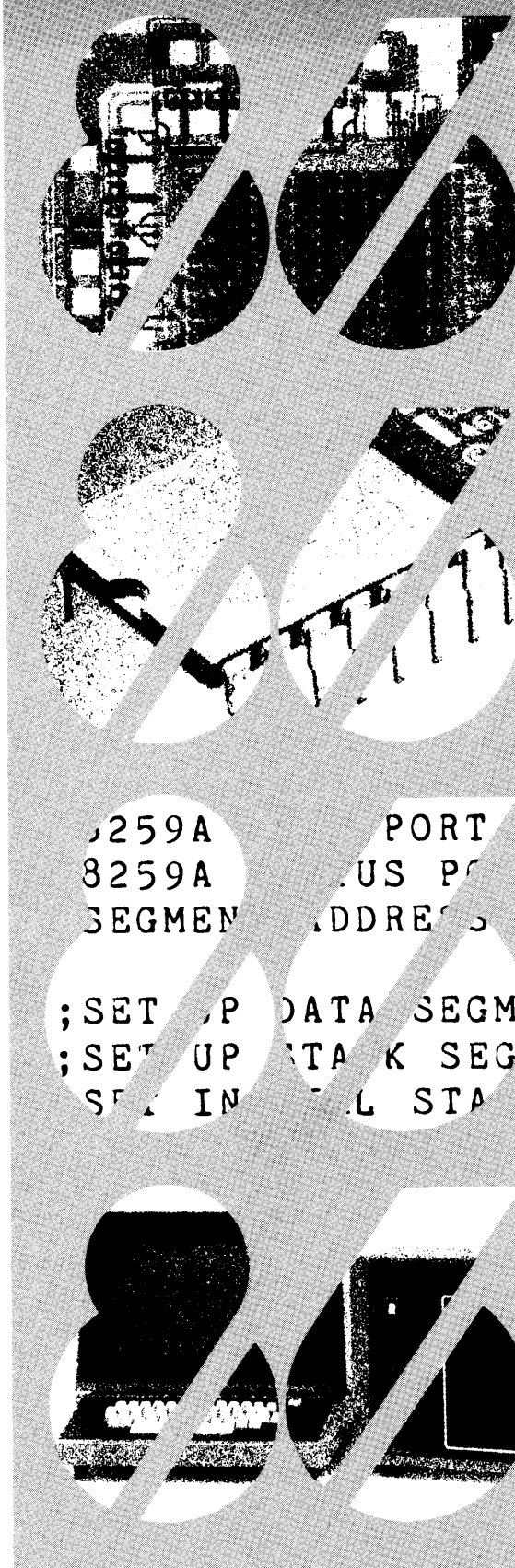
Chapter 4
Hardware Reference Information



CHAPTER 4
HARDWARE REFERENCE INFORMATION

4.1 Introduction

This chapter presents specific hardware information regarding the operation and functions of the 8086 family processors: the 8086 and 8088 Central Processing Units (CPUs) and the 8089 I/O Processor (IOP). Abbreviated descriptions of the 8086 family support circuits and their circuit functions appear where appropriate within the processor descriptions. For more specific information on any of the 8086 family support circuits, refer to the corresponding data sheets in Appendix B.

4.2 8086 and 8088 CPUs

The 8086 and 8088 CPUs are characterized by a 20-bit (1 megabyte) address bus and an identical instruction/function format, and differ essentially from one another by their respective data bus widths (the 8086 uses a 16-bit data bus, and the 8088 uses an 8-bit data bus). Except where expressly noted, the ensuing descriptions are applicable to both CPUs.

Both the 8086 and 8088 feature a combined or "time-multiplexed" address and data bus that permits a number of the pins to serve dual functions and consequently allows the complete CPU to be incorporated into a single, 40-pin package. As explained later in this chapter, a number of the CPU's control pins are defined according to the strapping of a single input pin (the MN/$\overline{MX}$ pin). In the "minimum mode," the CPU is configured for small, single-processor systems, and the CPU itself provides all control signals. In the "maximum mode," an Intel® 8288 Bus Controller, rather than the CPU, provides the control signal outputs and allows a number of the pins previously delegated to these control functions to be redefined in order to support multiprocessing applications. Figures 4-1 and 4-2 describe the pin assignments and signal definitions for the 8086 and 8088, respectively.

CPU Architecture

As shown in figures 4-3 and 4-4, both CPUs incorporate two separate processing units: the Execution Unit or "EU" and the Bus Interface Unit or "BIU." The EU for each processor is identical. The BIU for the 8086 incorporates a 16-bit data bus and a 6-byte instruction queue whereas the 8088 incorporates an 8-bit data bus and a 4-byte instruction queue.

The EU is responsible for the execution of all instructions, for providing data and addresses to the BIU, and for manipulating the general registers and the flag register. Except for a few control pins, the EU is completely isolated from the "outside world." The BIU is responsible for executing all external bus cycles and consists of the segment and communications registers, the instruction pointer and the instruction object code queue. The BIU combines segment and offset values in its dedicated adder to derive 20-bit addresses, transfers data to and from the EU on the ALU data bus and loads or "prefetches" instructions into the queue from which they are fetched by the EU.

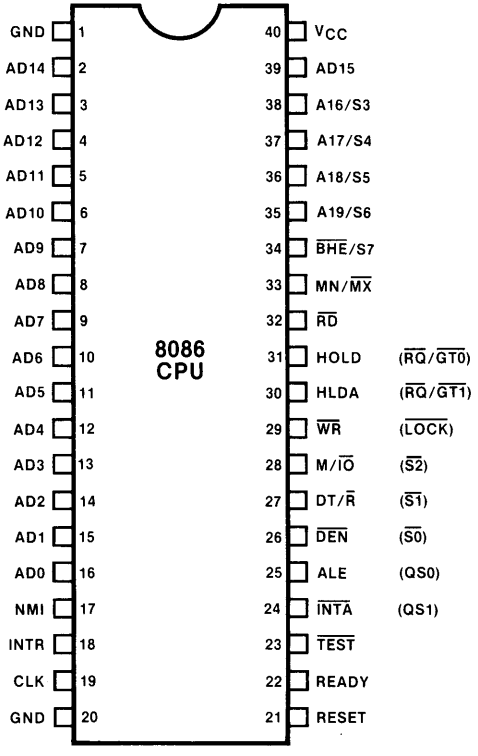
The EU, when it is ready to execute an instruction, fetches the instruction object code byte from the BIU's instruction queue and then executes the instruction. If the queue is empty when the EU is ready to fetch an instruction byte, the EU waits for the instruction byte to be fetched. In the course of instruction execution, if a memory location or I/O port must be accessed, the EU requests the BIU to perform the required bus cycle.

The two processing sections of the CPU operate independently. In the 8086 CPU, when two or more bytes of the 6-byte instruction queue are empty and the EU does not require the BIU to perform a bus cycle, the BIU executes instruction fetch cycles to refill the queue. In the 8088 CPU, when one byte of the 4-byte instruction queue is empty, the BIU executes an instruction fetch cycle. Note that the 8086 CPU, since it has a 16-bit data bus, can access two instruction object code bytes in a single bus cycle, while the 8088 CPU, since it has an 8-bit data bus, accesses one instruction object code byte per bus cycle. If the EU issues a request for bus access while the BIU is in the process of an instruction fetch bus cycle, the BIU completes the cycle before honoring the EU's request.

Common Signals		
Name	**Function**	**Type**
AD15-AD0	Address/Data Bus	Bidirectional, 3-State
A19/S6-A16/S3	Address/Status	Output, 3-State
$\overline{BHE}$/S7	Bus High Enable/ Status	Output, 3-State
MN/$\overline{MX}$	Minimum/Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output, 3-State
$\overline{TEST}$	Wait On Test Control	Input
READY	Wait State Control	Input
RESET	System Reset	Input
NMI	Non-Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
V_{CC}	+5V	Input
GND	Ground	

Minimum Mode Signals (MN/MX = V_{CC})		
Name	**Function**	**Type**
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{WR}$	Write Control	Output, 3-State
M/$\overline{IO}$	Memory/IO Control	Output, 3-State
DT/$\overline{R}$	Data Transmit/ Receive	Output, 3-State
$\overline{DEN}$	Data Enable	Output, 3-State
ALE	Address Latch Enable	Output
$\overline{INTA}$	Interrupt Acknowledge	Output

Maximum Mode Signals (MN/MX = GND)		
Name	**Function**	**Type**
$\overline{RQ}$/$\overline{GT1, 0}$	Request/Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3-State
$\overline{S2}$-$\overline{S0}$	Bus Cycle Status	Output, 3-State
QS1, QS0	Instruction Queue Status	Output



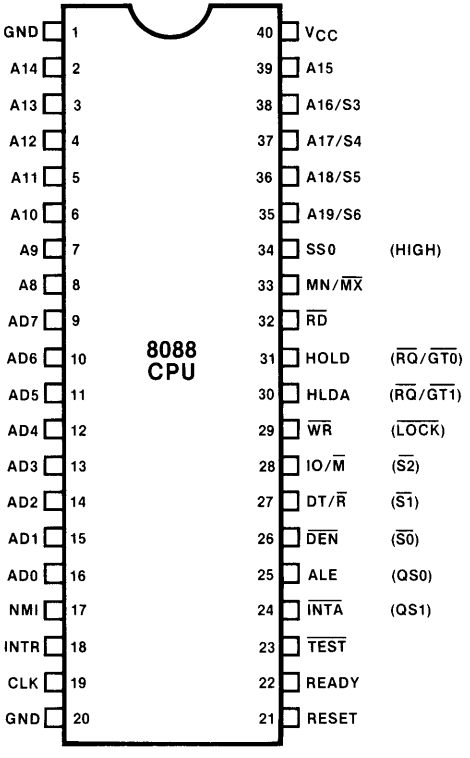
MAXIMUM MODE PIN FUNCTIONS (e.g., $\overline{LOCK}$) ARE SHOWN IN PARENTHESES

Figure 4-1. 8086 Pin Definitions

Common Signals		
Name	**Function**	**Type**
AD7-AD0	Address/Data Bus	Bidirectional, 3-State
A15-A8	Address Bus	Output, 3-State
A19/S6-A16/S3	Address/Status	Output, 3-State
MN/$\overline{MX}$	Minimum/Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output, 3-State
$\overline{TEST}$	Wait On Test Control	Input
READY	Wait State Control	Input
RESET	System Reset	Input
NMI	Non-Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
V_{CC}	+5V	Input
GND	Ground	

Minimum Mode Signals (MN/MX = V_{CC})		
Name	**Function**	**Type**
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{WR}$	Write Control	Output, 3-State
IO/$\overline{M}$	IO/Memory Control	Output, 3-State
DT/$\overline{R}$	Data Transmit/Receive	Output, 3-State
$\overline{DEN}$	Data Enable	Output, 3-State
ALE	Address Latch Enable	Output
$\overline{INTA}$	Interrupt Acknowledge	Output
SS0	S0 Status	Output, 3-State

Maximum Mode Signals (MN/MX = GND)		
Name	**Function**	**Type**
$\overline{RQ}$/GT1, 0	Request/Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3-State
$\overline{S2}$-$\overline{S0}$	Bus Cycle Status	Output, 3-State
QS1, QS0	Instruction Queue Status	Output



MAXIMUM MODE PIN FUNCTIONS (e.g., $\overline{LOCK}$) ARE SHOWN IN PARENTHESES

Figure 4-2. 8088 Pin Definitions

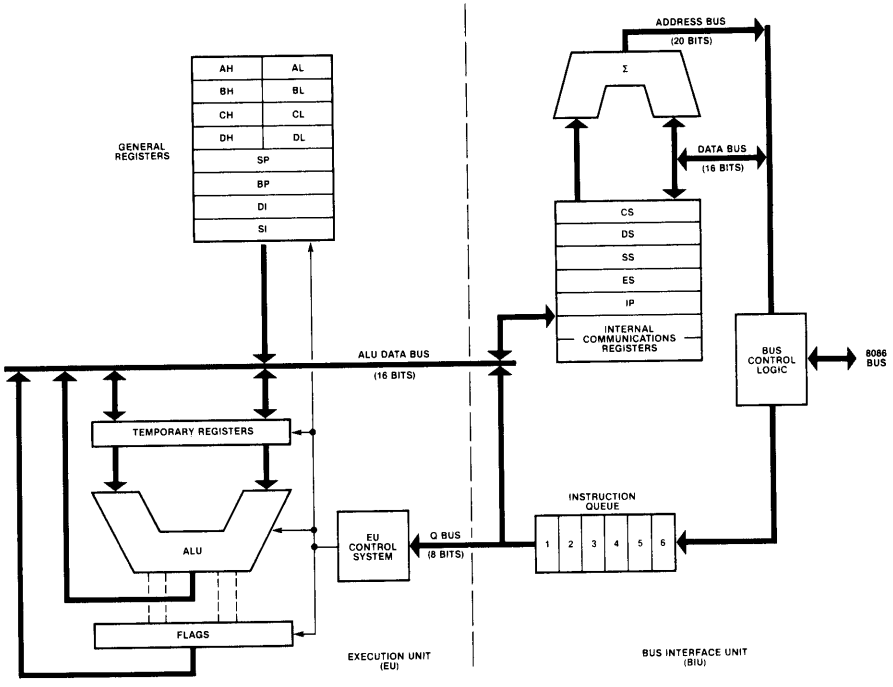


Figure 4-3. 8086 Elementary Block Diagram

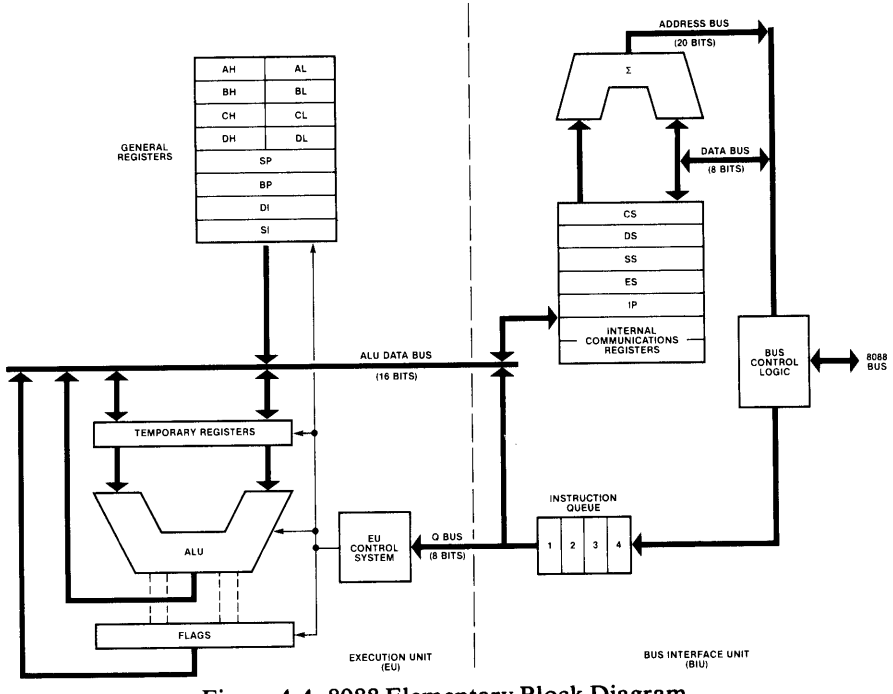


Figure 4-4. 8088 Elementary Block Diagram

Bus Operation

To explain the operation of the time-multiplexed bus, the BIU's bus cycle must be examined. Essentially, a bus cycle is an asynchronous event in which the address of an I/O peripheral or memory location is presented, followed by either a read control signal (to capture or "read" the data from the addressed device) or a write control signal and the associated data (to transmit or "write" the data to the addressed device). The selected device (memory or I/O peripheral) accepts the data on the bus during a write cycle or places the requested data on the bus during a read cycle. On termination of the cycle, the device latches the data written or removes the data read.

As shown in figure 4-5, all bus cycles consist of a minimum of four clock cycles or "T-states" identified as T_1, T_2, T_3 and T_4. The CPU places the address of the memory location or I/O device on the bus during state T_1. During a write bus cycle, the CPU places the data on the bus from state T_2 until state T_4. During a read bus cycle, the CPU accepts the data present on the bus in states T_3 and T_4, and the multiplexed address/data bus is floated in state T_2 to allow the CPU to change from the write mode (output address) to the read mode (input data).

It is important to note that the BIU executes a bus cycle only when a bus cycle is requested by the EU as part of instruction execution or when it must fill the instruction queue. Consequently, clock periods in which there is no BIU activity can occur between bus cycles. These inactive clock periods are referred to as idle states (T_I). While idle clock states result from several conditions (e.g., bus access granted to a coprocessor), as an example, consider the case of the execution of a "long" instruction. In the following example, an 8-bit register multiply (MUL) instruction (which requires between 70 and 77 clock cycles) is executed by the 8086. Assuming that the multiplication routine is entered as a result of a program jump (which causes the instruction queue to be reinitialized when the jump is executed) and, as will be explained later in this chapter, that the object code bytes are aligned on even-byte boundaries, the BIU's bus cycle sequence would appear as shown in figure 4-6.

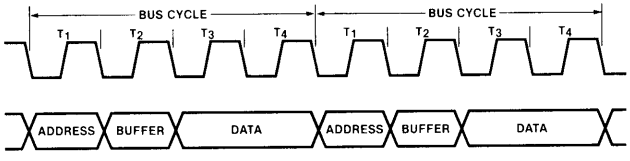


Figure 4-5. Typical BIU Bus Cycles

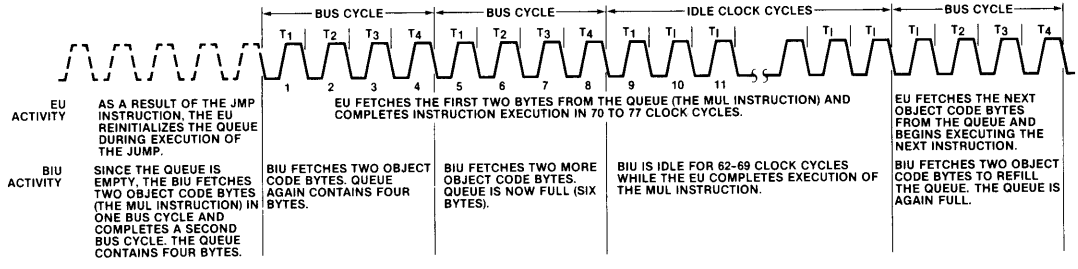


Figure 4-6. BIU Idle States

In addition to the idle state previously described, both the 8086 and 8088 CPUs include a mechanism for inserting additional T-states in the bus cycle to compensate for devices (memory or I/O) that cannot transfer data at the maximum rate. These extra T-states are called wait states (T_W) and, when required, are inserted between states T_3 and T_4. During a wait state, the data on the bus remains unchanged. When the device can complete the transfer (present or accept the data), it signals the CPU to exit the wait state and to enter state T_4.

As shown in the following timing diagrams, the actual bus cycle timing differs between a read and a write bus cycle and varies between the two CPUs. Note that the timing diagrams illustrated are for the minimum mode. (Maximum mode timing is described later in this chapter.)

Referring to figures 4-7 and 4-8, the 8086 CPU places a 20-bit address on the multiplexed address/data bus during state T_1. During state T_2, the CPU removes the address from the bus and either three-states (floats) the lower 16 address/data lines in preparation for a read cycle (figure 4-7) or places write data on these lines (figure 4-8). At this time, bus cycle status is available on the address/status lines. During state T_3, bus cycle status is maintained on the address/status lines and either the write data is maintained or read data is sampled on the lower 16 address/data lines. The bus cycle is terminated in state T_4 (control lines are disabled and the addressed device deselects from the bus).

The 8088 CPU, like the 8086, places a 20-bit address on the multiplexed address/data bus during state T_1 as shown in figures 4-9 and 4-10. Unlike the 8086, the 8088 maintains the address on the address lines (A_{15}-A_8) for the entire bus cycle. During state T_2, the CPU removes the address on the address/data lines (AD_7-AD_0) and either floats these lines in preparation for a read cycle (figure 4-9) or places write data on these lines (figure 4-10). At this time, bus cycle status is available on the address/status lines. During state T_3, bus cycle status is maintained on the address/status lines and either write data is maintained or read data is sampled on the address/data lines. The bus cycle is terminated in state T_4 (control lines are disabled and the addressed device deselects from the bus).

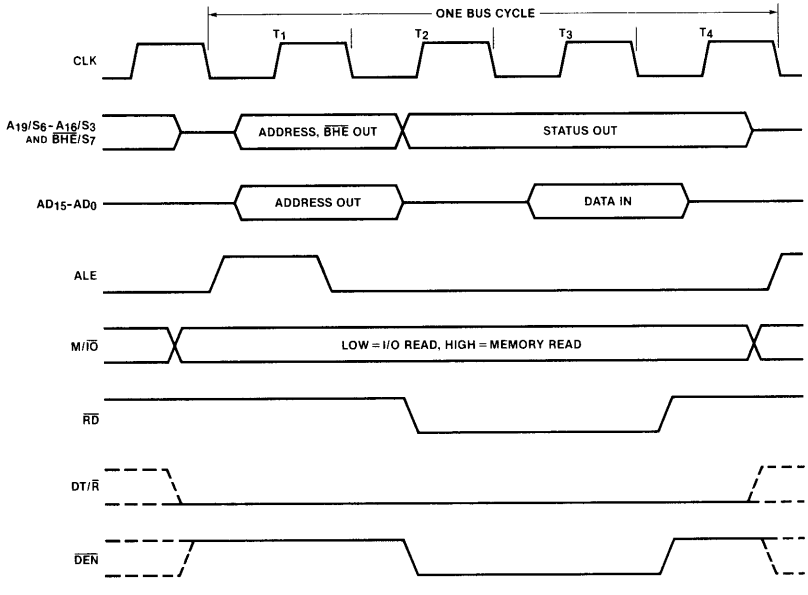


Figure 4-7. 8086 Read Bus Cycle

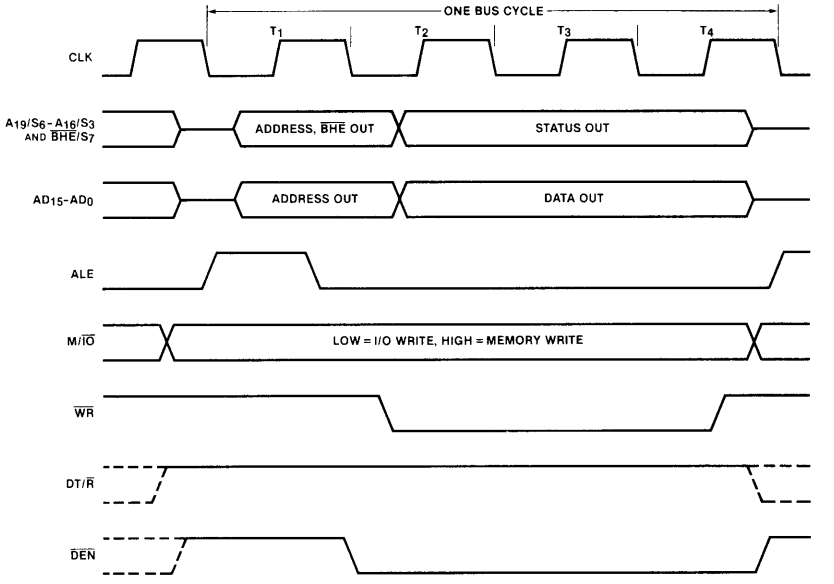


Figure 4-8. 8086 Write Bus Cycle

A majority of system memories and peripherals require a stable address for the duration of the bus cycle (certain MCS-85™ components can operate with a multiplexed address/data bus). During state T_1 of every bus cycle, the ALE (Address Latch Enable) control signal is output (either directly from the microprocessor in the minimum mode or indirectly through an 8288 Bus Controller in the maximum mode) to permit the address to be latched (the address is valid on the trailing-edge of ALE). This "demultiplexing" of the address/data bus can be done remotely at each device in the system or locally at the CPU and distributed throughout the system as a separate address bus. For optimum system performance and for compatibility with multiprocessor systems or with the Intel Multibus architecture, the locally-demultiplexed address bus is recommended. To latch the address, Intel® 8282 (non-inverting) or 8283 (inverting) Octal Latches are offered as part of the 8086 product family and are implemented as shown in figure 4-11. These circuits, in addition to providing the desired latch function, provide increased current drive capability and capacitive load immunity.

