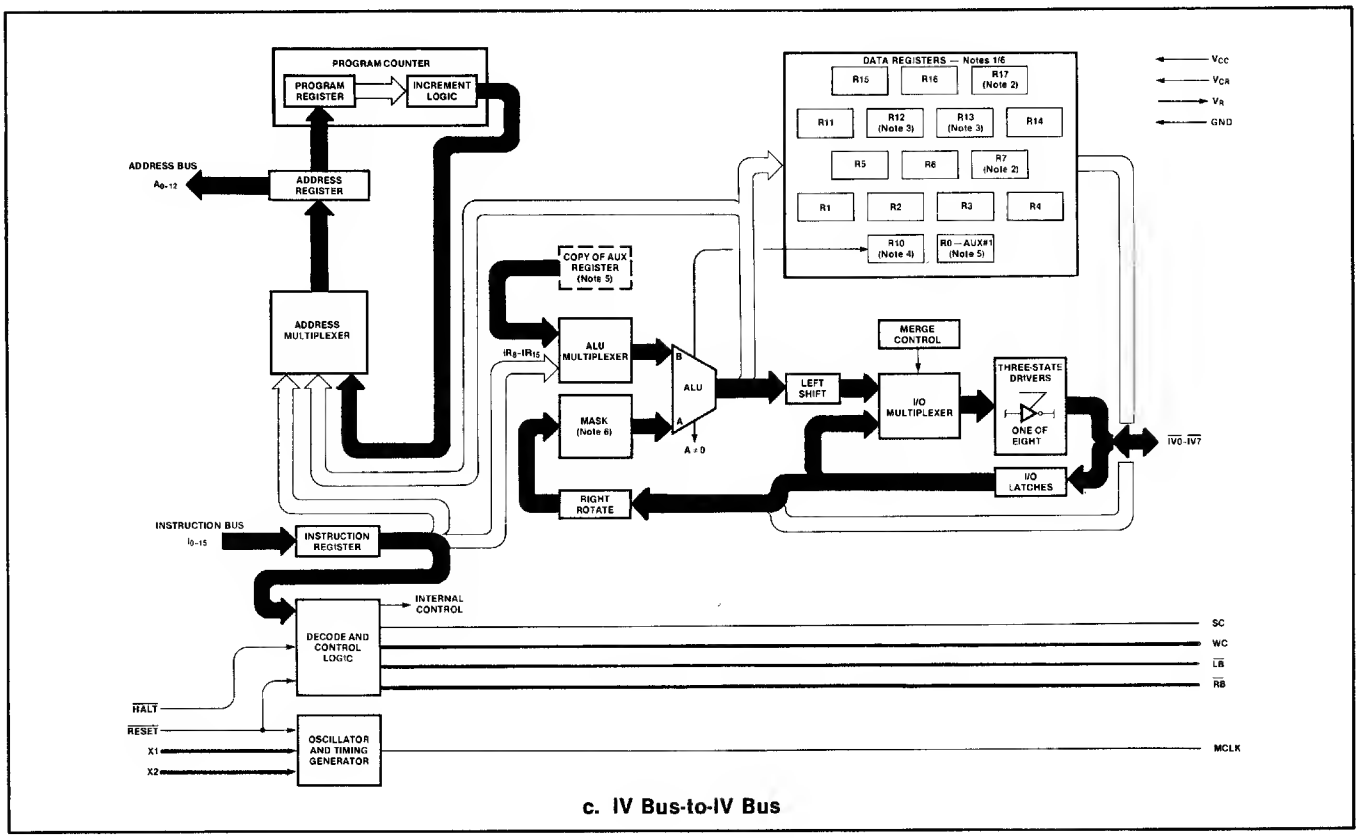
Permitted Operand Values:

S_0 = 0-7

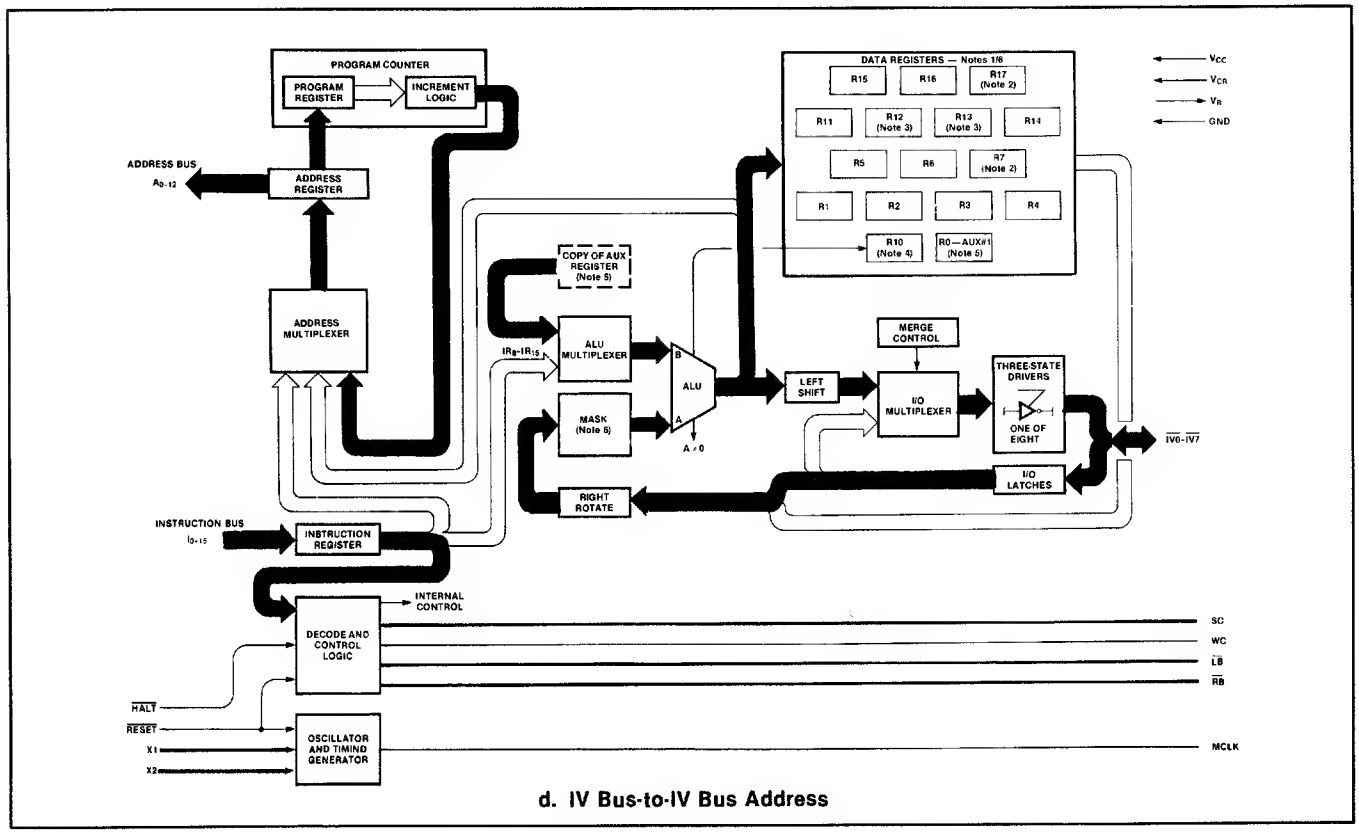
S_1 = 2 or 3

L = 0-7 (L = 0 selects an 8-bit field)

D = 0-7



c. IV Bus-to-IV Bus



d. IV Bus-to-IV Bus Address

Figure 2-11 (Continued)

2.4.3 Operations That Use Format JR

Operations that use Format JR are NZT-Register and XEC-Register.

NZT-Register Operation

Paths for the address, the instruction, the data flow, and the required control lines for the NZT-Register operation are shown in Figure 2-12a. A summary of operational parameters pertaining to the NZT-Register instruction is given below.

Format: See Format JR, Figure 2-8.
Description: If S ≠ 0, jump to address formed by replacing 8 LSBs of Address and Program Registers with value of J field; if S = 0, increment Program counter by one, where,
S = register which is subject of test (A input to ALU)
J = 8-bit integer that modifies address
The order of operation is:
- Read contents of source register.
- Test contents of source register for all zeroes.
- If contents are not zero, replace 8-LSBs of Address Register and Program Counter with 8-bit integer in J-field.
- If contents are zero, increment Program Counter by one.

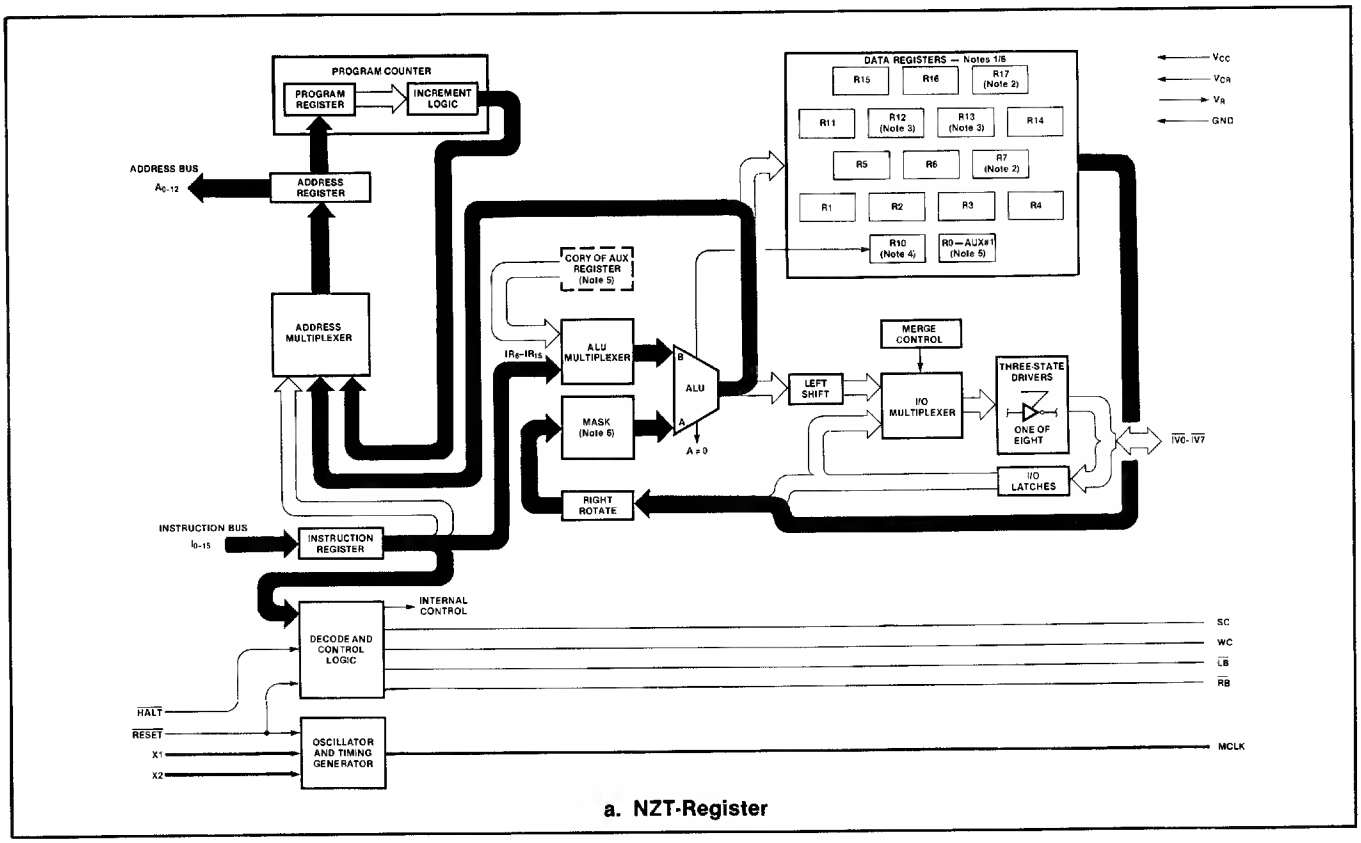
Permitted Operand Values:
S = R0-R17
J = $0\text{-}377_8$

XEC-Register Operation

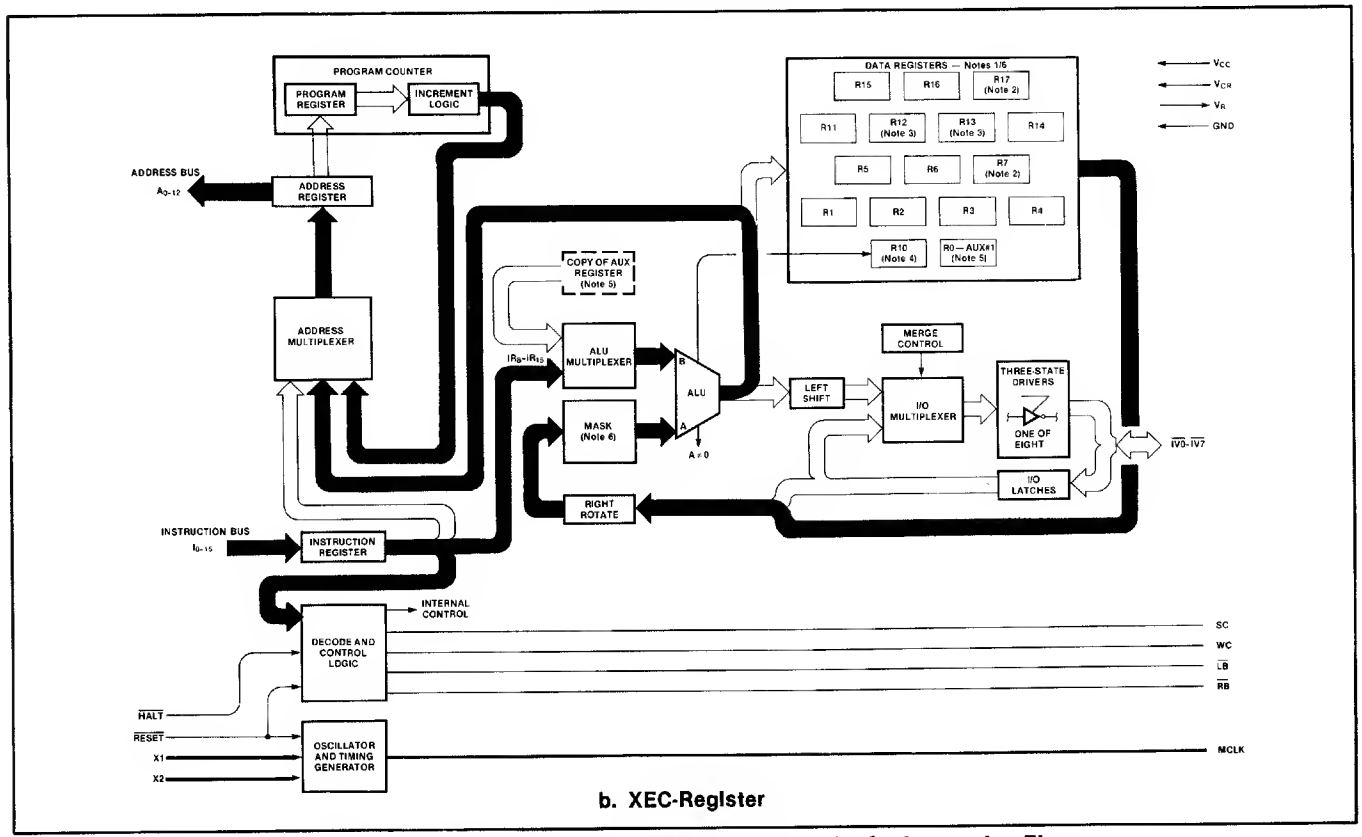
Paths for the address, the instruction, the data flow, and the required control lines for the XEC-Register operation are shown in Figure 2-12b. A summary of operational parameters pertaining to the XEC-Register is given below.

Format: See Format JR, Figure 2-8
Description: Execute instruction at the address formed by replacing 8 LSBs of Address Register with the 8-bit sum of J and the contents of register specified by S, where,
S = source register
J = 8-bit integer used in address modification
The order of operation is:
- Read contents of source register
- Form 8-bit sum by adding value of J-field to contents of source register
- Modify Address Register with the 8-bit sum

The Program Counter is not altered by the XEC instruction, that is, the original address within the 256 instruction page ($0 \leq J \leq 377_8$) is retained. During the instruction to be executed, the Program Counter is incremented by one in the normal manner to point to the instruction following the XEC instruction. However, if the executed instruction is a JMP or NZT, the Program Counter can be changed to the jump address and instruction execution does not return to the address following the XEC instruction.
Permitted Operand Values:
S = R0-R17
J = $0\text{-}377_8$ (When sum of S + J is greater than 255_{10} (377_8), only the 8-LSBs are used; Overflow (OVF) Register R10 is not changed.)



a. NZT-Register



b. XEC-Register

Figure 2-12. Data Flow for JR Format—See Figure 2-8 for Instruction Flow

2.4.4 Operations That Use Format XR

Operations that use Format XR are XMIT-Register, XMIT-IV Address, and XMIT-8 Bits Immediate IV Bus.

XMIT-Register

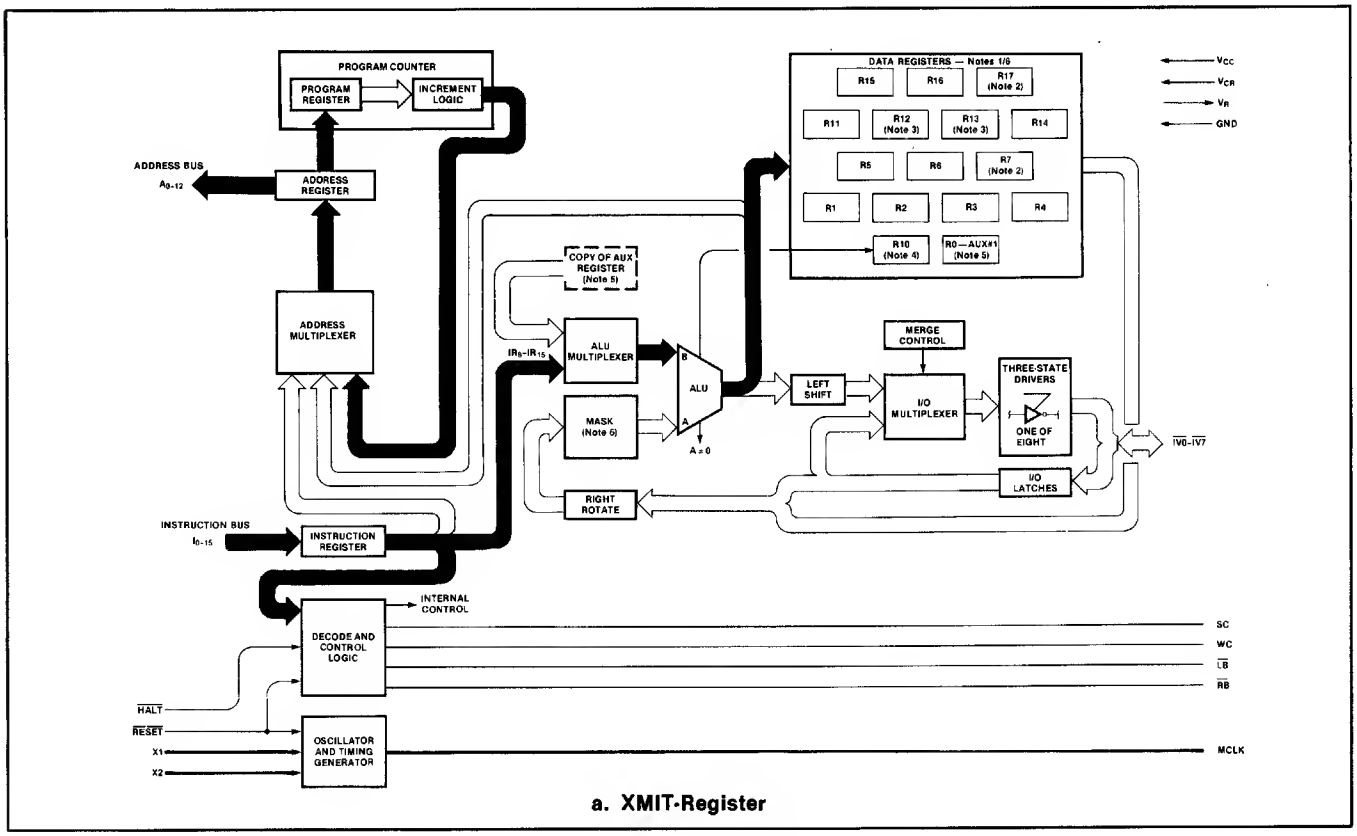
Paths for the address, the instruction, the data flow, and the required control lines for the XMIT-Register instruction are shown in Figure 2-13a. Operational parameters for these instructions are shown below.

Format: See Format XR, Figure 2-8
Description: Store value of 8-bit integer specified by J in register specified by D.
Permitted Operand Values:
D = R0-R6, R11, R14-F16
J = $0\text{-}377_8$

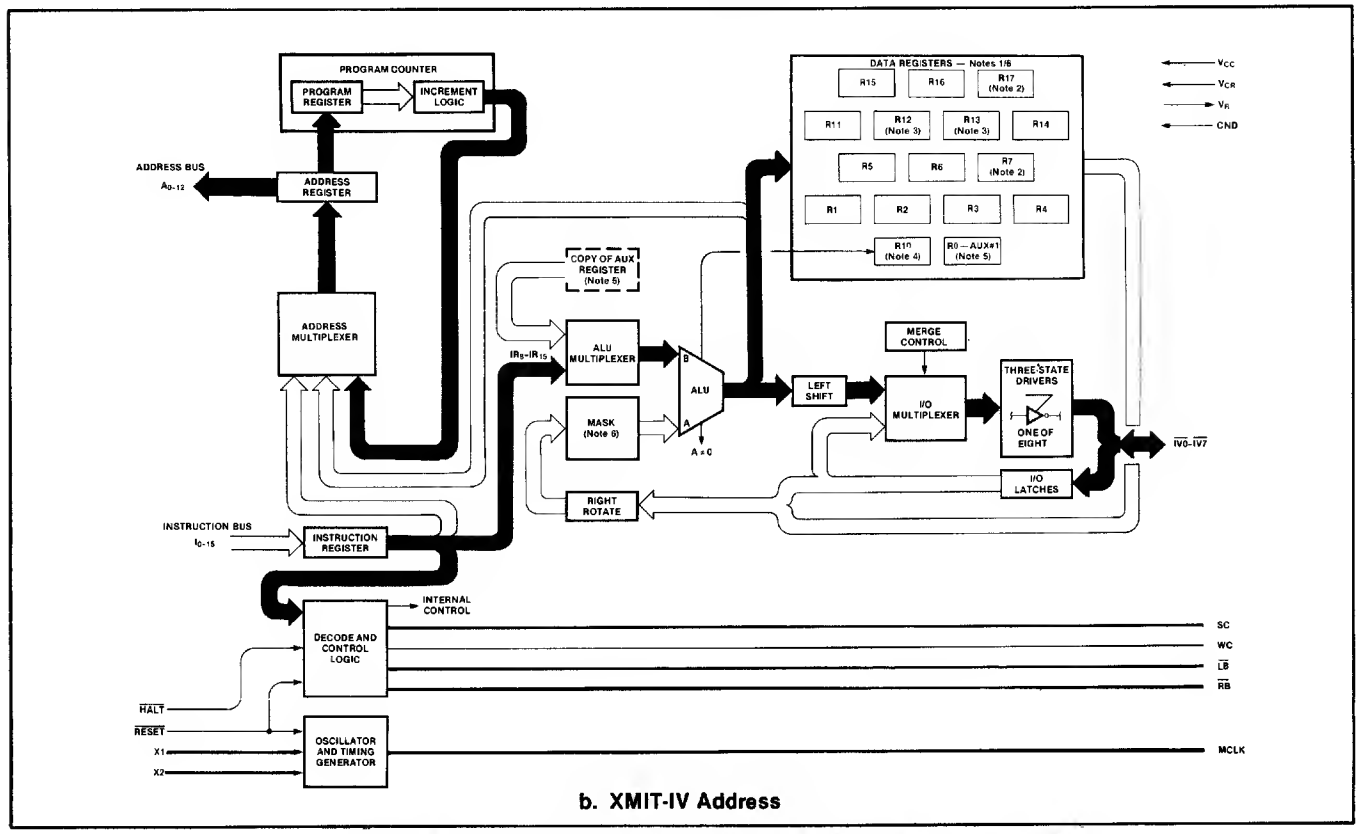
XMIT-IV Address

Paths for the address, the instruction, the data flow, and the required control lines for the XMIT-IV Address instruction are shown in Figure 2-13b. Operational parameters for this instruction are described below.

Format: See Format XR, Figure 2-8
Description: Enable I/O device on the IV bus bank specified by D, whose address is the 8-bit integer specified by J, where,
D = destination bank of IV bus for address data
J = 8-bit address of port to be enabled
Permitted Operand Values:
D = 07 (Left Bank) or 17 (Right Bank)
J = $0\text{-}377_8$
Note: The address is also stored in the selected register, R7 or R17.



a. XMIT-Register



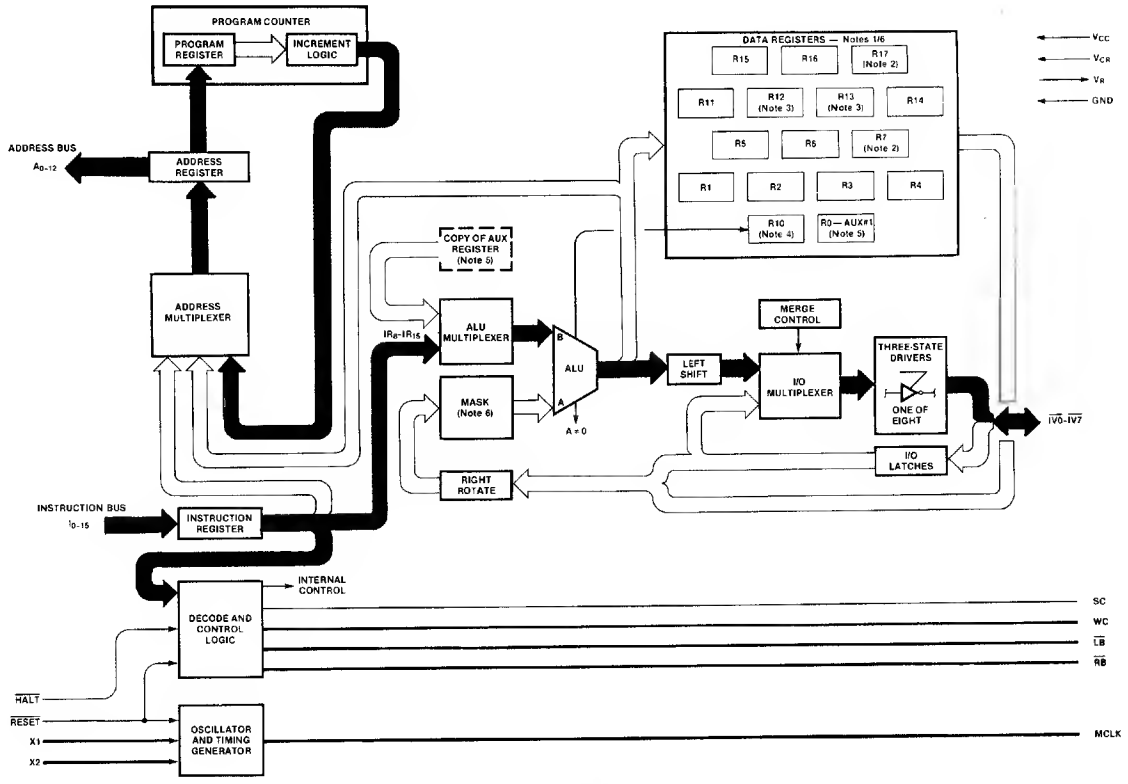
b. XMIT-IV Address

Figure 2-13. Data Flow for XR Format—See Figure 2.8 for Instruction Format

XMIT-8 Bits Immediate IV Bus

Paths for the address, the instruction, the data flow, and the required control lines for the XMIT-8 Bits Immediate IV Bus are shown in Figure 2-13c. Operational parameters for this instruction are described below.

Format: See Format XR, Figure 2-8
Description: Transmit value of 8-bit integer specified by J to Left Bank (R12) or to Right Bank (R13) as data. The contents of R12 and R13 remain unchanged.
Permitted Operand Values:
D =12 (Left Bank) or 13 (Right Bank)
J =0-377_8



c. XMIT-8-Bits Immediate IV Bus

Figure 2-13 (Continued)

2.4.5 Operations That Use Format JB

Operations that use Format JB are NZT-IV Bus and XEC-IV Bus.

NZT-IV Bus

Paths for the address, the instruction, the data flow, and the required control lines for the NZT-IV Bus instruction are shown in Figure 2-14a. A summary of operational parameters pertaining to this instruction is shown below.

Format: See Format JB, Figure 2-8

Description: If contents of IV bus source data field as specified by S and L is non-zero, replace 5 LSBs of both Address Register and Program Counter with the 5-bit integer specified by J; if the contents of the source field is zero, increment the Program Counter by one, where,

S_0 = least significant bit of IV input data field after rotation.

S_1 = specifies bank of IV bus; 2 specifies Left Bank and 3 specifies Right Bank.

L = number of bits in masked field.

J = 5-bit integer used in address modification.

The order of operation is:

- Read contents of selected IV data, as specified by S_1, into I/O latches.
- Right rotate the input data field as specified by S_0.
- Mask the least significant L bits of the rotated data.
- Test contents of masked field.
- If contents of masked field are not zero, replace 5-LSBs of Address Register and Program Counter with 5-bit integer in J-field.

Permitted Operand Values:

S_0 = 0-7

S_1 = 2 or 3

L = 0-7

J = $0\text{-}37_8$

XEC-IV Bus

Paths for the address, the instruction, the data flow, and the required control lines for the XEC-IV Bus instruction are shown in Figure 2-14b. A summary of operational parameters pertaining to this instruction is shown below.

Format: See Format JB, Figure 2-8

Description: Execute instruction at the address formed by replacing 5-LSBs of Address Register with the 5-bit sum of J and the contents of the IV Bus field, as specified by S and L, where,

S_0 = least signicant bit of IV input data field after rotation.

S_1 = specifies bank of IV bus; 2 specifies Left Bank and 3 specifies Right Bank.

L = number of bits in masked field.

J = 5-bit integer used in address modification.

The order of operation is:

- Read contents of selected IV data as specified by S_1 into I/O latches.
- Right rotate the input data field as specified by S_0.
- Mask the least significant L bits of the rotated data.
- Add masked field to 5-bit integer specified by J.
- Replace 5-LSBs of Address Register with 5-bit result of the ADD operation.

The Program Counter is not altered by the XEC instruction, that is, the original address within the page ($0 \leq J \leq 37_8$) is retained. During the instruction to be executed, the Program Counter is incremented by one in the normal manner to point to the instruction following the XEC instruction. However, if the executed instruction is a JMP or NZT, the Program Counter can be changed to the jump address and instruction execution does not return to the address following the XEC instruction.

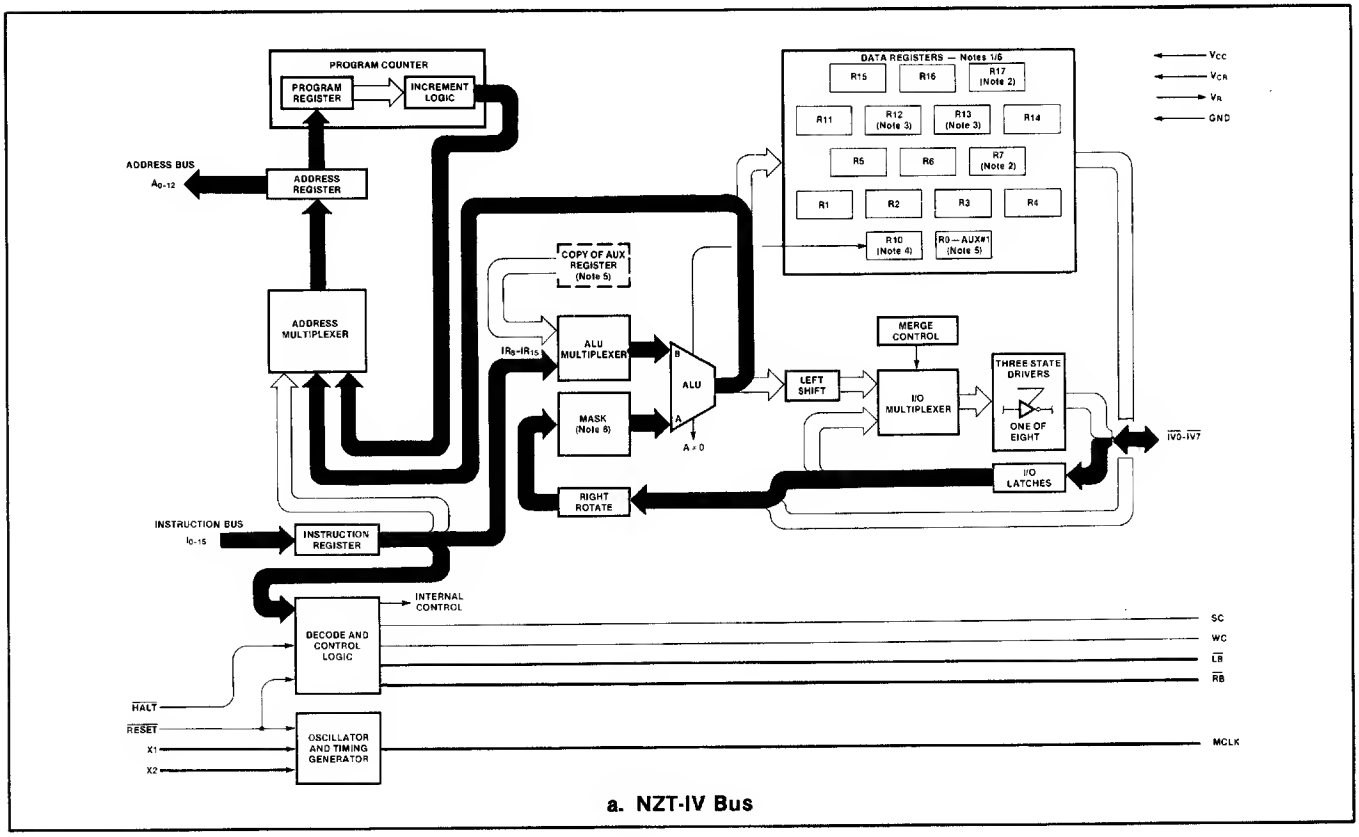
Permitted Operand Values:

S_0 = 0-7

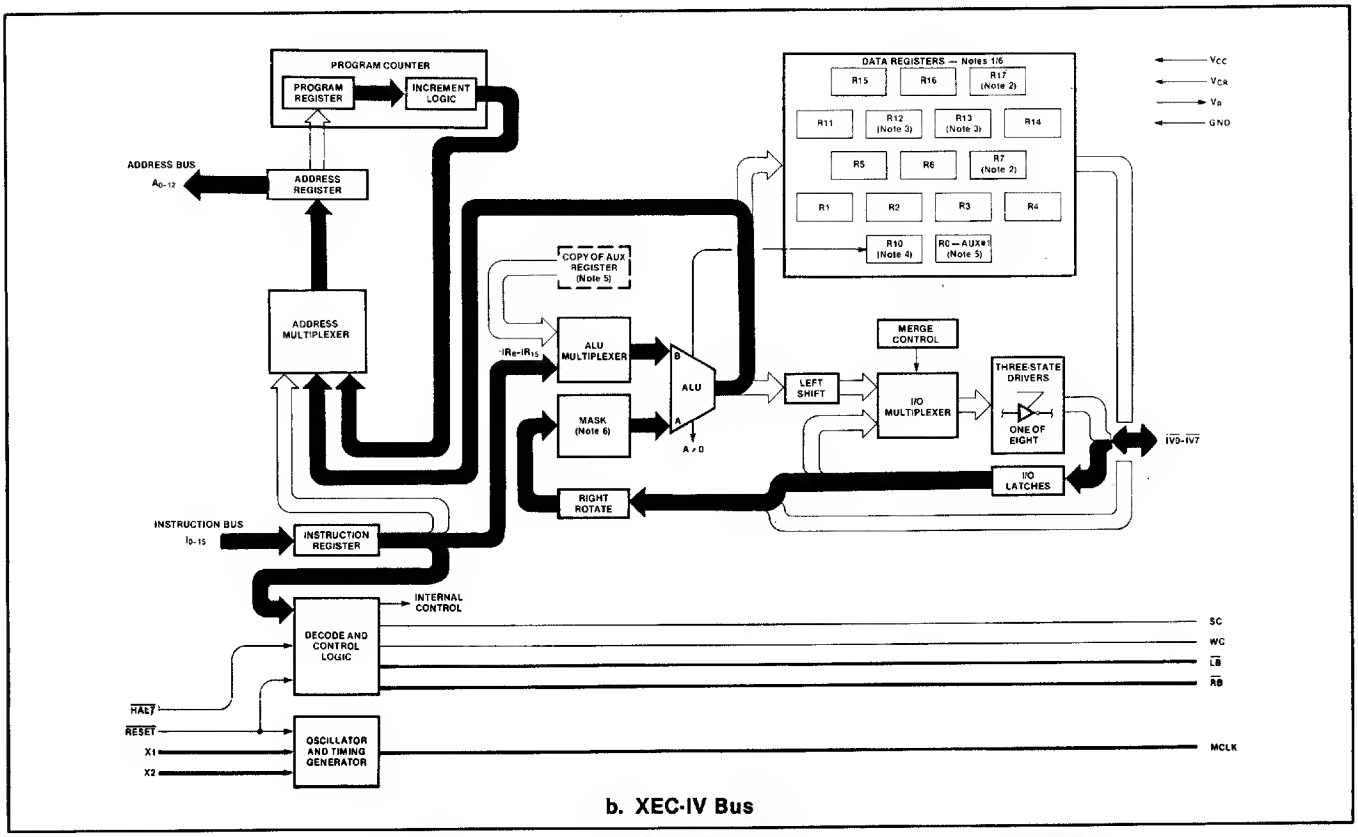
S_1 = 2 or 3

L = 1-5

J = $0\text{-}37_8$ [When sum of S + J is greater than 37_8 only the 5-LSBs are used; overflow (OVF) Register R10 is not changed.]



a. NZT-IV Bus



b. XEC-IV Bus

Figure 2-14. Data Flow for JB Format—See Figure 2.8 for Instruction Formats

2.4.6 Operation That Uses Format XB

The XB format is used to implement a XMIT Variable-Bit Field Immediate IV Bus instruction. Paths for the address, the instruction, the data flow, and the required control lines for this operation are shown in Figure 2-15; operational parameters are described below.

Format: See Format XB, Figure 2-8

Description: Transmit the least significant L bits of the field to the L-bit field of the IV bus specified by D; if L is greater than 5-bits, the most significant bits of the destination field are filled with zeroes. Operand designations are,

D_0 = bit position of IV data with which LSB of J-field data is to be aligned; this implies that the input data field is left-shifted until bit 7 is aligned with bit D_0.

D_1 = specifies bank of IV bus; 2 specifies Left Bank and 3 specifies Right Bank.

L = number of bits in destination field.

J = 5-bit integer.

The order of operation is:

- Read contents of destination IV port into I/O latches.
- Read 5-LSBs of instruction word.
- Left shift data field as specified by D_0.
- Merge shifted field of data, as specified by L, with contents of I/O latches and output result to IV bus. (Note: Data in the I/O latches outside the field specified by D_0 and L is not altered.)

Permitted Operand Values:

D_0 = 0-7

D_1 = 2 or 3

L = (L = 0 selects an 8-bit field)

J = $0\text{-}37_8$

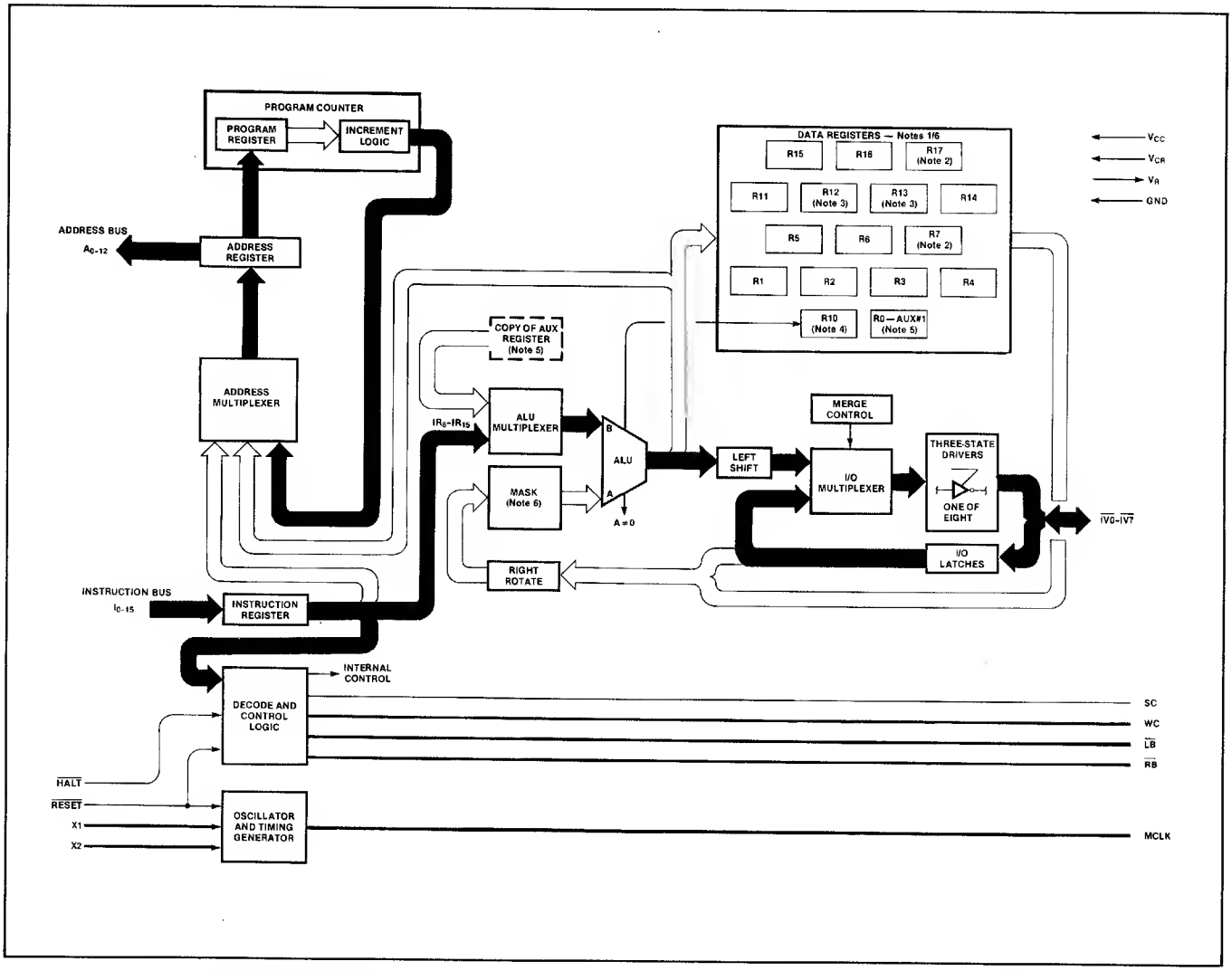


Figure 2-15. Data Flow for XB Format (XMIT IV Bus)—See Figure 2-8 for Instruction Format

2.4.7 Operation That Uses Format JA

The JA format is used to implement a JMP instruction. Paths for the address, the instruction, the data flow, and the required control lines for this operation are shown in Figure 2-16; operational parameters are described below.

Format: See Format JA, Figure 2-8
Description: Jump to the instruction address specified by A and continue normal program execution from that address. The contents of the 13-bit A-field are loaded into both the Address Register and Program Counter. The next instruction to be executed is then the instruction at the new address.
Permitted Operand Value:
A $=0\text{-}17777_8\ (8191_{10})$

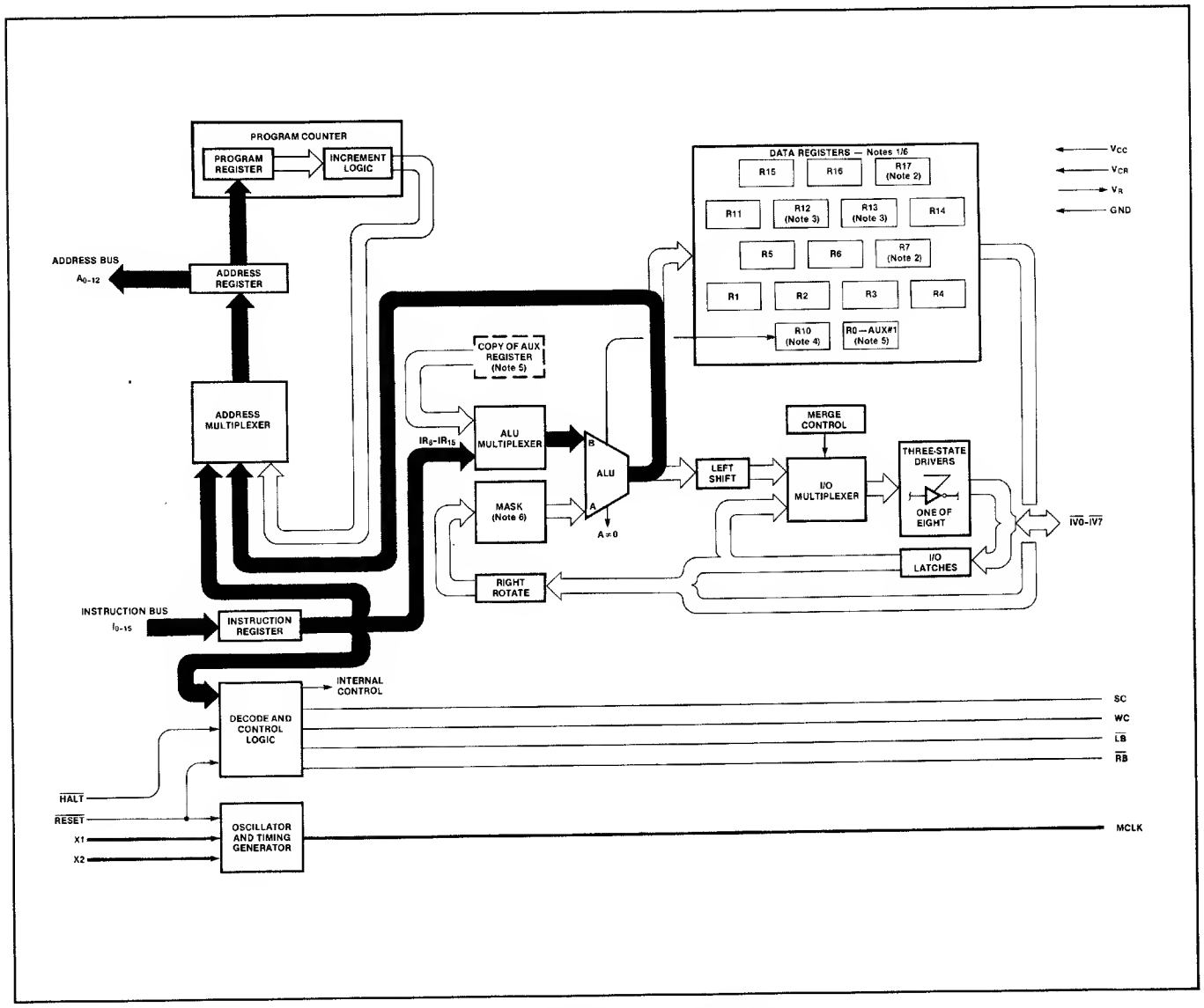


Figure 2-16. Data Flow for JA Format (JUMP)—See Figure 2-8 for Instruction Format

2.5 HALT AND RESET OPERATIONS

Processing of data by the 8X305 may be suspended by use of the $\overline{\text{HALT}}$ input to the chip. Precise timing relationships of this to other signals may be found in the 8X305 Data Sheet.

The $\overline{\text{HALT}}$ signal is sampled via internal chip logic prior to the end of the first internal quarter of each instruction cycle. If, when sampled, the $\overline{\text{HALT}}$ signal is low (active), a halt is immediately invoked and the current instruction cycle is terminated. The halt cycle does not inhibit MCLK nor does it affect any internal registers of the 8X305. As long as the $\overline{\text{HALT}}$ line is low (active), the SC and WC lines are low (inactive) and the IV bus remains in the three-state mode of operation. The halt cycle continues until, when sampled during the next first quarter cycle the $\overline{\text{HALT}}$ line is found to be high. At this time the 8X305 will continue processing with the instruction currently being presented by the program memory, which should be the same instruction as was about to be executed when the halt was invoked.

Note also that changes made in the instruction being presented to the 8X305 will affect the configuration of the $\overline{\text{LB}}$ and $\overline{\text{RB}}$ signals even during the halt condition of the device, as $\overline{\text{HALT}}$ does not disable $\overline{\text{LB}}$ and $\overline{\text{RB}}$.

The 8X305 may also be brought to a reset state by use of the $\overline{\text{RESET}}$ input to the chip.

The $\overline{\text{RESET}}$ line can be driven from a high (inactive) state to a low (active) state at any time with respect to the system clock, that is, the reset function is asynchronous. To ensure proper operation, the $\overline{\text{RESET}}$ line must be held low (active) for at least one full instruction cycle time. When the line is driven from a high state to a low (active) state, several events occur. The precise instant of occurrence is a function of the propagation delay for that particular event. These events are:

- The Program Counter and Address Register are set to an all-zero configuration and remain in that state as long as the $\overline{\text{RESET}}$ line is low. Other than PC and AR, $\overline{\text{RESET}}$ does not affect internal registers.
- The IV bus goes three-state and remains in that mode as long as the $\overline{\text{RESET}}$ line is low.
- The Select Command and Write Command signals are driven low (inactive) and remain low as long as the $\overline{\text{RESET}}$ line is low.
- The Left Bank/Right Bank signals are high (inactive) for the period in which the $\overline{\text{RESET}}$ line is low.

During the time $\overline{\text{RESET}}$ is low (active), MCLK is inhibited. If the $\overline{\text{RESET}}$ line is driven low during the last two quarter-cycles, MCLK may be shortened for that particular machine cycle. When the $\overline{\text{RESET}}$ line is driven high (inactive) then one-quarter to one full instruction cycle later, a normal MCLK pulse precedes the resumption of normal operation. As long as the $\overline{\text{RESET}}$ line is low (active), the $\overline{\text{HALT}}$ signal is not sampled by internal logic of the 8X305.

Chapter 3

INSTRUCTION SET

The instruction set of the 8X305 enables the device to perform a wide variety of complex tasks within a single instruction cycle. A general description of the instruction format can be found in Chapter 2 (Functional Operation). An overview of the structure of the Instruction Set is given in Table 3-1. Individual commands and their functions are described in subsequent paragraphs; these commands are defined in MCCAP notations. Refer to the MCCAP manual for further detail.

Table 3-1. Instruction Set Summary

INSTRUCTION WORD	DESCRIPTION	STATE OF CONTROL SIGNAL DURING INSTRUCTION CYCLE — SEE FIGURE 3		
		CONTROL SIGNAL	INPUT PHASE	OUTPUT PHASE
CLASS = MOVE OPCODE = 0 OPERATION = (S) → D				
$\overline{IV}$ Bus-to-Register (Note) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | S (S_1 : S_0) | L | D $S = 20_8$-37_8 $D \neq 10_8, 20_8$-37_8	Move right-rotated $\overline{IV}$ bus (source) data specified by the S-field to internal register specified by the D-field. The L-field specifies the length of source data starting from the LSB-position and, if less than 8 bits, the remaining bits are filled with zeros.	SC	L	H if $D = 07_8, 17_8$
		WC	L	L
		$\overline{LB}$	L if $S = 20_8$-27_8	L if $D = 07_8$
		$\overline{RB}$	L if $S = 30_8$-37_8	L if $D = 17_8$
$\overline{IV}$ Bus-to-$\overline{IV}$ Bus (Note) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | S (S_1 : S_0) | L | D (D_1 D_0) $S = 20_8$-37_8 $D = 20_8$-37_8	Move right-rotated $\overline{IV}$ bus (source) data specified by the S-field to the I/O latches. Before outputting on $\overline{IV}$ bus, shift data as specified by the D-field; then merge source and latched I/O data as specified by the L (length) field.	SC	L	L
		WC	L	H
		$\overline{LB}$	L if $S = 20_8$-27_8	L if $D = 20_8$-27_8
		$\overline{RB}$	L if $S = 30_8$-37_8	L if $D = 30_8$-37_8
CLASS = ADD OPCODE = 1 OPERATION = (S) + (AUX) → D				
Same as MOVE instruction class	Same as MOVE instruction class except that contents of AUX (R0) register are ADDed to the source data. If there is a "carry" from MSB, then R10 (OVF) = 1 (overflow), otherwise OVF = 0.	Same as MOVE instruction class		
CLASS = AND OPCODE = 2 OPERATION = (S) ∧ (AUX) → D				
Same as MOVE instruction class	Same as MOVE instruction class except that contents of AUX (R0) register are ANDed with source data.	Same as MOVE instruction class		
CLASS = XOR OPCODE = 3 OPERATION = (S) ⊕ (AUX) → D				
Same as MOVE instruction class	Same as MOVE instruction class except that contents of AUX (R0) register are exclusively ORed with source data.	Same as MOVE instruction class		
CLASS = XEC OPCODE = 4 OPERATION = Refer to Description				
Register Immediate 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | S | J $S \neq 20_8$-37_8 $J = 000_8$-377_8	Execute instruction at current page address offset by J (literal) + (S). Return to normal instruction flow unless a branch is encountered.	SC	L	L
		WC	L	L
		$\overline{LB}$	H	H
		$\overline{RB}$	H	H
	Execute instruction at an address determined by replacing the low-order 8 bits of the Address Register with the following derived sum: Value of literal (J-field) plus contents of internal register specified by S-field The PC is not incremented and the overflow status (OVF) is not changed.			
$\overline{IV}$ Bus Immediate (Note) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | S (S_1 : S_0) | L | J $S = 20_8$-37_8 $J = 00_8$-37_8	Execute instruction at an address determined by replacing the low-order 5 bits of Address Register with the following derived sum: 5-bit value of literal (J-field) plus value of rotated source data specified by S-field. The L-field specifies the length of source data starting from the LSB position and, if less than 8 bits, the remaining bits are filled with zeros; the Program Counter is not incremented and the overflow status (OVF) is not changed.	SC	L	L
		WC	L	L
		$\overline{LB}$	L if $S = 20_8$-27_8	H
		$\overline{RB}$	L if $S = 30_8$-37_8	H

Table 3-1. Instruction Set Summary (continued)

INSTRUCTION WORD	DESCRIPTION	STATE OF CONTROL SIGNAL DURING INSTRUCTION CYCLE — SEE FIGURE 3		
		CONTROL SIGNAL	INPUT PHASE	OUTPUT PHASE
CLASS = NZT OPCODE = 5 OPERATION = Refer to Description				
Register Immediate 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | S | J $S \neq 20_8\text{-}37_8$ $J = 000_8\text{-}377_8$	If data specified by the S-field is not equal to zero, jump to current page address offset by value of J-field; otherwise, increment the Program Counter. If contents of internal register specified by S-field is non-zero, transfer to address determined by replacing the low-order 8 bits of Address Register and Program Counter with "J", otherwise, increment PC.	SC	L	L
		WC	L	L
		$\overline{LB}$	H	H
		$\overline{RB}$	H	H
$\overline{IV}$ Bus Immediate (Note) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | S (S_1 : S_0) | L | J $S = 20_8\text{-}37_8$ $J = 00_8\text{-}37_8$	If right-rotated and masked $\overline{IV}$ bus is non-zero, transfer to address determined by replacing low-order 5 bits of Address Register and Program Counter with "J", otherwise, increment PC. (The L-field specifies the length of source I/O data starting from the LSB-position and, if less than 8 bits, the remaining bits are filled with zeros.)	SC	L	L
		WC	L	L
		$\overline{LB}$	L if $S = 20_8\text{-}27_8$	H
		$\overline{RB}$	L if $S = 30_8\text{-}37_8$	H
CLASS = XMIT OPCODE = 6 OPERATION = J → D				
XMIT, Register 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | D | J $D \neq 10_8, 12_8, 13_8, 07_8, 17_8, 20_8\text{-}37_8$ $J = 000_8\text{-}377_8$	Store 8-bit value specified by "J" into register specified by "D".	SC	L	L
		WC	L	L
		$\overline{LB}$	H	H
		$\overline{RB}$	H	H
XMIT, $\overline{IV}$ Bus Address 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | D | J $D = 07_8, 17_8$ $J = 000_8\text{-}377_8$	Enable I/O device on the bank specified by "D", whose address is the 8-bit integer specified by "J". Address "J" is stored in register "D".	SC	L	H
		WC	L	L
		$\overline{LB}$	H	L if $D = 07_8$
		$\overline{RB}$	H	L if $D = 17_8$
XMIT 8 Bits Immediate, $\overline{IV}$ Bus (Note) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | D | J $D = 12_8\text{-}13_8$ $J = 000_8\text{-}377_8$	Store value of 8-bit integer in the previously enabled I/O port, at the bank destination ($\overline{LB}$ or $\overline{RB}$) specified by "D". Contents of R12 or R13 remain unchanged.	SC	L	L
		WC	L	H
		$\overline{LB}$	H	L if $D = 12_8$
		$\overline{RB}$	H	L if $D = 13_8$
XMIT Variable Bit Field Immediate, $\overline{IV}$ Bus (Note) 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | D (D_1 : D_0) | L | J $D = 20_8\text{-}37_8$ $J = 00_8\text{-}37_8$	Transmit Least Significant "L" bits of "J" field to "L-bit" field of $\overline{IV}$ bus specified by "D"; if "L" is greater than 5 bits, the MSB bits of destination field is filled with zeroes.	SC	L	L
		WC	L	H
		$\overline{LB}$	L if $D = 20_8\text{-}27_8$	L if $D = 20_8\text{-}27_8$
		$\overline{RB}$	L if $D = 30_8\text{-}37_8$	L if $D = 30_8\text{-}37_8$
CLASS = JMP OPCODE = 7 OPERATION = Refer to Description				
Address Immediate 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 OPCODE | A $A = 00000_8\text{-}17777_8$	Jump to address in program storage specified by A-field; this address is loaded into the Address Register and the Program Counter.	SC	L	L
		WC	L	L
		$\overline{LB}$	H	H
		$\overline{RB}$	H	H

Note:
S_0 specifies the LSB of rotated input data field
S_1 specifies the bank of $\overline{IV}$ bus from which source data will be input
D_0 specifies bit position in I/O device with which LSB of processed data will be aligned and
D_1 specifies the bank of $\overline{IV}$ bus which will be the destination.