The data bus cannot be demultiplexed due to the timing differences between read and write cycles and the various read response times among peripherals and memories. Consequently, the multiplexed data bus either can be buffered or used directly. When memory and I/O peripherals are connected directly to an unbuffered bus, it is essential that during a read cycle, a device is prevented from corrupting the address present on the bus during state T_1. To ensure that the address is not corrupted, a device's output drivers should be enabled by an output enable function (rather than the device's chip select function) controlled by the CPU's read signal. (The MCS-86 family processors guarantee that the read signal will not be valid until after the address has been latched by ALE.) Many Intel peripheral, ROM/EPROM, and RAM circuits provide an output enable function to allow interface to an unbuffered multiplexed address/data bus. The alternative of using a buffered data bus should be considered since it simplifies the interfacing requirements and offers both increased drive current capability and capacitive load immunity. The Intel® 8286 (non-inverting) and 8287 (inverting)

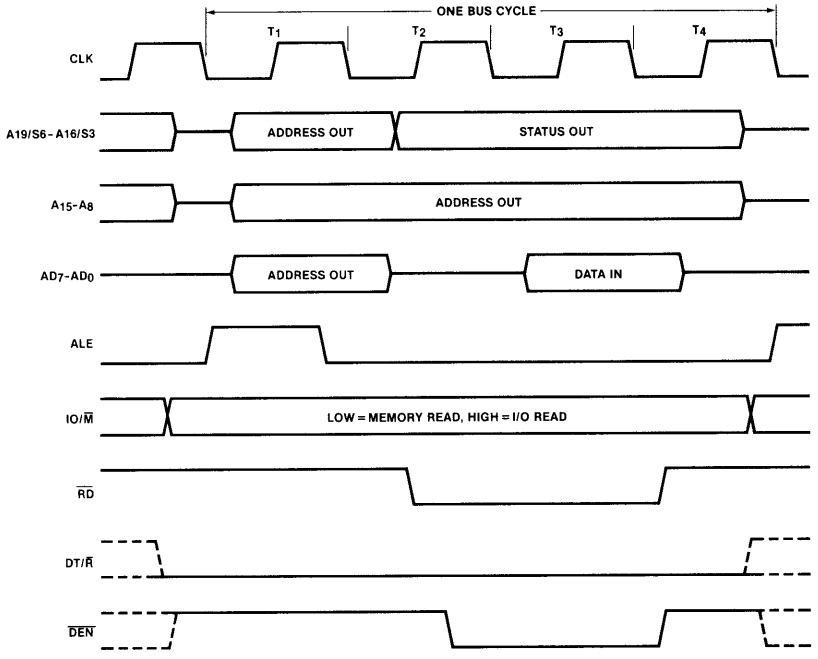


Figure 4-9. 8088 Read Bus Cycle

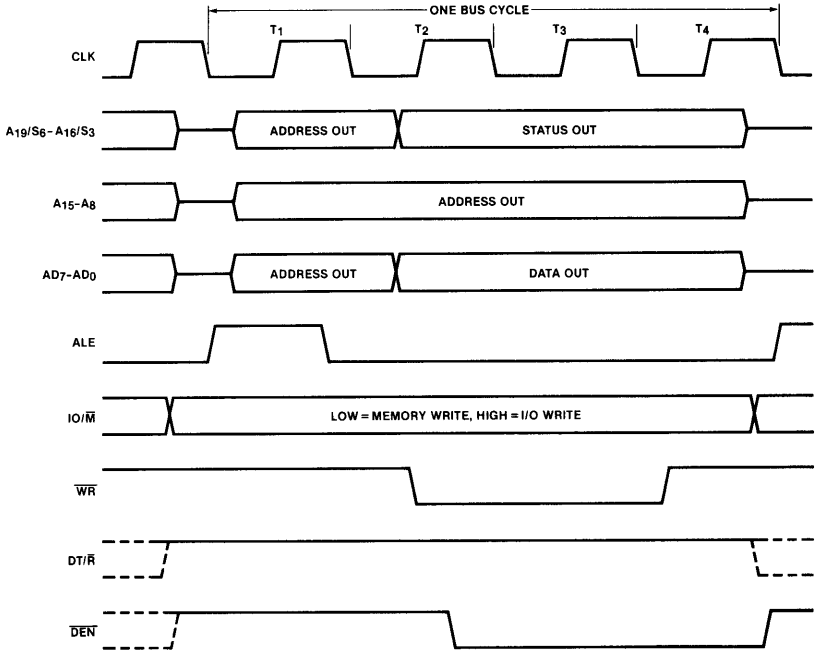


Figure 4-10. 8088 Write Bus Cycle

Octal Bus Transceivers, shown in figure 4-12, are expressly designed to buffer the data bus. These transceivers use the CPU's $\overline{DEN}$ (Data Enable) and DT/$\overline{R}$ (Data Transmit/Receive) control signals to enable and control the direction of data on the bus. These signals provide the proper timing relationship to guarantee isolation of the address that is present on the multiplexed bus during state T_1.

Except where noted, all subsequent discussions and examples in this chapter assume a locally demultiplexed address bus and a buffered data bus. The resultant address and data buses from the address latches and data transceivers to the memory and I/O devices will be referred to collectively as the "system" bus.

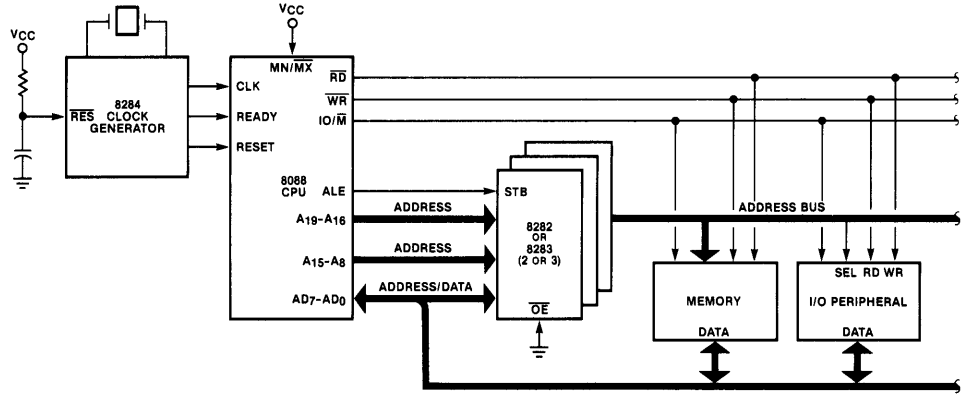


Figure 4-11. Minimum Mode 8088 Demultiplexed Address Bus

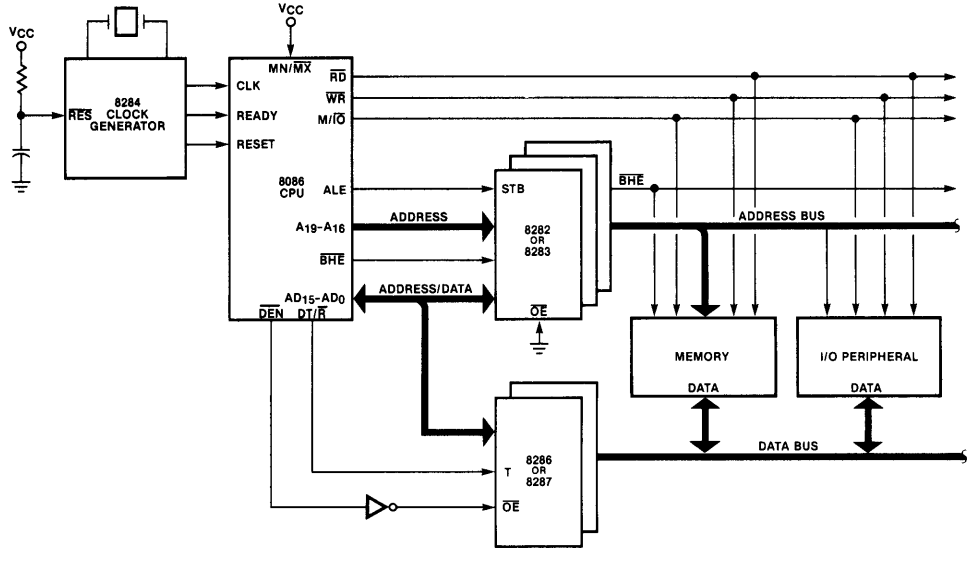


Figure 4-12. Minimum Mode 8086 Buffered Data Bus

Clock Circuit

To establish the bus cycle time, the CPU requires an external clock signal. As an integral part of the 8086 family, Intel offers the 8284 Clock Generator/Driver for this purpose. In addition to providing the primary (system) clock signal, this device provides both the hardware reset interface and the mechanism for the insertion of wait states in the bus cycle.

The clock generator/driver requires an external series-resonant crystal input (or external frequency source) at three times the required system clock frequency (i.e., to operate the CPU at 5 MHz, a 15 MHz fundamental frequency source is required). The divided-by-three output (CLK) from the 8284 is routed directly to the CPU's CLK input. The clock generator/driver provides a second clock output called PCLK (Peripheral Clock) at one half the frequency of the CLK output and a buffered TTL level OSC (oscillator) output at the applied crystal input frequency. These outputs are available for use by system devices.

The 8284's hardware reset function is accomplished with an internal Schmitt trigger circuit that is activated by the $\overline{RES}$ (Reset) input. When this input is pulled low (i.e., a contact closure to ground), the RESET output is activated synchronously with the CLK signal. This signal must be active for four clock cycles and causes the CPU to fetch and execute the instruction at location FFFF0H. An external RC circuit is connected to the RES input to provide the power-on reset function (on power-on, the $\overline{RES}$ input must be active for 50 microseconds). The RESET output is coupled directly to the RESET input of the CPU as well as being available to system peripherals as the system reset signal.

The insertion of wait states in the CPU's bus cycle is accomplished by deactivating one of the 8284's RDY inputs (RDY1 or RDY2). Either of these inputs, when enabled by its corresponding $\overline{AEN1}$ or $\overline{AEN2}$ input, can be deactivated directly by a peripheral device when it must extend the CPU's bus cycle (when it is not ready to present or accept data) or by a "wait state generator" circuit (a logic circuit that holds the RDY input inactive for a given number of clock cycles).

The READY output, which is synchronized to the CLK signal is coupled directly to the CPU's READY input. As shown in figure 4-13, when the addressed device needs to insert one or more wait states in a bus cycle, it deactivates the 8284's RDY input prior to the end of state T_2 which causes the READY output to be deactivated at the end of state T_2. The resultant wait state (T_W) is inserted between states T_3 and T_4. To exit the wait state, the device activates the 8284's RDY input which causes the READY input to the CPU to go active at the end of the current wait state and allows the CPU to enter state T_4.

Minimum/Maximum Mode

A unique feature of the 8086 and 8088 CPUs is the ability of a user to define a subset of the CPU's control signal outputs in order to tailor the CPU to its intended system environment. This "system tailoring" is accomplished by the strapping of the CPU's MN/$\overline{MX}$ (minimum/maximum) input pin. Table 4-1 defines the 8086 and 8088 pin assignments in both the minimum and maximum modes.

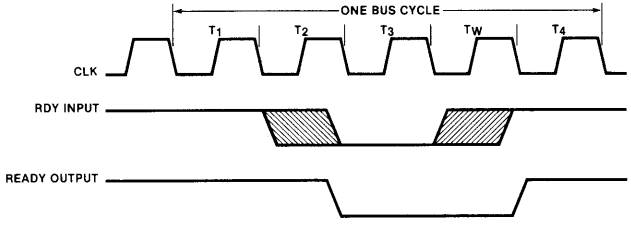


Figure 4-13. Wait State Timing

Table 4-1. Minimum/Maximum Mode Pin Assignments

8086			8088		
Pin	Mode		Pin	Mode	
	Minimum	Maximum		Minimum	Maximum
31	HOLD	$\overline{RQ}/\overline{GT0}$	31	HOLD	$\overline{RQ}/\overline{GT0}$
30	HLDA	$\overline{RQ}/\overline{GT1}$	30	HLDA	$\overline{RQ}/\overline{GT1}$
29	$\overline{WR}$	$\overline{LOCK}$	29	$\overline{WR}$	$\overline{LOCK}$
28	M/$\overline{IO}$	$\overline{S2}$	28	IO/$\overline{M}$	$\overline{S2}$
27	DT/$\overline{R}$	$\overline{S1}$	27	DT/$\overline{R}$	$\overline{S1}$
26	$\overline{DEN}$	$\overline{S0}$	26	$\overline{DEN}$	$\overline{S0}$
25	ALE	QS0	25	ALE	QS0
24	$\overline{INTA}$	QS1	24	$\overline{INTA}$	QS1
			34	SS0	High State

Minimum Mode

In the minimum mode (MN/$\overline{MX}$ pin strapped to +5V), the CPU supports small, single-processor systems that consist of a few devices and that use the system bus rather than support the Multibus™ architecture. In the minimum mode, the CPU itself generates all bus control signals (DT/$\overline{R}$, $\overline{DEN}$, ALE and either M/$\overline{IO}$ or IO/$\overline{M}$) and the command output signal ($\overline{RD}$, $\overline{WR}$ or $\overline{INTA}$), and provides a mechanism for requesting bus access (HOLD/HLDA) that is compatible with bus master type controllers (e.g., the Intel® 8237 and 8257 DMA Controllers).

In the minimum mode, when a bus master requires bus access, it activates the HOLD input to the CPU (through its request logic). The CPU, in response to the "hold" request, activates HLDA as an acknowledgement to the bus master requesting the bus and simultaneously floats the system bus and control lines. Since a bus request is asynchronous, the CPU samples the HOLD input on the positive transition of each CLK signal and, as shown in figure 4-14, activates HLDA at the end of either the current bus cycle (if a bus cycle is in progress) or idle clock period. The hold state is maintained until the bus master inactivates the HOLD input at which time the CPU regains control of the system bus. Note that during a "hold" state, the CPU will continue to execute instructions until a bus cycle is required.

Note that in the minimum mode, the I/O-memory control line for the 8088 CPU is the converse of the corresponding control line for the 8086 CPU (M/$\overline{IO}$ on the 8086 and IO/$\overline{M}$ on the 8088). This was done to provide the 8088 CPU, since it is an 8-bit device, compatibility with existing MCS-85™ systems and specific MCS-85™ family devices (e.g., the Intel® 8155/56).

Maximum Mode

In the maximum mode (MN/$\overline{MX}$ pin strapped to ground), an Intel® 8288 Bus Controller is added to provide a sophisticated bus control function and compatibility with the Multibus architecture (combining an Intel® 8289 Arbiter with the 8288 permits the CPU to support multiple processors on the system bus). As shown in figure 4-15, the bus controller, rather than the CPU, provides all bus control and command outputs, and allows the pins previously delegated to these functions to be redefined to support multiprocessing functions.

$\overline{S2}$, $\overline{S1}$ and $\overline{S0}$

Referring to figure 4-15, the 8288 Bus Controller uses the $\overline{S2}$, $\overline{S1}$ and $\overline{S0}$ status bit outputs from the CPU (and the 8089 IOP) to generate all bus control and command output signals required for a bus cycle. The status bit outputs are decoded as outlined in table 4-2. (For a detailed description of the operation of the 8288 Bus Controller, refer to the associated data sheet in Appendix B.)

The 8088 CPU, in the minimum mode, provides an SS0 status output. This output is equivalent to S0 in the maximum mode and can be decoded with DT/$\overline{R}$ and IO/$\overline{M}$ (inverted), which are equivalent to $\overline{S1}$ and $\overline{S2}$ respectively, to provide the same CPU cycle status information defined in table 4-2. This type of decoding could be used in a minimum mode 8088-based system to allow dynamic RAM refresh during passive CPU cycles.

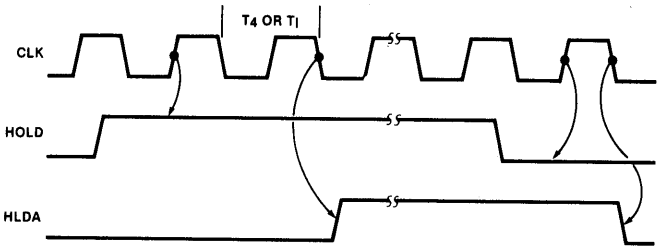


Figure 4-14. HOLD/HLDA Timing

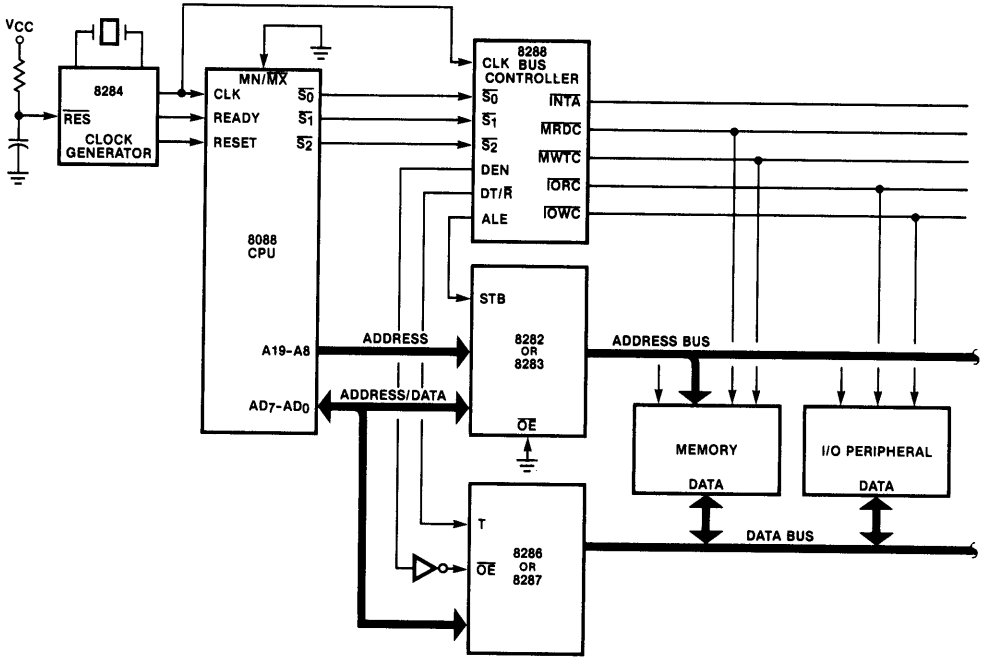


Figure 4-15. Elementary Maximum Mode System

Table 4-2. Status Bit Decoding

Status Inputs			CPU Cycle	8288 Command
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$		
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

$\overline{RQ}/\overline{GT1}$, $\overline{RQ}/\overline{GT0}$

The Request/Grant signal lines ($\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$) provide the CPU's bus access mechanism in the maximum mode (replacing the HOLD/HLDA function available in the minimum mode) and are designed expressly for multiprocessor applications using the 8089 I/O Processor in its local mode or other processors that can support this function. These lines are unique in that the request/grant function is accomplished over a single line ($\overline{RQ}/\overline{GT0}$ or $\overline{RQ}/\overline{GT1}$) rather than the two-line HOLD/HLDA function.

As shown in figure 4-16, the request/grant sequence is a three-phase cycle: request, grant and release. The sequence is initiated by another processor on the system bus when it outputs a pulse on one of the $\overline{RQ}/\overline{GT}$ lines to request bus access (request phase). In response, the CPU outputs a pulse (on the same line) at the end of either the current bus cycle (if a bus cycle is in progress) or idle clock period to indicate to the requesting processor that it has floated the system bus and that it will logically disconnect from the bus controller on the next clock cycle (grant phase) and enter a "hold" state. Note that the CPU's execution unit (EU) continues to execute the instructions in the queue until an instruction requiring bus access is encountered or until the queue is empty. In the third (release) phase, the requesting processor again outputs a pulse on the $\overline{RQ}/\overline{GT}$ line. This pulse alerts the CPU that the processor is ready to release the bus. The CPU regains bus access on its next clock cycle. Note that the exchange of pulses is synchronized and, accordingly, both the CPU and requesting processor must be referenced to the same clock signal.

The request/grant lines are prioritized with $\overline{RQ}/\overline{GT0}$ taking precedence over $\overline{RQ}/\overline{GT1}$. If a request arrives on both lines simultaneously, the processor on $\overline{RQ}/\overline{GT0}$ is granted the bus (the request on $\overline{RQ}/\overline{GT1}$ is granted when the bus is released by the first processor following a one or two clock channel transfer delay). Both $\overline{RQ}/\overline{GT}$ lines (and the HOLD line in minimum mode) have a higher priority than a pending interrupt.

Request/grant latency (the time interval between the receipt of a request pulse and the return of a grant pulse) for several conditions is given in table 4-3.

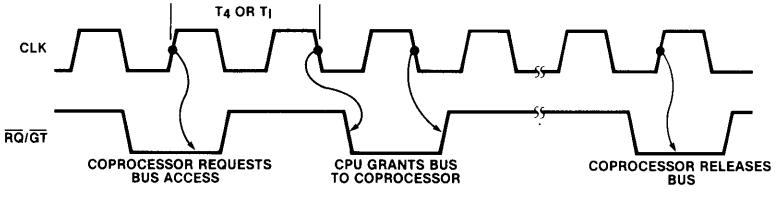


Figure 4-16. Request/Grant Timing

Table 4-3. Request/Grant Latency

Operating Condition	Request/Grant Delay	
	8086	8088
Normal Instruction Processing—$\overline{LOCK}$ inactive	3-6 (10*) clocks	3-10 clocks
$\overline{INTA}$ Cycle Executing—$\overline{LOCK}$ active	15 clocks	15 clocks
Locked XCHG Instruction Processing—$\overline{LOCK}$ active	24-31 (39*) clocks	24-39 clocks

*The number of clocks in parentheses applies when the instruction being executed references a word operand at an odd address boundary.

Latency during normal instruction processing ($\overline{LOCK}$ inactive) can be as short as three clock cycles (e.g., during execution of an instruction that does not reference memory) and no more than ten clock cycles. Whenever the $\overline{LOCK}$ output is active ($\overline{LOCK}$ is activated during an interrupt acknowledge cycle or during execution of an instruction with a Lock prefix), latency is increased. In the case of the execution of a locked XCHG instruction (used during semaphore examination), maximum latency is limited to 39 clock cycles. Greater latencies occur when a "long" instruction is locked. This, however, is neither necessary nor recommended.

At the end of processor activity, the 8086 or 8088 will not redirve its control and data buses until two clock cycles following receipt of the release pulse (or two clock cycles after HOLD goes inactive in the minimum mode).

A Hold request is honored immediately following CPU reset if the HOLD line is active when the RESET line goes inactive. This action facilitates the downloading of programs and, more specifically, the setting of memory location FFFF0H prior to CPU activation. Note that the same result can be effected in the maximum mode through the $\overline{RQ}/\overline{GT}$ line by generating the request pulse in the first or second clock cycle after RESET goes inactive.

$\overline{LOCK}$

The $\overline{LOCK}$ output is used in conjunction with an Intel 8289® Bus Arbiter to guarantee exclusive access of a shared system bus for the duration of an instruction. This output is software controlled and is effected by preceding the instruction requiring exclusive access with a one byte "lock" prefix (see instruction set description in Chapter 2).

When the lock prefix is decoded by the EU, the EU informs the BIU to activate the $\overline{LOCK}$ output during the next clock cycle. This signal remains active until one clock cycle after the execution of the associated instruction is concluded.

QS1, QS0

The QS1 and QS0 (Queue Status) outputs permit external monitoring of the CPU's internal instruction queue to allow instruction set extension processing by a coprocessor. (The corresponding Intel ICE modules use these status bits during "trace" operations.) The encoding of the QS1 and QS0 bits is shown in table 4-4.

Table 4-4. Queue Status Bit Decoding

QS1	QS0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

The queue status is valid during the clock cycle after the indicated activity has occurred.

External Memory Addressing

The 8086 and 8088 CPUs have a 20-bit address bus and are capable of accessing one megabyte of memory address space.

The 8086 memory address space consists of a sequence of up to one million individual bytes in which any two consecutive bytes can be accessed as a 16-bit data word. As shown in figure 4-17, the memory address space is physically divided into two banks of up to 512k bytes each.

One bank is associated with the lower half of the CPU's 16-bit data bus (data bits D7-D0), and the other bank is associated with the upper half of the data bus (data bits D15-D8). Address bits A19 through A1 are used to simultaneously address a specific byte location in both the upper and lower banks, and the A0 address bit is *not* used in memory addressing. Instead, A0 is used in memory bank selection. The lower bank, which

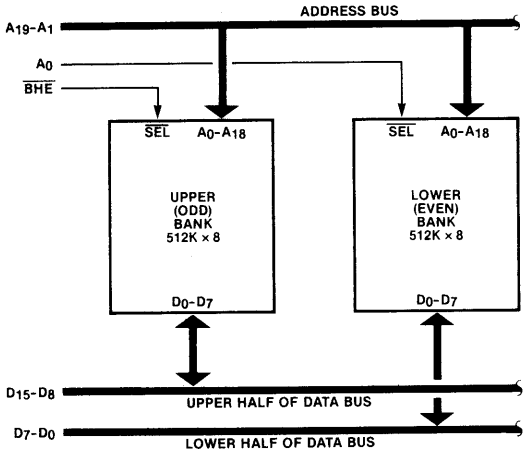


Figure 4-17. 8086 Memory Interface

contains even-address bytes, is selected when A0=0. The upper bank, containing odd address bytes (A0=1), is selected by a separate signal, Bus High Enable ($\overline{BHE}$). Table 4-5 defines the $\overline{BHE}$-A0 bank selection mechanism.

Table 4-5. Memory Bank Selection

$\overline{BHE}$	A0	Byte Transferred
0 (low)	0	Both bytes
0	1	Upper byte to/from odd address
1 (high)	0	Lower byte to/from even address
1	1	None

When accessing a data byte at an even address, the byte is transferred to or from the lower bank on the lower half of the data bus (D7-D0). In this case, the inactive level of the A0 address bit enables the addressed byte in the lower bank, and the inactive level of the $\overline{BHE}$ signal disables the addressed byte in the upper bank. Conversely, when performing a byte access at an odd address, the data byte is transferred to or from the upper bank on the upper half of the data bus (D15-D8). The active level of the $\overline{BHE}$ signal enables the upper bank, and the active level of the A0 address bit disables the lower bank.

As indicated in table 4-5, the 8086 can access a byte in both the upper and lower banks simultaneously as a 16-bit word. When the low-order byte of the word to be accessed is on an even address boundary (that is, when the low-order byte is in the lower bank), the word is said to be "aligned" and can be accessed in a single operation (a single bus cycle). As with the byte transfers previously described, address bits A19 through A1 address both banks, except that now $\overline{BHE}$ is active (selecting the upper bank) and A0 is inactive (selecting the lower bank) to access both bytes.

When the low-order byte of the word to be accessed is on an odd address boundary (when the low-order byte is in the upper bank), the word is "not aligned" and must be accessed in two bus cycles. During the first cycle, the low-order byte of the word is transferred to or from the upper bank as described for a byte access at an odd address (A0 and $\overline{BHE}$ active). The memory address is then incremented, which causes A0 to shift to an inactive level (selecting the lower bank), and a byte access at an even address is performed during the next bus cycle to transfer the word's high-order byte to or from the lower bank. The above sequence is initiated automatically by the 8086 whenever a word access at an odd address is performed. Also, the directing of the high- and low-order bytes of the 8086's internal word registers to the appropriate halves of the data bus is performed automatically and, except for the additional four clock cycles required to execute the second bus cycle, the entire operation is transparent to the program.

The 8088 memory address space is logically organized as a linear array of up to one million bytes. Since the 8088 uses an 8-bit-wide data bus, memory consists of a single bank. Address bit A0 is used to address memory, and a $\overline{BHE}$ signal is not provided.

Word (16-bit) operands can be located at odd- or even-address boundaries. The low-order byte of the word is stored in the lower-valued address location, and the high-order byte is stored in the next, higher-valued address location. The 8088 automatically executes two bus cycles when accessing word operands.

I/O Interfacing

The 8086 and 8088 CPUs support both I/O mapped I/O and memory mapped I/O. I/O mapped I/O permits an I/O device to reside in a separate address space (first 64k of address space), and the standard I/O instruction set is

available for device communications. Memory mapped I/O permits an I/O device to reside anywhere in memory and allows the complete CPU instruction set to be used for I/O operations.

The 8086 supports both 8-bit and 16-bit I/O devices. An 8-bit I/O device may be associated with either the upper or lower half of the data bus. (Assigning an equal number of devices to each half of the data bus distributes bus loading.) When an I/O device is assigned to the lower half of the bus (D7-D0), all I/O addresses must be even (A0 equal "0"), and when an I/O device is assigned to the upper half of the bus, all I/O addresses must be odd (A0 equal "1"). Note that since A0 always will be either a "1" or a "0" for a specific device, it cannot be used as an address input to select registers within the I/O device. When an I/O device on the upper half of the bus and an I/O device on the lower half of the bus are assigned addresses that differ only by the state of A0 (adjacent odd and even addresses), A0 and $\overline{BHE}$ both must be conditions of device selection to prevent a write operation to one device from overwriting data in the other device.

To permit data transfers to 16-bit I/O devices to be performed in a single bus cycle, the device is assigned an even address. To ensure that the I/O device is selected only for word transfers, A0 and $\overline{BHE}$ both must be conditions of device selection.

The 8088, since its data bus is eight bits wide, is designed to support 8-bit I/O devices and places no restrictions on odd or even addresses.

When the 8086 or the 8088 is operated in the minimum mode, the CPU's read and write commands ($\overline{RD}$ and $\overline{WR}$) are common for memory and I/O devices. If the memory and I/O address spaces overlap, device selection must be qualified by M/$\overline{IO}$ (8086) or IO/$\overline{M}$ (8088) to determine if the device is memory or I/O. This restriction does not apply to systems in which I/O and memory addresses do not overlap or to systems that use memory-mapped I/O exclusively. In the maximum mode, the CPU generates (through the bus controller) separate memory read/write and I/O read/write commands in place of the M/$\overline{IO}$ or IO/$\overline{M}$ signal. In a maximum mode system, an I/O device is assigned to an I/O address or to a memory address (memory mapped I/O) by connecting either the memory or I/O read/write command lines to the device's command inputs. When the I/O and memory address spaces overlap, device selection is determined by the appropriate read/write command set.

Interrupts

CPU interrupts can be software or hardware initiated. Software interrupts originate directly from program execution (i.e., execution of a breakpointed instruction) or indirectly through program logic (i.e., attempting to divide by zero). Hardware interrupts originate from external logic and are classified as either non-maskable or maskable. All interrupts, whether software or hardware initiated, result in the transfer of control to a new program location. A 256-entry vector table, which contains address pointers to the interrupt routines, resides in absolute locations 0 through 3FFH. Each entry in this table consists of two 16-bit address values (four bytes) that are loaded into the code segment (CS) and the instruction pointer (IP) registers as the interrupt routine address when an interrupt is accepted. Figure 4-18 illustrates the organization of the 256-entry vector table.

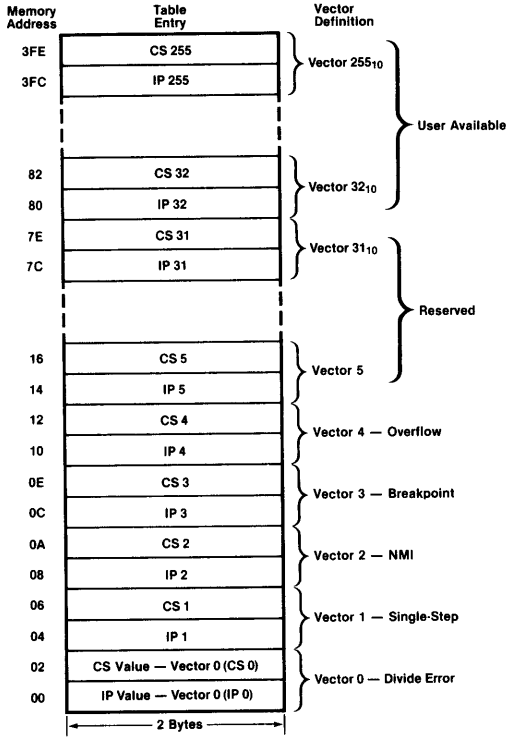


Figure 4-18. Interrupt Vector Table

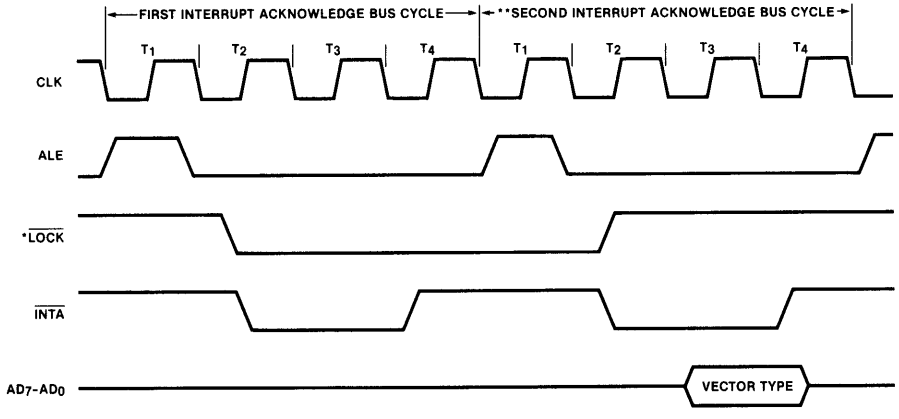
As shown in figure 4-18, the first five interrupt vectors are associated with the software-initiated interrupts and the hardware non-maskable interrupt (NMI). The next 27 interrupt vectors are reserved by Intel and should not be used if compatibility with future Intel products is to be maintained. The remaining interrupt vectors (vectors 32 thorugh 255) are available for user interrupt routines.

The non-maskable interrupt (NMI) occurs as a result of a positive transition at the CPU's NMI input pin. This input is asynchronous and, in order to ensure that it is recognized, is required to have a minimum duration of two clock cycles. NMI is typically used with power fail circuitry, error correcting memory or bus parity detection logic to allow fast response to these fault conditions. When NMI is activated, control is transferred to the interrupt service routine pointed to by vector 2 following execution of the current instruction. When a non-maskable interrupt is acknowledged, the current contents of the flags register are pushed onto the stack (the stack pointer is decremented by two), the interrupt enable and trap bits in the flags register are cleared (disabling maskable and single-step interrupts), and the vector 2 CS and IP address pointers are loaded into the CS and IP registers as the interrupt service routine address.

The CPU provides a single interrupt request input (INTR) that can be software masked by clearing the interrupt enable bit in the flags register through the execution of a CLI instruction. The INTR input is level triggered and is synchronized internally to the positive transition of the CLK signal. In order to be accepted before the next instruction, INTR must be active during the clock period preceding the end of the current instruction (and the interrupt enable bit must be set).

As shown in figure 4-19, when a maskable interrupt is acknowledged, the CPU executes two interrupt acknowledge bus cycles.

During the first bus cycle, the CPU floats the address/data bus and activates the $\overline{\text{INTA}}$ (Interrupt Acknowledge) command output during states T_2 through T_4. In the minimum mode, the CPU will not recognize a hold request from another bus master until the full interrupt acknowledge sequence is completed. In the maximum mode, the CPU activates the $\overline{\text{LOCK}}$ output from state T_2 of the first bus cycle until state T_2 of the second bus cycle to signal all 8289 Bus Arbiters in the system that the bus should not be accessed by any other processor. During the second bus cycle, the CPU again activates its $\overline{\text{INTA}}$ command output. In response to the



*MAXIMUM MODE ONLY
**SEVERAL (3 TYPICAL) IDLE CLOCK STATES OCCUR BETWEEN THE FIRST AND SECOND INTERRUPT ACKNOWLEDGE BUS CYCLES IN THE 8086 CPU (DURING THIS INTERVAL THE BUS IS DRIVEN). INTERRUPT ACKNOWLEDGE BUS CYCLES OCCUR BACK-TO-BACK IN THE 8088 CPU.

Figure 4-19. Interrupt Acknowledge Sequence

second $\overline{\text{INTA}}$, the external interrupt system (e.g., an Intel® 8259A Programmable Interrupt Controller) places a byte on the data bus that identifies the source of the interrupt (the vector number or vector "type"). This byte is read by the CPU and then multiplied by four with the resultant value used as a pointer into the interrupt vector table. Before calling the corresponding interrupt routine, the CPU saves the machine status by pushing the current contents of the flags register onto the stack. The CPU then clears the interrupt enable and trap bits in the flags register to prevent subsequent maskable and single-step interrupts, and establishes the interrupt routine return linkage by pushing the current CS and IP register contents onto the stack before loading the new CS and IP register values from the vector table.

The four classes of interrupts are prioritized with software-initiated interrupts having the highest priority and with maskable and single-step interrupts sharing the lowest priority (see section 2.6). Since the CPU disables maskable and single-step interrupts when acknowledging any interrupt, if recognition of maskable interrupts or single-step operation is required as part of the interrupt routine, the routine first must set these bits.

The processing times for the various classes of interrupts are given in table 4-6. (These times also are included with the 8086/8088 instruction times cited in section 2.7.)

Table 4-6. Interrupt Processing Time

Interrupt Class	Processing Time
External Maskable Interrupt (INTR)	61 clocks
Non-Maskable Interrupt (NMI)	50 clocks
INT (with vector)	51 clocks
INT Type 3	52 clocks
INTO	53 clocks
Single Step	50 clocks

Note that the times shown in table 4-6 represent only the time required to process the interrupt request after it has been recognized. To determine interrupt latency (the time interval between the posting of the interrupt request and the execution of "useful" instructions within the interrupt routine), additional time must be included for the completion on an instruction being executed when the interrupt is posted (interrupts are generally processed only at instruction boundaries), for saving the contents of any additional registers prior to interrupt processing (interrupts automatically save only CS, IP and Flags) and for any wait states that may be incurred during interrupt processing.

Machine Instruction Encoding and Decoding

Writing a MOV instruction in ASM-86 in the form:

```
MOV destination,source
```

will cause the assembler to generate 1 of 28 possible forms of the MOV machine instruction. A programmer rarely needs to know the details of machine instruction formats or encoding. An exception may occur during debugging when it may be necessary to monitor instructions fetched on the bus, read unformatted memory dumps, etc. This section provides the information necessary to translate or decode an 8086 or 8088 machine instruction.

To pack instructions into memory as densely as possible, the 8086 and 8088 CPUs utilize an efficient coding technique. Machine instructions vary from one to six bytes in length. One-byte instructions, which generally operate on single registers or flags, are simple to identify. The keys to decoding longer instructions are in the first two bytes. The format of these bytes can vary, but most instructions follow the format shown in figure 4-20.

The first six bits of a multibyte instruction generally contain an opcode that identifies the basic instruction type: ADD, XOR, etc. The following bit, called the D field, generally specifies the "direction" of the operation: 1 = the REG field in the second byte identifies the destination operand, 0 = the REG field identifies the source operand. The W field distinguishes between byte and word operations: 0 = byte, 1 = word.

One of three additional single-bit fields, S, V or Z, appears in some instruction formats. S is used in conjunction with W to indicate sign extension

of immediate fields in arithmetic instructions. V distinguishes between single- and variable-bit shifts and rotates. Z is used as a compare bit with the zero flag in conditional repeat and loop instructions. All single-bit field settings are summarized in table 4-7.

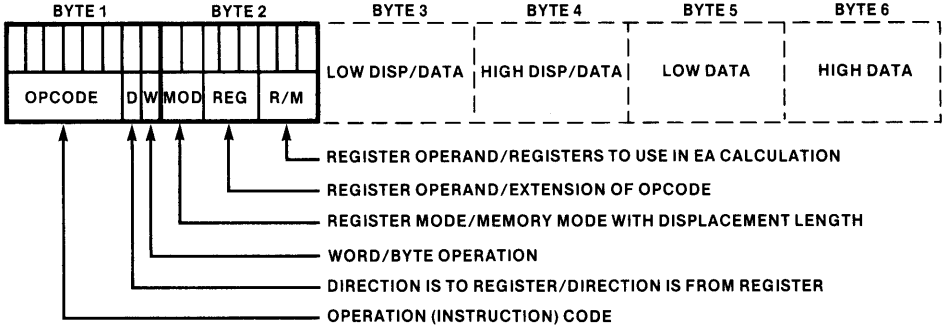


Figure 4-20. Typical 8086/8088 Machine Instruction Format

Table 4-7. Single-Bit Field Encoding

Field	Value	Function
S	0	No sign extension
	1	Sign extend 8-bit immediate data to 16 bits if W=1
W	0	Instruction operates on byte data
	1	Instruction operates on word data
D	0	Instruction source is specified in REG field
	1	Instruction destination is specified in REG field
V	0	Shift/rotate count is one
	1	Shift/rotate count is specified in CL register
Z	0	Repeat/loop while zero flag is clear
	1	Repeat/loop while zero flag is set

The second byte of the instruction usually identifies the instruction's operands. The MOD (mode) field indicates whether one of the operands is in memory or whether both operands are registers (see table 4-8). The REG (register) field identifies a register that is one of the instruction operands (see table 4-9). In a number of instructions, chiefly the immediate-to-memory variety, REG is used as an extension of the opcode to identify the type of operation. The encoding of the R/M (register/memory) field (see table 4-10) depends on how the mode field is set. If MOD = 11 (register-to-register mode), then R/M identifies the second register operand. If MOD selects memory mode, then R/M indicates how the effective address of the memory operand is to be calculated. Effective address calculation is covered in detail in section 2.8.

Bytes 3 through 6 of an instruction are optional fields that usually contain the displacement value of a memory operand and/or the actual value of an immediate constant operand.

Table 4-8. MOD (Mode) Field Encoding

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

Table 4-9. REG (Register) Field Encoding

REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

There may be one or two displacement bytes; the language translators generate one byte whenever possible. The MOD field indicates how many displacement bytes are present. Following Intel convention, if the displacement is two bytes, the most-significant byte is stored second in the instruction. If the displacement is only a single byte, the 8086 or 8088 automatically sign-extends this quantity to 16-bits before using the information in further address calculations. Immediate values always follow any displacement values that may be present. The second byte of a two-byte immediate value is the most significant.

Table 4-12 lists the instruction encodings for all 8086/8088 instructions. This table can be used to predict the machine encoding of any ASM-86 instruction. Table 4-13 lists the 8086/8088 machine instructions in order by the binary value of their first byte. This table can be used to decode any machine instruction from its binary representation. Table 4-11 is a key to the abbreviations used in tables 4-12 and 4-13. Table 4-14 is a more compact instruction decoding guide.

Table 4-10. R/M (Register/Memory) Field Encoding

MOD=11			EFFECTIVE ADDRESS CALCULATION			
R/M	W=0	W=1	R/M	MOD=00	MOD=01	MOD=10
000	AL	AX	000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	CL	CX	001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	DL	DX	010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	BL	BX	011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16
100	AH	SP	100	(SI)	(SI)+D8	(SI)+D16
101	CH	BP	101	(DI)	(DI)+D8	(DI)+D16
110	DH	SI	110	DIRECT ADDRESS	(BP)+D8	(BP)+D16
111	BH	DI	111	(BX)	(BX)+D8	(BX)+D16

Table 4-11. Key to Machine Instruction Encoding and Decoding

IDENTIFIER	EXPLANATION
MOD	Mode field; described in this chapter.
REG	Register field; described in this chapter.
R/M	Register/Memory field; described in this chapter.
SR	Segment register code: 00=ES, 01=CS, 10=SS, 11=DS.
W, S, D, V, Z	Single-bit instruction fields; described in this chapter.
DATA-8	8-bit immediate constant.
DATA-SX	8-bit immediate value that is automatically sign-extended to 16-bits before use.
DATA-LO	Low-order byte of 16-bit immediate constant.
DATA-HI	High-order byte of 16-bit immediate constant.
(DISP-LO)	Low-order byte of optional 8- or 16-bit unsigned displacement; MOD indicates if present.
(DISP-HI)	High-order byte of optional 16-bit unsigned displacement; MOD indicates if present.
IP-LO	Low-order byte of new IP value.
IP-HI	High-order byte of new IP value
CS-LO	Low-order byte of new CS value.
CS-HI	High-order byte of new CS value.
IP-INC8	8-bit signed increment to instruction pointer.
IP-INC-LO	Low-order byte of signed 16-bit instruction pointer increment.
IP-INC-HI	High-order byte of signed 16-bit instruction pointer increment.
ADDR-LO	Low-order byte of direct address (offset) of memory operand; EA not calculated.
ADDR-HI	High-order byte of direct address (offset) of memory operand; EA not calculated.
——	Bits may contain any value.
XXX	First 3 bits of ESC opcode.
YYY	Second 3 bits of ESC opcode.
REG8	8-bit general register operand.
REG16	16-bit general register operand.
MEM8	8-bit memory operand (any addressing mode).
MEM16	16-bit memory operand (any addressing mode).
IMMED8	8-bit immediate operand.
IMMED16	16-bit immediate operand.
SEGREG	Segment register operand.
DEST-STR8	Byte string addressed by DI.

Table 4-11. Key to Machine Instruction Encoding and Decoding (Cont'd.)

IDENTIFIER	EXPLANATION
SRC-STR8	Byte string addressed by SI.
DEST-STR16	Word string addressed by DI.
SRC-STR16	Word string addressed by SI.
SHORT-LABEL	Label within ±127 bytes of instruction.
NEAR-PROC	Procedure in current code segment.
FAR-PROC	Procedure in another code segment.
NEAR-LABEL	Label in current code segment but farther than −128 to +127 bytes from instruction.
FAR-LABEL	Label in another code segment.
SOURCE-TABLE	XLAT translation table addressed by BX.
OPCODE	ESC opcode operand.
SOURCE	ESC register or memory operand.