3.1 MOVE INSTRUCTIONS

Description:

These instructions move data. The contents of S are transferred to D; the contents of S are unaffected. If both S and D are registers, R/L specifies a right rotate of the source data during the move. Otherwise, R/L specifies the length of the source and/or destination I/O data field. If the MOVE is between Left Bank and Right Bank I/O Port, an 8-bit field will always be moved, regardless of the L-field value.

Example:

Store the least significant 3 bits of internal register 5 (R5) in bits 6, 5 and 4 of the I/O Port previously addressed on the Left Bank of the IV bus—see Figure 3-1. Details of the various MOVE instructions are described in subsequent paragraphs.

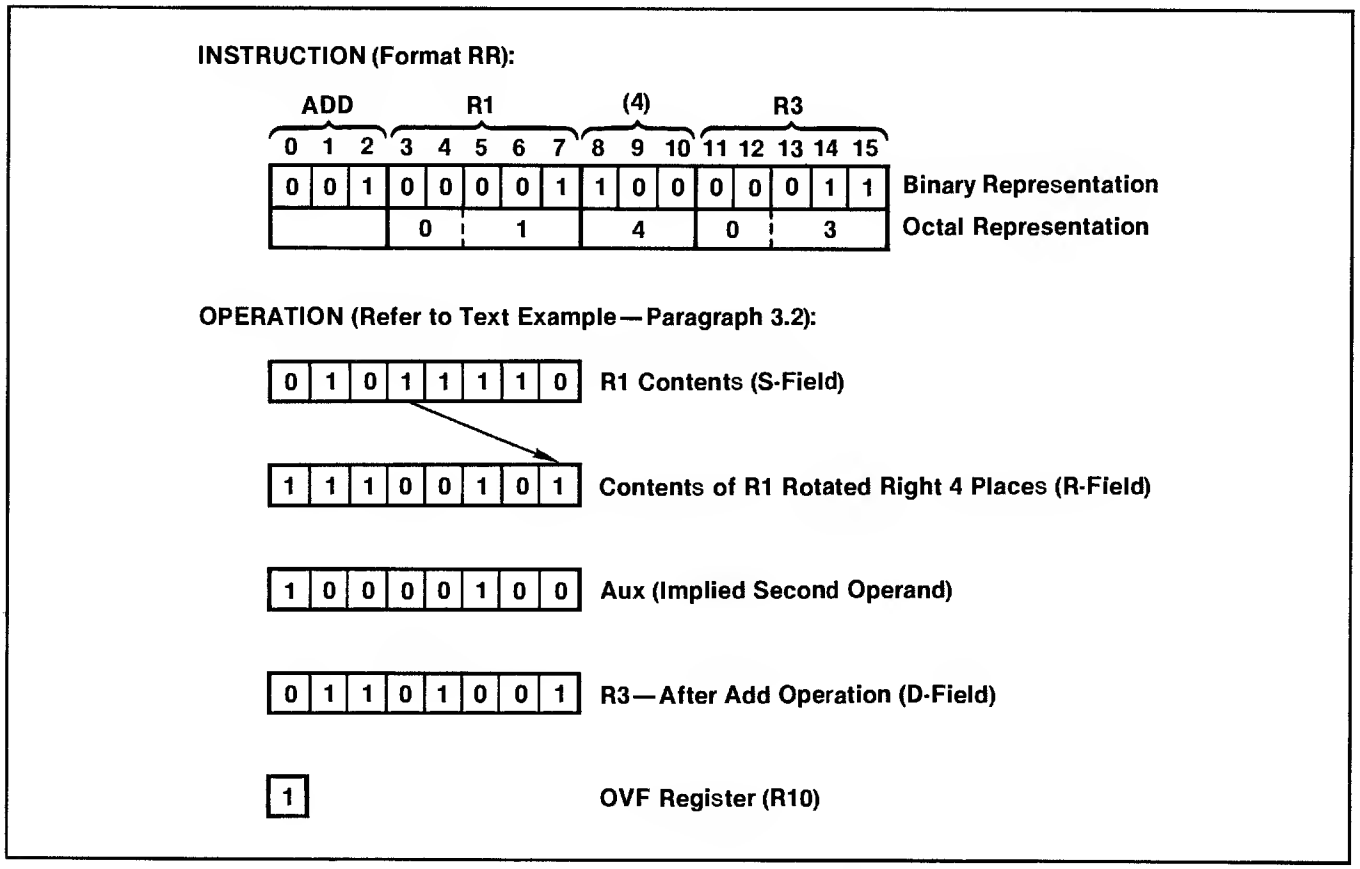
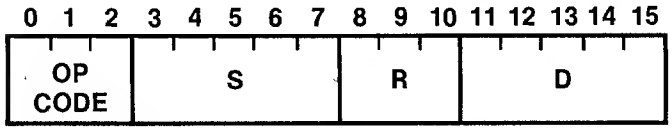


Figure 3-1. Example of MOVE Instruction

3.1.1 MOVE-Register, Register

Format RR:



Description:

The contents of the register specified by S are right rotated as specified by R and placed in the destination register specified by D. The contents of the source register remain unchanged unless it is also the destination. The original contents of the destination register are lost. Note that when registers R7 or R17 specified as the D-field, the instruction is a Register-to-IV-Bus-Address instruction, described below.

S specifies the source register.
R specifies the number of places that the source data is to be right rotated.
D specifies the destination register.

The order of operation is:

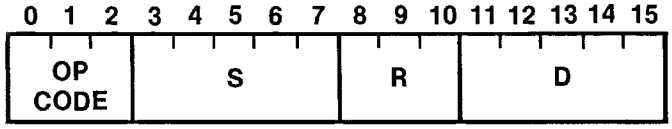
- Read the contents of the source register
- Right rotate the source data R places
- Move the rotated data to the destination register

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 00/01/02/03/04/05/06/11/12/13/14/ 15/16

3.1.2 MOVE-Register, IV Bus Address

Format RR:



Description:

Enable the I/O device on the bank specified by D, whose address is given by the right rotated contents of the register specified by S.

S specifies the source register.

R specifies the number of places that the source data is to be right rotated.

D specifies the destination bank of the IV bus for the address data:

D = 07 specifies the Left Bank
D = 17 specifies the Right Bank

The order of operation is:

- Rotate the source data from register S by R places
- Output the result to the IV bus as an address and also store in register specified by D.

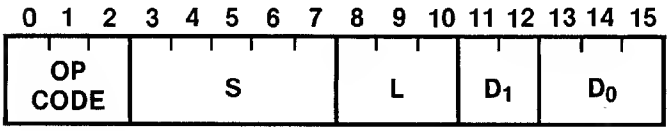
The contents of the source register remain unchanged after the instruction.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/
11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 07/17

3.1.3 MOVE-Register, IV Bus

Format RB:



Description:

Move the least significant L bits of the register specified by S to the selected port on the IV bus, after merging with the port data read into the IV bus latches during the instruction input phase. The port was selected during a preceding IV bus address instruction.

S specifies the source register.

L specifies the length (number of bits) of the masked data field that is to be merged with the latched port data. L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:

$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank

D_0 specifies the bit position in the latched port data with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that its LSB is aligned with bit D_0 of the IV bus.

The order of operation is:

- Read the contents of the selected port into the IV latches
- Read the contents of the source register
- Shift the source data as specified by D_0
- Merge the least significant L bits of the shifted source data with the data in the IV latches
- Output the modified data field to the IV bus

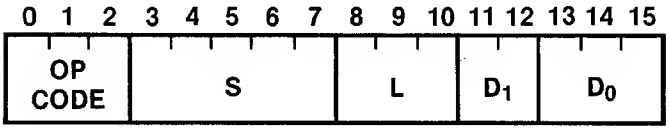
Note that the original port data outside the merged L-bit field remains unaltered. The contents of the source register remain unchanged.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.1.4 MOVE-IV Bus, Register

Format RB:



Description:

Move the L-bit field of the data from the selected port on the IV bus to the least significant L bits of the register specified by D.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_0 = 3$ selects the Right Bank

L specifies the least significant bit of the desired input data field. L specifies the length (number of bits) of the masked field of the port data. L = 0 selects an 8-bit field.

D specifies the address of the internal destination register.

The order of operation is:

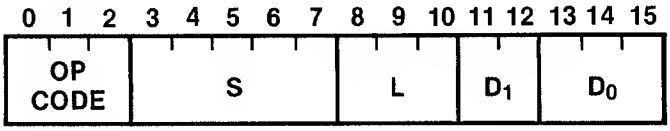
- Read data on IV bus bank specified by S_1 to input latches
- Right rotate the input data field as given by S_0
- Mask off the least significant L bits of the rotated field
- Move the masked field to the least significant L bits of the destination register, with zeros in the unmasked positions

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 00/01/02/03/04/05/06/11/12/13/14/15/16

3.1.5 MOVE-IV Bus, IV Bus

Format RB:



Description:

Move the variable length field, specified by S_0 and L, from the port on the IV bus bank specified by S_1 to the field and bank specified by D_1 and D_0.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field that is to be merged with the existing IV bus data. L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank

D_0 specifies the bit position in the data from the input latches with which the least significant bit of the data field should be aligned. This means that the data field is left-shifted so that its LSB is aligned with bit D_0 of the input latches.

The order of operation is:

- Read the data from the source port into the input latches
- Right rotate the input data and as specified by S_0
- Mask off the least significant L bits
- Shift left as specified by D_0
- Merge the L-bit field with the data from the input latches
- Output 8 bits of data to IV bus

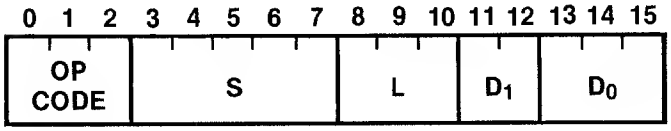
Since the data in the IV latches originated in the source port, data in the destination port will be lost if it is located on the other bank. The original values of the source field outside of the masked field will be preserved.

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.1.6 MOVE-IV Bus, IV Bus Address

Format RB:



Description:

Move the data from the IV bus as specified by S_1, right rotate the data field until bit S_0 is in the least significant position, mask the least significant L bits and output the result to the bank of the IV bus specified by D, as an I/O Port address. Bits of the output field outside the mask are set to zero.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field. L = 0 selects an 8-bit field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies Left Bank address (IVL) and register R7
D = 17 specifies Right Bank address (IVR) and register R17

The order of operation is:

- Input data from the IV bus
- Right rotate the input data as given by S_0
- Mask the least significant L bits
- Output result with zeros in positions outside mask and also place data in destination register
- Output result to IV bus with SC high to enable port, LB or RB low

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 07/17

3.2 ADD INSTRUCTIONS

Description:

These instructions perform unsigned 2's complement 8-bit addition. The contents of S are added to the contents of the AUX Register. The result is stored in D; if overflow occurs during the addition, bit 7 of R10, the OVF Register, is set to one. Otherwise it is set to zero. If both S and D are registers, R/L specifies a right rotate of the source (S) data before the operation. Otherwise, R/L specifies the length of the source and/or destination I/O data fields. S and AUX are unaffected unless specified as the destination.

Example:

Add the contents of R1 rotated 4 places to AUX and store the result in R3—see Figure 3-2. Details of the various ADD instructions are described in subsequent paragraphs.

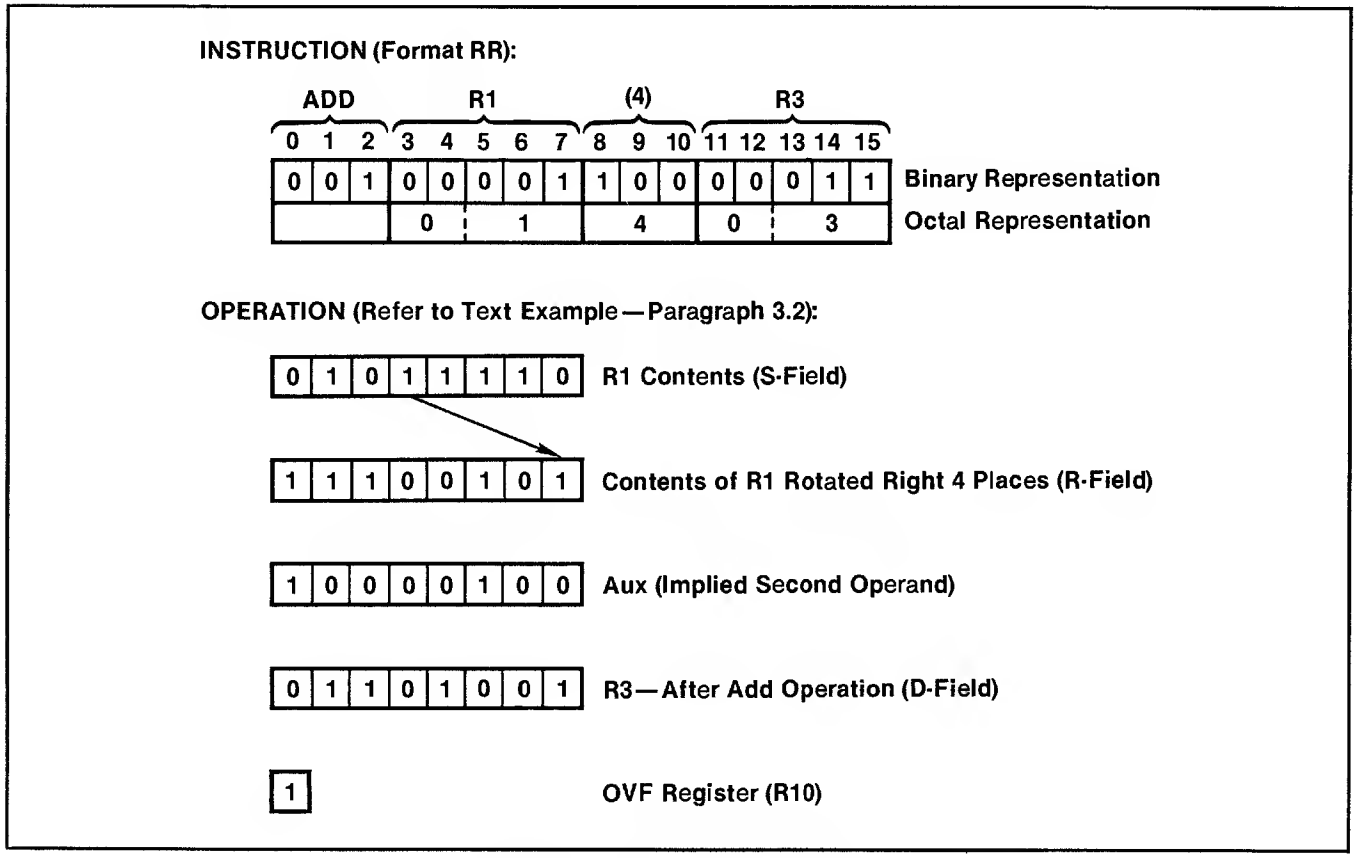
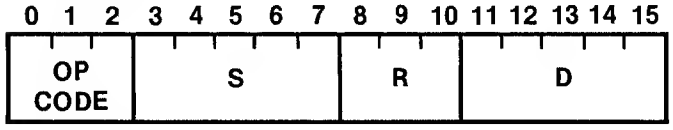


Figure 3-2. Example of ADD Instruction

3.2.1 ADD-Register, Register

Format RR:



Description:

Add the right rotated (if rotation is specified) contents of register S to the contents of the AUX register and place the result in register D. If overflow occurs during the addition, the LSB of the OVF register is set to 1, otherwise it is set to 0. Note that specifying either R_7 or R_{17} in the D-field results in a IV bus-IV bus address instruction, which is described below.

S specifies the source register.
R specifies the number of places that the source data is to be rotated.
D specifies the destination register.

The order of operation is:

- Read the contents of the source register
- Right rotate the source data.
- Add the right rotated data to the contents of the AUX register
- Set the overflow indication as appropriate in the OVF register
- Move the result to the destination register

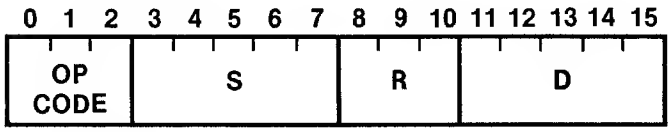
The contents of the source and AUX registers remain unchanged after the instruction unless one of these is also specified as the destination.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 00/01/02/03/04/05/06/11/12/13/14/15/16

3.2.2 ADD-Register, IV Bus Address

Format RR:



Description:

Enable the I/O Port whose address is given by the sum of the right rotated (if rotation is specified) contents of the source register and the contents of the AUX register on the IV bus bank specified by D. This port address is also stored in the selected destination register.

S specifies the source register.
R specifies the number of places that the source data is to be rotated.
D specifies the destination bank of the IV bus for the address data:
D = 07 specifies Left Bank address (IVL) and register R7
D = 17 specifies Right Bank address (IVR) and register R17

The order of operation is:

- Rotate the copied contents of register (S) by R places
- Add the rotated data field to the contents of AUX
- Set the overflow indication as appropriate in the OVF register
- Output the sum to the IV bus as an address and also to the selected internal register

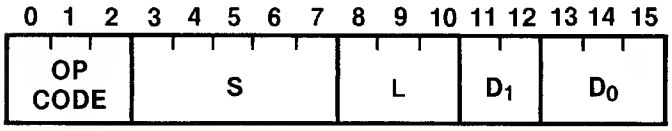
The contents of the source register remain unchanged after the instruction.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 07/17

3.2.3 ADD-Register, IV Bus

Format RB:



Description:

Add the contents of the source register to the contents of the AUX register and move the least significant L bits of the result to the selected I/O Port on the IV bus bank given by D_1, after shifting as required by D_0.

S specifies the source register.

L specifies the length (number of bits) of the masked field that is to be merged with the latched port data. L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank

D_0 specifies the bit position in the latched port data with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that the LSB is aligned with bit D_0 of the IV bus.

The order of operation is:

- Read the contents of the destination port into the IV latches
- Read the contents of the source register and add to the contents of the AUX register
- Set the overflow indication as appropriate in the OVF register
- Shift the result left as specified by D_0
- Merge the shifted data field with the original contents of the I/O Port and output to the IV bus

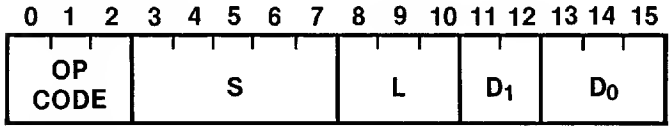
Note that the bits of the output data field outside the L-bit masked field retain their original values. The contents of the source register remain unchanged after the instruction.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.2.4 ADD-IV Bus, Register

Format RB:



Description:

Add the L-bit field of the I/O Port source data to the contents of the AUX register and place the result in the destination register. Set the overflow indicator as appropriate.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field. L = 0 selects an 8-bit field.

D specifies the address of the internal destination register.

The order of operation is:

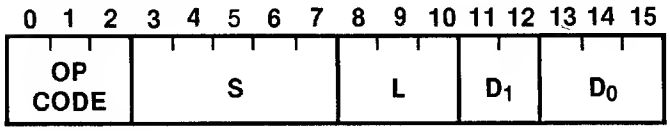
- Read the source I/O Port data into the input latches
- Right rotate the input data as given by S_0
- Mask the rotated data field as specified by L
- Add the masked data to the contents of the AUX register
- Set the overflow indicator as appropriate in the OVF register
- Move the 8 bit result of the addition to the destination register

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 00/01/02/03/04/05/06/11/12/13/14/15/16

3.2.5 ADD-IV Bus, IV Bus

Format RB:



Description:

Add the L-bit field of the source I/O Port data to the contents of the AUX register and move the least significant L bits of the result to the IV bus field specified by D_0.

Transfers to opposite banks are 8-bit transfers, regardless of the value of the L-field.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field that is to be processed and merged with the existing IV bus data. L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank

D_0 specifies the bit position in the IV bus with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that the LSB is aligned with bit D_0 of the IV bus.

The order of operation is:

- Read the data from the source port into the IV latches
- Right rotate the input data as specified by S_0
- Mask off the least significant L bits
- Add the L-bit field to the contents of the AUX register
- Left-shift the sum as given by D_0
- Merge the least significant L bits of the shifted field with the contents of the IV latches
- Output the merged 8-bit field to the bank of the IV bus given by D_1

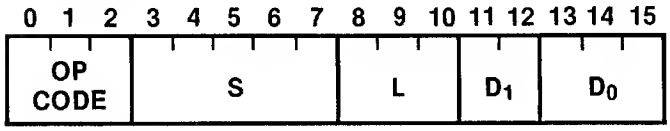
Since the data in the IV latch originated in the source port, original data in the destination port will be lost if it is located on the other bank. The original values of the source field outside of the masked field will be preserved in the destination.

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.2.6 ADD-IV Bus, IV Bus Address

Format RB:



Description:

Enable the port, on the IV bus bank specified by D, whose address is given by the sum of the L-bit field of the source data and the contents of the AUX register. Destination data is also placed in the internal register specified by the D-field.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field. L = 0 selects an 8-bit field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies Left Bank address (IVL) and register R7
D = 17 specifies Right Bank address (IVR) and register R17

The order of operation is:

- Read the data from the I/O Port into the input latches
- Right rotate the input data as given by S_0
- Mask off the least significant L bits
- Add the masked field to the contents of the AUX register
- Set the overflow indicator as appropriate in the OVF register
- Output the data as in IV bus address at the bank specified by D and store in the selected register

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 07/17

3.3 AND INSTRUCTIONS

Description:

These instructions perform logical AND operations. The AND of the source field and the AUX register is stored into the destination. If both S and D are registers, R/L specifies a right rotate of the source (S) data before the AND operation. Otherwise R/L specifies the length of the source and/or destination I/O data fields. S and AUX are unaffected unless specified as a destination.

Example:

Store the AND of the selected Right Bank I/O Port and AUX in R4. The Right Bank data field is 4 bits long and located in bits 2, 3, 4 and 5—see Figure 3-3. Details of the various AND instructions are described in subsequent paragraphs.

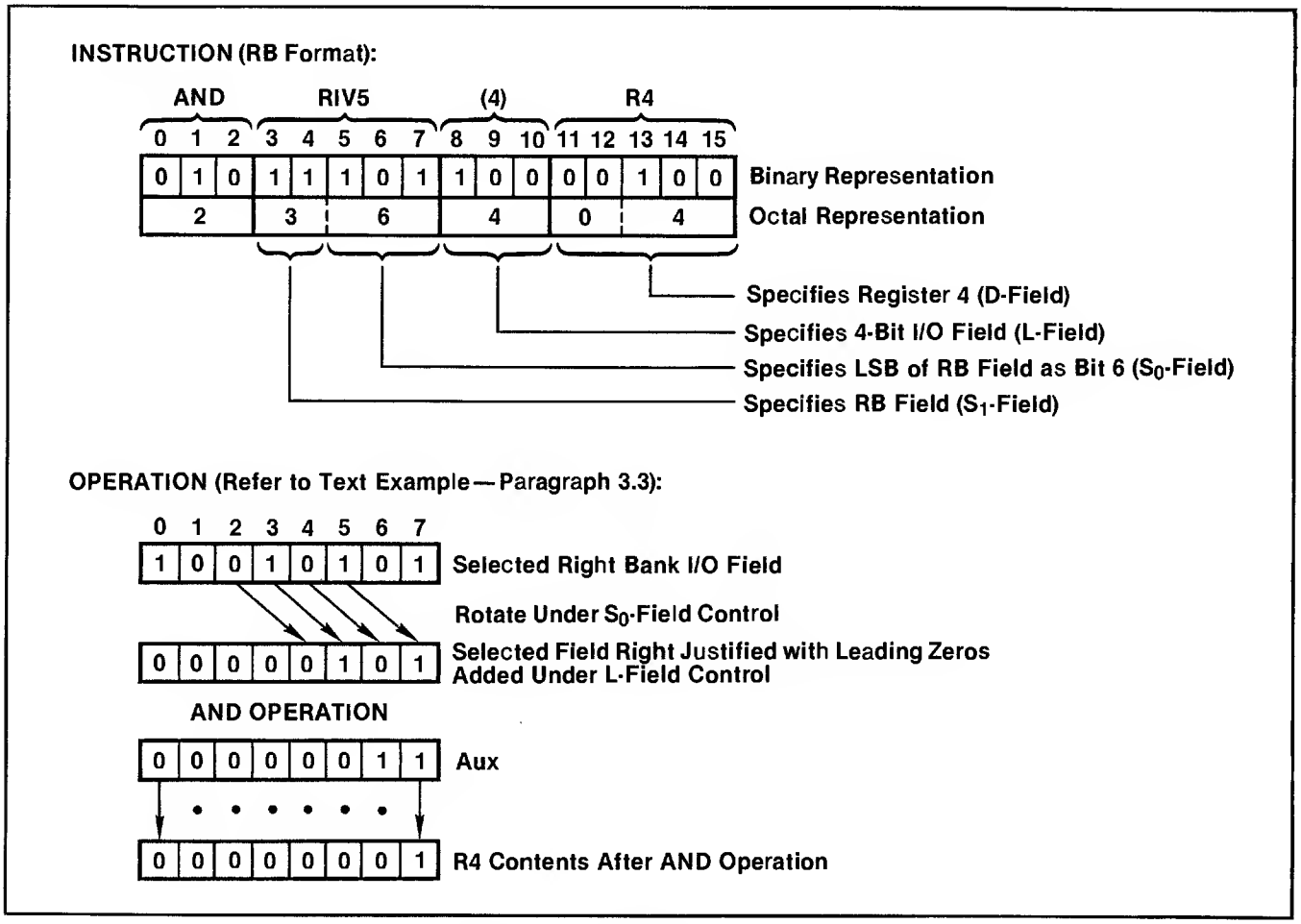
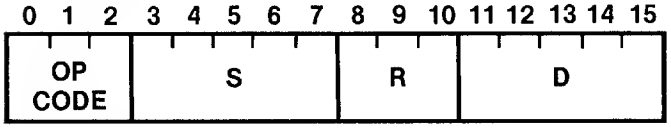


Figure 3-3. Example of AND Instruction

3.3.1 AND-Register, Register

Format RR:



Description:

AND the right rotated (if rotation is specified) contents of register S with the contents of the AUX register and place the result in register D. The original contents of D are not saved.