Table 4-12. 8086 Instruction Encoding

DATA TRANSFER

MOV = Move:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1			
Memory to accumulator	1 0 1 0 0 0 0 w	addr-lo	addr-hi			
Accumulator to memory	1 0 1 0 0 0 1 w	addr-lo	addr-hi			
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		

PUSH = Push:				
Register/memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m	(DISP-LO)	(DISP-HI)
Register	0 1 0 1 0 reg			
Segment register	0 0 0 reg 1 1 0			

POP = Pop:				
Register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)
Register	0 1 0 1 1 reg			
Segment register	0 0 0 reg 1 1 1			

Table 4-12. 8086 Instruction Encoding (Cont'd.)

DATA TRANSFER (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
XCHG = Exchange:						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Register with accumulator	1 0 0 1 0 reg					
IN = Input from:						
Fixed port	1 1 1 0 0 1 0 w	DATA-8				
Variable port	1 1 1 0 1 1 0 w					
OUT = Output to:						
Fixed port	1 1 1 0 0 1 1 w	DATA-8				
Variable port	1 1 1 0 1 1 1 w					
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1					
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m	(DISP-LO)	(DISP-HI)		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(DISP-LO)	(DISP-HI)		
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(DISP-LO)	(DISP-HI)		
LAHF = Load AH with flags	1 0 0 1 1 1 1 1					
SAHF = Store AH into flags	1 0 0 1 1 1 1 0					
PUSHF = Push flags	1 0 0 1 1 1 0 0					
POPF = Pop flags	1 0 0 1 1 1 0 1					

ARITHMETIC

ADD = Add:						
Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w=1			
ADC = Add with carry:						
Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w=1			
INC = Increment:						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)		
Register	0 1 0 0 0 reg					
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1					
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1					

Table 4-12. 8086 Instruction Encoding (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ARITHMETIC (Cont'd.)						
SUB = Subtract:						
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w=1			
SBB = Subtract with borrow:						
Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w=1			
DEC Decrement:						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)		
Register	0 1 0 0 1 reg					
NEG Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		
CMP = Compare:						
Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=1
Immediate with accumulator	0 0 1 1 1 1 0 w	data				
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1					
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1					
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)		
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)		
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	(DISP-LO)	(DISP-HI)		
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m	(DISP-LO)	(DISP-HI)		
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)		
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	(DISP-LO)	(DISP-HI)		
CBW Convert byte to word	1 0 0 1 1 0 0 0					
CWD Convert word to double word	1 0 0 1 1 0 0 1					
LOGIC						
NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)		
SHL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)		
SHR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)		
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)		
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)		

Table 4-12. 8086 Instruction Encoding (Cont'd.)

LOGIC (Cont'd.)	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)		
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)		
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		

AND = And:						
Reg/memory with register to either	0 0 1 0 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1			

TEST = And function to flags no result:						
Register/memory and register	0 0 0 1 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data				

OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w=1			

XOR = Exclusive or:						
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	0 0 1 1 0 1 0 w	data	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w=1			

STRING MANIPULATION	
REP = Repeat	1 1 1 1 0 0 1 z
MOVS = Move byte/word	1 0 1 0 0 1 0 w
CMPS = Compare byte/word	1 0 1 0 0 1 1 w
SCAS = Scan byte/word	1 0 1 0 1 1 1 w
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STDS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

Table 4-12. 8086 Instruction Encoding (Cont'd.)

CONTROL TRANSFER

CALL = Call:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within segment	1 1 1 0 1 0 0 0	IP-INC-LO	IP-INC-HI			
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)		
Direct intersegment	1 0 0 1 1 0 1 0	IP-lo	IP-hi			
		CS-lo	CS-hi			
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		

JMP = Unconditional Jump:						
Direct within segment	1 1 1 0 1 0 0 1	IP-INC-LO	IP-INC-HI			
Direct within segment-short	1 1 1 0 1 0 1 1	IP-INC8				
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)		
Direct intersegment	1 1 1 0 1 0 1 0	IP-lo	IP-hi			
		CS-lo	CS-hi			
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)		

RET = Return from CALL:			
Within segment	1 1 0 0 0 0 1 1		
Within seg adding immed to SP	1 1 0 0 0 0 1 0	data-lo	data-hi
Intersegment	1 1 0 0 1 0 1 1		
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0	data-lo	data-hi
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0	IP-INC8	
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0	IP-INC8	
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0	IP-INC8	
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0	IP-INC8	
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0	IP-INC8	
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0	IP-INC8	
JO = Jump on overflow	0 1 1 1 0 0 0 0	IP-INC8	
JS = Jump on sign	0 1 1 1 1 0 0 0	IP-INC8	
JNE/JNZ = Jump on not equal/not zer0	0 1 1 1 0 1 0 1	IP-INC8	
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1	IP-INC8	
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1	IP-INC8	
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1	IP-INC8	
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1	IP-INC8	
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1	IP-INC8	
JNO = Jump on not overflow	0 1 1 1 0 0 0 1	IP-INC8	

Table 4-12. 8086 Instruction Encoding (Cont'd.)

CONTROL TRANSFER (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
RET = Return from CALL:						
JNS = Jump on not sign	0 1 1 1 1 0 0 1	IP-INC8				
LOOP = Loop CX times	1 1 1 0 0 0 1 0	IP-INC8				
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1	IP-INC8				
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0	IP-INC8				
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1	IP-INC8				

INT = Interrupt:

Type specified	1 1 0 0 1 1 0 1	DATA-8				
Type 3	1 1 0 0 1 1 0 0					
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0					
IRET = Interrupt return	1 1 0 0 1 1 1 1					

PROCESSOR CONTROL

CLC = Clear carry	1 1 1 1 1 0 0 0			
CMC = Complement carry	1 1 1 1 0 1 0 1			
STC = Set carry	1 1 1 1 1 0 0 1			
CLD = Clear direction	1 1 1 1 1 1 0 0			
STD = Set direction	1 1 1 1 1 1 0 1			
CLI = Clear interrupt	1 1 1 1 1 0 1 0			
STI = Set interrupt	1 1 1 1 1 0 1 1			
HLT = Halt	1 1 1 1 0 1 0 0			
WAIT = Wait	1 0 0 1 1 0 1 1			
ESC = Escape (to external device)	1 1 0 1 1 x x x	m o d y y y r / m	(DISP-LO)	(DISP-HI)
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0			
SEGMENT = Override prefix	0 0 1 reg 1 1 0			

Table 4-13. Machine Instruction Decoding Guide

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3, 4, 5, 6	ASM-86 INSTRUCTION FORMAT	
00	0000 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG8/MEM8,REG8
01	0000 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG16/MEM16,REG16
02	0000 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG8,REG8/MEM8
03	0000 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG16,REG16/MEM16
04	0000 0100	DATA-8		ADD	AL,IMMED8
05	0000 0101	DATA-LO	DATA-HI	ADD	AX,IMMED16
06	0000 0110			PUSH	ES
07	0000 0111			POP	ES

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
08	0000 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG8/MEM8,REG8
09	0000 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG16/MEM16,REG16
0A	0000 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG8,REG8/MEM8
0B	0000 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	OR REG16,REG16/MEM16
0C	0000 1100	DATA-8		OR AL,IMMED8
0D	0000 1101	DATA-LO	DATA-HI	OR AX,IMMED16
0E	0000 1110			PUSH CS
0F	0000 1111			(not used)
10	0001 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG8/MEM8,REG8
11	0001 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG16/MEM16,REG16
12	0001 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG8,REG8/MEM8
13	0001 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	ADC REG16,REG16/MEM16
14	0001 0100	DATA-8		ADC AL,IMMED8
15	0001 0101	DATA-LO	DATA-HI	ADC AX,IMMED16
16	0001 0110			PUSH SS
17	0001 0111			POP SS
18	0001 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG8/MEM8,REG8
19	0001 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG16/MEM16,REG16
1A	0001 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG8,REG8/MEM8
1B	0001 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	SBB REG16,REG16/MEM16
1C	0001 1100	DATA-8		SBB AL,IMMED8
1D	0001 1101	DATA-LO	DATA-HI	SBB AX,IMMED16
1E	0001 1110			PUSH DS
1F	0001 1111			POP DS
20	0010 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG8/MEM8,REG8
21	0010 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG16/MEM16,REG16
22	0010 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG8,REG8/MEM8
23	0010 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	AND REG16,REG16/MEM16
24	0010 0100	DATA-8		AND AL,IMMED8
25	0010 0101	DATA-LO	DATA-HI	AND AX,IMMED16
26	0010 0110			ES: (segment override prefix)
27	0010 0111			DAA
28	0010 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	SUB REG8/MEM8,REG8
29	0010 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	SUB REG16/MEM16,REG16
2A	0010 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	SUB REG8,REG8/MEM8
2B	0010 1011	MOD REG R/M	(DISP-LO,(DISP-HI)	SUB REG16,REG16/MEM16
2C	0010 1100	DATA-8		SUB AL,IMMED8
2D	0010 1101	DATA-LO	DATA-HI	SUB AX,IMMED16
2E	0010 1110			CS: (segment override prefix)
2F	0010 1111			DAS
30	0011 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG8/MEM8,REG8
31	0011 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG16/MEM16,REG16
32	0011 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG8,REG8/MEM8
33	0011 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	XOR REG16,REG16/MEM16
34	0011 0100	DATA-8		XOR AL,IMMED8
35	0011 0101	DATA-LO	DATA-HI	XOR AX,IMMED16
36	0011 0110			SS: (segment override prefix)

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
37	0011 0110			AAA
38	0011 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG8/MEM8,REG8
39	0011 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG16/MEM16,REG16
3A	0011 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG8,REG8/MEM8
3B	0011 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	CMP REG16,REG16/MEM16
3C	0011 1100	DATA-8		CMP AL,IMMED8
3D	0011 1101	DATA-LO	DATA-HI	CMP AX,IMMED16
3E	0011 1110			DS: (segment override prefix)
3F	0011 1111			AAS
40	0100 0000			INC AX
41	0100 0001			INC CX
42	0100 0010			INC DX
43	0100 0011			INC BX
44	0100 0100			INC SP
45	0100 0101			INC BP
46	0100 0110			INC SI
47	0100 0111			INC DI
48	0100 1000			DEC AX
49	0100 1001			DEC CX
4A	0100 1010			DEC DX
4B	0100 1011			DEC BX
4C	0100 1100			DEC SP
4D	0100 1101			DEC BP
4E	0100 1110			DEC SI
4F	0100 1111			DEC DI
50	0101 0000			PUSH AX
51	0101 0001			PUSH CX
52	0101 0010			PUSH DX
53	0101 0011			PUSH BX
54	0101 0100			PUSH SP
55	0101 0101			PUSH BP
56	0101 0110			PUSH SI
57	0101 0111			PUSH DI
58	0101 1000			POP AX
59	0101 1001			POP CX
5A	0101 1010			POP DX
5B	0101 1011			POP BX
5C	0101 1100			POP SP
5D	0101 1101			POP BP
5E	0101 1110			POP SI
5F	0101 1111			POP DI
60	0110 0000			(not used)
61	0110 0001			(not used)
62	0110 0010			(not used)
63	0110 0011			(not used)
64	0110 0100			(not used)
65	0110 0101			(not used)
66	0110 0110			(not used)
67	0110 0111			(not used)

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
68	0110 1000			(not used)
69	0110 1001			(not used)
6A	0110 1010			(not used)
6B	0110 1011			(not used)
6C	0110 1100			(not used)
6D	0110 1101			(not used)
6E	0110 1110			(not used)
6F	0110 1111			(not used)
70	0111 0000	IP-INC8		JO SHORT-LABEL
71	0111 0001	IP-INC8		JNO SHORT-LABEL
72	0111 0010	IP-INC8		JB/JNAE/JC SHORT-LABEL
73	0111 0011	IP-INC8		JNB/JAE/JNC SHORT-LABEL
74	0111 0100	IP-INC8		JE/JZ SHORT-LABEL
75	0111 0101	IP-INC8		JNE/JNZ SHORT-LABEL
76	0111 0110	IP-INC8		JBE/JNA SHORT-LABEL
77	0111 0111	IP-INC8		JNBE/JA SHORT-LABEL
78	0111 1000	IP-INC8		JS SHORT-LABEL
79	0111 1001	IP-INC8		JNS SHORT-LABEL
7A	0111 1010	IP-INC8		JP/JPE SHORT-LABEL
7B	0111 1011	IP-INC8		JNP/JPO SHORT-LABEL
7C	0111 1100	IP-INC8		JL/JNGE SHORT-LABEL
7D	0111 1101	IP-INC8		JNL/JGE SHORT-LABEL
7E	0111 1110	IP-INC8		JLE/JNG SHORT-LABEL
7F	0111 1111	IP-INC8		JNLE/JG SHORT-LABEL
80	1000 0000	MOD 000 R/M	(DISP-LO),(DISP-HI),DATA-8	ADD REG8/MEM8,IMMED8
80	1000 0000	MOD 001 R/M	(DISP-LO),(DISP-HI),DATA-8	OR REG8/MEM8,IMMED8
80	1000 0000	MOD 010 R/M	(DISP-LO),(DISP-HI),DATA-8	ADC REG8/MEM8,IMMED8
80	1000 0000	MOD 011 R/M	(DISP-LO),(DISP-HI),DATA-8	SBB REG8/MEM8,IMMED8
80	1000 0000	MOD 100 R/M	(DISP-LO),(DISP-HI),DATA-8	AND REG8/MEM8,IMMED8
80	1000 0000	MOD 101 R/M	(DISP-LO),(DISP-HI),DATA-8	SUB REG8/MEM8,IMMED8
80	1000 0000	MOD 110 R/M	(DISP-LO),(DISP-HI),DATA-8	XOR REG8/MEM8,IMMED8
80	1000 0000	MOD 111 R/M	(DISP-LO),(DISP-HI),DATA-8	CMP REG8/MEM8,IMMED8
81	1000 0001	MOD 000 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	ADD REG16/MEM16,IMMED16
81	1000 0001	MOD 001 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	OR REG16/MEM16,IMMED16
81	1000 0001	MOD 010 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	ADC REG16/MEM16,IMMED16
81	1000 0001	MOD 011 R/M	(DISP-LO),(DISP-HI),DATA-LO,DATA-HI	SBB REG16/MEM16,IMMED16

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT	
81	1000 0001	MOD 100 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	AND	REG16/MEM16,IMMED16
81	1000 0001	MOD 101 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	SUB	REG16/MEM16,IMMED16
81	1000 0001	MOD 110 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	XOR	REG16/MEM16,IMMED16
81	1000 0001	MOD 111 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	CMP	REG16/MEM16,IMMED16
82	1000 0010	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-8	ADD	REG8/MEM8,IMMED8
82	1000 0010	MOD 001 R/M		(not used)	
82	1000 0010	MOD 010 R/M	(DISP-LO),(DISP-HI), DATA-8	ADC	REG8/MEM8,IMMED8
82	1000 0010	MOD 011 R/M	(DISP-LO),(DISP-HI), DATA-8	SBB	REG8/MEM8,IMMED8
82	1000 0010	MOD 100 R/M		(not used)	
82	1000 0010	MOD 101 R/M	(DISP-LO),(DISP-HI), DATA-8	SUB	REG8/MEM8,IMMED8
82	1000 0010	MOD 110 R/M		(not used)	
82	1000 0010	MOD 111 R/M	(DISP-LO),(DISP-HI), DATA-8	CMP	REG8/MEM8,IMMED8
83	1000 0011	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-SX	ADD	REG16/MEM16, IMMED8
83	1000 0011	MOD 001 R/M		(not used)	
83	1000 0011	MOD 010 R/M	(DISP-LO), (DISP-HI), DATA-SX	ADC	REG16/MEM16,IMMED8
83	1000 0011	MOD 011 R/M	(DISP-LO),(DISP-HI), DATA-SX	SBB	REG16/MEM16,IMMED8
83	1000 0011	MOD 100 R/M		(not used)	
83	1000 0011	MOD 101 R/M	(DISP-LO),(DISP-HI), DATA-SX	SUB	REG16/MEM16,IMMED8
83	1000 0011	MOD 110 R/M		(not used)	
83	1000 0011	MOD 111 R/M	(DISP-LO),(DISP-HI), DATA-SX	CMP	REG16/MEM16,IMMED8
84	1000 0100	MOD REG R/M	(DISP-LO),(DISP-HI)	TEST	REG8/MEM8,REG8
85	1000 0101	MOD REG R/M	(DISP-LO),(DISP-HI)	TEST	REG16/MEM16,REG16
86	1000 0110	MOD REG R/M	(DISP-LO),(DISP-HI)	XCHG	REG8,REG8/MEM8
87	1000 0111	MOD REG R/M	(DISP-LO),(DISP-HI)	XCHG	REG16,REG16/MEM16
88	1000 1000	MOD REG R/M	(DISP-LO),(DISP-HI)	MOV	REG8/MEM8,REG8
89	1000 1001	MOD REG R/M	(DISP-LO),(DISP-HI)	MOV	REG16/MEM16/REG16
8A	1000 1010	MOD REG R/M	(DISP-LO),(DISP-HI)	MOV	REG8,REG8/MEM8
8B	1000 1011	MOD REG R/M	(DISP-LO),(DISP-HI)	MOV	REG16,REG16/MEM16
8C	1000 1100	MOD 0SR R/M	(DISP-LO),(DISP-HI)	MOV	REG16/MEM16,SEGREG
8C	1000 1100	MOD 1-- R/M		(not used)	
8D	1000 1101	MOD REG R/M	(DISP-LO),(DISP-HI)	LEA	REG16,MEM16
8E	1000 1110	MOD 0SR R/M	(DISP-LO),(DISP-HI)	MOV	SEGREG,REG16/MEM16
8E	1000 1110	MOD 1-- R/M		(not used)	
8F	1000 1111	MOD 000 R/M	(DISP-LO),(DISP-HI)	POP	REG16/MEM16
8F	1000 1111	MOD 001 R/M		(not used)	
8F	1000 1111	MOD 010 R/M		(not used)	

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
8F	1000 1111	MOD 011 R/M		(not used)
8F	1000 1111	MOD 100 R/M		(not used)
8F	1000 1111	MOD 101 R/M		(not used)
8F	1000 1111	MOD 110 R/M		(not used)
8F	1000 1111	MOD 111 R/M		(not used)
90	1001 0000			NOP (exchange AX,AX)
91	1001 0001			XCHG AX,CX
92	1001 0010			XCHG AX,DX
93	1001 0011			XCHG AX,BX
94	1001 0100			XCHG AX,SP
95	1001 0101			XCHG AX,BP
96	1001 0110			XCHG AX,SI
97	1001 0111			XCHG AX,DI
98	1001 1000			CBW
99	1001 1001			CWD
9A	1001 1010	DISP-LO	DISP-HI,SEG-LO, SEG-HI	CALL FAR_PROC
9B	1001 1011			WAIT
9C	1001 1100			PUSHF
9D	1001 1101			POPF
9E	1001 1110			SAHF
9F	1001 1111			LAHF
A0	1010 0000	ADDR-LO	ADDR-HI	MOV AL,MEM8
A1	1010 0001	ADDR-LO	ADDR-HI	MOV AX,MEM16
A2	1010 0010	ADDR-LO	ADDR-HI	MOV MEM8,AL
A3	1010 0011	ADDR-LO	ADDR-HI	MOV MEM16,AL
A4	1010 0100			MOVS DEST-STR8,SRC-STR8
A5	1010 0101			MOVS DEST-STR16,SRC-STR16
A6	1010 0110			CMPS DEST-STR8,SRC-STR8
A7	1010 0111			CMPS DEST-STR16,SRC-STR16
A8	1010 1000	DATA-8		TEST AL,IMMED8
A9	1010 1001	DATA-LO	DATA-HI	TEST AX,IMMED16
AA	1010 1010			STOS DEST-STR8
AB	1010 1011			STOS DEST-STR16
AC	1010 1100			LODS SRC-STR8
AD	1010 1101			LODS SRC-STR16
AE	1010 1110			SCAS DEST-STR8
AF	1010 1111			SCAS DEST-STR16
B0	1011 0000	DATA-8		MOV AL,IMMED8
B1	1011 0001	DATA-8		MOV CL,IMMED8
B2	1011 0010	DATA-8		MOV DL,IMMED8
B3	1011 1011	DATA-8		MOV BL,IMMED8
B4	1011 0100	DATA-8		MOV AH,IMMED8
B5	1011 0101	DATA-8		MOV CH,IMMED8
B6	1011 0110	DATA-8		MOV DH,IMMED8
B7	1011 0111	DATA-8		MOV BH,IMMED8
B8	1011 1000	DATA-LO	DATA-HI	MOV AX,IMMED16
B9	1011 1001	DATA-LO	DATA-HI	MOV CX,IMMED16
BA	1011 1010	DATA-LO	DATA-HI	MOV DX,IMMED16
BB	1011 1011	DATA-LO	DATA-HI	MOV BX,IMMED16

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT
BC	1011 1100	DATA-LO	DATA-HI	MOV SP,IMMED16
BD	1011 1101	DATA-LO	DATA-HI	MOV BP,IMMED16
BE	1011 1110	DATA-LO	DATA-HI	MOV SI,IMMED16
BF	1011 1111	DATA-LO	DATA-HI	MOV DI,IMMED16
C0	1100 0000			(not used)
C1	1100 0001			(not used)
C2	1100 0010	DATA-LO	DATA-HI	RET IMMED16 (intraseg)
C3	1100 0011			RET (intrasegment)
C4	1100 0100	MOD REG R/M	(DISP-LO),(DISP-HI)	LES REG16,MEM16
C5	1100 0101	MOD REG R/M	(DISP-LO),(DISP-HI)	LDS REG16,MEM16
C6	1100 0110	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-8	MOV MEM8,IMMED8
C6	1100 0110	MOD 001 R/M		(not used)
C6	1100 0110	MOD 010 R/M		(not used)
C6	1100 0110	MOD 011 R/M		(not used)
C6	1100 0110	MOD 100 R/M		(not used)
C6	1100 0110	MOD 101 R/M		(not used)
C6	1100 0110	MOD 110 R/M		(not used)
C6	1100 0110	MOD 111 R/M		(not used)
C7	1100 0111	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	MOV MEM16,IMMED16
C7	1100 0111	MOD 001 R/M		(not used)
C7	1100 0111	MOD 010 R/M		(not used)
C7	1100 0111	MOD 011 R/M		(not used)
C7	1100 0111	MOD 100 R/M		(not used)
C7	1100 0111	MOD 101 R/M		(not used)
C7	1100 0111	MOD 110 R/M		(not used)
C7	1100 0111	MOD 111 R/M		(not used
C8	1100 1000			(not used)
C9	1100 1001			(not used)
CA	1100 1010	DATA-LO	DATA-HI	RET IMMED16 (intersegment)
CB	1100 1011			RET (intersegment)
CC	1100 1100			INT 3
CD	1100 1101	DATA-8		INT IMMED8
CE	1100 1110			INTO
CF	1100 1111			IRET
D0	1101 0000	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL REG8/MEM8,1
D0	1101 0000	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR REG8/MEM8,1
D0	1101 0000	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL REG8/MEM8,1
D0	1101 0000	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR REG8/MEM8,1
D0	1101 0000	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL REG8/MEM8,1
D0	1101 0000	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR REG8/MEM8,1
D0	1101 0000	MOD 110 R/M		(not used)
D0	1101 0000	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR REG8/MEM8,1
D1	1101 0001	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL REG16/MEM16,1
D1	1101 0001	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR REG16/MEM16,1
D1	1101 0001	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL REG16/MEM16,1
D1	1101 0001	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR REG16/MEM16,1
D1	1101 0001	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL REG16/MEM16,1

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT	
D1	1101 0001	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR	REG16/MEM16,1
D1	1101 0001	MOD 110 R/M		(not used)	
D1	1101 0001	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR	REG16/MEM16,1
D2	1101 0010	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL	REG8/MEM8,CL
D2	1101 0010	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR	REG8/MEM8,CL
D2	1101 0010	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL	REG8/MEM8,CL
D2	1101 0010	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR	REG8/MEM8,CL
D2	1101 0010	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL	REG8/MEM8,CL
D2	1101 0010	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR	REG8/MEM8,CL
D2	1101 0010	MOD 110 R/M		(not used)	
D2	1101 0010	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR	REG8/MEM8,CL
D3	1101 0011	MOD 000 R/M	(DISP-LO),(DISP-HI)	ROL	REG16/MEM16,CL
D3	1101 0011	MOD 001 R/M	(DISP-LO),(DISP-HI)	ROR	REG16/MEM16,CL
D3	1101 0011	MOD 010 R/M	(DISP-LO),(DISP-HI)	RCL	REG16/MEM16,CL
D3	1101 0011	MOD 011 R/M	(DISP-LO),(DISP-HI)	RCR	REG16/MEM16,CL
D3	1101 0011	MOD 100 R/M	(DISP-LO),(DISP-HI)	SAL/SHL	REG16/MEM16,CL
D3	1101 0011	MOD 101 R/M	(DISP-LO),(DISP-HI)	SHR	REG16/MEM16,CL
D3	1101 0011	MOD 110 R/M		(not used)	
D3	1101 001 1	MOD 111 R/M	(DISP-LO),(DISP-HI)	SAR	REG16/MEM16,CL
D4	1101 0100	00001010		AAM	
D5	1101 0101	00001010		AAD	
D6	1101 0110			(not used)	
D7	1101 0111			XLAT	SOURCE-TABLE
D8	1101 1000	MOD 000 R/M			
	1XXX	MOD YYY R/M	(DISP-LO), (DISP-HI)	ESC	OPCODE,SOURCE
DF	1101 1111	MOD 111 R/M			
E0	1110 0000	IP-INC-8		LOOPNE/ LOOPNZ	SHORT-LABEL
E1	1110 0001	IP-INC-8		LOOPE/ LOOPZ	SHORT-LABEL
E2	1110 0010	IP-INC-8		LOOP	SHORT-LABEL
E3	1110 0011	IP-INC-8		JCXZ	SHORT-LABEL
E4	1110 0100	DATA-8		IN	AL,IMMED8
E5	1110 0101	DATA-8		IN	AX,IMMED8
E6	1110 0110	DATA-8		OUT	AL,IMMED8
E7	1110 0111	DATA-8		OUT	AX,IMMED8
E8	1110 1000	IP-INC-LO	IP-INC-HI	CALL	NEAR-PROC
E9	1110 1001	IP-INC-LO	IP-INC-HI	JMP	NEAR-LABEL
EA	1110 1010	IP-LO	IP-HI,CS-LO,CS-HI	JMP	FAR-LABEL
EB	1110 1011	IP-INC8		JMP	SHORT-LABEL
EC	1110 1100			IN	AL,DX
ED	1110 1101			IN	AX,DX
EE	1110 1110			OUT	AL,DX
EF	1110 1111			OUT	AX,DX
F0	1111 0000			LOCK	(prefix)
F1	1111 0001			(not used)	
F2	1111 0010			REPNE/REPNZ	
F3	1111 0011			REP/REPE/REPZ	
F4	1111 0100			HLT	
F5	1111 0101			CMC	

Table 4-13. Machine Instruction Decoding Guide (Cont'd.)

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3,4,5,6	ASM-86 INSTRUCTION FORMAT	
F6	1111 0110	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-8	TEST	REG8/MEM8,IMMED8
F6	1111 0110	MOD 001 R/M		(not used)	
F6	1111 0110	MOD 010 R/M	(DISP-LO),(DISP-HI)	NOT	REG8/MEM8
F6	1111 0110	MOD 011 R/M	(DISP-LO),(DISP-HI)	NEG	REG8/MEM8
F6	1111 0110	MOD 100 R/M	(DISP-LO),(DISP-HI)	MUL	REG8/MEM8
F6	1111 0110	MOD 101 R/M	(DISP-LO),(DISP-HI)	IMUL	REG8/MEM8
F6	1111 0110	MOD 110 R/M	(DISP-LO),(DISP-HI)	DIV	REG8/MEM8
F6	1111 0110	MOD 111 R/M	(DISP-LO),(DISP-HI)	IDIV	REG8/MEM8
F7	1111 0111	MOD 000 R/M	(DISP-LO),(DISP-HI), DATA-LO,DATA-HI	TEST	REG16/MEM16,IMMED16
F7	1111 0111	MOD 001 R/M		(not used)	
F7	1111 0111	MOD 010 R/M	(DISP-LO),(DISP-HI)	NOT	REG16/MEM16
F7	1111 0111	MOD 011 R/M	(DISP-LO),(DISP-HI)	NEG	REG16/MEM16
F7	1111 0111	MOD 100 R/M	(DISP-LO),(DISP-HI)	MUL	REG16/MEM16
F7	1111 0111	MOD 101 R/M	(DISP-LO),(DISP-HI)	IMUL	REG16/MEM16
F7	1111 0111	MOD 110 R/M	(DISP-LO),(DISP-HI)	DIV	REG16/MEM16
F7	1111 0111	MOD 111 R/M	(DISP-LO),(DISP-HI)	IDIV	REG16/MEM16
F8	1111 1000			CLC	
F9	1111 1001			STC	
FA	1111 1010			CLI	
FB	1111 1011			STI	
FC	1111 1100			CLD	
FD	1111 1101			STD	
FE	1111 1110	MOD 000 R/M	(DISP-LO),(DISP-HI)	INC	REG8/MEM8
FE	1111 1110	MOD 001 R/M	(DISP-LO),(DISP-HI)	DEC	REG8/MEM8
FE	1111 1110	MOD 010 R/M		(not used)	
FE	1111 1110	MOD 011 R/M		(not used)	
FE	1111 1110	MOD 100 R/M		(not used)	
FE	1111 1110	MOD 101 R/M		(not used)	
FE	1111 1110	MOD 110 R/M		(not used)	
FE	1111 1110	MOD 111 R/M		(not used)	
FF	1111 1111	MOD 000 R/M	(DISP-LO),(DISP-HI)	INC	MEM16
FF	1111 1111	MOD 001 R/M	(DISP-LO),(DISP-HI)	DEC	MEM16
FF	1111 1111	MOD 010 R/M	(DISP-LO),(DISP-HI)	CALL	REG16/MEM16 (intra)
FF	1111 1111	MOD 011 R/M	(DISP-LO),(DISP-HI)	CALL	MEM16 (intersegment)
FF	1111 1111	MOD 100 R/M	(DISP-LO),(DISP-HI)	JMP	REG16/MEM16 (intra)
FF	1111 1111	MOD 101 R/M	(DISP-LO),(DISP-HI)	JMP	MEM16 (intersegment)
FF	1111 1111	MOD 110 R/M	(DISP-LO),(DISP-HI)	PUSH	MEM16
FF	1111 1111	MOD 111 R/M		(not used)	

Table 4-14. Machine Instruction Encoding Matrix

HI \ Lo	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD b,f,r/m	ADD w,f,r/m	ADD b,t,r/m	ADD w,t,r/m	ADD b,ia	ADD w,ia	PUSH ES	POP ES	OR b,f,r/m	OR w,f,r/m	OR b,t,r/m	OR w,t,r/m	OR b,i	OR w,i	PUSH CS	
1	ADC b,f,r/m	ADC w,f,r/m	ADC b,t,r/m	ADC w,t,r/m	ADC b,i	ADC w,i	PUSH SS	POP SS	SBB b,f,r/m	SBB w,f,r/m	SBB b,t,r/m	SBB w,t,r/m	SBB b,i	SBB w,i	PUSH DS	POP DS
2	AND b,f,r/m	AND w,f,r/m	AND b,t,r/m	AND w,t,r/m	AND b,i	AND w,i	SEG =ES	DAA	SUB b,f,r/m	SUB w,f,r/m	SUB b,t,r/m	SUB w,t,r/m	SUB b,i	SUB w,i	SEG =CS	DAS
3	XOR b,f,r/m	XOR w,f,r/m	XOR b,t,r/m	XOR w,t,r/m	XOR b,i	XOR w,i	SEG =SS	AAA	CMP b,f,r/m	CMP w,f,r/m	CMP b,t,r/m	CMP w,t,r/m	CMP b,i	CMP w,i	SEG =DS	AAS
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6																
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	Immed b,r/m	Immed w,r/m	Immed b,r/m	Immed is,r/m	TEST b,r/m	TEST w,r/m	XCHG b,r/m	XCHG w,r/m	MOV b,f,r/m	MOV w,f,r/m	MOV b,t,r/m	MOV w,t,r/m	**MOV sr,f,r/m**	LEA	**MOV sr,t,r/m**	POP r/m
9	XCHG AX	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI	CBW	CWD	CALL l,d	WAIT	PUSHF	POPF	SAHF	LAHF
A	MOV m → AL	MOV m → AX	MOV AL → m	MOV AX → m	**MOVS**	**MOVS**	**CMPS**	**CMPS**	**TEST b,I,a**	**TEST w,I,a**	**STOS**	**STOS**	**LODS**	**LODS**	**SCAS**	**SCAS**
B	MOV i → AL	MOV i → CL	MOV i → DL	MOV i → BL	MOV i → AH	MOV i → CH	MOV i → DH	MOV i → BH	MOV i → AX	MOV i → CX	MOV i → DX	MOV i → BX	MOV i → SP	MOV i → BP	MOV i → SI	MOV i → DI
C			RET, (i+SP)	RET	LES	LDS	MOV b,i,r/m	MOV w,i,r/m			RET, I,(i+SP)	RET I	INT Type 3	INT (Any)	INTO	IRET
D	Shift b	Shift w	Shift b,v	Shift w,v	AAM	AAD		XLAT	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	**IN b**	**IN w**	**OUT b**	**OUT w**	CALL d	JMP d	JMP I,d	JMP si,d	**IN v,b**	**IN v,w**	**OUT v,b**	**OUT v,w**
F	LOCK		REP	REP z	HLT	CMC	Grp 1 b,r/m	Grp 1 w,r/m	CLC	STC	CLI	STI	CLD	STD	Grp 2 b,r/m	Grp 2 w,r/m

where:

mod□r/m	000	001	010	011	100	101	110	111
Immed	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
Shift	ROL	ROR	RCL	RCR	SHL/SAL	SHR	—	SAR
Grp 1	TEST	—	NOT	NEG	MUL	IMUL	DIV	IDIV
Grp 2	INC	DEC	CALL id	CALL I,id	JMP id	JMP I,id	PUSH	—

b = byte operation
d = direct
f = from CPU reg
i = immediate
ia = immed. to accum.
id = indirect
is = immed. byte, sign ext.
I = long ie. intersegment
m = memory
r/m = EA is second byte
si = short intrasegment
sr = segment register
t = to CPU reg
v = variable
w = word operation
z = zero

8086 Instruction Sequence

Figure 4-22 illustrates the internal operation and bus activity that occur as an 8086 CPU executes a sequence of instructions. This figure presents the signals and timing relationships that are important in understanding 8086 operation. The following discussion is intended to help in the interpretation of the figure.

Figure 4-22 shows the repeated execution of an instruction loop. This loop is defined in both machine code and assembly language by figure 4-21. A loop was chosen both to demonstrate the effects of a program jump on the queue and to make the instruction sequence easy to follow. The program sequence shown was selected for several reasons. First, consisting of seven instructions and 16 bytes, the sequence is typical of the tight loops found in many application programs. Second, this particular sequence contains several short, fast-executing instructions that demonstrate both the effect of the queue on CPU performance and the interaction between the execution unit (EU) fetching code from the queue and the bus interface unit (BIU) filling the queue and performing the requested bus cycles. Last, for the purpose of this discussion, code, stack, and memory data references were arranged to be aligned on even word boundaries.

ASSEMBLY LANGUAGE	MACHINE CODE
MOV AX, 0F802H	B802F8
PUSH AX	50
MOV CX, BX	8BCB
MOV DX, CX	8BD1
ADD AX, [SI]	0304
ADD SI, 8086H	81C68680
JMP $ −14	EBF0

Figure 4-21. Instruction Loop Sequence

Figure 4-22 can be more easily interpreted by keeping the following guidelines in mind.

- The queue status lines (QS0, QS1) are the key indicators of EU activity.
- Status lines $\overline{S2}$ through $\overline{S0}$ are the main indicators of 8086/8088 bus activity.
- Interaction of the BIU and EU is via the queue for prefetched opcodes and via the EU for requested bus cycles for data operands.

Keeping these guidelines in mind, the instruction sequence depicted in figure 4-22 can be described as follows. Starting the loop arbitrarily in clock cycle 1 with the queue reinitialization that occurs as part of the JMP instruction, JMP instruction execution is completed by the EU, while the BIU performs an opcode fetch to begin refilling the queue. (Note that a shorthand notation has been used in the figure to represent the two queue status lines and the three status lines—active periods on any of these lines are noted and the binary value of the lines is indicated above each active region.)

In clock cycle 8, the queue status lines indicate that the first byte of the MOV immediate instruction has been removed from the queue (one clock cycle after it was placed there by the BIU fetch) and that execution of this instruction has begun. The second byte of this instruction is taken from the queue in clock cycle 10 and then, in clock cycle 12, the EU pauses to wait one clock cycle for the BIU's second opcode fetch to be completed and for the third byte of the MOV immediate instruction to be available for execution (remember the queue status lines indicate queue activity that has occurred in the previous clock cycle).

Clock cycle 13 begins the execution of the PUSH AX instruction, and in clock cycle 15, the BIU begins the fourth opcode fetch. The BIU finishes the fourth fetch in clock cycle 18 and prepares for another fetch when it receives a request from the EU for a memory write (the stack push). Instead of completing the opcode fetch and forcing the EU to wait four additional clock cycles, the BIU immediately aborts the fetch cycle (resulting in two idle clock cycles (T_I) in clock cycles 19 and 20) and performs the required memory write. This interaction between the EU and BIU results in a single clock extension to the execution time of the PUSH AX instruction, the maximum delay that can occur in response to an EU bus cycle request.

Execution continues in clock cycle 24 with the execution of back-to-back, register-to-register MOV instructions. The first of these instructions takes full advantage of the prefetched opcode to complete this operation in two clock cycles. The second MOV instruction, however, depletes the queue and requires two additional clock cycles (clock cycles 28 and 29).

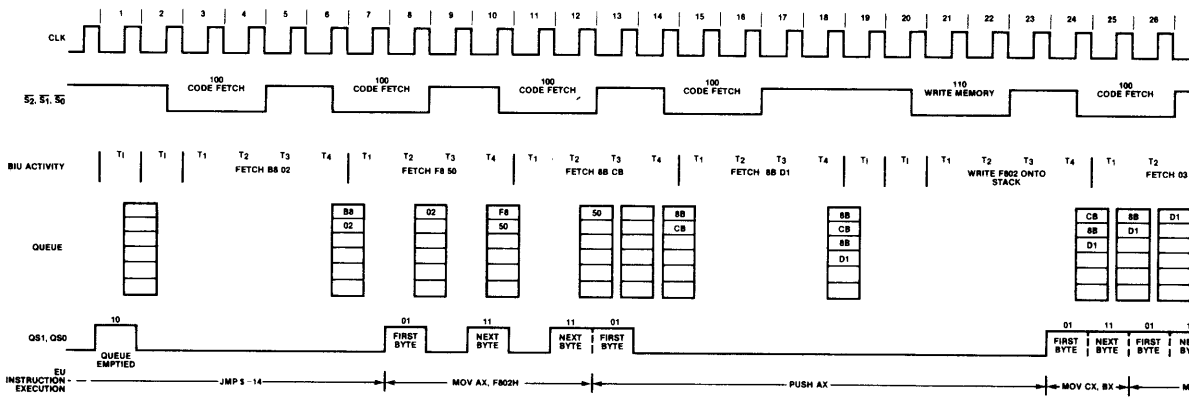


Figure 4-22. Sample Instruction Sequence Execution

In clock cycle 30, the ADD memory indirect to AX instruction begins. In the time required to execute this instruction, the BIU completes two opcode fetch cycles and a memory read and begins a fourth opcode fetch cycle. Note that in the case of the memory read, the EU's request for a bus cycle occurs at a point in the BIU fetch cycle where it can be incorporated directly (idle states are not required and no EU delay is imposed).

In clock cycle 44, the EU begins the ADD immediate instruction, taking four bytes from the queue and completing instruction execution in four clock cycles. Also during this time, the BIU senses a full queue in clock cycle 45 and enters a series of bus idle states (five or six bytes constitute a full queue in the 8086; the BIU waits until it can fetch a full word of opcode before accessing the bus).

At clock cycle 47, the BIU again begins a bus cycle sequence, one that is destined to be an "overfetch" since the EU is executing a JMP instruction. As part of the JMP instruction, the queue reinitialization (which began the instruction sequence) occurs.

The entire sequence of instructions has taken 55 clock cycles. Eighteen opcode bytes were fetched, one word memory read occurred, and one word stack write was performed.

This example was, by design, partially bus limited and indicates the types of EU and BIU interaction that can occur in this situation. Most application code sequences, however, use a higher proportion of more complex, longer-executing instructions and addressing modes, and therefore tend to be execution limited. In this case, less BIU-EU interaction is required, the queue more often is full, and more idle states occur on the bus.

The previous example sequence can be easily extended to incorporate wait states in the bus access cycles. In the case of a single wait state, each bus cycle would be lengthened to five clock cycles with a wait state (T_W) inserted between every T_3 and T_4 state of the bus cycle. As a first approximation, the instruction sequence exection time would appear to be lengthened by 10 clock cycles, one cycle for each useful read or write bus cycle that occurs. Actually, this approximation for the number of wait states inserted is incorrect since the queue can compensate for wait states by making use of previously idle bus time. For the example sequence, this compensation reduced the actual execution time by one wait state, and the sequence was completed in 64 clock cycles, one less than the approximated 65 clock cycles.

4.3 8089 I/O Processor

The Intel® 8089 I/O Processor (IOP) combines the functions of a DMA controller with the processing capabilities of a microprocessor. In addition to the normal DMA function of transferring data, the 8089 is capable of dynamically translating and comparing the data as it is

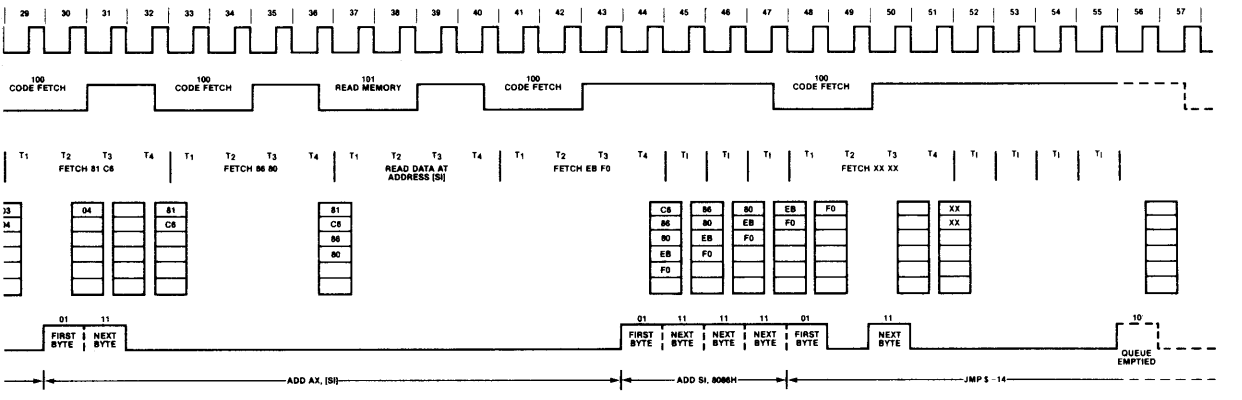


Figure 4-22. Sample Instruction Sequence Execution

transferred and of supporting a number of terminate conditions including byte count expired, data compare or miscompare and the occurrence of an external event. The 8089 contains two separate DMA channels, each with its own register set. Depending on the established priorities (both inherent and program determined), the two channels can alternate (interleave) their respective operations.

Designed expressly to relieve the 8086 or 8088 CPU of the overhead associated with I/O operations, the 8089, when configured in the remote mode, can perform a complete I/O task while the CPU is performing data processing tasks. The 8089, when it has completed its I/O task, can then interrupt the CPU.

Transfer flexibility is an integral part of the 8089's design. In addition to routine transfers between an I/O peripheral and memory, transfers can be performed between two I/O devices or between two areas of memory. Transfers between dissimilar bus widths are automatically handled by the 8089. When data is transferred from an 8-bit peripheral bus to a 16-bit memory bus, the 8089 reads two bytes from the peripheral, assembles the bytes into a 16-bit word and then writes the single word to the addressed memory location. Also, both 8- and 16-bit peripherals can reside on the same (16-bit) bus; byte transfers are performed with the 8-bit peripheral, and word transfers are performed with the 16-bit peripheral.

System Configuration

The 8089 can be implemented in one of two system configurations: a "local" mode in which the 8089 shares the system bus with an 8086 or 8088 CPU and a "remote" mode in which the 8089 has exclusive access to its own dedicated bus as well as access to the system bus. Note that in either the local or remote mode, the 8089 can address a full megabyte of system memory and 64k bytes of I/O space.

Local Mode

In the local mode, the 8089 acts as a slave to an 8086 or 8088 CPU that is operating in the maximum mode. In this configuration, the 8089 shares the system address latches, data transceivers and bus controller with the CPU as shown in figure 4-23.

Since the IOP and CPU share the system bus, either the IOP or the CPU will have access to the bus at any one time. When one processor is using the bus, the other processor floats its address/data and control lines. Bus access between the IOP and CPU is determined through the request/grant function. Recalling the CPU's request/grant sequence, the IOP requests the bus from the CPU, the CPU grants the bus to the IOP, and the IOP relinquishes the bus to the CPU when its operation is complete. Remember that the CPU cannot request the bus from the IOP (the CPU is only capable of granting the bus and

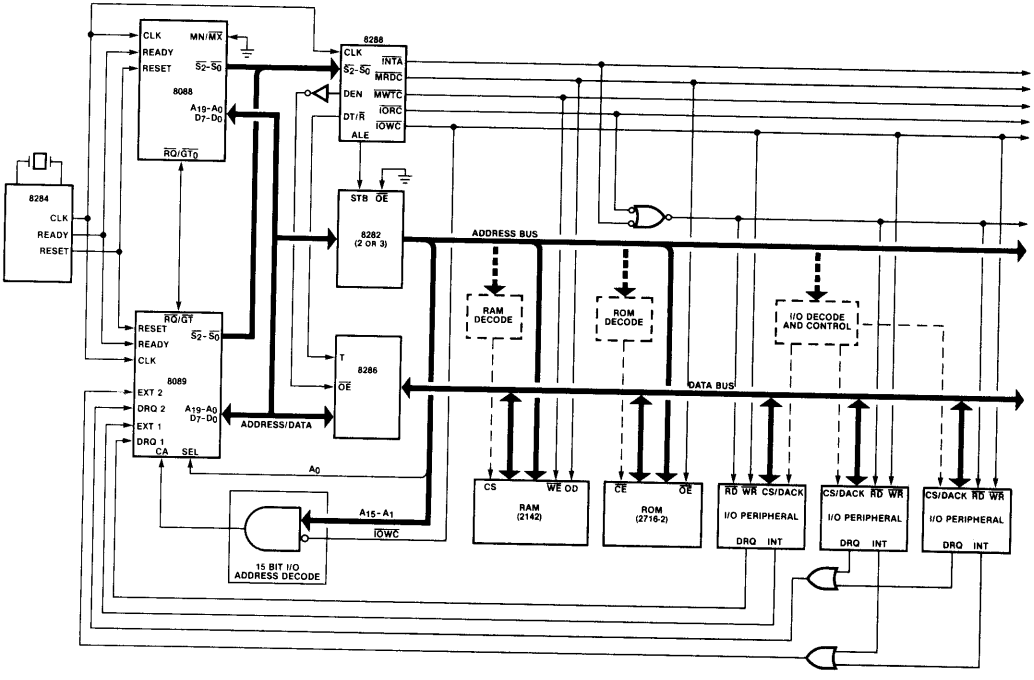


Figure 4-23. Typical 8088/8089 Local Mode Configuration

must wait for the IOP to release the bus). Also, since the request/grant pulse exchange must be synchronized, both the CPU and IOP must be referenced to the same clock signal.