S specifies the source register.
R specifies the number of places that the source data is to be rotated.
D specifies the destination register.

The order of operation is:

- Read the contents of the source register
- Right rotate the source data
- AND the right rotated data to the contents of the AUX register
- Move the result to the destination register

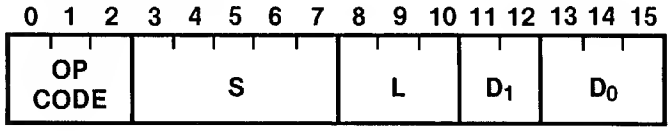
The contents of the source and AUX registers remain unchanged after the instruction unless one of these is also the destination register.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 00/01/02/03/04/05/06/11/12/13/14/15/16

3.3.2 AND-Register, IV Bus Address

Format RB:



Description:

Enable the I/O Port on the IV bus bank specified by D, whose address is given by the AND operation on the right rotated (if rotation is specified) contents of the source register and the contents of the AUX register.

S specifies the source register.
R specifies the number of places that the source data is to be rotated.
D specifies the destination bank of the IV bus for the address data:

D = 07 specifies Left Bank address and register R7
D = 17 specifies Right Bank address and register R17

The order of operation is:

- Read the contents of the source register
- Right rotate the source data
- Move the rotated data to the destination register

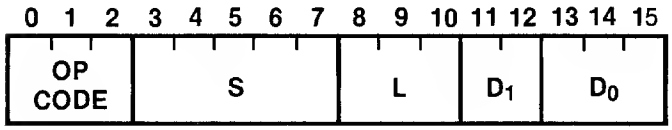
The contents of the source register remain unchanged after the instruction.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 07/17

3.3.3 AND-Register, IV Bus

Format RB:



Description:

Perform an AND operation on the source register contents with the contents of the AUX register and move the least significant L bits of the result to the destination I/O Port.

S specifies the source register.
L specifies the length (number of bits) of the masked field that is to be merged with the existing port data. L = 0 selects an 8-bit field.
D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank
D_0 specifies the bit position in the latched bus data with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that the LSB is aligned with bit D_0 of the IV bus.

The order of operation is:

- Read the contents of the destination port into the input latches
- Read the contents of the source register and AND with the contents of the AUX register
- Left shift the result as specified by D
- Merge the shifted data field with the original contents of the IV latches and output to the destination port

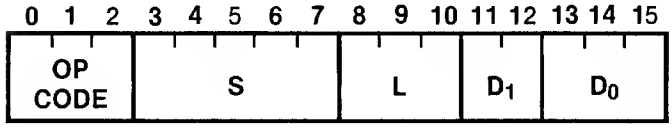
Note that the bits of the output data field outside the L-bit processed field retain their original values. The contents of the source register remain unchanged after the instruction.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.3.4 AND-IV Bus, Register

Format RB:



Description:

Perform an AND operation on the L-bit field of the I/O Port data and the contents of the AUX register, and store the result in the internal destination register.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank
S_0 specifies the least significant bit of the rotated input data field.
L specifies the length (number of bits) of the masked field. L = 0 selects an 8-bit field.
D specifies the address of the internal destination register

The order of operation is:

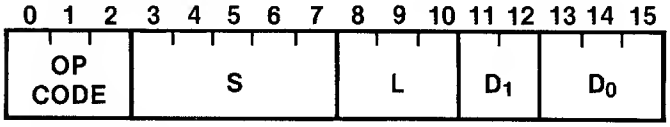
- Read the I/O Port data into the IV latches
- Right rotate the input data as given by S_0
- Mask of the least significant L bits of the rotated field
- AND the masked data to the contents of the AUX register
- Move the 8 bit result to the destination register

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 00/01/02/03/04/05/06/11/12/13/14/15/16

3.3.5 AND-IV Bus, IV Bus

Format RB:



Description:

Perform an AND operation on the L-bit field of the source I/O Port data with the contents of the AUX register, then move the least significant L bits of the result to the destination I/O Port. Bank to bank transfers are 8-bit transfers, regardless of the L-field value.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field that is to be processed and merged with the existing port data. L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank

D_0 specifies the bit position in the data from the input latches with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that the LSB is aligned with bit D_0 of the input latches.

The order of operation is:

- Read the data from the I/O Port into the input latches
- Right rotate the input data as given by S_0
- Mask off the least significant L bits
- AND the L bit field to the contents of the AUX register
- Left-shift the result as given by D_0
- Merge the least significant L bits of the shifted field with the contents of the input latches.
- Output the merged 8-bit field to the bank of the IV bus given by D_1

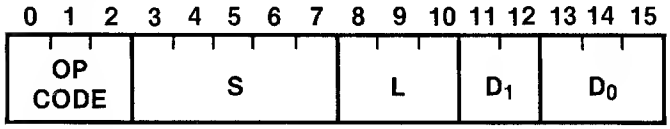
Since the data in the IV latches originated in the source port, original data in the destination port will be lost if it is located on the other bank. The original values of the source field outside of the masked field will be preserved in the destination.

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.3.6 AND-IV Bus, IV Bus Address

Format RB:



Description:

Enable the I/O Port on the bank specified by D, whose address is the result of the AND operation on the L-bit field of the I/O Port data and the contents of the AUX register. Destination data is also placed in the internal register selected by the D-field.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the rotated input data field.

L specifies the length (number of bits) of the masked field. L = 0 selects an 8-bit field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies Left Bank address (IVL) and register R7
D = 17 specifies Right Bank address (IVR) and register R17

The order of operation is:

- Read the data from the port into the IV latches
- Right rotate the input data as given by S_0
- Mask off the least significant L bits
- Perform the AND operation with the contents of the AUX registers
- Output the data as an address to the bank specified by D and store in the selected register

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 07/17

3.4 XOR INSTRUCTIONS

Description:

These instructions perform exclusive-OR operations. The exclusive-OR of the source field and the AUX Register is stored in the destination. If both S and D are registers, R/L specifies a right rotate of the source (S) data before the XOR operation. Otherwise R/L specifies the length of the source and/or destination I/O data fields. S and AUX are unaffected unless specified as a destination.

Example:

Replace the selected I/O data field with the XOR of the source field and AUX. The I/O data field is 5 bits in length and located in bits 3, 4, 5, 6 and 7 of left bank—see Figure 3-4. Details of the various XOR instructions are described in subsequent paragraphs.

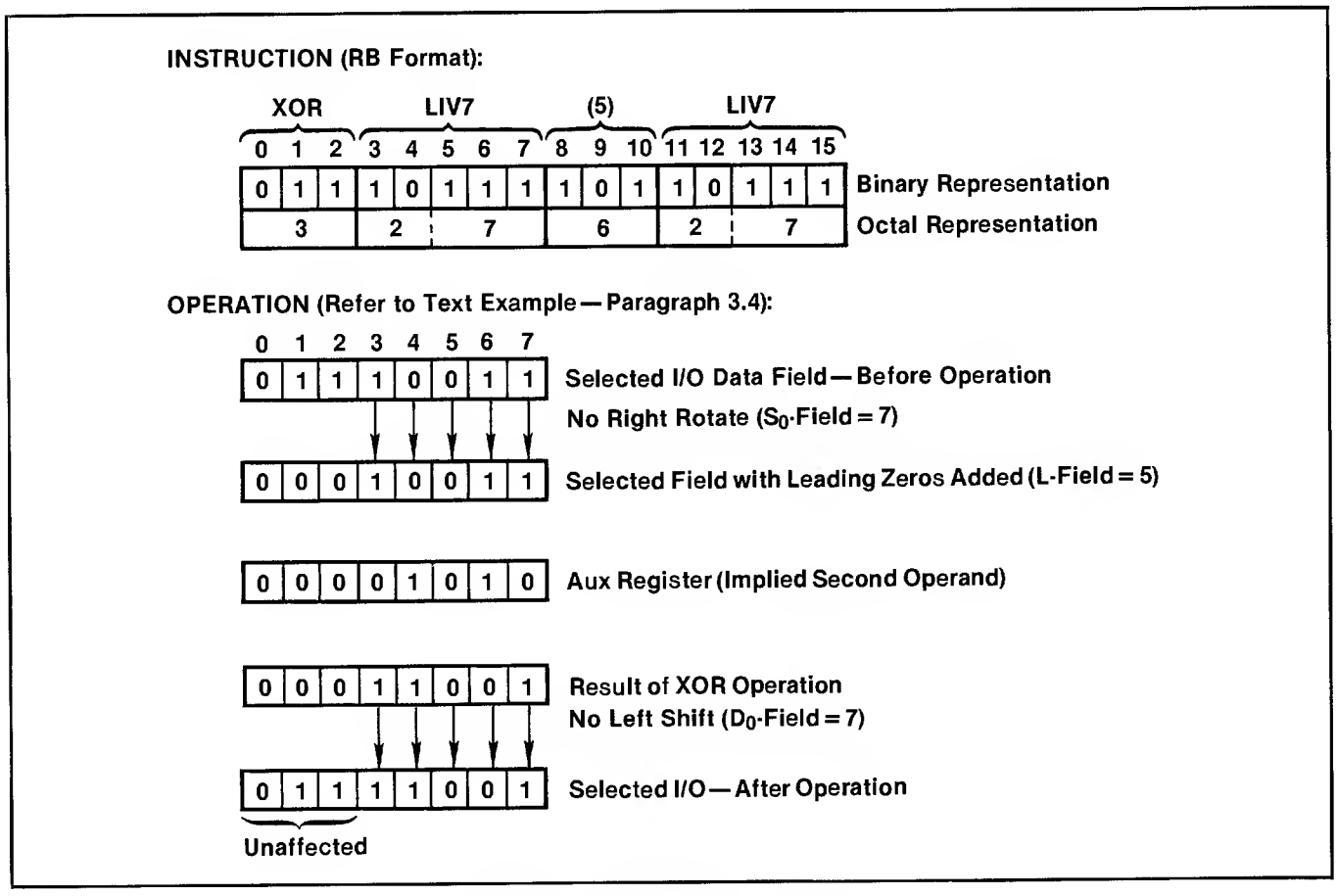
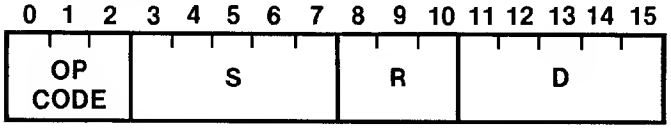


Figure 3-4. Example of XOR Instruction

3.4.1 XOR-Register, Register

Format RR:



Description:

Perform an exclusive-OR operation on the right rotated (if rotation is specified) contents of the source register and the contents of the AUX register, and store the result in the destination register.

S specifies the source register.
R specifies the number of places that the source data is to be rotated.
D specifies the destination register.

The order of operation is:

- Read the contents of the source register
- Right rotate the source data
- XOR the right rotated data with the contents of the AUX register
- Move the result to the destination register

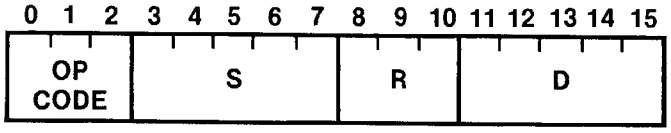
The contents of the source and AUX registers remains unchanged after the instruction unless one of them is specified as the destination register.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 00/01/02/03/04/05/06/11/12/13/14/15/16

3.4.2 XOR-Register, IV Bus Address

Format RR:



Description:

Enable the I/O Port on the bank specified by D, whose address is the result of the XOR operation on the right rotated (if rotation is specified) contents of the source register and the contents of the AUX register.

S specifies the source register.
R specifies the number of places that the source data is to be right rotated.
D specifies the destination bank of the IV bus for the address data:

D = 07 specifies Left Bank address (IVL) and register R7
D = 17 specifies Right Bank address (IVR) and register R17

The order of operation is:

- Rotate the source contents of register S by R places
- XOR the rotated data field to the contents of AUX
- Output the result to the IV bus as an I/O Port address and also store in the destination register

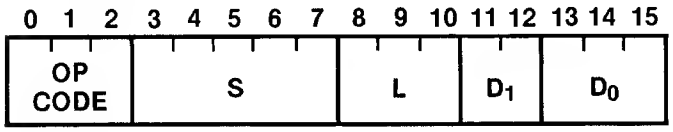
The contents of the source and AUX registers remain unchanged after the instruction, unless also specified as a destination.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
R: 0/1/2/3/4/5/6/7
D: 07/17

3.4.3 XOR-Register, IV Bus

Format RB:



Description:

Perform an exclusive-OR operation on the contents of the source register and the contents of the AUX register. Move the least significant L bits of the result to the L-bit field of the IV bus port.

S specifies the source register.
L specifies the length (number of bits) of the masked field that is to be merged with the latched port data. L = 0 selects an 8-bit field.
D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank
D_0 specifies the bit position in the port data with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that the LSB is aligned with bit D_0 of the IV bus.

The order of operation is:

- Read the data of the destination port into the IV latches
- Read the contents of the source register and perform an XOR operation with the contents of the AUX register
- Left-shift the result as specified by D_0
- Merge the least significant L bits of the shifted field with the contents of the IV latches
- Output the merged data to the destination port

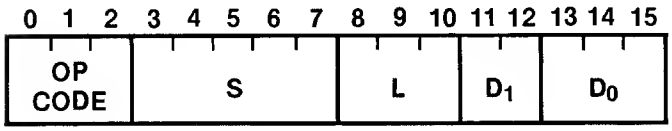
Note that the bits of the output data field outside the L-bit masked field retain their original values. The contents of the source register remain unchanged after the instruction.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.4.4 XOR-IV Bus, Register

Format RB:



Description:

Perform an "XOR" operation with the L-bit field of the IV bus source data and the contents of the AUX register. Move the 8-bit result to the register specified by D.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank
S_0 specifies the least significant bit of the rotated input data field.
L specifies the length (number of bits) of the desired port data field. L = 0 selects an 8-bit field.
D specifies the address of the internal destination register.

The order of operation is:

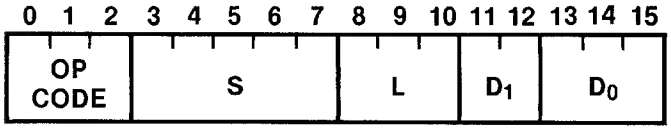
- Read the IV bus data into the input latches
- Right rotate the input field as given by S_0
- Mask off the least significant L bits of the rotated field
- XOR the masked data with the contents of the AUX register
- Move the 8-bit result to the destination register

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 00/01/02/03/04/05/06/11/12/13/14/15/16

3.4.5 XOR-IV Bus, IV Bus

Format RB:



Description:

Perform an exclusive-OR operation on the L-bit field of the IV bus source data and the contents of the AUX register and move the least significant L bits of the result to the destination on the IV bus, given by D. Transfers from one bank to another implies an 8-bit transfer, regardless of the value of the L-field.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the least significant bit of the input data field after rotation.

L specifies the length (number of bits) of the masked field that is to be processed and merged with the existing IV bus data. L = 0 selects an 8-bit field.

D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank

D_0 specifies the bit position in the data from the input latches with which the least significant bit of the processed data field should be aligned. This means that the processed data field is left-shifted so that the LSB is aligned with bit D_0 of the input latches.

The order of operation is:

- Read the IV bus port data into the input latches
- Right rotate the input data field until S_0 becomes LSB
- Mask the least significant L bits
- XOR the masked field with the contents of the AUX register
- Left-shift the result until the LSB is aligned with bit D_0
- Merge the least significant L bits with the original IV bus port data from the input latches
- Output the merged 8-bit field to the IV bus destination port

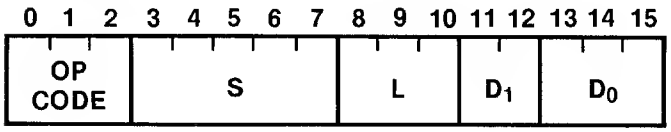
Since the data in the IV latch originated in the source port, data in the destination port will be lost if it is located on the other bank. The original values of the source field outside of the masked field will be preserved.

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D_0: 0/1/2/3/4/5/6/7
D_1: 2/3

3.4.6 XOR-IV Bus, IV Bus Address

Format RB:



Description:

Enable the I/O Port on the bank specified by D, whose address is the result of the XOR operation on the L-bit field of the IV bus port data and the contents of the AUX register. Destination data is also placed in the internal register specified by the D-field.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank

S_0 specifies the bit which will be the least significant bit of the input data field after rotation.

L specifies the length (number of bits) of the masked field. L = 0 selects an 8-bit field.

D specifies the destination bank of the IV bus for the address data:
D = 07 specifies Left Bank address (IVL) and register R7
D = 17 specifies Right Bank address (IVR) and register R17

The order of operation is:

- Read the IV bus port data into the input latches
- Right rotate the input data field until bit S_0 the LSB
- Mask the least significant L bits
- XOR the masked field with the contents of the AUX register
- Move the resulting 8-bit field to the IV bus as an address at the bank specified by D and also place in the internal register

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
D: 07/17

3.5 XEC INSTRUCTIONS

Description:

The XEC or Execute instructions enable the programmer to perform a single instruction out of sequence from the normal program flow. This is accomplished through an indexed offset scheme where the high order bits of the Address Register are set from the Program Counter while the low order bits are determined from the sum of the source data and a literal contained in the instruction. The source may be an internal register, in which case the low order 8 bits are replaced, or IV bus data, in which case only the low order 5 bits are replaced. This results in an accessible address range of the instruction of 256 or 32 words respectively. The address range is referred to as a page of memory.

The instruction located at the computed address is then executed. This instruction is referred to as the target instruction.

The Program Counter is not altered by the XEC instruction. The target instruction, however, performs whatever action it normally would on the Program Counter. This results in the following possibilities:

1. For MOVE, ADD, AND, XOR, XMIT, and unsatisfied NZT instructions, the Program Counter is incremented by 1. After execution of the target instruction, the next sequential instruction after the XEC will be executed.
2. For JMP and satisfied NZT instructions, the target instruction will change the Program Counter to an address specified in the instruction. After execution of the target instruction, the instruction at the address specified in the target instruction will be executed.
3. The target instruction of the XEC may be another XEC. The target XEC, like the initial XEC, will not in and of itself, modify the Program Counter, but will in turn point to another target instruction. XEC instructions can be chained indefinitely in this manner. Since none of the XECs in such a chain modify the Program Counter, they must reside in the same address page as the initial XEC and the eventual target instruction. After execution of the eventual target, the next sequential instruction after the *first* XEC in the chain will be executed, unless the eventual target instruction is a JMP or a satisfied NZT.

Example:

Execute the specified JMP from a table of JMP instructions determined by the value of the selected I/O Port on the Left Bank. The table follows immediately after the XEC instruction and the I/O field is called INTERPT and is a 3-bit field located in bits 4, 5 and 6—see Figure 3-5. Details of the XEC instructions are described in subsequent paragraphs.

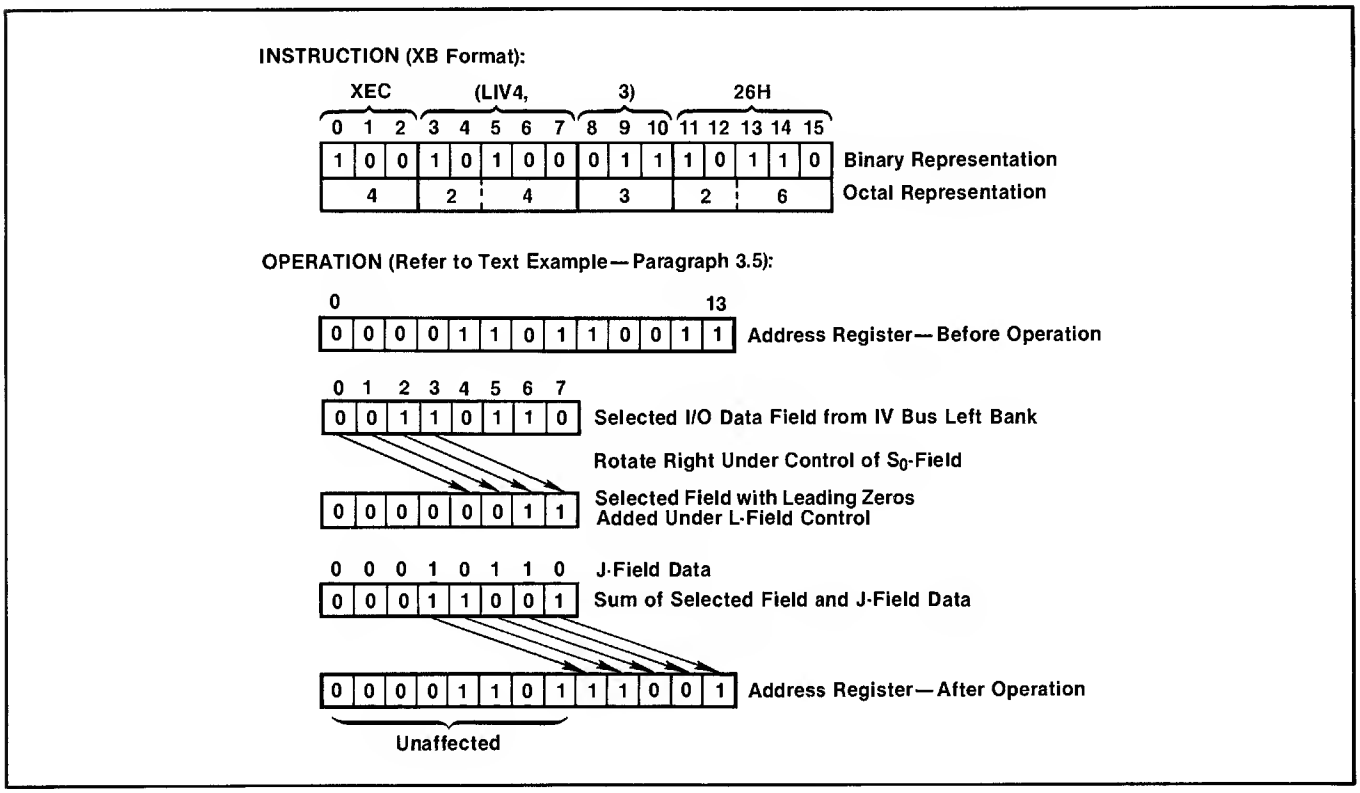
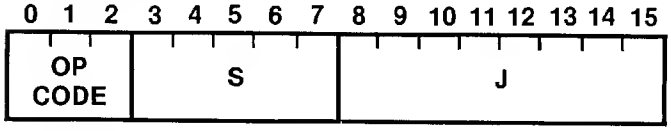


Figure 3-5. Example of XEC Instruction

3.5.1 XEC-Register

Format JR:



Description:

Perform the instruction at the address formed by replacing the 8 least significant bits of the Address Register with the 8-bit sum of J and the contents the register specified by S.

S specifies the source register.
J is the 8-bit integer value for address modification.

The order of operation is:

- Read the data from the source register
- Form the 8-bit sum of the J field value and the source register contents
- Replace the least significant 8 bits of the Address Register with the 8 bit sum

Only the least significant 8 bits of the Address Register can be changed by this instruction, so that a range of 256 addresses can be affected. This range of 256 addresses is termed the address page, which is defined by the five most significant bits of the Address Register. When the sum of (S)+J is greater than 255 (377_8) only the least significant 8 bits are used; the overflow register is not changed.

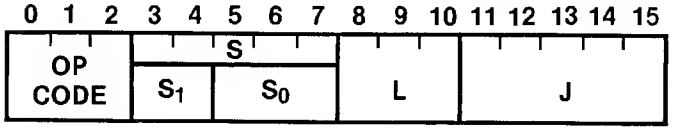
The Program Counter is not modified by the XEC instruction. It is updated by the target instruction as described in 3.5 above. Normally, the next instruction executed after the target instruction will be the next sequential instruction after the XEC, unless the target instruction is a JMP or a satisfied NZT.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
J: 0 to 377_8

3.5.2 XEC-IV Bus

Format JB:



Description:

Perform the instruction at the address formed by replacing the 5 least significant bits of the Address Register with the 5-bit sum of J and the IV bus data specified by S.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ selects the Left Bank
$S_1 = 3$ selects the Right Bank
S_0 specifies the bit which will be the least significant bit of the input data field after rotation.
J is the 5-bit integer value for index offset.
L specifies the length (number of bits) of the masked IV bus data that is to be processed. Note that a maximum of 5 bits will be interpreted by this instruction.

The order of operation is:

- Read the IV bus data into the input latches
- Rotate the input data field as specified by S_0
- Mask off the least significant L bits
- Form the 5 bit sum of the J field value and the rotated and masked IV bus data
- Replace the least significant 5 bits of the Address Register with the 5 bit sum

Only the least significant 5 bits of the address register can be changed by this instruction, so that a range of 32 addresses can be affected. This range of 32 addresses is termed the address page, which is defined by the eight most significant bits of the Address Register. When the sum (S)+J is greater than 31 (37_8) only the least significant 5 bits are used: the overflow register is not changed.

The Program Counter is not modified by the XEC instruction. It is updated by the target instruction as described in 3.5 above. Normally, the next instruction executed after the target instruction will be the next sequential instruction after the XEC, unless the target instruction is a JMP or a satisfied NZT.

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7
I: 0 to 37_8

3.6 NZT INSTRUCTIONS

Description:

These instructions perform conditional branches. If the data specified by the S field is non-zero, the low order bits of the Program Counter and Address Register are replaced with a literal. Otherwise, processing continues with the next instruction in sequence. If S is a register, the low order 8 bits of the Program Counter and Address Register are replaced; if S is an I/O data field, the low order 5 bits of the Program Counter and Address Register are replaced, resulting in an NZT range of 256 and 32, respectively.

Example:

Jump to program address ALPHA if the selected Left Bank I/O field is non-zero. The field name is OVERFLO and it is a 1-bit field located in bit 3—see Figure 3-6. Details of the NZT instructions are described in subsequent paragraphs.