The 8089 IOP, when used in the local mode, can be added to an 8086 or 8088 maximum mode configuration with little affect on component count (channel attention decoding logic as required) and offers the benefits of intelligent DMA (scan/match, translate, variable termination conditions), modular programming in a full megabyte of memory address space and a set of optimized I/O instructions that are unavailable to the 8086 and 8088 CPUs. The major disadvantage to the local configuration is that since the system bus is shared, bus contention always exists between the CPU and IOP. The use of the bus load limit field in the channel control word can help reduce IOP bus access during task block program execution (bus load limiting has no affect on DMA transfers) although, for I/O intensive systems, the remote mode should be considered.

Remote Mode

The 8089, when used in the remote mode, provides a multiprocessor system with true parallel processing. In this mode, the 8089 has a separate (local) bus and memory for I/O peripheral communications, and the system bus is completely isolated from the I/O peripheral(s). Accordingly, I/O transfers between an I/O peripheral and the IOP's local memory can occur simultaneously with CPU operations on the system bus.

As shown in figure 4-24, to interface the 8089 to the system bus, data transceivers and address latches are used to separate the IOP's local bus from the system bus, an 8288 Bus Controller is used to generate the bus control signals for both the local and system buses as well as to govern the operation of the transceivers/latches, and an 8289 Bus Arbiter is used to control access to the system bus (each processor in the system would have an associated 8289 Bus Arbiter). To interface the 8089 to its local bus, another set of address

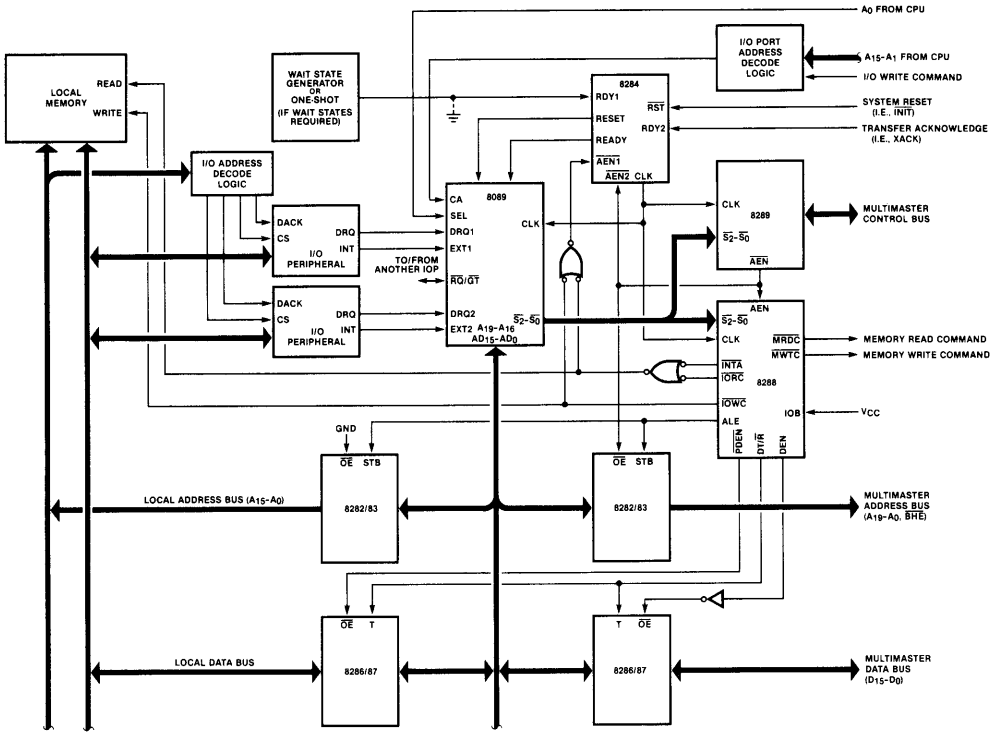


Figure 4-24. Typical 8089 Remote Mode Configuration

latches is required (unless MCS-85™ multiplexed address components are exclusively interfaced) and, depending on the bus loading demands, one (8-bit bus) or two (16-bit bus) data transceivers would be used.

In the remote mode, the IOP's local bus is treated as I/O space (up to 64k bytes), and the system bus is treated as memory space (1 megabyte). The 8288 Bus Controller's I/O command outputs control the local (I/O) bus, and its memory command outputs control the system (memory) bus. The 8289 Bus Arbiter, which is operated in its IOB (I/O peripheral bus) mode, also decodes the IOP's $\overline{S2}$ through $\overline{S0}$ status outputs. In this mode, the 8289 will not request the multimaster system bus when the IOP indicates an operation on its local bus. If the IOP's bus arbiter currently has access to the system bus, the CPU's arbiter (or any other arbiter in the system) can acquire use of the system bus at this time (a bus arbiter maintains bus access until another arbiter requests the bus).

Bus Operation

The 8089 utilizes the same bus structure as an 8086 or 8088 CPU that is configured in the maximum mode and performs a bus cycle only on demand (e.g., to fetch an instruction during task block execution or to perform a data transfer). The bus cycle itself is identical to an 8086 or 8088 CPU's bus cycle in that all cycles consist of four T-states and use the same time-multiplexing technique of the addressdata lines. As shown in the following timing diagrams, the address (and ALE signal) is output during state T_1 for either a read or write cycle. Depending on the type of cycle indicated, the address/data lines are floated during state T_2 for a read cycle (figure 4-25) or data is output on these lines during a write cycle (figure 4-26). During state T_3, write data is maintained or read data is sampled, and the busy cycle is concluded in state T_4.

Since the 8089 is capable of transferring data to or from both 8-bit and 16-bit buses, when an 8-bit physical bus is specified (bus width is specified

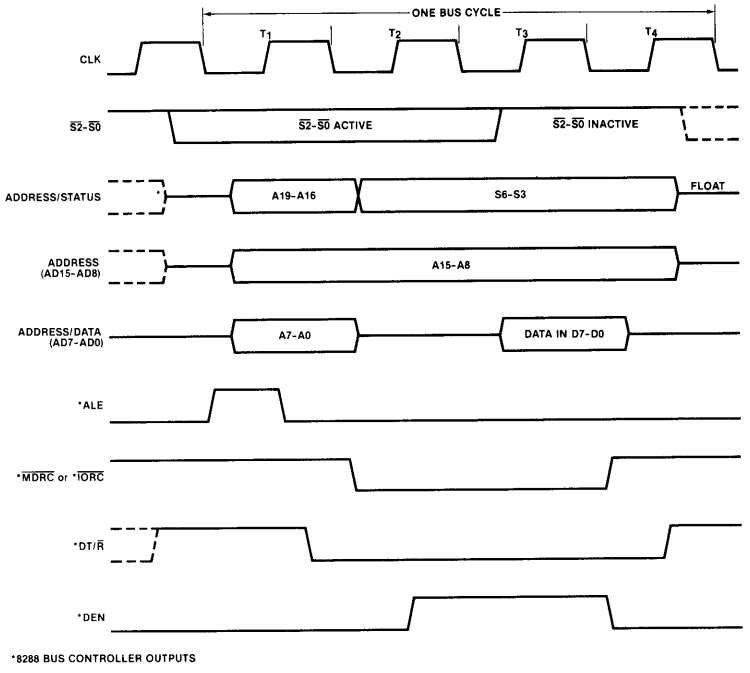


Figure 4-25. Read Bus Cycle (8-Bit Bus)

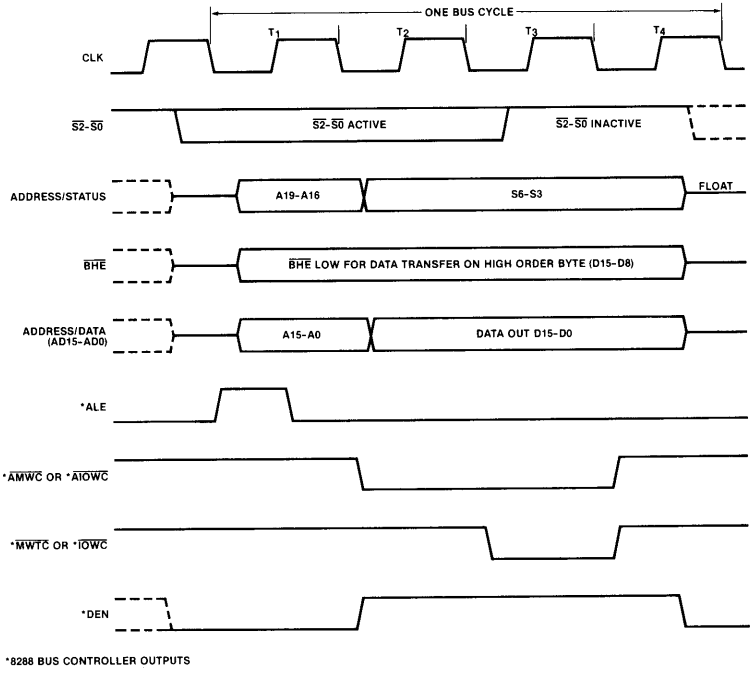


Figure 4-26. Write Bus Cycle (16-Bit Bus)

during the initialization sequence), the address present on the AD15 through AD8 address/data lines is maintained for the entire bus cycle as shown in figure 4-25 and, unless added drive capability is required, the associated address latch can be eliminated. An 8-bit data bus is compatible with the 8088 CPU and with the MCS-85™ multiplexed address peripherals (8155, 8185, etc.).

The 8089 operates identically to the 8086 CPU with respect to the use of the low- and high-order halves of the data bus. Table 4-14 defines the data bus use for the various combinations of bus width and address boundary.

The $\overline{S2}$ through $\overline{S0}$ status lines define the bus cycle to be performed. These lines are used by an 8288 Bus Controller to generate all memory and I/O command and control signals, and are decoded according to table 4-15.

Table 4-14. Data Bus Usage

Logical Bus Width[1]	Address Boundary	Physical Bus Width[2]		
		8	16	
			Byte Transfer	Word Transfer
8	Even	AD7-AD0 = DATA ($\overline{BHE}$ not used)	AD7-AD0 = DATA ($\overline{BHE}$ high)	N/A
	Odd	AD7-AD0 = DATA ($\overline{BHE}$ not used)	AD15-AD8 = DATA ($\overline{BHE}$ low)	N/A
16	Even	Illegal	AD7-AD0 = DATA ($\overline{BHE}$ high)	AD15-AD0 = DATA ($\overline{BHE}$ low)
	Odd	Illegal	AD15-AD8 = DATA ($\overline{BHE}$ low)	N/A[3]

Notes:

1. Logical bus width is specified by the WID instruction prior to the DMA transfer.
2. Physical bus width is specified when the 8089 is initialized.
3. A word transfer to or from an odd boundary is performed as two byte transfers. The first byte transferred is the low-order byte on the high-order data bus (AD15-AD8), and the second byte is the high-order byte on the low-order data bus (AD7-AD0). The 8089 automatically assembles the two bytes in their proper order.

Table 4-15. Bus Cycle Decoding

Status Output			Bus Cycle Indicated	Bus Controller Command Output
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$		
0	0	0	Instruction fetch from I/O space	$\overline{INTA}$
0	0	1	Data read from I/O space	$\overline{IORC}$
0	1	0	Data write to I/O space	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Not used	None
1	0	0	Instruction fetch from system memory	$\overline{MRDC}$
1	0	1	Data read from system memory	$\overline{MRDC}$
1	1	0	Data write to system memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

Note that the 8089 indicates an instruction fetch from I/O space as a status of zero ($\overline{S2}$, $\overline{S1}$ and $\overline{S0}$ equal 0). Since the 8288 Bus Controller decodes an input status value of zero as an interrupt acknowledge bus cycle, the bus controller's $\overline{INTA}$ output must be OR'ed with its $\overline{IORC}$ output to permit fetching of task block instructions from local 8089 memory (remote configuration) or system I/O space (local and remote configurations).

The $\overline{S2}$ through $\overline{S0}$ status lines become active in state T_4 if a subsequent bus cycle is to be performed. These lines are set to the passive state (all "ones") in the state immediately prior to state T_4 of the current bus cycle (state T_3 or T_W) and are floated when the 8089 does not have access to the bus.

The S6 through S3 status lines are multiplexed with the high-order address bits (A19-A16) and, accordingly, become valid in state T_2 of the bus cycle. The S4 and S3 status lines reflect the type of bus cycle being performed on the corresponding channel as indicated in table 4-16.

Table 4-16. Type of Cycle Decoding

Status Output		Type of Cycle
S4	S3	
0	0	DMA on Channel 1
0	1	DMA on Channel 2
1	0	Non-DMA on Channel 1
1	1	Non-DMA on Channel 2

The S6 and S5 status lines are always "1" on the 8089. Since these lines are not both "1" on the other processors in the 8086 family (S6 is always "0" on the 8086 and 8088 CPUs), these status lines can be used as a "signature" in a multiprocessor environment to identify the type of processor performing the bus cycle.

The 8089 includes the same provision as do the 8086 and 8088 CPUs for the insertion of wait states (T_W) in a bus cycle when the associated memory or I/O device cannot respond within the alloted time interval or when, in the remote mode, the 8089 must wait for access to the system bus. An 8284 Clock Generator/Driver is used to control the insertion of wait states which, when required, are inserted between states T_3 and T_4. The actual insertion of wait states is accomplished by deactivating one of the 8284's RDY inputs (RDY1 or RDY2). Either of these inputs, when enabled by its corresponding AEN1 or AEN2 input, can be deactivated directly by the memory or I/O device when it must extend the 8089's bus cycle (when the addressed device is not ready to present or accept data). The 8284's READY output, which is synchronized to the CLK signal, is directly connected to the 8089's READY input. As shown in figure 4-27, when the addressed device requires one or more wait states to be inserted into a bus cycle, it deactivates the 8284's RDY input prior to the end of state T_2. The READY output from the 8284 is subsequently deactivated at the end of state T_2 which causes the 8089 to insert wait states following state T_3. To exit the wait state, the device activates the 8284's RDY input which causes the READY input to the 8089 to go active on the next clock cycle and allows the 8089 to enter state T_4.

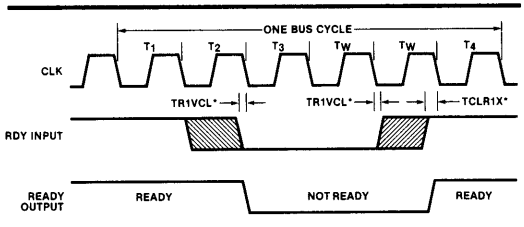


Figure 4-27. Wait State Timing

Periods of inactivity can occur between bus cycles. These inactive periods are referred to as idle states (T_I) and, as with the 8086 and 8088 CPUs, can result from the execution of a "long" instruction or the loss of the bus to another processor during task block instruction execution. Additionally, the 8089 can experience idle states when it is in the DMA mode and it is waiting for a DMA request from the addressed I/O device or when the bus load limit (BLL) function is enabled for a channel performing task block instruction execution and the other channel is idle.

Initialization

Initialization of the IOP is generally the responsibility of the host processor which, as stated in Chapter 3, prepares the communications data structure in shared memory. Initialization of the IOP itself begins with the activation of its RESET input. This input (originating typically from an

8284 Clock Generator/Driver) must be held active for at least five clock cycles to allow the 8089's internal reset sequence to be completed. Note that like the 8086 and 8088 CPUs, the RESET input must be held active for at least 50 microseconds when power is first applied. Following the reset interval, the host processor signals the IOP to begin its initialization sequence by activating the 8089's CA (Channel Attention) input. The 8089 will not recognize a pulse at its CA input until one clock cycle after the RESET input returns to an inactive level. Note that the minimum width for a CA pulse is one clock cycle and that this pulse may go active prior to RESET returning to an inactive level provided that the negative-going, trailing-edge of the CA pulse does not occur prior to one clock cycle after RESET goes inactive. Figure 4-28 illustrates the timing for this portion of the initialization sequence.

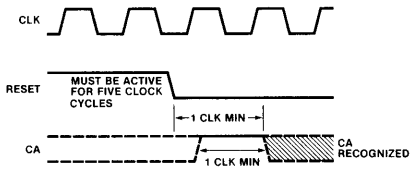


Figure 4-28. RESET-CA Initialization Timing

Coincident with the trailing edge of the first CA pulse following reset, the 8089 samples its SEL (Select) input from the host processor to determine master/slave status for its request/grant circuity. If the SEL input is low, the 8089 is designated a "master," and if the SEL input is high, the 8089 is designated a "slave." As a master, the 8089 assumes that it has the bus initially, and it will subsequently grant the bus to a requesting slave when the bus becomes available (i.e., the 8089 will respond to a "request" pulse on its $\overline{RQ}/\overline{GT}$ line with a "grant" pulse). A single 8089 in the remote configuration (or one of two 8089s in a remote configuration) would be designated a master. As a slave, the 8089 can only request the bus from a master processor (i.e., the 8089 initiates the request/grant sequence by outputting a "request" pulse on its $\overline{RQ}/\overline{GT}$ line). An 8089 that shares a bus with an 8086 or 8088 (or one of two 8089s in a remote configuration) would be designated a slave. Note that since the 8086 and 8088 CPUs can grant the bus only in response to a request, whenever an 8086 or 8088 and an 8089 share a common bus, the 8089 *must* be designated the slave. Also, when the $\overline{RQ}/\overline{GT}$ line is not used (i.e., a single 8089 in the remote configuration), the 8089 *must* be designated a master.

In addition to determining master/slave status, the CA pulse also causes the 8089 to begin execution of its internal ROM initialization sequence. Note that since the 8089 must have access to the *system* bus in order to perform this sequence, the 8089 immediately initiates a request/grant sequence (if designated a slave) and, if required, then requests the bus through the 8289 Arbiter. (If designated a master, the 8089 requests the bus through the 8289 Arbiter.) In the execution of the initialization sequence, the 8089 first fetches the SYSBUS byte from location FFFF6H. The W bit (bit 0) of this byte specifies the *physical* bus width of the *system* bus. Depending on the bus width specified, the 8089 then fetches the address of the system configuration block (SCB) contained in locations FFFF8H through FFFFBH in either two bus cycles (16-bit bus, W bit equal 1) or four bus cycles (8-bit bus, W bit equal 0). The SCB offset and segment address values fetched are combined into a 20-bit physical address that is stored in an internal register. Using this address, the 8089 next fetches the system operation command (SOC) byte. As explained in Chapter 3, this byte specifies both the request/grant operational mode (R bit) and the *physical* width of the I/O bus (I bit). After reading the SOC byte, the 8089 fetches the channel control block (CB) offset and segment address values. These values are combined into a 20-bit physical address and are stored in another internal register. To inform the host CPU that it has completed the initialization sequence, the 8089 clears the Channel 1 Busy flag in the channel control block by writing an all "zeroes" byte to CB + 1.

After the IOP has been initialized, the system configuration block may be altered in order to initialize another IOP. Once an IOP has been initialized, its channel control block in system memory cannot be moved since the CB address, which is internally stored by the IOP during the initialization sequence, is automatically accessed on every subsequent CA pulse.

As previously stated, the generation of the CA and SEL inputs to the IOP are the responsibility of the host CPU. Typically, these signals result from the CPU's execution of an I/O write instruction to one of two adjacent I/O ports (I/O port addresses that only differ by A0). Figure 4-29 illustrates a simple decoding circuit that could be used to generate the CA and SEL signals. Note that by qualifying the CA output with $\overline{\text{IOWC}}$, the SEL output, since it is latched for the entire I/O bus cycle, is guaranteed to be stable on the trailing edge of the CA pulse.

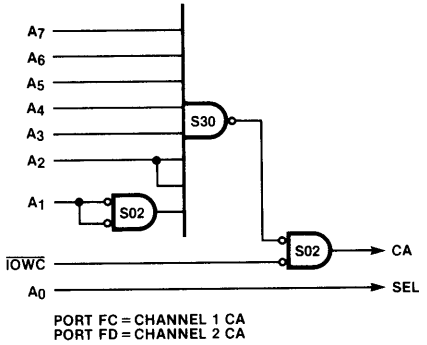


Figure 4-29. Channel Attention Decoding Circuit

I/O Dispatching

During normal operation, the I/O supervisory program running in the host CPU will receive a request to perform a specific I/O operation on one of the 8089's channels. In response to this request, the supervisory program will typically perform the following sequence of operations:

- Check the availability of the specified channel by examining the channel's busy flag in the Channel Control Block. If it is possible for another processor to access the channel, a semaphore operation (implemented by a locked XCHG instruction) is used to check channel availability.
- Load the variable parameters required for the intended operation into the channel's parameter block.
- Load the channel command word (CCW) into the channel control block.
- Establish the necessary linkages by writing the starting address of the channel program (task block) in the first four bytes of the parameter block and writing the address of the parameter block in the channel control block.
- Issue a channel attention (CA) to the specified channel.

In response to the CA, the 8089 interrupts any current activity at its first opportunity (see "Concurrent Channel Operation" in section 3.2) and begins execution of an internal instruction sequence that fetches and decodes the channel command word (CCW) and then performs the operation indicated (i.e., start, halt or continue channel program execution).

If the CCW specifies start channel program (start task block execution), the address of the parameter block is fetched from the channel control block, the address of the first channel program instruction (contained in the first four bytes of the parameter block) is fetched and then loaded into the TP (task pointer) register and, finally, task block execution is initiated from either system or I/O space. Task block execution continues, subject to the activity on the other channel as described in "Concurrent Channel Operation," until a XFER instruction is executed. Following execution of this instruction, the next sequential channel program instruction is executed before the channel enters the DMA transfer mode.

If the CCW specifies halt channel, the current operation on the specified channel is halted. If the channel is performing task block execution (either chained or not chained), channel operation is stopped at an instruction boundary, and if the channel is performing a DMA transfer, channel operation is stopped at a DMA transfer cycle boundary. Note that a channel will not stop a locked DMA transfer until the operation is completed. There are two unique halt channel commands. One command simply halts the channel and clears the busy flag in the channel control block. This command is used when the halted operation is to be discarded. The other command halts the channel, saves the task pointer and program status word (PSW) byte, and clears the busy flag. This command is used when the halted operation is to be resumed. Note that this halt command will not affect the integrity of resumed task block execution or a memory-to-memory DMA transfer, but could affect the integrity of a synchronized DMA transfer (a DMA request occuring while the channel is halted could be missed).

If the CCW specifies continue channel, an operation that has been previously halted is resumed (and the busy flag is set). Since this command restores the task pointer and PSW, it should be used only if the task pointer and PSW have been saved by a previous halt command.

Table 4-17 outlines the various CCW command execution times. Note that the times listed in the table for the halt commands do *not* include the time required to complete any current channel activity when the channel attention is received (completion of the current DMA transfer cycle or task block instruction).