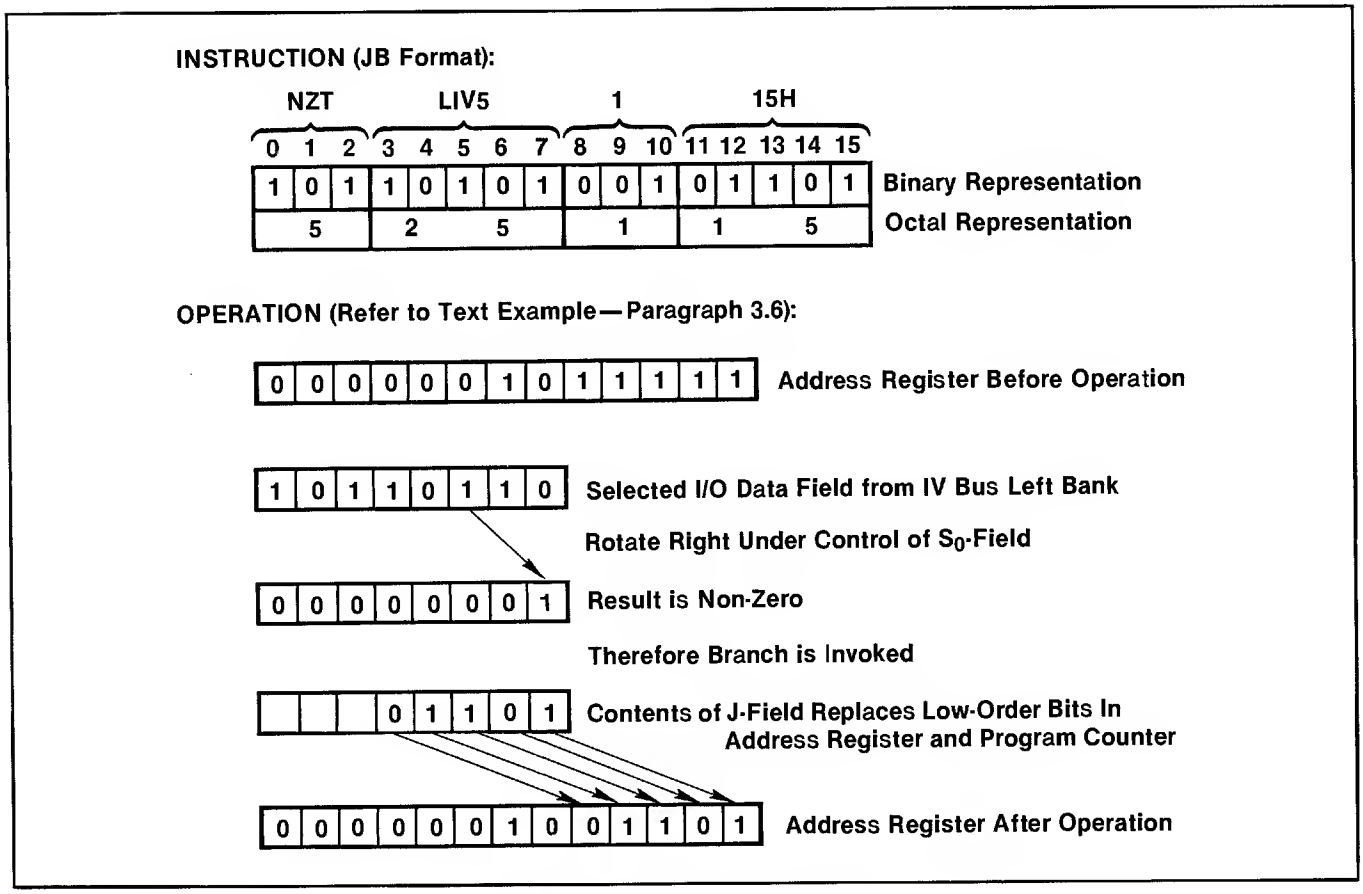
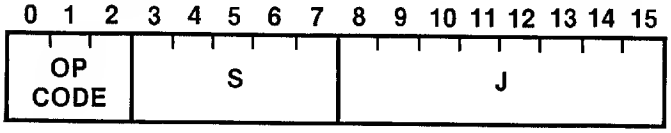


Figure 3-6. Example of NZT Instruction

3.6.1 NZT-Register

Format JR:



Description:

If the source data is non-zero, jump to the address formed by replacing the 8 least significant bits of the Address Register and Program Counter with the value in the J field. If the source data is zero, increment the Program Counter by one.

S specifies the register whose contents are subject to the test.

J specifies the 8-bit integer for address modification.

The order of operation is:

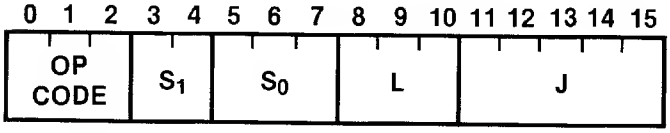
- Read the contents of the source register
- Test the register contents for all zeros
- If the contents are not zeros, replace the least significant 8 bits of the Address Register with the value of the J-field
- If the contents are zeros, increment the Program Counter by one.

Permitted Operand Values:

S: 00/01/02/03/04/05/06/07/10/11/12/13/14/15/16/17
J: 0 to 377_8

3.6.2 NZT-IV Bus

Format JB:



Description:

If the contents of the L-bit field of the IV bus data is non-zero, insert the value of the 5-bit J-field into the 5 least significant bits of the Address Register and Program Counter. If the contents are all zeros, the Program Counter is incremented by one.

S_1 specifies the bank of the IV bus which is the data source:
$S_1 = 2$ specifies the Left Bank
$S_1 = 3$ specifies the Right Bank
S_0 specifies the bit which will be the least significant bit of the input data field after rotation.
J is the 5-bit integer value for address modification.

L specifies the length (number of bits) of the masked field. L = 0 specifies an 8-bit field.

The order of operation is:

- Read the IV bus data into the input latches
- Rotate the input data until bit S_0 becomes the LSB
- Mask off the least significant L bits
- Test the contents of the masked field
- If the contents of the masked field are non-zero, replace the 5 least significant bits of the Address Register and Program Counter with the value of the J-field
- If the contents of the masked field are zero, increment the Program Counter by one

Permitted Operand Values:

S_0: 0/1/2/3/4/5/6/7
S_1: 2/3
L: 1/2/3/4/5/6/7/0
J: 0 to 37_8

3.7 XMIT INSTRUCTIONS

Description:

These instructions transmit literals to internal registers and IV bus ports. The literal field is stored in D. If D is a register, an 8-bit field is transferred. If D is an I/O Port, up to a 5-bit field is transferred. An 8-bit field can be transmitted to the IV bus only by specifying the D-field as registers R12 or R13.

Example:

Store the bit pattern 110 in the selected I/O Port selected on the Right Bank. The field name is VALUE and is located in bits 3, 4 and 5—see Figure 3-7. Details of the XMIT instructions are described in subsequent paragraphs.

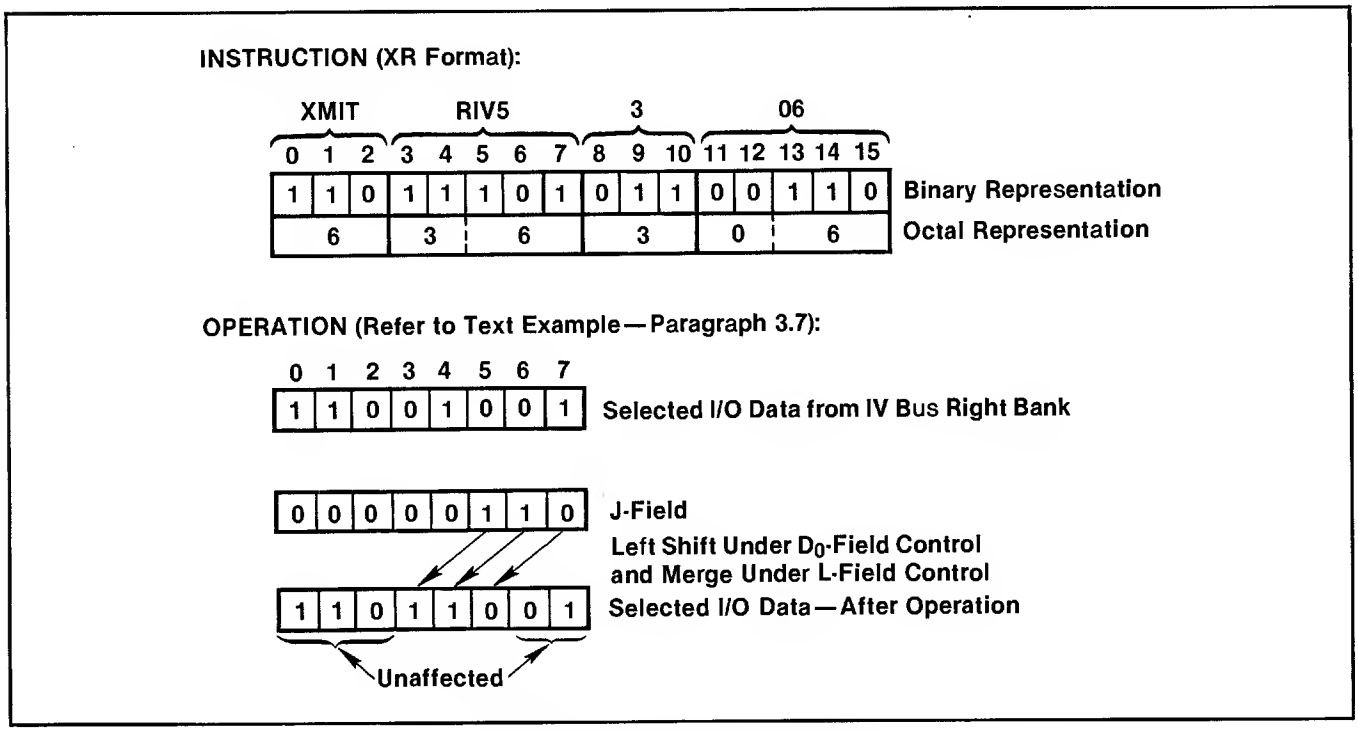
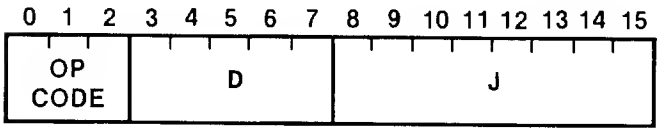


Figure 3-7. Example of XMIT Instruction

3.7.1 XMIT-Register

Format XR:



Description:

Store the value of the 8-bit integer, J, in the register specified by D. Note the values of D = 07 or 17 result in an IV Bus Address instruction, described in subsection 3.7.4. Also, values of D = 12 or D = 13 result in an IV bus instruction, described on the following page.

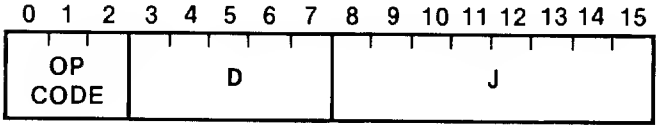
D specifies the register to be loaded.
J specifies the 8-bit field containing the value to be loaded into the register.

Permitted Operand Values:

D: 00/01/02/03/04/05/06/11/14/15/16
J: 0 to 377_8

3.7.2 XMIT Register, IV Bus

Format XR:



Description:

Store the value of the 8-bit integer in the selected IV bus port. Note: Contents of R12 or 13 are not changed.

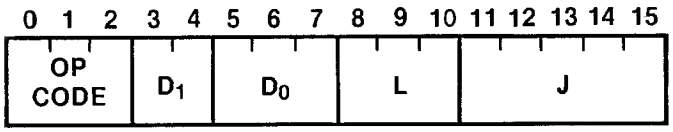
D specifies the bank of the IV bus which is the destination:
D = 12 selects the Left Bank and register R12
D = 13 selects the Right Bank and register R13
J specifies the 8-bit field containing the value to be loaded.

Permitted Operand Values:

D: 12/13
J: 0 to 377_8

3.7.3 XMIT-IV Bus

Format XB:



Description:

Transmit the least significant L bits of the J field to the I/O Port specified by D. If L is greater than 5 bits, the most significant bits of the destination field are filled with zeros.

D_1 specifies the bank of the IV bus which is the destination:
$D_1 = 2$ selects the Left Bank
$D_1 = 3$ selects the Right Bank

D_0 specifies the bit position in the IV bus with which the least significant bit of the J field data should be aligned. This means that the J field data is left-shifted so that the LSB is aligned with bit D_0 of the IV bus.

L specifies the length of the destination field (number of bits). L = 0 selects an 8-bit field.

J specifies the 5-bit field to be loaded.

The order of operation is:

- Read the contents of the destination port into the input latches
- Read the least significant 5 bits of the instruction word
- Left-shift the copied 5-bit field as specified by D_0
- Merge the shifted field as specified by L with the contents of the IV latches and output the result to the IV bus port

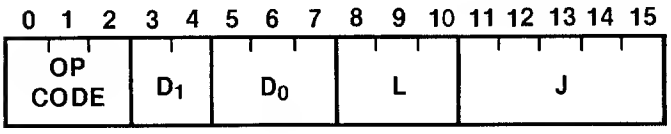
Note that the data in the IV latches outside the field specified by D_0 and L is not altered.

Permitted Operand Values:

D_0: 0/1/2/3/4/5/6/7
D_1: 2/3
L: 1/2/3/4/5/6/7/0
J: 0 to 37_8

3.7.4 XMIT-IV Bus Address

Format XB:



Description:

Enable the IV bus port at the bank specified by D whose address is the literal constant of the J-field. The constant is also placed in the register specified by the D-field.

D specifies the destination bank of the IV bus for the address data
D = 07 specifies Left Bank (IVL) and register R7
D = 17 specifies Right Bank (IVR) and register R17

J specifies the 8-bit literal to be used as address data.

Permitted Operand Values:

D: 07/17
J: 0 to 377_8

3.8 JMP INSTRUCTIONS

Description:

The literal value A is placed in the Program Counter and Address Register, and processing continues at location A. The A-field has a range of $0\text{-}17777_8$ (0-8191).

Example:

Jump to location ALPHA (0000101110001)—see Figure 3-8.

There is only one jump instruction, which is described in the following subsection.

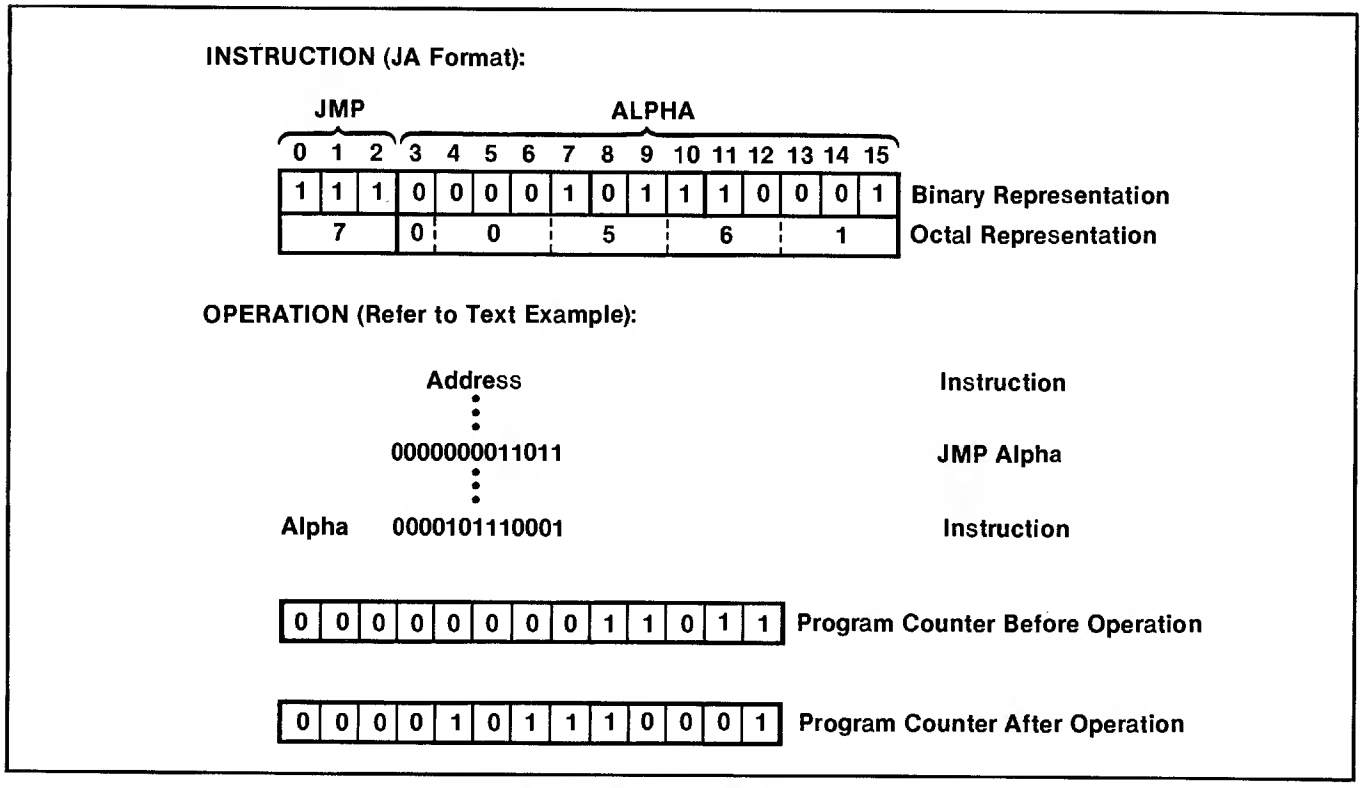
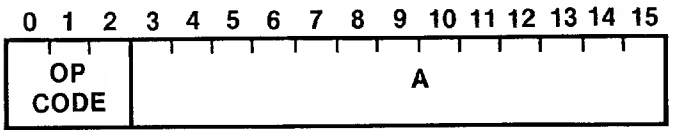


Figure 3-8. Example of JMP Instruction

3.8.1 JMP Address

Format JA



Description:

Jump to the instruction address specified by the A field, and continue normal program execution from that address. The contents of the 13-bit A field are loaded into the Program Counter and Address Register. The next instruction to be executed is then the instruction at the new address.

The order of operation is:

- Load the Address Register and Program Counter with the contents of the A field
- New address value is used for next instruction

Permitted Operand Values

A: 0 to 17777_8 (8191_{10})

Table 4-1. Definitions of Timing Parameters

PARAMETER (Note 1)		COMMENTS
T_{PC}	Processor cycle time	
T_{CP}	X1 clock period	
T_{CH}	X1 clock high time	
T_{CL}	X1 clock low time	
T_{MCH}	MCLK high delay	
T_{MCL}	MCLK low delay	
T_{W}	MCLK pulse width	Note 2
T_{AS}	X1 falling edge to address stable	Note 7
T_{MAS}	MCLK falling edge to address stable	Notes 2, 3, & 7
T_{IA}	Instruction to address	Notes 2, 3, & 8
T_{IVA}	Input data to address	Notes 3 & 9
T_{IS}	Instruction set up time (X1 rising edge)	Note 10
T_{MIS}	MCLK falling edge to instruction stable	Notes 2, 4, & 10
T_{IH}	Instruction hold time (X1 rising edge)	Note 11
T_{MIH}	Instruction hold time (MCLK falling edge)	Notes 2 & 11
T_{WH}	X1 falling edge to SC/WC rising edge	
T_{MWH}	MCLK falling edge to SC/WC rising edge	Note 2
T_{WL}	X1 falling edge to SC/WC falling edge	
T_{MWL}	MCLK falling edge to SC/WC falling edge	
T_{IBS}	X1 falling edge to $\overline{LB}/\overline{RB}$ (Input phase)	
T_{MIBS}	MCLK falling edge to $\overline{LB}/\overline{RB}$ (Input phase)	
T_{IIBS}	Instruction to $\overline{LB}/\overline{RB}$ (Input phase)	
T_{OBS}	X1 falling edge to $\overline{LB}/\overline{RB}$ (Output phase)	

PARAMETER (NOTE 1)		COMMENTS
T_{MOBS}	MCLK falling edge to $\overline{LB}/\overline{RB}$ (Output phase)	Note 2
T_{IDS}	Input data set up time (X1 falling edge)	
T_{MIDS}	MCLK falling edge to input data stable	Notes 2 & 5
T_{IDH}	Input data hold time (X1 falling edge)	
T_{MIDH}	Input data hold time (MCLK falling edge)	Note 2
T_{ODH}	Output data hold tim (X1 falling edge)	
T_{MODH}	Output data hold time (MCLK falling edge)	
T_{ODS}	Output data stable (X1 falling edge)	Notes 12 & 15
T_{MODS}	Output data stable (MCLK falling edge)	Notes 2, 12, & 15
T_{ODO}	Output driver turn-on time (X1 falling edge)	Note 14
T_{MODO}	Output driver turn-on time (MCLK falling edge)	Note 14
T_{DI}	Output driver turn-on time (SC/WC rising edge)	Note 16
T_{DD}	Input data to output data	Notes 13 & 15
T_{HS}	$\overline{HALT}$ set up time (X1 rising edge)	
T_{MHS}	MCLK falling edge to $\overline{HALT}$ falling edge	Notes 2 & 6
T_{HH}	$\overline{HALT}$ hold time (X1 rising edge)	
T_{MHH}	$\overline{HALT}$ hold time (MCLK falling edge)	Note 2
T_{ACC}	Program storage access time	
T_{IO}	I/O port output enable time ($\overline{LB}/\overline{RB}$ to valid $\overline{IV}$ data input)	

Notes.

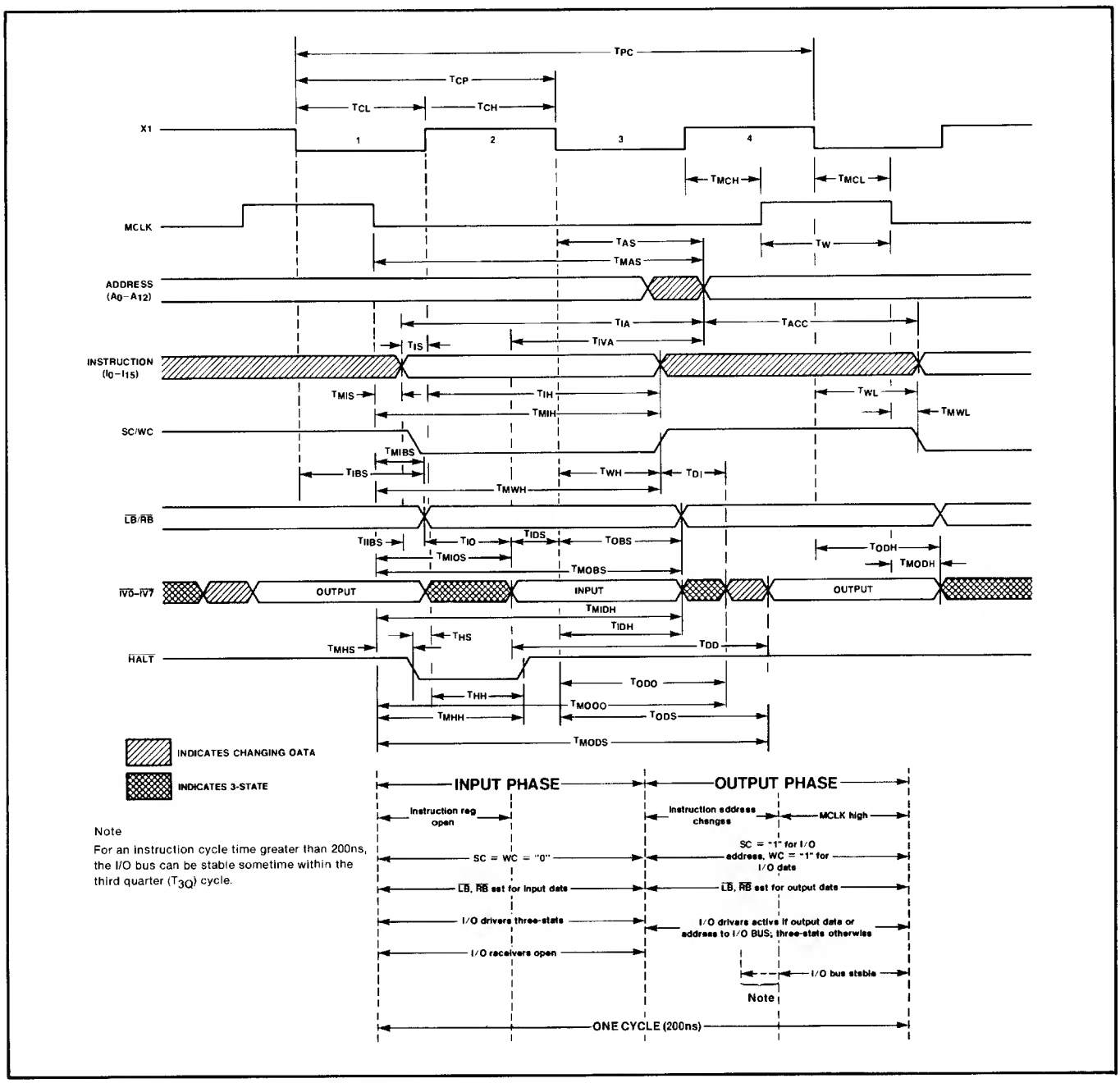
1. X1 and X2 inputs are driven by an external pulse generator with an amplitude of 1.5 volts; all timing parameters are measured at this voltage level.
2. Respectively, T_{1O}, T_{2O}, T_{3O}, and T_{4O} represent time intervals for the first, second, third, and fourth quarter cycles.
3. Capacitive loading for the address bus is 150 picofarads.
4. Same as TIS but referenced to falling edge of MCLK.
5. Same as TIDS but referenced to falling edge of MCLK.
6. Same as THS but referenced to falling edge of MCLK.
7. TAS is obtained by forcing a valid instruction and an I/O bus input to occur earlier than the specified minimum set up time; the TAS parameter then represents the earliest time that the address bus is valid.
8. TIA is obtained by forcing a valid instruction input to occur earlier than the minimum set up time.
9. TIVA is obtained by forcing a valid I/O bus input to just meet the minimum set up time.
10. TMIS represents the set up time required by internal latches of the 8X305 in system applications, the instruction input may have to be valid before the worst-case set up time in order for the system to respond with a valid I/O bus input that meets the I/O bus input set up time (TIDS and TMIDS).
11. TIH represents the hold time required by internal latches of the 8X305. To generate proper $\overline{LB}/\overline{RB}$ signals, the instruction must be held valid until the address bus changes.
12. TODS is obtained by forcing a valid I/O bus input to occur earlier than the I/O bus input set up time (TIDS); this timing parameter represents the earliest time that the I/O output data can be valid.
13. TDD is obtained by forcing a valid I/O bus input to just meet the minimum I/O bus input set up time; this timing parameter represents the latest time that the I/O output data can be valid.
14. The minimum figure for these parameters represents the earliest time that I/O bus output drivers of the 8X305 will turn on.
15. For TIDS ≥ 25ns, TODS or TMODS should be used to determine when the output data is stable.
16. This parameter represents the latest time that the output drivers of the input device should be turned off.

Chapter 4

TIMING

The efficiency of the 8X305 instruction cycle implies that the timing relationship of the signals which interface to the device is of great importance in optimizing the design. This chapter describes the generation and interaction of timing signals, including their effect on the internal workings of the device.

The block diagram in Figure 2-1 shows the latches and registers in the device. The basic timing diagram in Figure 4-1 and those in the following paragraphs show internal and external clocking of data during an instruction cycle. All latches of the 8X305 MicroController are level-triggered.



Note
For an instruction cycle time greater than 200ns, the I/O bus can be stable sometime within the third quarter (T_{3Q}) cycle.

Figure 4-1. Timing Diagram of 8X305 MicroController

4.1 CLOCK GENERATION

The 8X305 contains an on-chip oscillator, which is driven from an external source via the X1 and X2 input pins. Any one of the four techniques described below will generate acceptable drive signals for these input pins.

4.1.1 Clock Generation Using a Crystal

This is the most desirable method of oscillator control, and is implemented through the use of a frequency determining crystal. The oscillator is designed such that its gain decreases as its operating frequency increases. It should be noted that in some instances with cycle times greater than 400ns it may be necessary to add a resistor in parallel with the crystal to prevent harmonic oscillation. Table 4-2 provides the crystal specifications to be used with the 8X305. The resonant frequency of the crystal, f_0 (in Hertz), is related to the desired instruction cycle time, (T in seconds), by the relationship $f_0 = 2/T$.

For example, if the desired instruction cycle time is 200 nanoseconds, or 200×10^{-9} seconds, then:

$$f_0 = \frac{2}{200 \times 10^{-9}} = 10 \times 10^6 \text{ Hz} = 10 \text{ MHz}.$$

Table 4-2. Crystal Specifications

Type	Impedance at Fundamental	Impedance at Harmonics and Spurs
Fundamental Mode, Series Resonant	35 ohms Maximum	50 ohms Minimum

4.1.2 Clock Generation Using Pulse Generator

This method is especially useful in applications that require the ability to vary the instruction cycle time. The X1 and X2 inputs of the 8X305 must be connected to the complementary outputs of the pulse generator as shown in Figure 4-2.

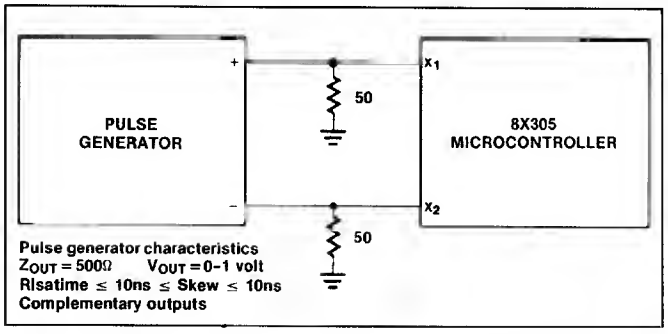


Figure 4-2. Clocking with a Pulse Generator

4.1.3 Clock Generation Using TTL Logic

The 8X305 can be synchronized with an external clock by simply connecting appropriate drive circuits to the X1 and X2 inputs. In these applications, the X1 and X2 lines can be interfaced to TTL logic as shown in Figure 4-3.

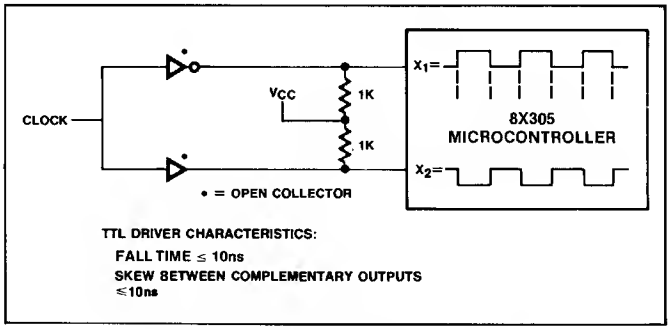


Figure 4-3. Clocking with TTL Signals

4.1.4 Clock Generation Using a Capacitor

The final control method, an external capacitor, is an imprecise way of controlling the speed of the 8X305 and its use should be restricted to low speed applications. Capacitor values are shown in Figure 4-4 along with the approximate cycle time.

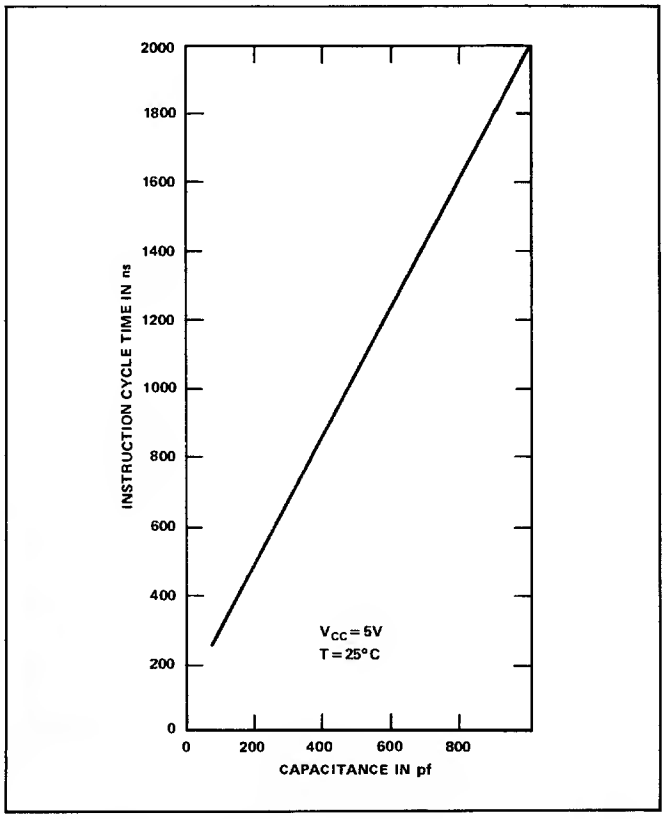


Figure 4-4. Typical Cycle Time Versus Capacitance

4.2 BASIC CYCLE TIMING

All timing and timing-control signals of the 8X305 are generated by the oscillator and sequencer. Outputs of the sequencer control both internal and external timing parameters of the system. Each internal quarter cycle bears a fixed relationship to X1 via the propagation delay of the sequencer and oscillator.

General and interactive timing relationships pertaining to I/O signals of the 8X305 are shown in Figure 4-5.

4.3 TIMING RELATIONSHIPS

Certain timing events must occur in specific quarter-cycles. In 8X305 systems operating with fast instruction cycle times, most delays within the MicroController are determined by propagation delays of internal gates. When operating with slower instruction cycle times, the delays appear to increase due to internal-clock gating. Timing relationships within the 8X305 chip are shown in Figure 4-6.

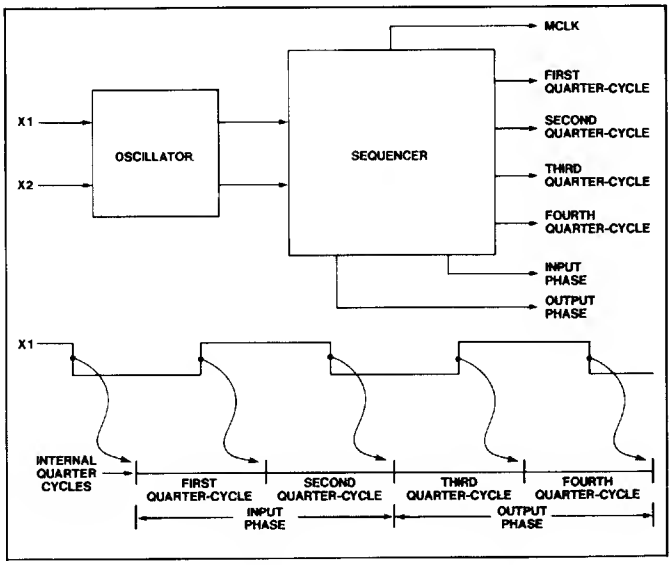


Figure 4-5. Timing and Timing Control Signals

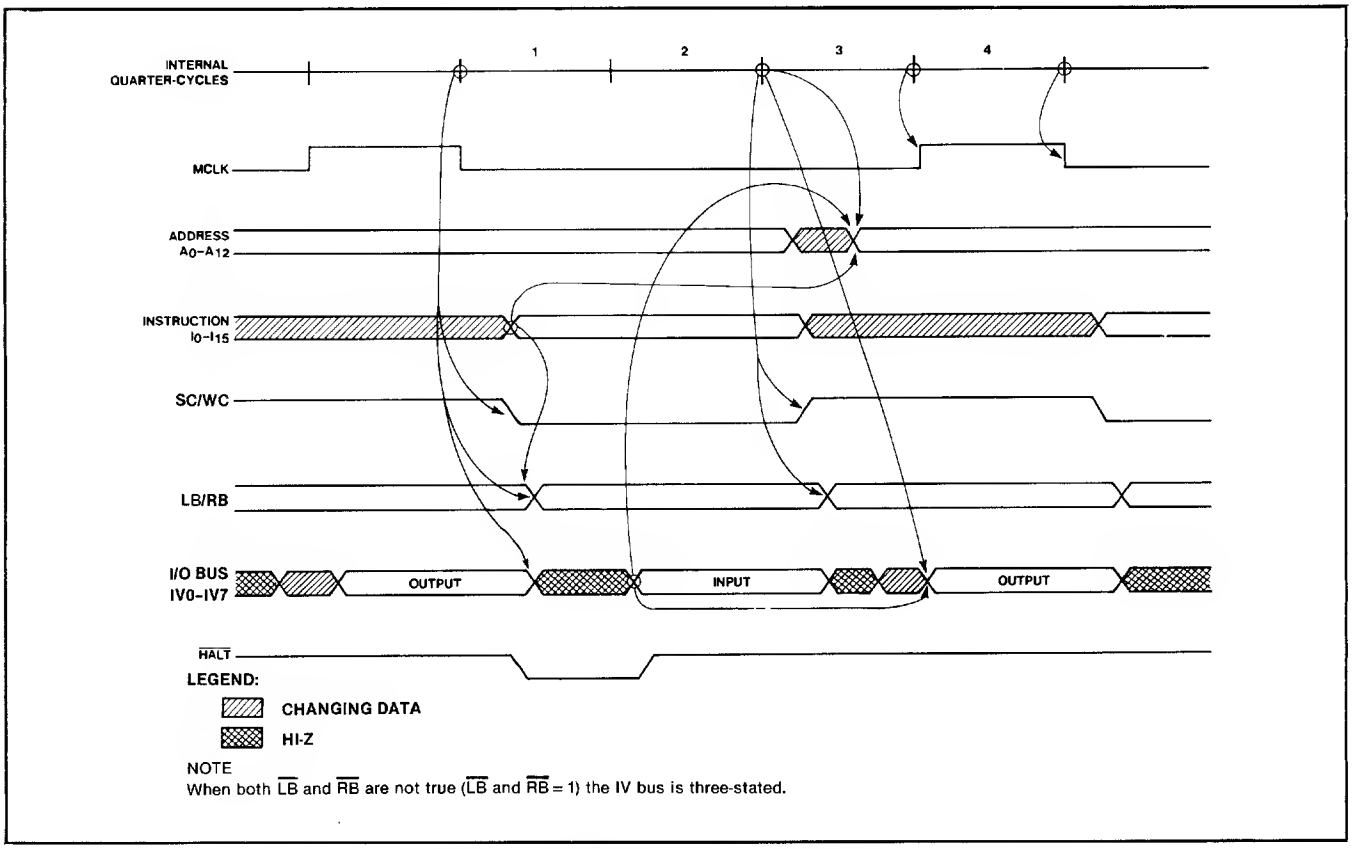


Figure 4-6. Typical Instruction Timing Relationships

4.3.1 Master Clock (MCLK)

The Master Clock (MCLK) signal marks the fourth quarter-cycle. The leading and trailing edges of MCLK approximately correspond to their counterparts of the fourth quarter-cycle.

The most significant parameter for the user is the opening of the Instruction Register Latch. In high performance applications, time can be gained by having the instruction valid at the inputs to the device slightly prior to the trailing edge of MCLK.

4.3.2 Address Latches

The opening and closing of the internal address latches occur with the leading edges of the third and fourth quarter-cycles, respectively. The address of the current instruction remains stable until the end of the second quarter-cycle. During the third quarter-cycle, the address of the next instruction becomes true on the A_0-A_{12} lines. The point at which the address becomes valid and stable depends on the following considerations:

1. The propagation delay from the address latch clock input to the address latch data output, when the address latch starts to open during the third quarter-cycle. This assumes valid address information is available at the start of the 3rd quarter cycle such as in a MOVE instruction.

2. The propagation delay from the time an instruction becomes stable to the address output (TIA), as in a JMP instruction.

3. The propagation delay from the beginning a valid I/O bus input to the address output (TIVA). This is the case in a conditional branch, and XEC instruction, in which data in a port or register is tested before a decision on the branch is made.

4.3.3 Instruction Latches

Input signals to the instruction latches (register) are open for the duration of the first quarter-cycle only. When considered at the chip level, there is a minimum set-up and hold time requirement with respect to the trailing edge of the first quarter-cycle. At a system level, there are further requirements as to when and how long the instruction inputs must be valid. The $\overline{LB}/\overline{RB}$ bus control signals are derived directly from the instruction input during the input phase and for $\overline{LB}$ and $\overline{RB}$ to remain stable, the instruction input must be stable until the end of the input phase. Since the address bus is stable for the entire input phase, the instruction output of the program storage should be stable for the entire input phase, thus satisfying the requirements for stability of the $\overline{LB}/\overline{RB}$ signals. A factor determining the latest point at which the instruction must become valid is the worst case I/O data input set-up time. This is discussed in the following subsection.

4.3.4 I/O and I/O Control

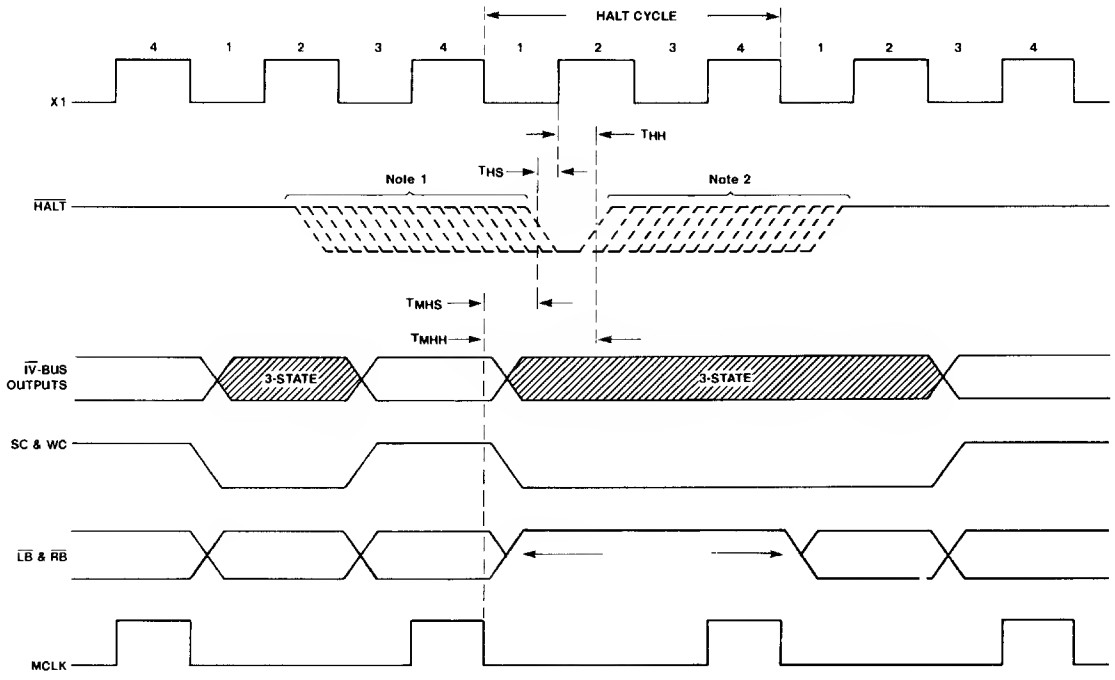
The availability of data on the IV bus during the input phase depends on the setting of the $\overline{LB}/\overline{RB}$ signals because these are used as enabling signals by the I/O ports. $\overline{LB}$ or $\overline{RB}$ becomes valid after either the leading edge of the first quarter-cycle or the beginning of a valid instruction, whichever occurs first. During the output phase, $\overline{LB}$ or $\overline{RB}$ becomes valid at the leading edge of the third quarter-cycle. This can be delayed by a longer (worst case) instruction also, as in the input phase.

The SC/WC signals change state at the falling edge of the fourth and at the leading edge of the third quarter-cycles and are not gating terms for bus data.

The IV bus has constraints during the input phase which are a function of the internal chip architecture. In addition to the IV bus latch, which remains open during the first and second quarter-cycles, there is an internal latch at the ALU. Due to gate delays between the chip inputs and this latch, data must be stable from the port some time previous to the leading edge of the third quarter-cycle which latches data into both the IV bus and the ALU latch. During the output phase the IV bus is driven by the 8X305, but data may become valid as late as the fourth quarter-cycle in 200 nsecs operation.

4.3.5 HALT Timing

The $\overline{HALT}$ signal is sampled via internal chip logic at the end of the first internal quarter of each instruction cycle. If, when sampled, the $\overline{HALT}$ signal is active-low, a halt is immediately executed and the current instruction cycle is terminated; however, the halt cycle does not inhibit MCLK nor does it affect any internal registers of the 8X305. As long as the $\overline{HALT}$ line is active-low, the SC and WC lines are low (inactive), the Left Bank ($\overline{LB}$)/Right Bank ($\overline{RB}$) signals are high (inactive), and the IV bus remains in the three-state mode of operation. Normal operation resumes at the next cycle in which $\overline{HALT}$ is high when sampled—see Figure 4-7.



NOTES
1. The HALT signal can switch from High to Low at any time during this interval.
2. The HALT signal can switch from Low to High at any time during this interval.
3. LB and RB are High during the time the HALT signal is active-Low.

Timing Descriptions:
T_{HS} — set-up time from HALT to X1 (independent of instruction cycle time)
t_{HH}—hold time from X1 to HALT (independent of instruction cycle time)
T_{MS}—set-up time from MCLK to HALT (dependent upon instruction cycle time)
T_{MHH}—hold time from MCLK to HALT (dependent upon instruction cycle time)

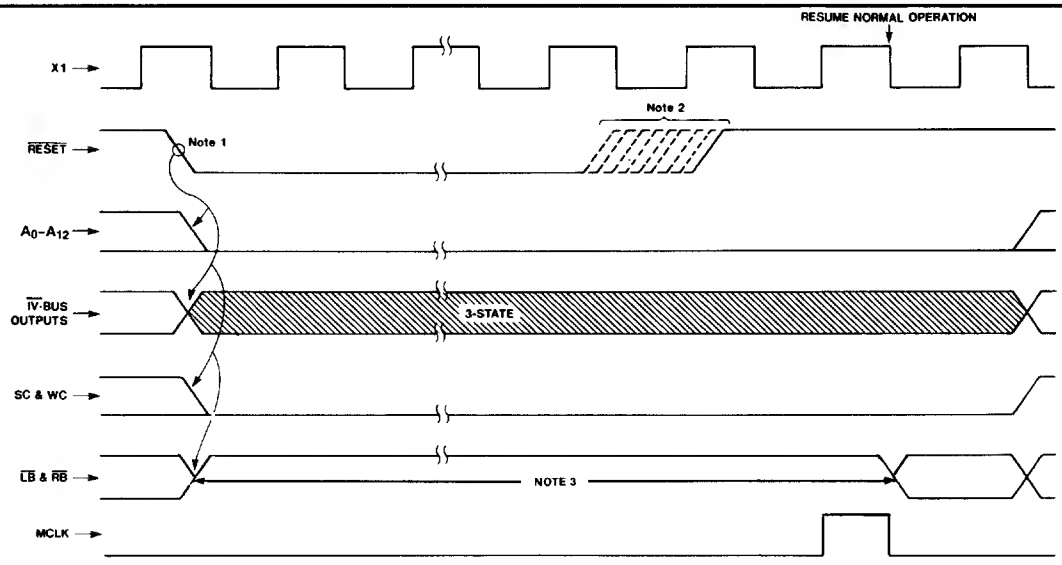
Figure 4-7. HALT Timing

4.3.6 RESET Timing

The RESET line can be driven from a high (inactive) state to a low (active) state at any time with respect to the system clock, that is, the reset function is asynchronous. To ensure proper operation, RESET must be held low (active) for one full instruction time. When the line is driven from a high state to an active-low state, several events occur—the precise instant of occurrence is basically a function of the propagation delay for that particular event. As shown in Figure 4-8, these events are:

- The Program Counter and Address Register are set to address zero and remain in that state as long as the RESET line is low. Other than PC and AR, RESET does not affect other internal registers.
- The input/output (IV) bus goes three-state and remains in that condition as long as the RESET line is low.
- The Select Command and Write Command signals are driven low and remain low as long as the RESET line is low.
- The Left Bank/Right Bank (LB/RB) signals are forced high asynchronously for the period in which the RESET line is low.

During the time RESET is active-low, MCLK is inhibited; moreover, if the RESET line is driven low during the last two quarter cycles, MCLK may be shortened for that particular machine cycle. When RESET line is driven high (inactive)—one quarter to one full instruction cycle later, MCLK appears just before normal operation is resumed. As long as the RESET line is active-low, the preceding HALT signal is not sampled by internal logic of the 8X305.



NOTES
1. A High to Low transition of the RESET signal will force the Address Bus to an all-zero configuration.
2. The RESET signal can switch from Low to High at any point within this time interval and, in all cases, MCLK will occur at least one-quarter cycle time later as shown.
3. LB and RB are high for the period of time RESET is low.

Figure 4-8. RESET Timing

4.4 CRITICAL TIMING CALCULATIONS

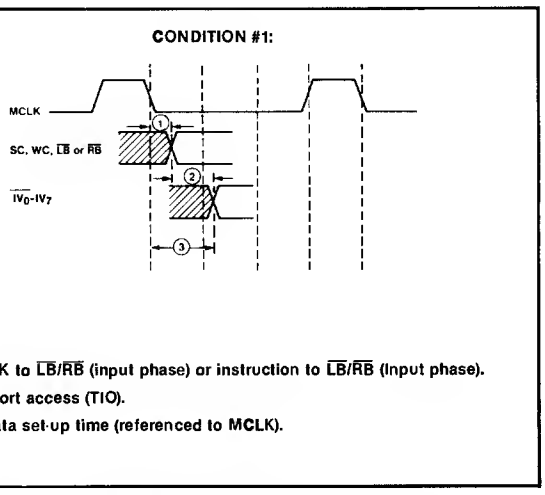
The timing parameters defined in Table 4-1 are normally measured with reference to X1 or MCLK. Principal system determinants for the instruction cycle time are:

- Propagation delays within the 8X305
- Access time of program storage
- Enable time of bus port devices

Normally, the instruction cycle time is constrained by one or more of the following conditions:

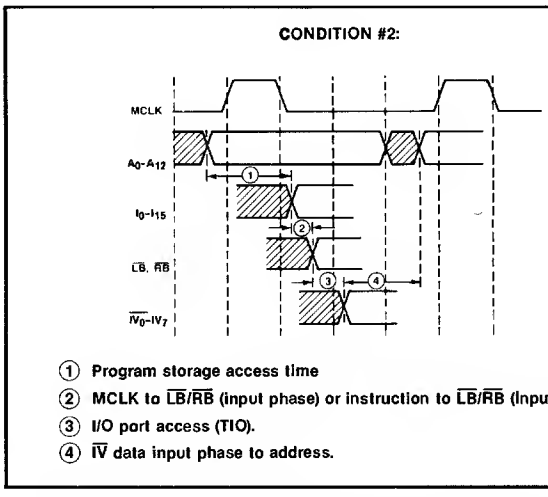
Condition 1—Instruction or MCLK to LR/RB (input phase) plus I/O port access time (TIO) ≤ IV data set up time—see Figure 4-9.

Condition 2—Program storage access time (TACC) plus instruction to LB/RB (input phase) plus I/O port access time (TIO) plus IV data (input phase) to address ≤ instruction cycle time—see Figure 4-10.



① MCLK to LB/RB (input phase) or instruction to LB/RB (input phase).
② I/O port access (TIO).
③ IV data set-up time (referenced to MCLK).

Figure 4-9. Calculating Access Time of I/O Port



① Program storage access time
② MCLK to LB/RB (input phase) or instruction to LB/RB (input phase).
③ I/O port access (TIO).
④ IV data input phase to address.

Figure 4-10. Calculating Access Time of Program Storage

Condition 3—Program storage access time plus instruction to address ≤ instruction cycle time—see Figure 4-11.

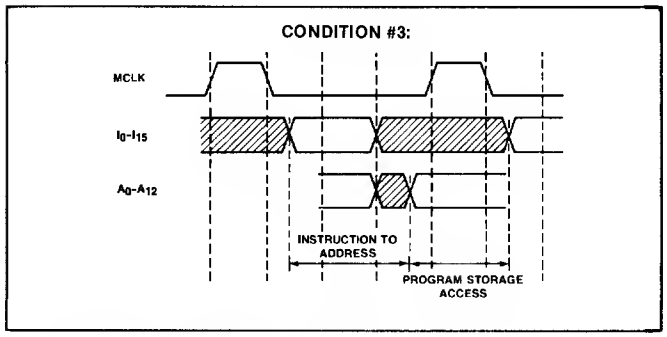


Figure 4-11. Verification of Program Storage Access Time

4.4.1 I/O Port Access Time

From condition #1, above, and with an instruction cycle time of 200ns, the I/O port access time (TIO) can be calculated as follows:

$$\begin{aligned} TMIBS + TIO &\leq TMIDS \\ \text{transposing, } TIO &\leq TMIDS - TMIBS \\ \text{substituting, } TIO &\leq 55ns - 25ns \\ \text{result, } TIO &\leq 30ns \end{aligned}$$

Using 30ns for TIO, the constraint imposed by condition #1 can also be used to calculate the minimum cycle time:

$$\begin{aligned} TMIBS + TIO &\leq TMIDS \\ \text{thus, } 25ns + 30ns &\leq T_{1Q} + T_{2Q} - 45 \\ \text{and, } 25ns + 30ns &\leq 1/2 \text{ cycle} - 45 \end{aligned}$$

therefore, the worst case instruction cycle time is 200ns. With subject parameters referenced to X1, the same calculations are valid:

$$\begin{aligned} TIBS + TIO + TIDS &\leq 1/2 \text{ cycle} \\ \text{thus, } 45ns + 30ns + 25ns &\leq 1/2 \text{ cycle} \end{aligned}$$

therefore, the worst-case instruction cycle time is again 200ns.

4.4.2 Program Storage Access Time

From condition #2, above, and with an instruction cycle time of 200ns, the program storage access time can be calculated:

$$\begin{aligned} TACC + TIIBS + TIO + TIVA &\leq 200ns \\ \text{transposing, } TACC &\leq 200ns - TIIBS - TIO - TIVA \\ \text{substituting, } TACC &\leq 200ns - 25ns - 30ns - 85ns \\ \text{thus, } TACC &\leq 60ns \end{aligned}$$

hence, for an instruction cycle time of 200ns, a program storage access time of 60ns is implied. The constraint imposed by condition #3 (Figure 4-11) can be used to verify the maximum program storage access time:

$$\begin{aligned} TIA + TACC &\leq \text{Instruction Cycle} \\ \text{thus, } TACC &\leq 200 \text{ ns} - 140ns \\ \text{and, } TACC &\leq 60ns, \end{aligned}$$

confirming that a program storage access time of 60ns is satisfactory.

For an instruction cycle time of 200ns and a program storage access time of 60ns (Figure 4-10), the instruction must be valid at the falling edge of MCLK. This relationship can be verified by the following calculation:

$$\begin{aligned} &200ns - TMAS - TACC \\ &= 200ns - 140ns - 60ns \\ &= 0ns \end{aligned}$$

It is important to note that, during the input phase, the beginning of a valid $\overline{LB}/\overline{RB}$ signal is determined by either the instruction to $\overline{LB}/\overline{RB}$ delay (TIIBS) or the delay from the falling edge of MCLK to $\overline{LB}/\overline{RB}$ (TMIBS). Assuming the instruction is valid at the falling edge of MCLK and adding the instruction-to-$\overline{LB}/\overline{RB}$ delay (TIIBS = 25ns), the $\overline{LB}/\overline{RB}$ signal will be valid 25ns after the falling edge of MCLK. With a fast program storage memory and with a valid instruction before the falling edge of MCLK—the $\overline{LB}/\overline{RB}$ signal will, due to the TMIBS delay, still be valid 25ns after the falling edge of MCLK. Using a worst-case instruction cycle time of 200ns, the user cannot gain a speed advantage by selecting a memory with faster access time. Under the same conditions, a speed advantage cannot be obtained by using an I/O port with fast access time (TIO) because the address bus will be stable 55ns (TAS) after the beginning of the third quarter cycle—no matter how early the IV data input is valid.