DMA Transfers

The number of bytes transferred during a single DMA cycle is determined by both the source and destination logical bus widths as well as by the address boundary (odd or even address). The 8089 performs DMA transfers between dissimilar bus widths by assembling bytes or disassembling words in its internal assembly register file. As explained in Chapter 3, the DMA source and destination bus widths are defined by the execution of a WID instruction during task block (channel command) execution. Note that the bus widths specified remain in force until changed by a subsequent WID instruction. Table 4-18 defines the various byte (B) and word (W) source/destination transfer combinations based on address boundary and bus width specified.

The 8089 additionally optimizes bus accesses during transfers between dissimilar bus widths whenever possible. When either the source or destination is a 16-bit memory bus (auto-incrementing) that is initially aligned on an odd

Table 4-17. CCW Command Execution Times

CCW Command	Minimum Time*	Maximum Time**
CA NOP	48 + 2n clocks	48 + 2n clocks
CA Halt (no save)	48 + 2n clocks	48 + 2n clocks
CA Halt (with save)	94 + 5n clocks	100 + 6n clocks
CA Start (memory)	108 + 6n clocks	124 + 10n clocks
CA Start (I/O)	96 + 5n clocks	108 + 8n clocks
CA Continue	95 + 5n clocks	103 + 6n clocks

Notes:
n is the number of wait states per bus cycle.

* Minimum time occurs when both the channel control block and parameter block addresses are aligned on an even address boundary and a 16-bit bus is used.

** Maximum time occurs when both the channel control block and parameter block addresses are aligned on an odd address boundary on a 16-bit bus or when an 8-bit bus is used.

Table 4-18. DMA Assembly Register Operation

Address Boundary (Source → Destination)	Logical Bus Width (Source → Destination)			
	8 → 8	8 → 16	16 → 8	16 → 16
Even → Even	B → B	B/B → W	W → B/B	W → W
Even → Odd	B → B	B → B	W → B/B	W → B/B
Odd → Even	B → B	B/B → W	B → B	B/B → W
Odd → Odd	B → B	B → B	B → B	B → B

address boundary (causing the first transfer cycle to be byte-to-byte), following the first transfer cycle, the memory address will be aligned on an even address boundary, and word transfers will subsequently occur. For example, when performing a memory-to-port transfer from a 16-bit bus to an 8-bit bus with the source beginning on an odd address boundary, the first transfer cycle will be byte-to-byte (B → B) as indicated in table 4-18, but subsequent transfers will be word-to-byte/byte (W → B/B).

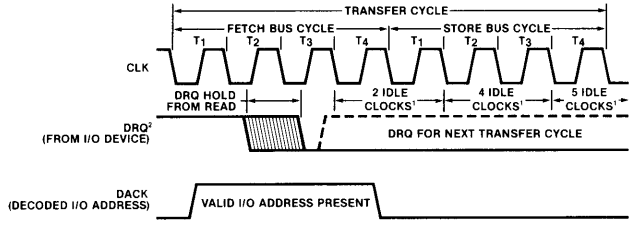
All DMA transfer cycles consist of at least two bus cycles; one bus cycle to fetch (read) the data form the source into the IOP, and one bus cycle to store (write) the data previously fetched from the IOP into the destination. Note that in all transfers, the data passes through the IOP to allow mask/compare and translate operations to be optionally performed during the transfer as well as to allow the data to be assembled or disassembled.

The IOP performs DMA transfers in one of three modes: unsynchronized, source synchronized or destination synchronized (the transfer mode is specified in the channel control register). The unsynchronized mode is used when both the source and destination devices do not provide a data request (DRQ) signal to the IOP as in the case of a memory-to-memory transfer. In the synchronized transfer modes, the source (source synchronized) or destination (destination synchronized) device initiates the transfer cycle by activating the IOP's DRQ1 (channel 1) or DRQ2 (channel 2) input.

The DRQ input is asynchronous and usually originates from an I/O device controller rather than from a memory circuit. This input is latched on the positive transition of the clock (CLK) signal and therefore must remain active for more than one clock period (more than 200 nanoseconds when using a 5 MHz clock) in order to guarantee that it is recognized.

During state T_1 of the associated fetch bus cycle (source synchronized) or store bus cycle (destination synchronized), the IOP outputs the address of the I/O device (the port address). This address must be decoded (by external circuitry) to generate the DMA acknowledge (DACK) signal to the I/O controller as the response to the controller's DMA request. An I/O controller will typically use DACK as a conditional input for the removal of DRQ. (After receipt of the DACK signal, most Intel peripheral controllers deactivate DRQ following receipt of the corresponding read or write signal.) Figures 4-30 and 4-31 illustrate the DRQ/DACK timing for both source synchronized (i.e., port-to-memory) and destination synchronized (i.e., memory-to-port) transfers.

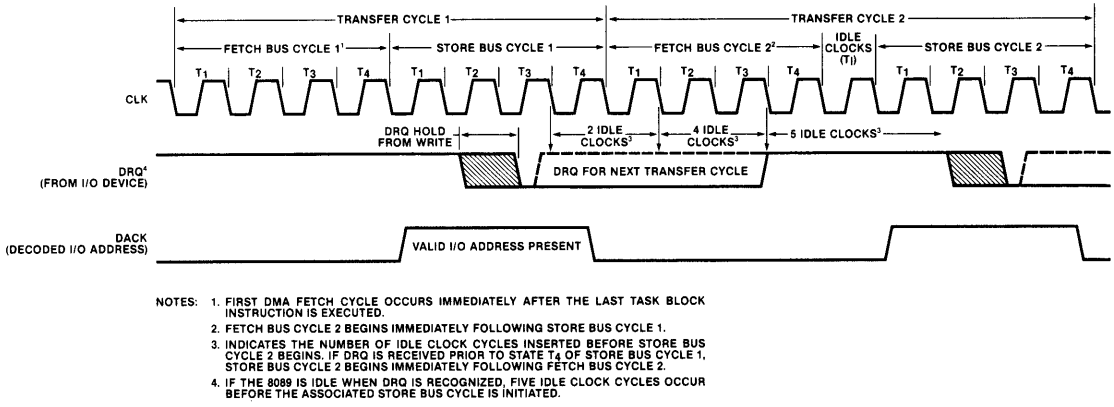
Table 4-19 defines the DMA transfer cycles in terms of the number of bus and clock cycles required. Note that the number of clocks required to complete a transfer cycle does not take into account the effects of possible concurrent operations on the other channel or wait states within any of the bus cycles.



NOTES:

1. INDICATES THE NUMBER OF IDLE CLOCK CYCLES INSERTED BEFORE THE NEXT TRANSFER CYCLE BEGINS. IF DRQ IS RECEIVED PRIOR TO STATE T_4 OF THE CURRENT FETCH CYCLE, THE NEXT FETCH CYCLE BEGINS IMMEDIATELY FOLLOWING THE CURRENT STORE CYCLE.
2. IF THE 8089 IS IDLE WHEN DRQ IS RECOGNIZED, FIVE IDLE CLOCK CYCLES OCCUR BEFORE THE ASSOCIATED TRANSFER CYCLE IS INITIATED.

Figure 4-30. Source Synchronized Transfer Cycle



NOTES:
1. FIRST DMA FETCH CYCLE OCCURS IMMEDIATELY AFTER THE LAST TASK BLOCK INSTRUCTION IS EXECUTED.
2. FETCH BUS CYCLE 2 BEGINS IMMEDIATELY FOLLOWING STORE BUS CYCLE 1.
3. INDICATES THE NUMBER OF IDLE CLOCK CYCLES INSERTED BEFORE STORE BUS CYCLE 2 BEGINS. IF DRQ IS RECEIVED PRIOR TO STATE T_4 OF STORE BUS CYCLE 1, STORE BUS CYCLE 2 BEGINS IMMEDIATELY FOLLOWING FETCH BUS CYCLE 2.
4. IF THE 8089 IS IDLE WHEN DRQ IS RECOGNIZED, FIVE IDLE CLOCK CYCLES OCCUR BEFORE THE ASSOCIATED STORE BUS CYCLE IS INITIATED.

Figure 4-31. Destination Synchronized Transfer Cycle

Table 4-19. DMA Transfer Cycles

Logical Bus Width		Transfer Mode					
		Unsynchronized		Source Synchronized		Destination Synchronized	
Source	Destination	Bus Cycles Required	Total[1] Clocks	Bus Cycles Required	Total[1] Clocks	Bus Cycles Required	Total[1] Clocks
8	8	2 (1 fetch, 1 store)	8[2]	2 (1 fetch, 1 store)	8[2]	2 (1 fetch, 1 store)	8[2]
8	16[3]	3 (2 fetch, 1 store)	12	3 (2 fetch, 1 store)	16[4]	3 (2 fetch, 1 store)	12
16[3]	8	3 (1 fetch, 2 store)	12	3 (1 fetch, 2 store)	12	3 (1 fetch, 2 store)	16[4]
16[3]	16[3]	2 (1 fetch, 1 store)	8	2 (1 fetch, 1 store)	8	2 (1 fetch, 1 store)	8

Notes:
1. The "Total Clocks Required" does not include wait states. One clock cycle per wait state must be added to each fetch and/or store bus cycle in which a wait state is inserted. When performing a memory-to-memory transfer, three additional clocks must be added to the total clocks required (the first fetch cycle of any memory-to-memory transfer requires seven clock cycles).
2. When performing a translate operation, one additional 7-clock bus cycle must be added to the values specified in the table.
3. Word transfers in the table assume an even address word boundary. Word transfers to or from odd address boundaries are performed as indicated in table 4-18 and are subject to the bus cycle/clock requirements for byte-to-byte transfers.
4. Transfer cycles that include two synchronized bus cycles (i.e., synchronous transfers between dissimilar logical bus widths) insert four idle clock cycles between the two synchronized bus cycles to allow additional time for the synchronzing device to remove its initial DMA request.

DACK latency is defined as the time required for the 8089 to acknowledge, by outputting the device's corresponding port address, a DMA request at its DRQ input. This response latency is dependent on a number of factors including the transfer cycle being performed, activity on the other channel, memory address boundaries, wait states present in either bus cycle and bus arbitration times.

Generally, when the other channel is idle, the maximum DACK latency is five clock cycles (1 microsecond at 5 MHz), excluding wait states and bus arbitration times. An exception occurs when performing a word transfer to or from an odd memory address boundary. This operation, since two store (source synchronized) or two fetch (destination synchronized) bus cycles are required to access memory, has a maximum possible latency of nine clock cycles. When the other channel is performing DMA transfers of equal priority ("P" bits equal), interleaving occurs at bus cycle boundaries, and the maximum latency is either nine clock cycles when the other channel is performing a normal 4-clock fetch or store bus cycle or twelve clock cycles when the other channel is performing the first fetch cycle of a memory-to-memory transfer. If the other channel is performing "chained" task block instruction execution of equal priority, maximum latency can be as high as 12 clock cycles (channel command instruction execution is interrupted at machine cycle boundaries which range from two to eight clock cycles).

DMA Termination

As stated in Chapter 3, a channel can exit the DMA transfer mode (and return to task block execution) on any of the following terminate conditions:

- Single cycle transfer
- Byte count expired
- Mask/compare match or mismatch
- External event

The terminate conditions are specified by individual fields in the channel control register. More than one terminate condition can be specified for a transfer (e.g., a transfer can be terminated when a specific byte count is reached or on the occurrence of an external event). When more than one terminate condition is possible, displacements (which are added to the task pointer register value) are specified to cause task block execution to resume at a unique entry point for each condition. Three reentry points are available: TP, TP + 4 and TP + 8. The time interval between the occurrence of a terminate condition and the resumption of task block execution is 12 clock cycles for reentry point TP and 15 clock cycles for reentry points TP + 4 and TP + 8.

Peripheral Interfacing

When interfacing a peripheral to an 8-bit physical data bus, the 8089 uses only the lower half of the address/data lines (AD7-AD0) as the bidirectional data bus, and the upper half of the address/data lines (AD15-AD8) maintain address information for the entire bus cycle. Consequently, with this bus configuration, only one octal latch (e.g., an Intel® 8282/83 Octal Latch) is required since only the lower half of the address/data lines is time-multiplexed (unless the address bus requires the increased current drive capability and capacitive load immunity provided by the latch).

When interfacing a peripheral to a 16-bit data bus, both the lower and upper halves of the address/data lines are time-multipelxed, and two octal latches are required. Note that unlike the 8086 and 8088 CPUs, the 8089 does not time-multiplex $\overline{\text{BHE}}$ (this signal is valid for the entire bus cycle). Both 8- and 16-bit peripherals can be interfaced to a 16-bit bus. An 8-bit peripheral can be connected to either the upper or lower half of the bus. An 8-bit peripheral on the lower half of the bus must use an even source/destination address, and an 8-bit peripheral on the upper half of the bus must use an odd source/destination address. To take advantage of word transfers, a 16-bit peripheral must use an even source/destination address.

To prepare a peripheral device for a DMA transfer, command and parameter data is written to the device's command/status port. This is usually accomplished using pointer register GC. Recalling that the 8089 executes one additional task block instruction following execution of the XFER instruction (the XFER instruction causes the 8089 to enter the DMA mode), this additional instruction is used to access the command port of an I/O device that immediately begins DMA

operation on receipt of the last command (the 8271 Floppy Disk Controller begins its DMA transfer on receipt of the last command parameter). Since a translate DMA operation requires the use of all three pointer registers (GA and GB specify the source and destination addresses; GC specifies the base address of the translation table), when it is necessary to use the last task block instruction to start the device, command port access can be accomplished relative to one of the pointer registers or relative to the PP register. If the device's data port address (GA or GB) is below the device's command port address, either an offset or an indexed reference can be used to access the command port.

A peripheral's (or peripheral controller's) DMA communication protocol with the 8089 is as follows:

- The peripheral (when source or destination synchronized) initiates a DMA transfer cycle by activating the 8089's DRQ (DMA request) input.
- The 8089 acknowledges the request by placing the peripheral's assigned data port address on the bus during state T_1 of the corresponding fetch (source synchronized) or store (destination synchronized) bus cycle. The peripheral is responsible for decoding this address as the DMA acknowledge (DACK) to its request.
- The data is transferred between the peripheral and the 8089 during the T_2 through T_4 state interval of the bus cycle. The peripheral must remove its DMA request during this interval.
- The peripheral, when ready, requests another DMA transfer cycle by again activating the DRQ input, and the above sequence is repeated.
- The peripheral can, as an option, end the DMA transfer by activating the 8089's EXT (external terminate) input.

The 8089 can support mulitple peripheral devices on a single channel provided that only one device is in the active transfer mode at any one time. To interface multiple devices, the DMA request (DRQ) lines are OR'ed together as are the external terminate (EXT) lines. Unique port addresses are, however, assigned to each device so that an individual DMA acknowledge (DACK) is returned to only the active device. DACK decoding can be accomplished with an Intel ® 8205 Binary Decoder or a ROM circuit. Note that the 8089 can only determine which device has requested service or terminated by the context of the task block program.

Most peripheral devices interfaced to the 8089 will use the decoded DMA acknowledge signal (DACK) as the "chip select" input. Peripheral devices that do not follow this convention must use DACK as a conditional input of chip select.

While most interrupts associated with the 8089 will be DMA requests or external terminates, non-DMA related interrupts can additionally be supported.

One technique that would be used when an 8089 is the local configuration (or when an 8086 or 8088 and an 8089 are locally connected as a remote module) is to allow the CPU to accept the interrupt and then direct the 8089 to the interrupt service routine. Another technique is to allow the 8089 to "poll" the device to determine when an interrupt has occurred (most peripheral controllers have an interrupt pending bit in a status word). The 8089's bit testing instructions are ideally suited for polling.

When the 8089 is in a remote configuration, non-DMA related interrupts can be supported with the addition of an Intel® 8259A Programmable Interrupt Controller. Systems that require this type of interrupt structure would dedicate one of the 8089's channels to interrupt servicing. In implementing this structure, the interrupt output from the 8259A is directly connected to the channel's external terminate (EXT) input, and the channel's DMA request (DRQ) input is not used. A task block program is initially executed to perform a source-synchronized DMA transfer (with an external terminate) on the "interrupt" channel to "arm" the interrupt mechanism. Since the DRQ input is not used, when the channel enters the DMA transfer mode, the channel idles while waiting for the first DMA request (which never occurs). The other channel, since the interrupt channel is idle, operates at maximum throughput. When an interrupt occurs, the "pseudo" DMA transfer is immediately terminated, and task block instruction execution is resumed. The task block program would write a "poll" command to the 8259A's command port and then read the

8259A's data port to acknowledge the interrupt and to determine the device responsible for the interrupt (the device is identified by a 3-bit binary number in the associated data byte). The device number read would be used by the task block program as a vector into a jump table for the device's interrupt service routine. Pertinent interrupt data could be written into the associated parameter block for subsequent examination by the host processor.

The interrupt mechanism previously described, since it uses the 8089's external terminate function, provides an extremely fast interrupt response time.

Note that when using dynamic RAM memory with the 8089, an Intel® 8202 Dynamic RAM Controller can be used to simplify the interface and to perform the RAM refresh cycle. When maximum transfer rates are required, the RAM refresh cycle can be externally initiated by the 8089. By connecting the decoded DACK (DMA acknowledge) signal to the 8202's REFRQ (refresh request) input, the refresh cycle will occur coincident with the I/O device bus cycle and therefore will not impose wait states in the memory bus cycle.

Instruction Encoding

Most 8089 programming will be performed at the assembly language level using ASM-89, the 8089 assembler. During program debugging, however, it may be necessary to work directly with machine instructions when monitoring the bus, reading unformatted memory dumps, etc. This section contains both a table to encode any ASM-89 instruction into its corresponding machine instruction (table 4-24) and a table to "disassemble" any machine instruction back into its associated assembly language equivalent (table 4-26).

Figure 4-32 shows the format of a typical 8089 machine instruction. Except for the LPDI and memory-to-memory forms of the MOV and MOVB instructions that are six bytes long, all 8089 machine instructions consist of from two to five bytes. The first two bytes are always present and are generally formatted as shown in figure 4-32 (table 4-24 contains the exact encoding of every instuction).

Bits 5 through 7 of the first byte of an instruction comprise the R/B/P field. This field identifies a register, bit select or pointer register operand as outlined in table 4-20.

Table 4-20. R/B/P Field Encoding

Code	Register	Bit	Pointer
000	GA	0	GA
001	GB	1	GB
010	GC	2	GC
011	BC	3	N/A
100	TP	4	TP
101	IX	5	N/A
110	CC	6	N/A
111	MC	7	N/A

The WB field (bits 3 and 4 of the first byte) indicates how many displacement/data bytes are present in the instruction as outlined in table 4-21. The displacement bytes are used in program transfers; one byte is present for short transfers, while long transfers contain a two-byte (word) displacement. As mentioned in Chapter 3, the

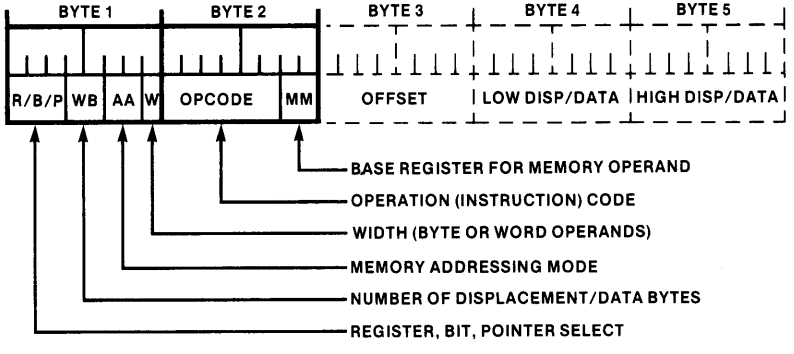


Figure 4-32. Typical 8089 Machine Instruction Format

displacement is stored in two's complement notation with the high-order bit indicating the sign. Data bytes contain the value of an immediate constant operand. A byte immediate instruction (e.g., MOVBI) will have one data byte, and a word immediate instruction (e.g., ADDI) will have two bytes (a word) of immediate data. An instruction may contain either displacement or data bytes, but not both (the TSL instruction is an exception and contains one byte of displacement and one byte of data). If an offset byte is present, the displacement/data byte(s) always follow the offset byte.

Table 4-21. WB Field Encoding

Code	Interpretation
00	No displacement/data bytes
01	One displacement/data byte
10	Two displacement/data bytes
11	TSL instruction only

The AA field specifies the addressing mode that the processor is to use in order to construct the effective address of a memory operand. Four addressing modes are available as outlined in table 4-22. (Address modes are described in detail in section 3.8.)

Table 4-22. AA Field Encoding

Code	Interpretation
00	Base register only
01	Base register plus offset
10	Base register plus IX
11	Base register plus IX, auto-increment

Bit 0 of the first instruction byte indicates whether the instruction operates on a byte (W=0) or a word (W=1).

Bits 7 through 2 of the second instruction byte specify the instruction opcode. The opcode, in conjunction with the W field of the first byte, identifies the instruction. For example, the opcode "111011" denotes the decrement instruction; if W=0, the assembly language instruction is DECB, while if W=1, the instruction is DEC. Table 4-26 lists, in hexadecimal order, the opcode of every assembly language instruction.

The MM field (bits 0 and 1) indicates which pointer (base) register is to be used to construct the effective address of a memory operand. Table 4-23 defines the MM field encoding. (Memory operand addressing is described in section 3.8.)

Table 4-23. MM Field Encoding

Code	Base Register
00	GA
01	GB
10	GC
11	PP

When the AA field value is "01" (base register + offset addressing), the third byte of the instruction contains the offset value. This unsigned value is added to the content of the base register specified by the MM field to form the effective address of the memory operand.

When the AA field value is "10," the IX register value is added to the content of the base register specified by the MM field to provide a 64k range of effective addresses. (Note that the upper four bits of the IX register are not sign-extended.)

When the AA field value is "11," the IX register value is added to the base register value to form the effective address as described for an AA field value of "10." In this addressing mode, however, the IX register value is incremented by one after every byte accessed.

Table 4-24. 8089 Instruction Encoding

DATA TRANSFER INSTRUCTIONS

MOV = Move word variable	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Memory to register	R R R 0 0 A A 1	1 0 0 0 0 0 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 1	1 0 0 0 0 1 M M	offset if AA=01			
Memory to memory	0 0 0 0 0 A A 1	1 0 0 1 0 0 M M	offset if AA=01	0 0 0 0 0 A A 1	1 1 0 0 1 1 M M	offset if AA=01

Table 4-24. 8089 Instruction Encoding (Cont'd.)