4.4.3 Instruction Cycle Time

When operating at slower instruction cycle times (TPC > 200 ns), there are two more timing conditions which must be satisfied in addition to those already mentioned. First, the I/O input data must be stable by the set-up time required by the input latches. The program storage access time (TACC) must be such that:

$$TAS + TACC + TIIBS + TIO + TIDS \leq TPC$$

Second, the instruction must be stable by the set-up time required by the instruction latches. The program storage access time must also be such that:

$$TAS + TACC + TIS \leq TPC - T_{2Q}$$

where T_{2Q} is the length of the second quarter-cycle (pulse width of X1 high).
For symmetric clock signals:

$$T_{2Q} = 1/4\ TPC, \text{ or}$$

$$TAS + TACC + TIS \leq 3/4\ TPC$$

Program storage access time must satisfy the worst case of all of the timing conditions mentioned.

Chapter 5

SYSTEM DESIGN

The optimal structure of a system which uses an 8X305 MicroController is a function of the specific application. Several useful techniques are available to the user which help optimize a design. Factors such as system cycle time, response time to an external stimulus, or type of memory used as working storage may also need to be considered.

A typical system structure based on the 8X305 is shown in Figure 5-1. While the preceding chapters contain a full discussion of the architecture and functioning of the 8X305, detailed system design parameters must be determined from the device Data Sheet. This Data Sheet is supplied separate from this manual, and provides full AC and DC characteristics for the device.

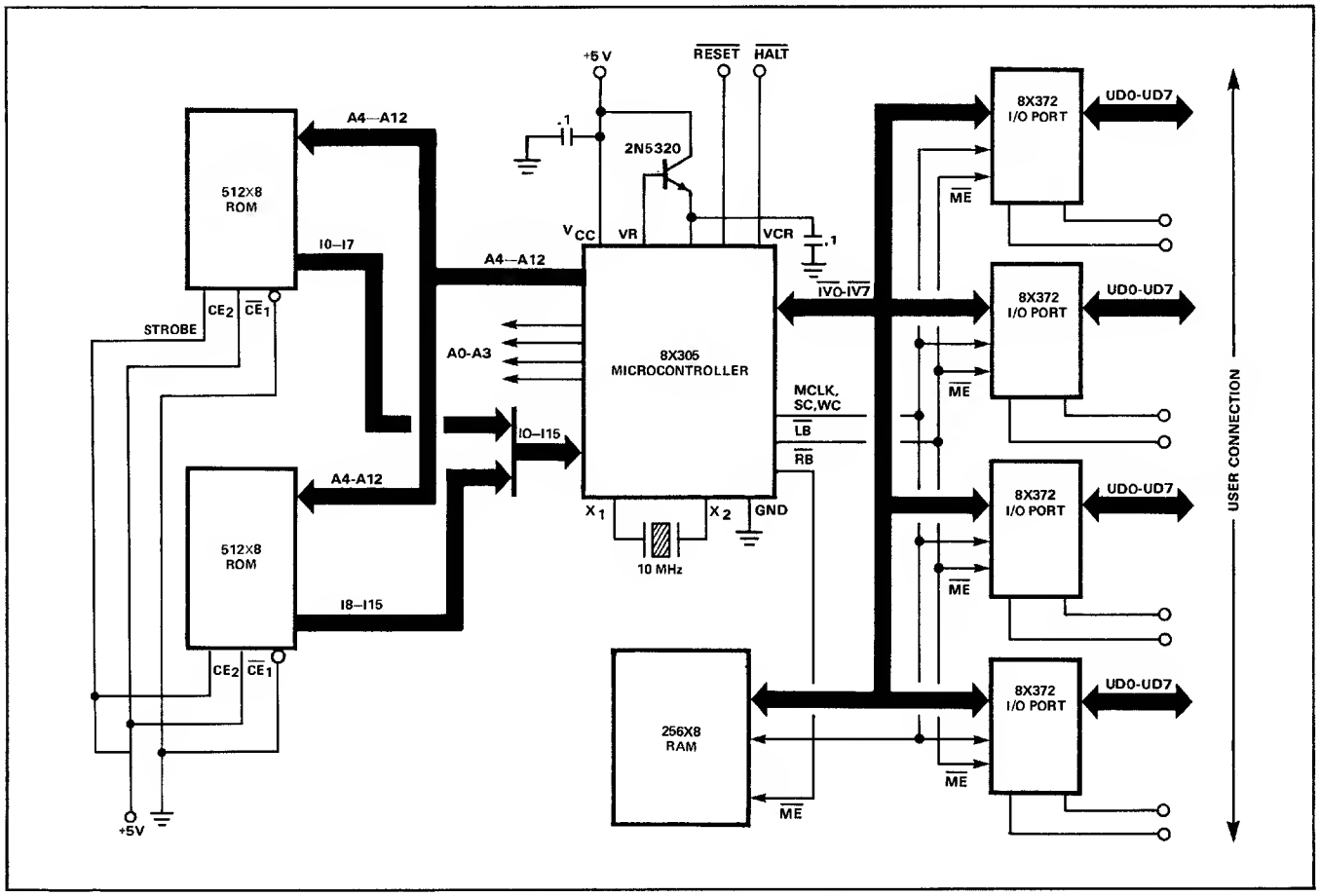


Figure 5-1. Representative Control System

Since the 8X305 is a member of the 8X300 Family of devices, it is recommended that the Product Capabilities Manual for the family be consulted for an appreciation of the full range of compatible support devices and development tools currently available to the designer. System development time can be greatly reduced through their use.

5.1 PROGRAM MEMORY

Program memory devices contain the instructions used to sequence the 8X305 through the tasks it is required to perform in the application. This may consist of up to 8K words of program.

Some degree of latitude in the choice of memory is available to the designer. In most cases, program memory will be bipolar PROM. The exact choice of device type depends on the memory size and access time required. Access time is in turn dependent on the cycle time of the 8X305 established for the particular design. Generally speaking, the faster the cycle time of the 8X305, the shorter the memory access time necessary to operate with it.

Applications where the program is subject to change may be solved more cost-effectively through the use of EPROM's as program memory. While still being non-volatile, these can be field-updated.

Certain applications may require the ability to customize a controller interface at some time other than initial manufacture, such as at power-up time. In such cases, a control program can be down-loaded from a host into a RAM program memory, which, in turn, may be used by the 8X305 as program memory.

5.1.1 Cycle Modification

With conventional clocking schemes, operating the 8X305 at maximum speed requires the use of very fast program memory. In some applications, however, slower memory devices may be employed in conjunction with a more sophisticated clocking scheme.

Program memory access occurs during the latter part of the instruction cycle. It is possible to run those quarter-cycles not involved with program memory access at full speed, while slowing down those used for memory access to a period compatible with the memory devices being used.

In other words, by stretching the positive phases of the X1 clock input, memory fetch cycles are elongated, with a corresponding reduction in required memory access time.

Such a scheme is shown in Figure 5-2. For an oscillator frequency of 6 Mhz, corresponding to a cycle time of 167nsecs, a delay of 20nsecs has been introduced, which results in a stretched X1 positive phase, and a correspondingly squeezed X2 positive phase.

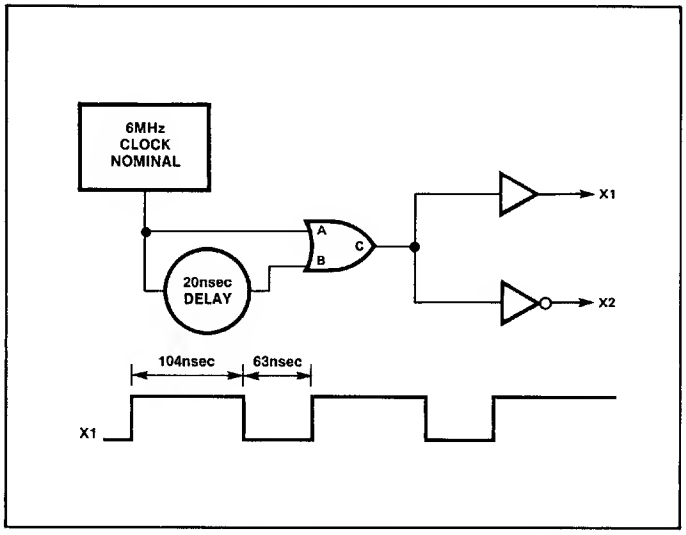


Figure 5-2. Elongated Positive Clock Cycle

5.2 EXTENDED MICROCODE

Many applications that use the 8X305 MicroController require special functions to be performed rapidly during program execution. Often, such functions can be initiated by a simple extension of the 8X305's instruction word beyond the 16 bits used by the MicroController itself. This technique, referred to as Extended Microcode, is implemented by extending the Instruction Address Bus to additional program memory devices. The additional bits accessed with each instruction can then be used to provide system control functions or status information.

Figure 5-3 shows a general implementation of this concept.

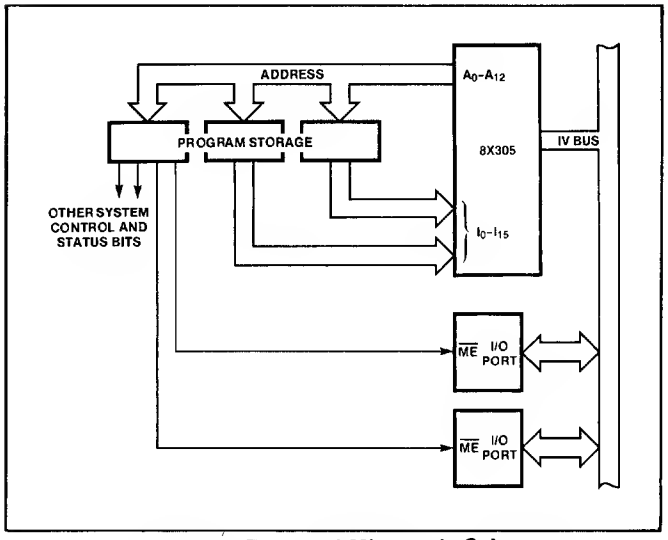


Figure 5-3. Extended Microcode Scheme

5.2.1 Fast I/O Select

Conventional usage of I/O ports on the IV bus requires that the port be enabled by means of an Address Select cycle (instruction with destination to the IV bus address registers) before the port can be used for data transfers. If an application involves a critical timing loop which includes the transfer of data to several different ports, the need for separate Address Select cycles for each different port access can be avoided if Extended Microcode is used to select each port.

The savings in processing time in such cases can be considerable. An example where the two alternate approaches are compared is shown in Table 5-1. For the given example, four of the instruction cycles (1, 3, 6, and 8) can be eliminated from the operation sequence by using the extended microcode technique.

Table 5-1. Comparison of Fast I/O Select

Instruc Number	Normal 8X305 Operation	Extended Microcode Operation	
		Extended Instruc	Instruc Proper
1	Select Command Port	Select Command Port	Read Command
2	Read Command	Select Data Input Port	Read Data Input
3	Select Data Input Port	(no operation)	Process Data
4	Read Data Input Port	Select Data Output Port	Write Data Output
5	Process Data	Select Status Port	Update Status
6	Select Data Output Port		
7	Write Data Output		
8	Select Status Port		
9	Update Status		

The 8X371 I/O Port device is well-suited for applications which utilize extended microcode. Due to the nature of the 8X305 timing cycle, the extended microcode bits from program storage need to be latched in order that they remain stable throughout the instruction cycle. Figure 5-4 shows how an edge triggered D flip-flop can be used to maintain the stability of the extended microcode bits throughout the instruction cycle.

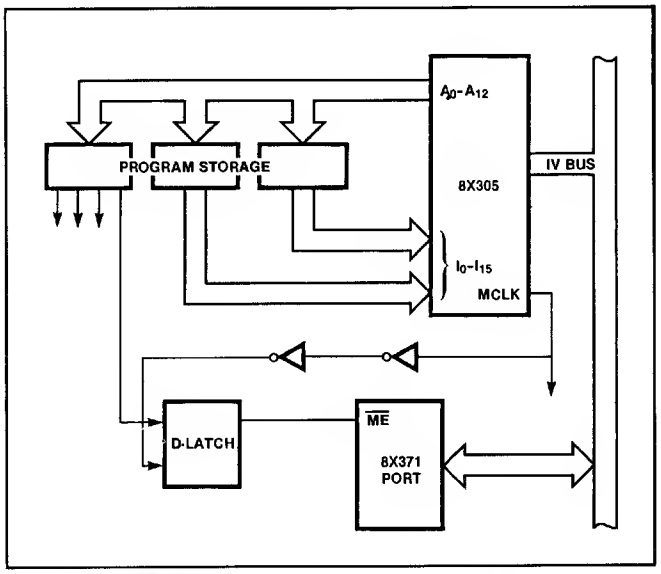


Figure 5-4. Port Enable Latching Scheme

Depending on the timing involved in the specific application, delays in the clocking term to the D flip-flop may be necessary to allow sufficient time for data setup. Since the clocking term is usually the system clock MCLK, the timing scheme should correspond to Figure 5-5.

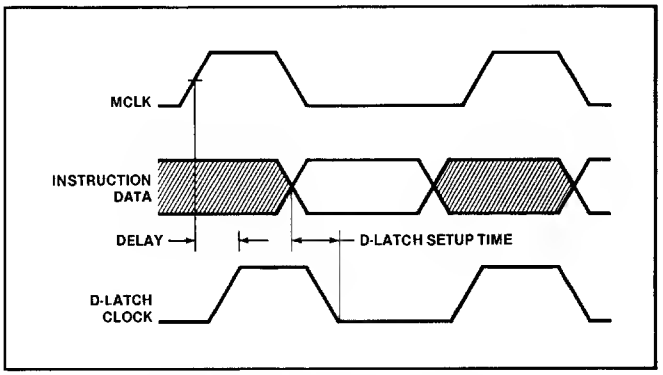


Figure 5-5. Latching Timing Diagram

In order that the port may present valid data at the correct point in the input phase, consideration must also be given to the timing of the port enable within the overall timing scheme. Each delay added to the enable time must be evaluated for its impact on this.

In systems where a small additional delay can be tolerated, the power of extended microcode bits can be expanded by means of a decoder. A scheme to provide 16 distinct signals from 4 bits is shown in Figure 5-6.

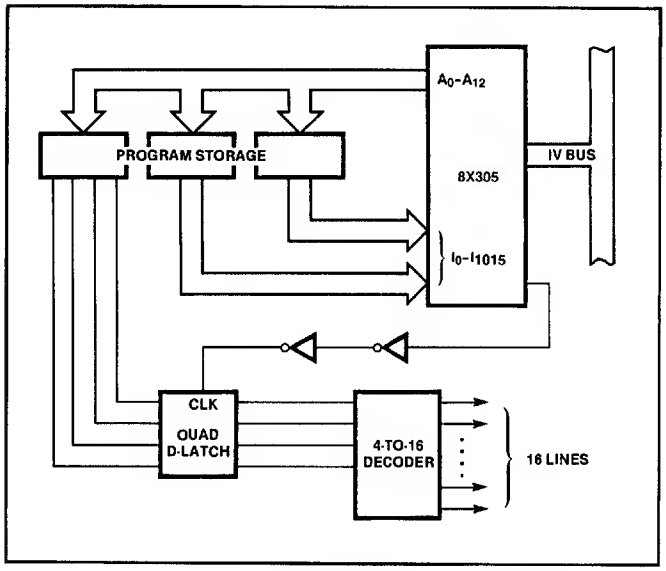


Figure 5-6. Extended Microcode Decoding

In systems which include multiple ports on the same bank, some or all of which are accessed via Extended Microcode, care must be taken to avoid having more than one port enabled at the same time.

5.2.2 Other Uses of Extended Microcode

Extended microcode can be used to enhance system performance in many ways besides Fast I/O Select. One example is to control the term which a multiplexer gates to a particular line of a port when it is polled by the system. This permits a number of status bytes to be accessed through the same port.

Other examples include selection of pages within working storage concurrent with data transfer, generation of control signals, generation of interrupts, and numerous other common design considerations. The specific use of the technique is determined by the application, but Extended Microcode should be considered where parallel control of a number of functions is required.

5.2.3 Cycle Delay

Sometimes it is desirable to use a slow memory device for working storage that has an access time in excess of the maximum acceptable for the chosen system cycle time. This can be done by providing the system with a hardware delay for program storage access, and can be implemented with a minimum of devices through use of a flip-flop and the $\overline{\text{HALT}}$ line. Such a scheme is shown in Figure 5-7.

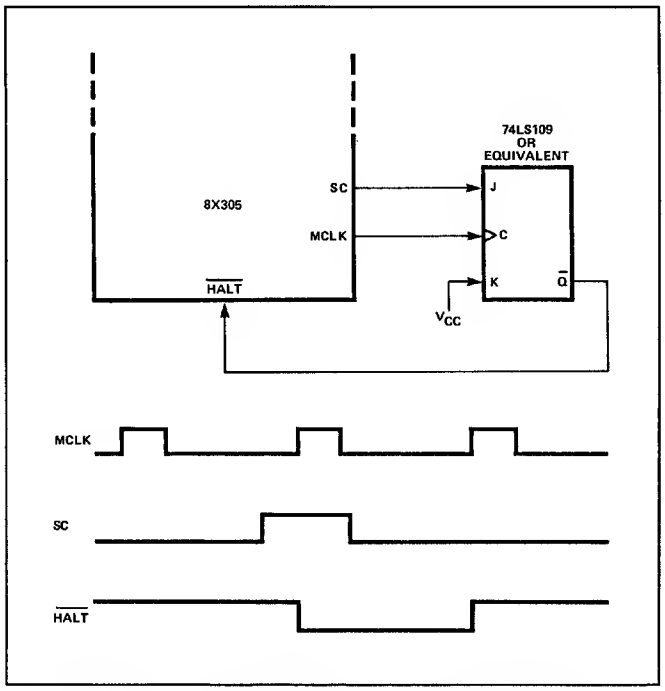


Figure 5-7. Working Storage Access Delay Generation

A variation of this, which allows the designer to combine program memories of differing performance in the same system is shown in Figure 5-8. The fast program memory can be used for time critical routines such as disk data transfer, with the slower memory containing other programming that can operate at lower speeds. Here, the delay is generated in a similar fashion, except that the program space has been partitioned and the delay is invoked only when the memory which requires it is being accessed.

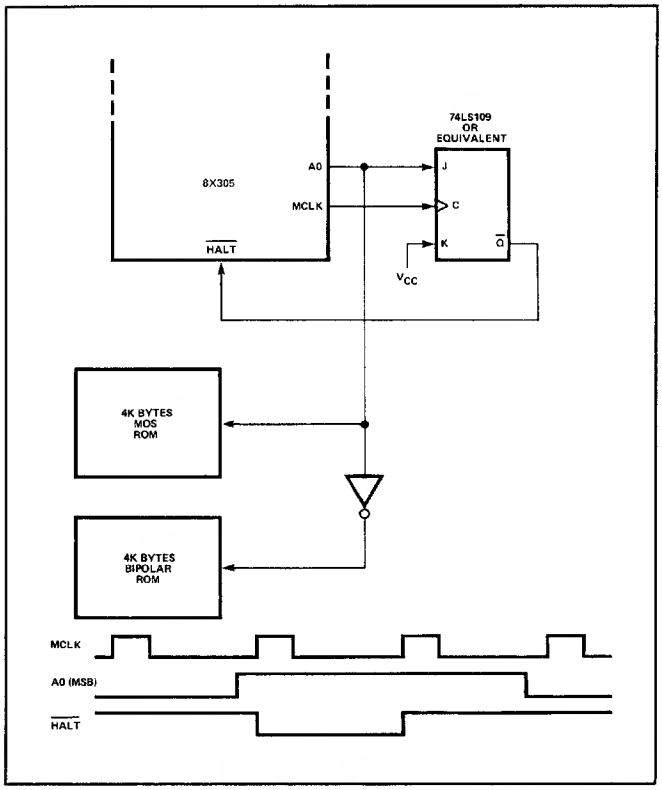


Figure 5-8. Delay Generation with Program Storage Partitioning

5.3 IV BUS INTERFACE

The bidirectional IV Bus, in conjunction with the SC WC, $\overline{LB}$, $\overline{RB}$, and MCLK control signals associated with it, constitutes the principal interface between the 8X305 and peripheral devices. While Signetics has developed a number of bus compatible peripheral support devices for use with the 8X305, there are applications where other devices are required. Systems which incorporate such devices must be designed with the hardware and software implications of this in mind. This section provides information for interfacing various devices to the IV bus, and also discusses fanout considerations.

5.3.1 8X300 Family Bus-Compatible Devices

A variety of devices compatible with the 8X305 IV bus have been developed by Signetics to solve common system design problems. Incorporation of these devices into an 8X305 system is extremely simple. Device pins are labeled corresponding to the 8X305 pins to which they connect, and a simple decision to connect the Master Enable ($\overline{ME}$) pin to either the $\overline{LB}$ or $\overline{RB}$ signals is all that need be made.

5.3.2 Non-8X300 Family Devices

Should a system require the use of a device not specifically designed for use on the IV bus, consideration of a number of factors critical to bus operation must be made.

A key factor is Address Selection, which is not featured in non-compatible devices. Circuitry must be included to provide the control signal decode, address comparison and address latch necessary for conventional device addressing on the IV bus. Alternatively, such devices can be selected by Extended Microcode techniques.

A common use of non-compatible devices is the substitution of octal latches for the 8X370/380 series of compatible I/O ports. While cost savings may be realized with this approach in some applications, certain programming capabilities are lost when using these devices. This stems from the absence of bidirectional capability in octal latches, which prevents the powerful 8X305 read-merge-write operation from being used, and results in the latch being able to deal only with 8-bit increments of data.

In general, the bus control signals provided by the 8X305 will require decoding in order to permit a non-compatible device to recognize the current operation. The additional logic necessary to decode this may add delays at critical points in the timing cycle, and must be carefully analyzed to ensure proper operation at the high speeds involved and avoid deterioration in system throughput.

5.3.3 IV Bus Operations

The IV bus control signals defined the type of activity occurring on the IV bus at any given point in time. The system designer must thoroughly understand the relationships between these signals before attempting a design using other than 8X300 Family bus-compatible devices. This section discusses the significance of various IV bus control signal configurations.

The 8X305 instruction cycle is divided into two phases, known as the Input Phase and the Output Phase. The Input Phase is signalled by the term:

$$\overline{SC}\cdot\overline{WC}\cdot(\overline{LB}+\overline{RB})$$

At this time the 8X305 expects the currently selected port on the bank being accessed to place a byte of input data in the IV bus. Ports on the other bank must be inactive at this time, as must all other ports on the bank being accessed.

The Output Phase can be either of two types, an Address Select or Data Output operation.

An Address Select operation is signalled by the term:

$$SC\cdot\overline{WC}\cdot(\overline{LB}+\overline{RB})\cdot MCLK$$

During this time, each I/O Port and support device connected to the bank being accessed examines the information on the IV bus and compares it to its own address. A device containing the address currently on the bus will become selected, while all other devices on the bank will become deselected. This operation has no effect on ports on the other bank of the bus.

A Data Output operation is signalled by the term:

$$\overline{SC} \cdot WC \cdot (\overline{LB} + \overline{RB}) \cdot MCLK$$

At this time, the currently selected device on the bank being accessed accepts the data currently placed on the IV bus by the 8X305. Note that if the device selected during the Output Phase is not the same as during the Input Phase, data can only be handled in 8 bit increments. If they are the same, however, advantage can be taken of the 8X305's ability to accept data from a port, perform bit manipulation on it, and restore the altered data to the same port.

All terms are positive true with the exception of the bank select signals ($\overline{LB}$ and $\overline{RB}$), which are negative true. These signals normally indicate the bank being selected, but may indicate that the current phase is not accessing the IV bus by the term:

$$RB \cdot LB$$

During this time, the IV bus is three-stated by the 8X305. It provides a direct method of monitoring when the 8X305 has read a device. It also provides a means to know when the IV bus is not in use by the 8X305, which is useful in systems incorporating multiprocessor access to common resources, DMA data transfer, or signal handshake techniques.

5.3.4 Bus Loading

The 8X305 and its associated family of compatible peripheral devices have been designed with sufficient AC and DC drive capability for all but the largest systems to be implemented without IV bus buffering.

In general, the IV bus can drive approximately twenty-five 8X300 Family bus compatible devices. A calculation to determine the adequacy of bus drive capability for the particular design should be made if this number is approached. Such a calculation should be made for any design that uses non-8X300 Family devices on the IV bus. Detailed specifications of bus drive capability can be found in the 8X305 Data Sheet.

In the event that additional bus drive is required, any of a variety of high speed buffering devices may be used.

When using bus buffering devices, system cycle time calculations must allow for the intrinsic delay of the buffers. The most convenient approach to this is simply to add the buffer delay to the I/O port delay. A buffered port is, in effect, a slower port, and this will have an impact on the instruction cycle that can be used in the system.

5.4 WORKING STORAGE

The internal registers in the 8X305 provide sufficient data storage capacity for most systems. Some sophisticated applications may require additional storage, usually in the form of RAM located on the IV bus. Various options are open to the designer to implement this. Some of the more common are discussed in the rest of this section.

5.4.1 Data Storage With the 8X350

The 8X350 is an IV bus-compatible device containing a 256x8 bit array of bipolar RAM. The most common system configuration incorporating this assigns one entire bank to the device, with the 256 bytes of RAM corresponding to the 256 port addresses available on that bank.

Systems which require additional storage can use additional 8X350s, which can be selected as memory pages via Extended Microcode or a control latch and the $\overline{ME}$ (Master Enable) pin of each individual chip.

5.4.2 Alternate Data Storage

Requirements for working storage much larger or smaller than 256 bytes or for I/O Ports on both banks of the IV bus may be better satisfied by data storage schemes other than the 8X350.

Use of such an approach requires development of an interface between the IV bus and the RAMs. The principal factors effecting such a design are RAM size, organization, and speed, all seen within the context of system performance. Since the normal addressing on the IV bus requires an Address Select Cycle which precedes the actual transfer of data, the interface must provide some form of latch to store the address currently selected. An 8X370 series I/O Port can be used if the RAM itself does not have address latches.

If the application requires high performance, Extended Microcode techniques can be used for address generation, provided that the RAM address is known to the program in advance.

In addition to addressing, consideration must be given to the transformation of the 8X305 bus control signals into

those required for control of the RAM devices, and to the interface of RAM data and the IV bus lines.

A general interface scheme for large RAM is shown in Figure 5-9. Two I/O ports are used to latch the High and Low address bytes of the 64K-byte storage, while a third handles RAM control signals and a fourth transfers the data between the 8X305 and the RAM. While being simple to implement, this scheme may not be suitable for applications which require high throughput due to the number of port accesses which must be made for each data transfer cycle.

Where memory size smaller than 256 bytes is required, a scheme which parallels the characteristics of the 8X350 can be used. Shown in Figure 5-10, it incorporates an 8X371 as an address latch and implements the control signal interface by means of standard gates. If the RAM devices have three-state outputs, they can be connected directly to the IV bus without an I/O Port.

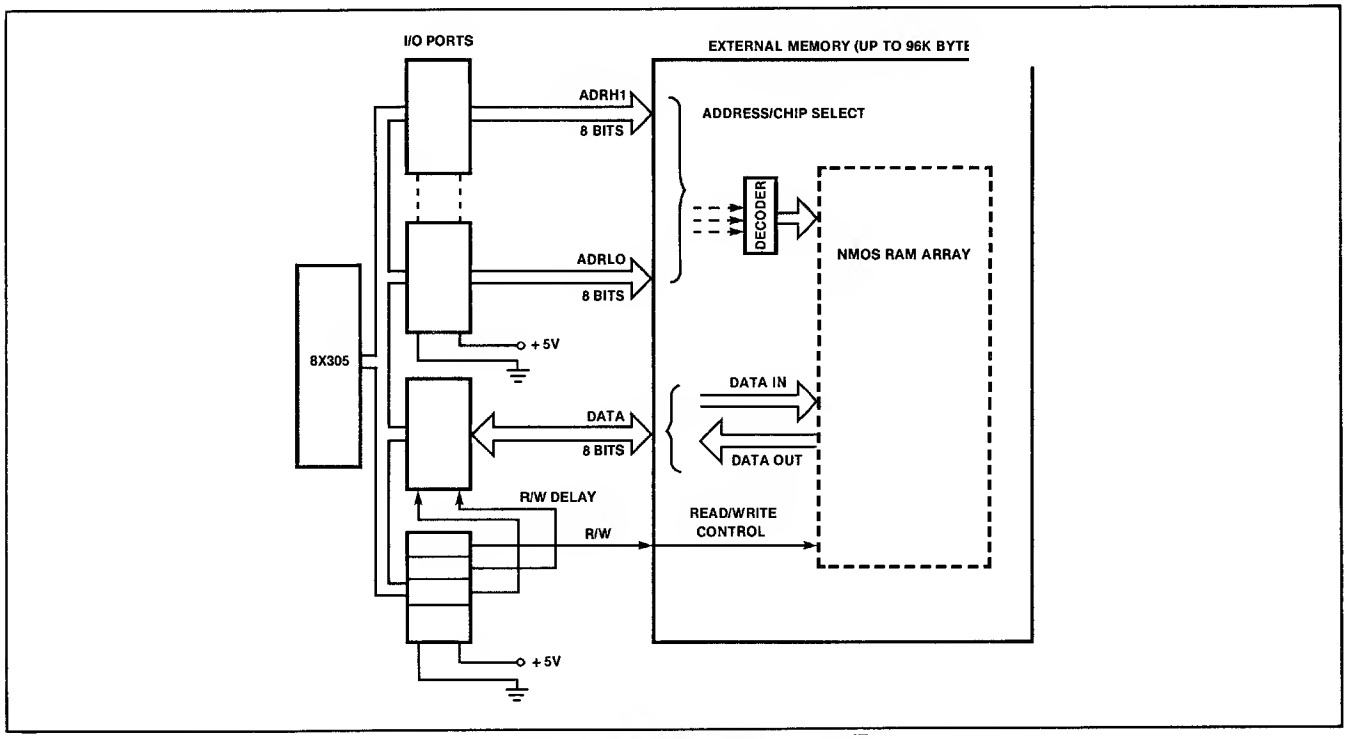


Figure 5-9. General Memory Scheme

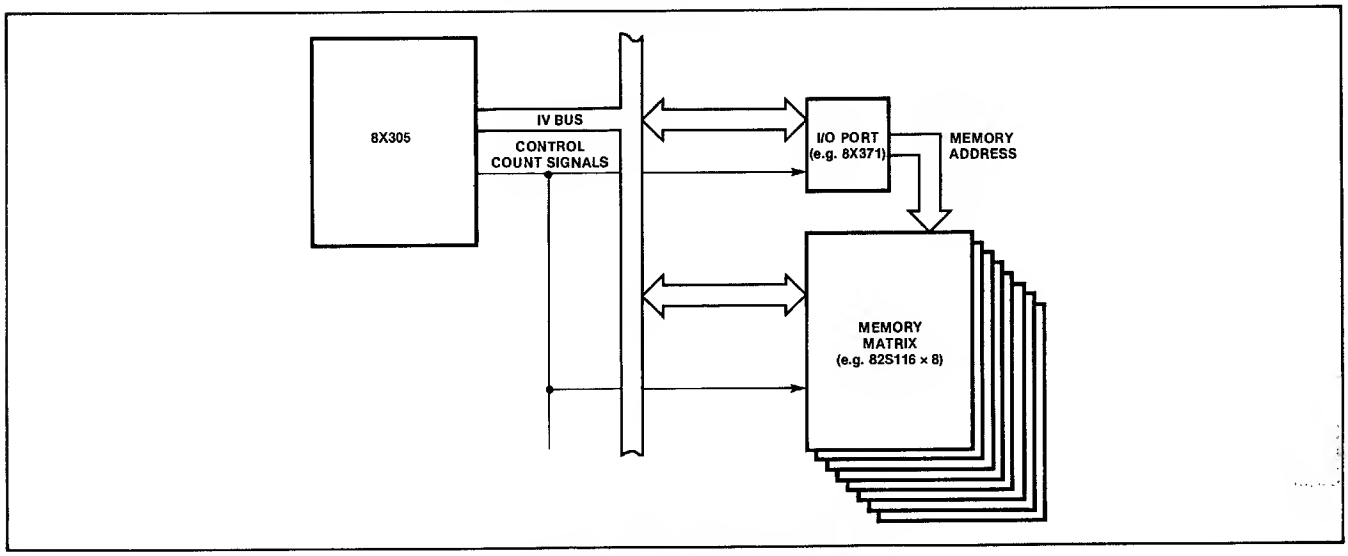


Figure 5-10. Small Data Storage Scheme

5.4.3 Implementation of the 8X360

The need to access large blocks of memory can result in program loops which are excessively long for high-performance applications. Access of memory normally requires a number of steps, which include:

- formation of the address by the 8X305
- testing the address for some termination value
- moving the address to the memory address register
- performing the data transfer (read or write)

The performance of this operation can be greatly improved using the 8X360 Memory Address Director (MAD). This device acts as a quasi-DMA controller, in that it generates sequential memory addresses, which are monitored internally by comparison logic to give simple loop count and block transfer operations. The 8X305 is released from the need to perform the repetitive address update and comparison operations, resulting in a faster and more efficient system.

A typical system usage is illustrated in Figure 5-11. The 16-bit address bus allows up to 64K-bytes of RAM to be addressed directly. Control of the addresses can be achieved in a number of ways, including external stimulus for optimal RAM access timing. Status can be monitored either through an on-board status register, or through external pins, as required.

Since the 8X360 is a sophisticated device, the relevant data sheet and the 8X300 Family Product Capabilities Manual should be consulted for a full appreciation of its capabilities and optimal usage.

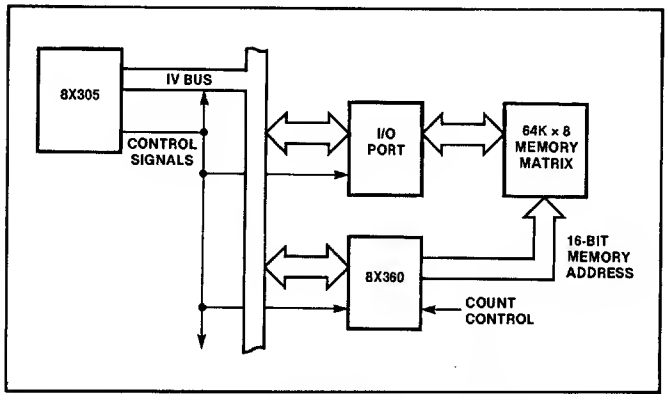


Figure 5-11. System Using the 8X360

5.5 INTERRUPT HANDLING

Many 8X305 applications require the monitoring of external events that occur randomly during normal program flow. Because of the speed of the MicroController itself, the occurrence of such can often be detected using polling techniques.

Because the 8X305 has no on-chip interrupt facility, it is necessary to add hardware to perform the following actions upon receipt of an interrupt signal:

- Save the current Address Register contents
- Make the program jump to a fixed (interrupt) address
- Detect a return instruction
- Restore the original contents of the Address Register

IVL and IVR addresses as well as any registers which are used in the interrupt routine must be saved and restored. This is normally done as part of the interrupt handling software. To save and restore these, at least 16 bytes of RAM is normally required.

5.5.1 Interrupts Using the 8X310

The most efficient approach for most applications which require the handling of asynchronous interrupts involves use of the 8X310 Interrupt Control Coprocessor (ICC). It provides a single-chip solution with minimum program overhead for interrupt handling, and also provides a subroutine call and return facility.

Unlike other members of the 8X300 Family the 8X310 does not connect to the IV bus but is connected to the Instruction and Instruction Address Busses in parallel with the program memory, as shown in Figure 5-12.

The 8X310 monitors the address and instruction bus plus the three maskable, prioritized interrupt input pins. When an interrupt occurs, the device acknowledges it, stores the current Program Address on a four-level stack, disables the program memory and forces the 8X305 to jump to the interrupt handling routine by supplying a JMP instruction.

The 8X310 then releases both busses and the program memory to allow normal program execution of the interrupt handler. It is normally necessary to include a routine in the interrupt handling software which stores the contents of some or all of the internal registers of the 8X305 in external RAM.

Return from the interrupt handler is performed by a RETURN instruction which, like all of the special 8X310

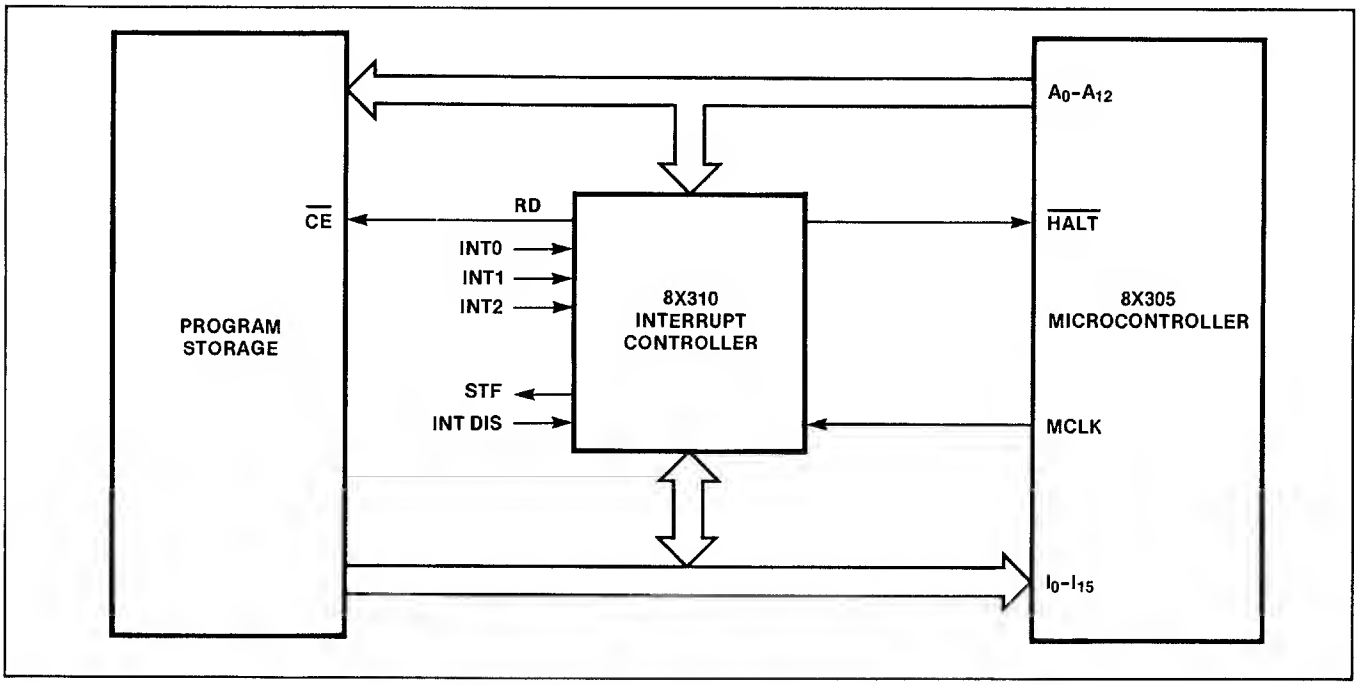


Figure 5-12. System with 8X310 Interrupt Control

instructions, is stored as a part of the program in memory and treated as a No-Op by the 8X305. A summary of the 8X310 instructions is given in Table 5-2.

Table 5-2. Special Interrupt Control Instructions

Machine Code	Instruction	Description
MOVE R5, R5	Set Mask	Sets software interrupt mask.
MOVE R4, R4	Clear Mask	Clears software interrupt mask.
MOVE R6, R6	Return	Returns control to the most recent return on the stack.
MOVE R3, R3	Call	Initiates subroutine and pushes current address onto stack.
MOVE R2, R2	Clear Interrupt	Clears all interrupt requests. A service routine in progress is not affected.

A subroutine can be initiated by a CALL instruction. A combination of up to four interrupts and subroutine calls may be in process at one time without overflowing the return address stack in the 8X310.

Further information on the 8X310 can be found in the 8X310 Data Sheet and the 8X300 Family Product Capabilities Manual.

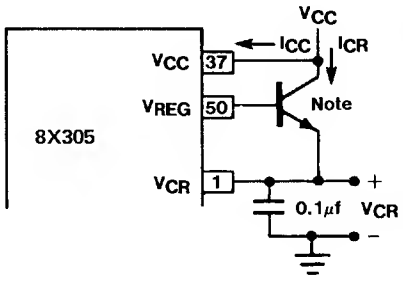
5.5.2 Interrupts Not Using the 8X310

If 8X310 is not used, the following functions have to be implemented in external logic to provide interrupt facility.

- Execute Op Code decoding (since interrupts during XEC cannot be permitted)
- Return decoding
- Return address register

5.6 VOLTAGE REGULATION

All internal logic of the 8X305 is powered by an on-chip voltage regulator that requires an external series-pass transistor. Electrical specifications for the off-chip power transistor and a typical hook-up are shown in Figure 5-13. To minimize lead inductance, the transistor should be as close as possible to the 8X305 package and the emitter should be AC-grounded via a 0.1-microfarad ceramic capacitor.



PARAMETER	CONDITIONS	LIMITS
h_{fe}	V_{CE} = 2V; 100mA < I_C < 500mA	>50
BE_{ON}	V_{CE} = 5V; I_C = 500mA	<1V
V_{CESAT}	I_C = 500mA; I_B = 50mA	<0.5V
BV_{CEO}		>15V
f_t		>30MHz

NOTE:
Typical approved parts—2N5320, 2N5337

Figure 5-13. Voltage Regulation

Chapter 6

SOFTWARE DESIGN

The efficiency of an 8X305 system is as much a function of the manner in which it is programmed as of the hardware architecture. This chapter provides an introduction to programming of the 8X305.

Where examples have been used, they have been written in MicroController Cross Assembler Program (MCCAP) notation. This is described in detail in the MCCAP Manual.

This chapter is not intended to be exhaustive. It serves only to indicate possible solutions to a few common software design problems.

6.1 ARITHMETIC AND LOGICAL OPERATIONS

One of the most powerful features of the 8X305 is the simplicity of structure of the instruction classes. Within each class a number of operations can be performed within one instruction cycle.

However, some common functions of a processor have not been implemented directly in the instruction set. This section outlines simple software routines to perform these functions.

6.1.1 Subtraction

Subtraction is performed using the two's complement method. A routine to perform an 8-bit subtraction of the contents of R2 from those of R1 follows:

```
XMIT  -1,  AUX  (Load AUX with FFFF16)
XOR   R2,  R3   (Store 1's complement of R2 in R3)
XMIT  -1,  AUX  (Load AUX with 000116)
ADD   R3,  AUX  (Add 1 to 1's complement of R2 to
                 form 2's complement)
ADD   R1,  R3   (Perform subtraction)
```

The result is located in R3 with R1 and R2 remaining unchanged. The routine executes in five instruction cycles.

6.1.2 Inclusive-OR

Implementation of a logical OR function can be made using the fact that:

$$A + B = (A + B) + (AB)$$

Using this logical equivalence, a routine to perform an inclusive-OR of two terms contained in R1 and R2 follows:

```
MOVE  R2,  AUX  (Fetch implicit operand from R2)
XOR   R1,  R3   (Perform exclusive OR of R1 and
                 R2 data)
AND   R1,  AUX  (Perform logical AND of R1 and R2
                 data)
XOR   R3,  R3   (XOR of AUX = (A + B) and
                 R3 = (A + B))
```

The result is stored in R3, with R1 and R2 remaining unchanged. The routine executes in four instruction cycles.

6.1.3 Rotate Left

Since the 8X305 has a full circular rotate capability, the equivalent of left rotation of n places can be accomplished by rotating (8-n) places to the right.

A left rotate of the contents of R4 by 3 places would thus be accomplished using the instruction:

```
MOVE R4(5),R4
```

6.2 PROGRAM CONTROL

This section provides information on such common program control techniques as conditional and multi-way branching.

6.2.1 Conditional Branching

During program execution, decisions must be made based on information available only at the time of execution. A conditional branch can be made in the 8X305 based on whether or not port or register data is currently zero. The instruction used to perform this is the NZT (Non-Zero Transfer) instruction.

The instruction requires two operands. The first defines the source of the data which is to be tested and the second defines the location where program execution is to continue if the data is found to be non-zero. Otherwise, program execution continues with the instruction following the NZT. Since the NZT instruction alters only the low order 5 or 8 bits of the program counter depending on the data source, the target instruction must lie within the same 32 or 256 word address page as the NZT instruction.

As an example, the instruction:

```
NZT R1, * + 2
```

examines the contents of internal register R1. If they are found to be non-zero, the next instruction executed will be located at the sum of this NZT instruction address plus 2.

6.2.2 Multi-Way Branching

An application may require the ability to transfer to different locations depending on the value contained in a particular location or I/O Port. This can be accomplished on the 8X305 by using the XEC (Execute) instruction coupled with a table of JMP (Jump) instructions. The XEC will modify the Address Register based on the contents of a register or I/O Port and execute the instruction at the address computed. If the target instruction is a JMP, the Program Counter will be changed and control will be transferred.

As an example, an I/O Port may contain a value of 0, 1, or 2 indicating the status of some external operation. Depending on the value, a different section of program must be executed. The 8X305 instructions to accomplish this are as follows:

```
          •
          XEC   JTABLE(IOPORT)
          •
          •
          •
JTABLE    JMP   LABEL0
          JMP   LABEL1
          JMP   LABEL2
```

Control will be passed to one of the three routines specified in the jump table, depending on the data from the I/O Port.

Since the XEC instruction modifies only the low order 5 or 8 bits of the address register depending on the data source (internal register or IV bus), all of the instructions in this type of routine must be contained in the same address page, including both the XEC and the JMP instructions in the table. The number of posssible jumps, then, is limited to 255 when an internal register is used in the XEC and 31 when IV bus data is used.

6.2.3 Subroutines

Subroutine linkage is most readily accomplished on the 8X305 through the use of the 8X310 Interrupt Control Coprocessor (ICC). If the specific design does not include this device, however, subroutine calls and returns can be performed through a simple software technique.

A unique return link value is assigned to each subroutine call and is loaded into an internal register or I/O Port prior to transfer to the subroutine. Upon completion, the subroutine uses a jump table to return to the calling location.

In the following example, internal register R11 is used to store the return link:

```
          •
          XMIT  0,R11       (Load return
                            link 0)
CALL0     JMP   SUBR        (Jump to
          •                 subroutine)
          •                 (Return location)
          •
          XMIT  1,R11       (Load return
                            link 1)
CALL1     JMP   SUBR        (Jump to
          •                 subroutine)
          •
          •
          XMIT  2,R11       (Load return
                            link 2)
CALL2     JMP   SUBR        (Jump to
                            subroutine)
          •                 (Return location)
          •
          •
SUBR      •                 (Start
                            subroutine)
          •
          •
          •
          XEC   RTAB(R11)   (Return)
          •
          •
          •
RTAB      JMP   CALL0 + 1
          JMP   CALL1 + 1
          JMP   CALL2 + 1
```

Upon completion, the subroutine uses the return link value stored in R11 as an index into a jump table containing the return addresses corresponding to each call.

6.2.4 Looping

Program loops can readily be implemented on the 8X305 using the NZT instruction. It is accomplished by loading an internal register prior to entering the loop, incrementing this register in the loop, and terminating the loop with an NZT that tests the register and exits when the value is zero.

In the example show below, the loop will be executed three times. The two's complement of three is preloaded

into R1. Each time the loop is executed, R1 is incremented by 1. The NZT will transfer control back to the beginning of the loop until R1 becomes 0, at which time the next sequential instruction following the NZT will be executed.

```
        •
        XMIT   11111101B,R1
LOOP    XMIT   1,AUX
        ADD    R1,R1
        •
        •
        •
        NZT    R1,LOOP
```

6.3 CODE CONVERSION

It is frequently required to convert from one code to another. Depending on the codes involved, this can sometimes be accomplished using a series of logical operations. The fastest technique, and the technique that is capable of conversions where a logical algorithm cannot be determined, is to use a basic table lookup scheme. The table is constructed in program memory, and contains a series of XMIT instructions that place the converted value into an internal register or I/O Port. The XMIT instructions in the table are sequenced so that the input code represents the offset from the origin of the table. An XEC instruction is then used to cause the proper XMIT to be executed.

In this example, a value from an I/O Port will be converted to its two's complement and stored in internal register R3.

```
        •
        XEC    CVT(INPUT)
        •
        •
CVT     XMIT   00000000B,R3
        XMIT   11111111B,R3
        XMIT   11111110B,R3
        XMIT   11111101B,R3
```

After conversion, processing will proceed with the instruction following the XEC. Regardless of the input code, the conversion requires two instruction cycles for execution.

The above example will convert codes from 0 to 3. It is possible to use the same technique to convert up to 31 codes of IV bus data or 255 codes of data from an internal register. This limitation caused by the restriction that all possible target instructions of an XEC must reside in the same address page. If it is necessary to convert all possible codes of 5 or 8 bit data, additional code is required to test for a particular code and leave space for the XEC in the same page as the table.

The following example performs a conversion of all possible values contained in internal register R2 and stores the results in R3. The zero code in R2 is tested prior to the XEC instruction and converted independently. The XEC itself is located at the end of the table and contains a literal value of 11111111B. This value, when added to the contents of R2 by the XEC will cause the offset from the start of the table to be 1 less than the contents of R2. Note that the table must be aligned on a 256 word page boundary.

```
        •
        NZT    NOTZ,R2          (Test for 0 code)
        XMIT   00000000B,R3     (Convert 0 code)
        JMP    DONE             (Bypass XEC)
NOTZ    JMP    DOIT             (XEC for non-0
                                code)
        •
        •
        ORG    256,256
CVT     XMIT   11111111B,R3     (Convert 1 code)
        XMIT   11111110B,R3     (Convert 2 code)
        XMIT   11111101B,R3     (Convert 3 code)
        •
        •
        •
        •
        XMIT   00000010B,R3     (Convert 254 code)
        XMIT   00000001B,R3     (Convert 255 code)
DOIT    XEC    11111111B(R2)    Perform non-0
                                convert)
DONE    •
        •
        •
```

The routine requires three cycles to execute for the zero code and four cycles for all other codes. After conversion, execution continues with the instruction at the location after the XEC instruction.

6.4 PERIPHERAL DEVICE OPERATION

Control of peripheral devices attached to the IV bus of the 8X305 is readily accomplished through the use of a number of techniques. This section outlines some of the more common approaches to control of these devices.

6.4.1 Device Polling

Most systems designed using the 8X305 require response to external stimuli that are not under the control of the MicroController and its program. If the system does not utilize an 8X310 Interrupt Control Coprocessor (ICC) or some equivalent circuitry, detection of these events requires the use of a polling scheme.

Polling implies that the 8X305 will monitor the status of the devices attached to its IV bus to determine what, if any, action has occurred externally. If the polling routine detects that something has occurred that requires action, it must transfer control to a routine that provides the appropriate service.

The polling normally must occur within windows of time that are dependent on the particular application. A good example is a disk controller, where data must be processed after it has been retrieved from the disk but before additional data has been read from the disk. Since the disk will continue its rotation regardless of the controller's action, the presence of the data must be detected in a timely manner and it must be processed rapidly.

Routines that perform polling functions are normally inserted into the program so that they execute at all times when the MicroController is in a "wait loop" and not performing other tasks. For some extremely time-critical applications, however, polling may be done at any point where the program can be interrupted from its flow should the external event being polled occur.

External events can be monitored through a number of techniques. Bits from an I/O Port can be tested for zero versus non-zero condition through a single instruction if the external signals are necessarily stable during the MicroController's read phase:

```
NZT   INPUT,LABEL
```

The I/O Port being tested is determined through previous selection or Extended Microcode techniques. If the contents of INPUT, after rotating and masking, are non-zero, control will transfer to the instruction at the location specified by LABEL.

If the event occurs asynchronously from the 8X305 cycle, signals may change on the IV bus during execution, resulting in unpredictable results from the NZT instruction. This condition can be avoided by using the synchronous I/O Port available with the 8X300 Family. In this case, it is necessary to preserve the status of the bus prior to the NZT as follows:

```
MOVE  INPUT,R2 (Store IV data in R2)
NZT   R2,LABEL (Test for zero)
```

The effect of this routine is the same as the single instruction above, except that data is stored and preserved in R2.

In some cases, it may be necessary to test for a particular bit pattern to recognize the event. The 8X305 efficiently handles this through its unique bit manipulation capability:

```
XMIT  01011B,AUX  (Store pattern in AUX)
XOR   INPUT,AUX   (Compare IV data)
NZT   AUX,LABEL   (Branch if different)
```

Another implementation technique is to connect all status signals to a single I/O Port that can be read and tested for all potential external stimuli. If this approach is used, the NZT instruction above may be used to test for a single bit or multiple bits, or the comparison technique may be used to test for any combination. This approach reduces the software overhead required to access multiple ports, and permits communication of a large amount of status information through a single device.

6.4.2 Fast I/O Select

Since Fast I/O Select is implemented through Extended Microcode techniques, the bits that control selection of the I/O Port are not available to the 8X305. If the Fast I/O Selection is used, the programmer must exercise care not to create bus contention by having multiple ports selected simultaneously, one by Fast I/O Select and one by conventional techniques.

6.4.3 Other 8X300 Family Devices

The 8X300 Family offers several devices that can be used to reduce the effort required by the hardware and software designer interfacing to various subsystems. These devices include the following:

- 8X310 Interrupt Control Coprocessor (ICC)—This device monitors external events through maskable, prioritized interrupt pins and is capable of interrupting the normal instruction flow of the 8X305, thus eliminating the need for device polling. It also contains a four level stack for return addresses and affords subroutine capability
- 8X320 Bus Interface Register Array—This device is a two-port memory that contains flag registers to signal changes in data that has been written from either port. The 8X320 simplifies interfacing to both 8 and 16 bit host system interfaces.
- 8X330 Floppy Disk Controller—This device contains all of the circuitry necessary to control multiple floppy disk drives with an 8X305, including PLL data separation and scratchpad memory. Its unique architecture supports all floppy disk formats and permits Error Checking and Correction (ECC).

- 8X350 256 x 8 Bipolar RAM—This device is compatible with 8X305 control signals and IV bus protocol, requiring no additional hardware and software for interface to a system. It is capable of supporting full speed operation of the 8X305.
- 8X360 Memory Address Director (MAD)—This device contains eleven address registers for the control of up to 64K bytes of external RAM for use as working storage by the 8X305. It eliminates the need for a considerable amount of interfacing hardware and numerous instructions that would be required to set up, modify, and monitor the 16 bit addresses that are required for a large memory.

Additional capability, hardware, and programming information on each of these devices can be found in the 8X300 Family Product Capabilities Manual and the specific device Data Sheets.

signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation
811 East Arques Avenue
P.O. Box 409
Sunnyvale, California 94086
Telephone 408/739-7700

98-3001-760

Printed in U.S.A. BA-GP10M0682