DATA TRANSFER INSTRUCTIONS (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
MOVB = Move byte variable						
Memory to register	R R R 0 0 A A 0	1 0 0 0 0 0 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 0	1 0 0 0 0 1 M M	offset if AA=01			
Memory to memory	0 0 0 0 0 A A 0	1 0 0 1 0 0 M M	offset if AA=01	0 0 0 0 0 A A 0	1 1 0 0 1 1 M M	offset if AA=01
MOVBI = Move byte immediate						
Immediate to register	R R R 0 1 0 0 0	0 0 1 1 0 0 0 0	data-8			
Immediate to memory	0 0 0 0 1 A A 0	0 1 0 0 1 1 M M	offset if AA=01	data-8		
MOVI = Move word immediate						
Immediate to register	R R R 1 0 0 0 1	0 0 1 1 0 0 0 0	data-lo	data-hi		
Immediate to memory	0 0 0 1 0 A A 1	0 1 0 0 1 1 M M	offset if AA=01	data-lo	data-hi	
MOVP = Move pointer						
Memory to pointer register	P P P 0 0 A A 1	1 0 0 0 1 1 M M	offset if AA=01			
Pointer register to memory	P P P 0 0 A A 1	1 0 0 1 1 0 M M	offset if AA=01			
LPD = Load pointer with doubleword variable	P P P 0 0 A A 1	1 0 0 0 1 0 M M	offset if AA=01			
LPDI = Load pointer with doubleword immediate	P P P 1 0 0 0 1	0 0 0 0 1 0 0 0	offset-lo	offset-hi	segment-lo	segment-hi

ARITHMETIC INSTRUCTIONS

ADD = Add word variable						
Memory to register	R R R 0 0 A A 1	1 0 1 0 0 0 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 1	1 1 0 1 0 0 M M	offset if AA=01			
ADDB = Add byte variable						
Memory to register	R R R 0 0 A A 0	1 0 1 0 0 0 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 0	1 1 0 1 0 0 M M	offset if AA=01			
ADDI = Add word immediate						
Immediate to register	R R R 1 0 0 0 1	0 0 1 0 0 0 0 0	data-lo	data-hi		
Immediate to memory	0 0 0 1 0 A A 1	1 1 0 0 0 0 M M	offset if AA=01	data-lo	data-hi	

Table 4-24. 8089 Instruction Encoding (Cont'd.)

ARITHMETIC INSTRUCTIONS (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ADDBI = Add byte immediate						
Immedaite to register	R R R 0 1 0 0 0	0 0 1 0 0 0 0 0	data-8			
Immediate to memory	0 0 0 0 1 A A 0	1 1 0 0 0 0 M M	offset if AA=01	data-8		
INC = Increment word by 1						
Register	R R R 0 0 0 0 0	0 0 1 1 1 0 0 0				
Memory	0 0 0 0 0 A A 1	1 1 1 0 1 0 M M	offset if AA=01			
INCB = Increment byte by 1	0 0 0 0 0 A A 0	1 1 1 0 1 0 M M	offset if AA=01			
DEC = Decrement word by 1						
Register	R R R 0 0 0 0 0	0 0 1 1 1 1 0 0				
Memory	0 0 0 0 0 A A 1	1 1 1 0 1 1 M M	offset if AA=01			
DECB = Decrement byte by 1	0 0 0 0 0 A A 0	1 1 1 0 1 1 M M	offset if AA=01			

LOGICAL AND BIT MANIPULATION INSTRUCTIONS

AND = AND word variable						
Memory to register	R R R 0 0 A A 1	1 0 1 0 1 0 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 1	1 1 0 1 1 0 M M	offset if AA=01			
ANDB = AND byte variable						
Memory to register	R R R 0 0 A A 0	1 0 1 0 1 0 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 0	1 1 0 1 1 0 M M	offset if AA=01			
ANDI = AND word immediate						
Immediate to register	R R R 1 0 0 0 1	0 0 1 0 1 0 0 0	data-lo	data-hi		
Immediate to memory	0 0 0 1 0 A A 1	1 1 0 0 1 0 M M	offset if AA=01	data-lo	data-hi	
ANDBI = AND byte immediate						
Immediate to register	R R R 0 1 0 0 0	0 0 1 0 1 0 0 0	data-8			
Immediate to memory	0 0 0 0 1 A A 0	1 1 0 0 1 0 M M	offset if AA=01	data-8		
OR = OR word variable						
Memory to register	R R R 0 0 A A 1	1 0 1 0 0 1 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 1	1 1 0 1 0 1 M M	offset if AA=01			

Table 4-24. 8089 Instruction Encoding (Cont'd.)

LOGICAL AND BIT MANIPULATION INSTRUCTIONS (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ORB = OR byte variable						
Memory to register	R R R 0 0 A A 0	1 0 1 0 0 1 M M	offset if AA=01			
Register to memory	R R R 0 0 A A 0	1 1 0 1 0 1 M M	offset if AA=01			
ORI = OR word immediate						
Immediate to register	R R R 1 0 0 0 1	0 0 1 0 0 1 0 0	data-lo	data-hi		
Immediate to memory	0 0 0 1 0 A A 1	1 1 0 0 0 1 M M	offset if AA=01	data-lo	data-hi	
ORBI = OR byte immediate						
Immediate to register	R R R 0 1 0 0 0	0 0 1 0 0 1 0 0	data-8			
Immediate to memory	0 0 0 0 1 A A 0	1 1 0 0 0 1 M M	offset if AA=01	data-8		
NOT = NOT word variable						
Register	R R R 0 0 0 0 0	0 0 1 0 1 1 0 0				
Memory	0 0 0 0 0 A A 1	1 1 0 1 1 1 M M	offset if AA=01			
Memory to register	R R R 0 0 A A 1	1 0 1 0 1 1 M M	offset if AA=01			
NOTB = NOT byte variable						
Memory	0 0 0 0 0 A A 0	1 1 0 1 1 1 M M	offset if AA=01			
Memory to register	R R R 0 0 A A 0	1 0 1 0 1 1 M M	offset if AA=01			
SETB = Set bit to 1	B B B 0 0 A A 0	1 1 1 1 0 1 M M	offset if AA=01			
CLR = Clear bit to 0	B B B 0 0 A A 0	1 1 1 1 1 0 M M	offset if AA=01			

PROGRAM TRANSFER INSTRUCTIONS

***CALL** = Call	1 0 0 0 1 A A 1	1 0 0 1 1 1 M M	offset if AA=01	disp-8	
LCALL = Long call	1 0 0 1 0 A A 1	1 0 0 1 1 1 M M	offset if AA=01	disp-lo	disp-hi
***JMP** = Jump unconditional	1 0 0 0 1 0 0 0	0 0 1 0 0 0 0 0	disp-8		
LJMP = Long jump unconditional	1 0 0 1 0 0 0 1	0 0 1 0 0 0 0 0	disp-lo	disp-hi	

*The ASM-89 Assembler will automatically generate the long form of a program transfer instruction when the target is known to be beyond the byte-displacement range.

Table 4-24. 8089 Instruction Encoding (Cont'd.)

PROGRAM TRANSFER INSTRUCTIONS (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
***JZ** = Jump if word is 0						
Label to register	R R R 0 1 0 0 0	0 1 0 0 0 1 0 0	disp-8			
Label to memory	0 0 0 0 1 A A 1	1 1 1 0 0 1 M M	offset if AA=01	disp-8		
LJZ = Long jump if word is 0						
Label to register	R R R 1 0 0 0 0	0 1 0 0 0 1 0 0	disp-lo	disp-hi		
Label to memory	0 0 0 1 0 A A 1	1 1 1 0 0 1 M M	offset if AA=01	disp-lo	disp-hi	
***JZB** = Jump if byte is 0	0 0 0 0 1 A A 0	1 1 1 0 0 1 M M	offset if AA=01	disp-8		
LJZB = Long jump if byte is 0	0 0 0 1 0 A A 0	1 1 1 0 0 1 M M	offset if AA=01	disp-lo	disp-hi	
***JNZ** = Jump if word not 0						
Label to register	R R R 0 1 0 0 0	0 1 0 0 0 0 0 0	disp-8			
Label to memory	0 0 0 0 1 A A 1	1 1 1 0 0 0 M M	offset if AA=01	disp-8		
LJNZ = Long jump if word not 0						
Label to register	R R R 1 0 0 0 0	0 1 0 0 0 0 0 0	disp-lo	disp-hi		
Label to memory	0 0 0 1 0 A A 1	1 1 1 0 0 0 M M	offset if AA=01	disp-lo	disp-hi	
***JNZB** = Jump if byte not 0	0 0 0 0 1 A A 0	1 1 1 0 0 0 M M	offset if AA=01	disp-8		
LJNZB = Long jump if byte not 0	0 0 0 1 0 A A 0	1 1 1 0 0 0 M M	offset if AA=01	disp-lo	disp-hi	
***JMCE** = Jump if masked compare equal	0 0 0 0 1 A A 0	1 0 1 1 0 0 M M	offset if AA=01	disp-8		
LJMCE = Long jump if masked compare equal	0 0 0 1 0 A A 0	1 0 1 1 0 0 M M	offset if AA=01	disp-lo	disp-hi	
***JMCNE** = Jump if masked compare not equal	0 0 0 0 1 A A 0	1 0 1 1 0 1 M M	offset if AA=01	disp-8		
LJMCNE = Long jump if masked compare not equal	0 0 0 1 0 A A 0	1 0 1 1 0 1 M M	offset if AA=01	disp-lo	disp-hi	
***JBT** = Jump if bit is 1	B B B 0 1 A A 0	1 0 1 1 1 1 M M	offset if AA=01	disp-8		

*The ASM-89 Assembler will automatically generate the long form of a program transfer instruction when the target is known to be beyond the byte-displacement range.

Table 4-24. 8089 Instruction Encoding (Cont'd.)

PROGRAM TRANSFER INSTRUCTIONS (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
LJBT = Long jump if bit is 1	B B B 1 0 A A 0	1 0 1 1 1 1 M M	offset if AA=01	disp-lo	disp-hi	
***JNBT** = Jump if bit is not 1	B B B 0 1 A A 0	1 0 1 1 1 0 M M	offset if AA=01	disp-8		
LJNBT = Long jump if bit is not 1	B B B 1 0 A A 0	1 0 1 1 1 0 M M	offset if AA=01	disp-lo	disp-hi	

PROCESSOR CONTROL INSTRUCTIONS

TSL = Test and set while locked	0 0 0 1 1 A A 0	1 0 0 1 0 1 M M	offset if AA=01	data-8	disp-8
WID = Set logical bus widths	1 S D* 0 0 0 0 0	0 0 0 0 0 0 0 0			
XFER = Enter DMA mode	0 1 1 0 0 0 0 0	0 0 0 0 0 0 0 0			
SINTR = Set interrupt service bit	0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0			
HLT = Halt channel program	0 0 1 0 0 0 0 0	0 1 0 0 1 0 0 0			
NOP = No operation	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0			

*S=source width, D=destination width; 0=8 bits, 1=16 bits

*The ASM-89 Assembler will automatically generate the long form of a program transfer instruction when the target is known to be beyond the byte-displacement range.

Table 4-26 lists all of the 8089 machine instructions in hexadecimal/binary order by their *second* byte. This table may be used to "decode" an assembled machine instruction into its ASM-89 symbolic form. The preceding table (table 4-25) defines the notation used in table 4-26.

Table 4-25. Key to 8089 Machine Instruction Decoding Guide

Identifier	Explanation
S	Logical width of source bus; 0=8, 1=16
D	Logical width of destination bus; 0=8, 1=16
PPP	Pointer register encoded in R/B/P field
RRR	Register encoded in R/B/P field
AA	AA (addressing mode) field
BBB	Bit select encoded in R/B/P field
offset-lo	Low-order byte of offset word in doubleword pointer
offset-hi	High-order byte of offset word in doubleword pointer
segment-lo	Low-order byte of segment word in doubleword pointer
segment-hi	High-order byte of segment word in doubleword pointer
data-8	8-bit immediate constant
data-lo	Low-order byte of 16-bit immediate constant
data-hi	High-order byte of 16-bit immediate constant
disp-8	8-bit signed displacement
disp-lo	Low-order byte of 16-bit signed displacement
disp-hi	High-order byte of 16-bit signed displacement
(offset)	Optional 8-bit offset used in offset addressing

Table 4-26. 8089 Machine Instruction Decoding Guide

Byte 1	Byte 2		Bytes 3, 4, 5, 6	ASM89 Instruction Format
	Hex	Binary		
00000000	00	00000000		NOP
01000000	00	00000000		SINTR
1SD00000	00	00000000		WID source-width,dest-width
01100000	00	00000000		XFER
	01	00000001		} not used
	↓	↓		
	07	00000111		
PPP10001	08	00001000	offset-lo,offset-hi,segment-lo,segment-hi	LPDI ptr-reg,immed32
	09	00001001		} not used
	↓	↓		
	1F	00011111		
RRR01000	20	00100000	data-8	ADDBI register,immed8
RRR10001	20	00100000	data-lo,data-hi	ADDI register,immed16
10001000	20	00100000	disp-8	JMP short-label
10010001	20	00100000	disp-lo,disp-hi	LJMP long-label
	21	00100001		} not used
	↓	↓		
	23	00100011		
RRR01000	24	00100100	data-8	ORBI register,immed8
RRR10001	24	00100100	data-lo,data-hi	ORI register,immed16
	25	00100101		} not used
	↓	↓		
	27	00100111		
RRR01000	28	00101000	data-8	ANDBI register,immed8

Table 4-26. 8089 Machine Instruction Decoding Guide (Cont'd.

Byte 1	Byte 2		Bytes 3, 4, 5, 6	ASM89 Instruction Format
	Hex	Binary		
RRR10001	28	00101000	data-lo,data-hi	ANDI register,immed16
	29	00101001		} not used
	↓	↓		
	2B	00101011		
RRR00000	2C	00101100		NOT register
	2D	00101101		} not used
	↓	↓		
	2F	00101111		
RRR01000	30	00110000	data-8	MOVBI register,immed8
RRR10001	30	00110000	data-lo,data-hi	MOVI register,immed16
	31	00110001		} not used
	↓	↓		
	37	00110111		
RRR00000	38	00111000		INC register
	39	00111001		} not used
	↓	↓		
	3B	00111011		
RRR00000	3C	00111100		DEC register
	3D	00111101		} not used
	↓	↓		
	3F	00111111		
RRR01000	40	01000000	disp-8	JNZ register,short-label
RRR10000	40	01000000	disp-lo,disp-hi	LJNZ register,long-label
	41	01000001		} not used
	↓	↓		
	43	01000011		
RRR01000	44	01000100	disp-8	JZ register,short-label
RRR10000	44	01000100	disp-lo,disp-hi	LJZ register,short-label
	45	01000101		} not used
	↓	↓		
	47	01000111		
00100000	48	01001000		HLT
	49	01001001		} not used
	↓	↓		
	4B	01001011		
00001AA0	4C	010011MM	} (offset),data-8	} MOVBI mem8,immed8
↓	↓	↓		
00001AA0	4F	010011MM		
00010AA1	4C	010011MM	} (offset),data-lo,data-hi	} MOVI mem16,immed16
↓	↓	↓		
00010AA1	4F	010011MM		
	50	01010000		} not used
	↓	↓		
	7F	01111111		
RRR00AA0	80	100000MM	} (offset)	} MOVB register,mem8
↓	↓	↓		
RRR00AA0	83	100000MM		

Table 4-26. 8089 Machine Instruction Decoding Guide (Cont'd.

Byte 1	Byte 2 Hex	Byte 2 Binary	Bytes 3, 4, 5, 6	ASM89 Instruction Format
RRR00AA1	80	100000MM	(offset)	MOV register,mem16
↓	↓	↓		
RRR00AA1	83	100000MM		
RRR00AA0	84	100001MM	(offset)	MOVB mem8,register
↓	↓	↓		
RRR00AA0	87	100001MM		
RRR00AA1	84	100001MM	(offset)	MOV mem16,register
↓	↓	↓		
RRR00AA1	87	100001MM		
PPP00AA1	88	100010MM	(offset)	LPD ptr-reg,mem32
↓	↓	↓		
PPP00AA1	8B	100010MM		
PPP00AA1	8C	100011MM	(offset)	MOVP ptr-reg,mem24
↓	↓	↓		
PPP00AA1	8F	100011MM		
00000AA0	90	100100MM	(offset),00000AA0,110011MM,(offset)	MOVB mem8,mem8
↓	↓	↓		
00000AA0	93	100100MM		
00000AA1	90	100100MM	(offset),00000AA1,110011MM,(offset)	MOV mem16,mem16
↓	↓	↓		
00000AA1	93	100100MM		
00011AA0	94	100101MM	(offset),data-8,disp-8	TSL mem8,immed8,short-label
↓	↓	↓		
00011AA0	97	100101MM		
PPP00AA1	98	100110MM	(offset)	MOVP mem24,ptr-reg
↓	↓	↓		
PPP00AA1	9B	100110MM		
10001AA1	9C	100111MM	(offset),disp-8	CALL mem24,short-label
↓	↓	↓		
10001AA1	9F	100111MM		
10010AA1	9C	100111MM	(offset),disp-lo,disp-hi	LCALL mem24,long-label
↓	↓	↓		
10010AA1	9F	100111MM		
RRR00AA0	A0	101000MM	(offset)	ADDB register,mem8
↓	↓	↓		
RRR00AA0	A3	101000MM		
RRR00AA1	A0	101000MM	(offset)	ADD register,mem16
↓	↓	↓		
RRR00AA1	A3	101000MM		
RRR00AA0	A4	101001MM	(offset)	ORB register,mem8
↓	↓	↓		
RRR00AA0	A7	101001MM		
RRR00AA1	A4	101001MM	(offset)	OR register,mem16
↓	↓	↓		
RRR00AA1	A7	101001MM		
RRR00AA0	A8	101010MM	(offset)	ANDB mem8,register
↓	↓	↓		
RRR00AA0	AB	101010MM		

Table 4-26. 8089 Machine Instruction Decoding Guide (Cont'd.

Byte 1	Byte 2		Bytes 3, 4, 5, 6	ASM89 Instruction Format
	Hex	Binary		
RRR00AA1	A8	101010MM	(offset)	AND mem16,register
↓	↓	↓		
RRR00AA1	AB	101010MM		
RRR00AA0	AC	101011MM	(offset)	NOTB register,mem8
↓	↓	↓		
RRR00AA0	AF	101011MM		
RRR00AA1	AC	101011MM	(offset)	NOT register,mem16
↓	↓	↓		
RRR00AA1	AF	101011MM		
00001AA0	B0	101100MM	(offset),disp-8	JMCE mem8,short-label
↓	↓	↓		
00001AA0	B3	101100MM		
00010AA0	B0	101100MM	(offset),disp-lo,disp-hi	LJMCE mem8,long-label
↓	↓	↓		
00010AA0	B3	101100MM		
00001AA0	B4	101101MM	(offset),disp-8	JMCNE mem8,short-label
↓	↓	↓		
00001AA0	B7	101101MM		
00010AA0	B4	101101MM	(offset),disp-lo,disp-hi	LJMCNE mem8,long-label
↓	↓	↓		
00010AA0	B7	101101MM		
BBB01AA0	B8	101110MM	(offset),disp-8	JNBT mem8,bit-select,short-label
↓	↓	↓		
BBB01AA0	BB	101110MM		
BBB10AA0	B8	101110MM	(offset),disp-lo,disp-hi	LJNBT mem8,bit-select,long-label
↓	↓	↓		
BBB10AA0	BB	101110MM		
BBB01AA0	BC	101111MM	(offset),disp-8	JBT mem8,bit-select,short-label
↓	↓	↓		
BBB01AA0	BF	101111MM		
BBB10AA0	BC	101111MM	(offset),disp-lo,disp-hi	LJBT mem8,bit-select,long-label
↓	↓	↓		
BBB10AA0	BF	101111MM		
00001AA0	C0	110000MM	(offset),data-8	ADDBI mem8,immed8
↓	↓	↓		
00001AA0	C3	110000MM		
00010AA1	C0	110000MM	(offset),data-lo,data-hi	ADDI mem16,immed16
↓	↓	↓		
00010AA1	C3	110000MM		
00001AA0	C4	110001MM	(offset),data-8	ORBI mem8,immed8
↓	↓	↓		
00001AA0	C7	110001MM		
00010AA1	C4	110001MM	(offset),data-lo,data-hi	ORI mem16,immed16
↓	↓	↓		
00010AA1	C7	110001MM		
00001AA0	C8	110010MM	(offset),data-8	ANDBI mem8,immed8
↓	↓	↓		
00001AA0	CB	110010MM		

Table 4-26. 8089 Machine Instruction Decoding Guide (Cont'd.

Byte 1	Byte 2		Bytes 3, 4, 5, 6	ASM89 Instruction Format
	Hex	Binary		
00010AA1	C8	110010MM	(offset),data-lo,data-hi	ANDI mem16,immed16
↓	↓	↓		
00010AA1	CB	110010MM		
	CC	11001100		not used
	↓	↓		
	CF	11001111		
RRR00AA0	D0	110100MM	(offset)	ADDB mem8,register
↓	↓	↓		
RRR00AA0	D3	110100MM		
RRR00AA1	D0	110100MM	(offset)	ADD mem16,register
↓	↓	↓		
RRR00AA1	D3	110100MM		
RRR00AA0	D4	110101MM	(offset)	ORB mem8,register
↓	↓	↓		
RRR00AA0	D7	110101MM		
RRR00AA1	D4	110101MM	(offset)	OR mem16,register
↓	↓	↓		
RRR00AA1	D7	110101MM		
RRR00AA0	D8	110110MM	(offset)	ANDB mem8,register
↓	↓	↓		
RRR00AA0	DB	110110MM		
RRR00AA1	D8	110110MM	(offset)	AND mem16,register
↓	↓	↓		
RRR00AA1	DB	110110MM		
RRR00AA0	DC	110111MM	(offset)	NOTB mem8,register
↓	↓	↓		
RRR00AA0	DF	110111MM		
RRR00AA1	DC	110111MM	(offset)	NOT mem16,register
↓	↓	↓		
RRR00AA1	DF	110111MM		
00001AA0	E0	111000MM	(offset),disp-8	JNZB mem8,short-label
↓	↓	↓		
00001AA0	E3	111000MM		
00001AA1	E0	111000MM	(offset),disp-8	JNZ mem16,short-label
↓	↓	↓		
00001AA1	E3	111000MM		
00010AA0	E0	111000MM	(offset),disp-lo,disp-hi	LJNZB mem8,long-label
↓	↓	↓		
00010AA0	E3	111000MM		
00010AA1	E0	111000MM	(offset),disp-lo,disp-hi	LJNZ mem16,longlabel
↓	↓	↓		
00010AA1	E3	111000MM		
00001AA0	E4	111001MM	(offset),disp-8	JZB mem8,short-label
↓	↓	↓		
00001AA0	E7	111001MM		
00001AA1	E4	111001MM	(offset),disp-8	JZ mem16,short-label
↓	↓	↓		
00001AA1	E7	111001MM		

Table 4-26. 8089 Machine Instruction Decoding Guide (Cont'd.

Byte 1	Byte 2		Bytes 3, 4, 5, 6	ASM89 Instruction Format
	Hex	Binary		
00010AA0 ↓ 00010AA0	E4 ↓ E7	111001MM ↓ 111001MM	(offset),disp-lo,disp-hi	LJZB mem8,long-label
00010AA1 ↓ 00010AA1	E4 ↓ E7	111001MM ↓ 111001MM	(offset),disp-lo,disp-hi	LJZ mem16,long-label
00000AA0 ↓ 00000AA0	E8 ↓ EB	111010MM ↓ 111010MM	(offset)	INCB mem8
00000AA1 ↓ 00000AA1	E8 ↓ EB	111010MM ↓ 111010MM	(offset)	INC mem16
00000AA0 ↓ 00000AA0	EC ↓ EF	111011MM ↓ 111011MM	(offset)	DECB mem8
00000AA1 ↓ 00000AA1	EC ↓ EF	111011MM ↓ 111011MM	(offset)	DEC mem16
	F0 ↓ F3	11110000 ↓ 11110000		not used
BBB00AA0 ↓ BBB00AA0	F4 ↓ F7	111101MM ↓ 111101MM	(offset)	SETB mem8,0-7
BBB00AA0 ↓ BBB00AA0	F8 ↓ FB	111110MM ↓ 111110MM	(offset)	CLR mem8,0-7
	FC ↓ FF	11111100 ↓ 11111111		not used