BIBLIOGRAPHY

Because of the technical level of this book, the major sources of further information on the topics discussed are manufacturer's data books, application notes, and articles in current engineering periodicals. With the foundation you get from this book you should be able to comfortably read these materials. Listed below, by chapter, are some materials that will give you more details for many of the topics we discuss in the book. Following the chapter listings is a list of periodicals which we have found to be particularly helpful in keeping up with the latest advances in microcomputer evolution and applications.

Chapter 1

Hall, Douglas V., *Digital Circuits and Systems*, McGraw-Hill, Inc., New York, 1989.

Chapters 2–6

Mick, John, and Jim Brick; *Bit-Slice Microprocessor Design*, McGraw-Hill, Inc., New York, 1980.

A History of Microprocessor Development at Intel, Intel Article Reprint/AR-173, Intel Corporation, Santa Clara, Calif.

Microprocessors, Intel Corporation, Santa Clara, Calif., latest edition (Databook).

Turbo Assembler User's Guide, Borland International, Scotts Valley, Calif., 1988.

Macro Assembler User's Guide, Microsoft Corp., Bellingham, Wash., 1989.

Peripherals, Intel Corporation, Santa Clara, Calif., latest edition (Databook).

SDK-86 MCS-86 System Design Kit User's Guide, Intel Corporation, Santa Clara, Calif., 1981.

8086 System Design, Application Note, Intel Corporation/AP-67, Intel Corporation, Santa Clara, Calif.

Chapters 9–10

Peripherals, Intel Corporation, Santa Clara, Calif., latest edition (Databook).

IBM PC TECHNICAL REFERENCE MANUAL, IBM Corp., Boca Raton, Fla., 1983.

Dorf, Richard C., *Robotics and Automated Manufacturing*, Reston Publishing Company, Reston, Va. 1983.

AMF Potter & Brumfield Catalog, Potter & Brumfield, Princeton, Ind., latest edition.

Optoelectronics Designer's Catalog, Hewlett-Packard, Palo Alto, Calif., latest edition.

Interfacing Liquid Crystal Displays in Digital Systems, Application Note AN-8, Beckman Instruments, Inc., Scottsdale, Ariz., latest edition.

Optoelectronics Device Data Book, DL118R1, Motorola Semiconductor Products Inc., Phoenix, Ariz., latest edition.

Sandhu, H. S., Hands-On Introduction to ROBOTICS—The Manual for the XR-Series Robots, Rhino Robots, Champaign, Ill., latest edition.

Slo-Syn DC Stepping Motors Catalog, DCM1078, Superior Electric Company, Bristol, Conn., latest edition.

Auslander, David M., and Paul Sagues, *Microprocessors for Measurement and Control*, Osborne/McGraw-Hill, Berkeley, Calif., 1981.

Allocca, John A., and Allen Stuart, *Transducers Theory and Applications*, Reston Publishing Company, Inc., Reston, Va., 1984.

Seippel, Robert G., *Transducers, Sensors and Detectors*, Reston Publishing Company, Inc., Reston, Va., 1983.

Johnson, Curtis D., *Process Control Instrumentation Technology*, John Wiley & Sons, New York, latest edition.

Sheingold, Daniel H. (ed.), *Transducer Interfacing Handbook — A Guide to Analog Signal Conditioning*, Analog Devices, Inc., Norwood, Mass., latest edition.

Analog Devices Industrial Control Series Data Sheet, Analog Devices, Inc., Norwood, Mass., latest edition.

Texas Instruments, Inc. *Third Generation TMS320 User's Guide*, Dallas, Tex., latest edition.

Chassaing, Rulph, and Darell W. Horning, *Digital Signal Processing with the TMS320C25*, John Wiley & Sons, New York, 1990.

Chapter 11

Peripherals, Intel Corporation, Santa Clara, Calif., latest edition (Databook).

Microprocessors, Intel Corporation, Santa Clara, Calif., latest edition (Databook).

IBM PC Technical Reference Manual, IBM Corp., Boca Raton, Fla., 1983.

IBM PC/AT Technical Reference Manual, IBM Corp., Florida, 1984.

8086 Macro Assembly Language Reference Manual for 8086-Based Development Systems, Intel Corporation, Santa Clara, Calif., latest edition.

Error Detecting and Correcting Codes, Application Note AP-46, Intel Corporation, Santa Clara, Calif., 1979.

Getting Started With the Numeric Data Processor, Application Note AP-113, Intel Corporation, Santa Clara, Calif., 1981.

Hall, Douglas V., *Digital Circuits and Systems*, McGraw-Hill, Inc., New York, 1989. (Chapter 15 on Electronic Design Automation.)

Chapter 12

Turbo C++ User's Manual; Turbo C++ Programmer's Guide; Turbo C++ Library Reference, Borland International, Scotts Valley, Calif., 1990.

Microsoft C 6.0 Optimizing Compiler, Language Reference, Codeview, and Utilities, Microsoft Corporation, Bellingham, Wash., 1987.

Schildt, Herbert, *Turbo C: The Complete Reference*, Borland-Osborne/McGraw-Hill, Berkeley, Calif., 1988.

Waite, Michael, and Stephen Prata, *New C Primer Plus*, Howard W. Sams & Company, Carmel, Ind., 1990.

Chapter 13

Peripherals, Intel Corporation, Santa Clara, Calif., latest edition (Databook).

Lesea, Austin, and Rodnay Zaks, *Microprocessor Interfacing Techniques*, Sybex Inc., Berkeley, Calif., latest edition.

An Intelligent Data Base System Using the 8272, Application Note AP-116, Intel Corporation, Santa Clara, Calif., 1981. (Old, but good basics.)

Sutty, George, and Steve Blair, *Programmer's Guide to the EGA/VGA*, Brady Books, New York, NY, 1988.

Stevens, Roger T., *Fractal Programming in C*, M&T Books, Redwood City, Calif., 1989. (Demo disk comes with book and provides much fun.)

DOS Technical Reference Manual, Microsoft Corporation, Bellingham, Wash., latest edition.

Jamsa, Kris, *DOS—The Complete Reference*, Osborne/McGraw-Hill, Berkeley, Calif., 1987.

Jamsa, Kris, *DOS Power User's Guide*, Osborne/McGraw-Hill, Berkeley, Calif., 1988.

Luther, Arch, *Digital Video in the PC Environment*, McGraw-Hill, New York, NY, 1989.

Phillips International, Inc. *Compact Disk-Interactive, A Designer's Overview*, McGraw-Hill, New York, NY, 1989.

Chapter 14

Microcommunications Handbook, Intel Corporation, Santa Clara, Calif., latest edition. (Two-volume databook with many application notes.)

Stallings, William D., *Local Networks, an Introduction*, Macmillan, New York, NY, 1984. (Older, but good, clear basics.)

Friend, George E., et al., *Understanding Data Communications*, Howard W. Sams, Indianapolis, Ind., 1987. (Good basics.)

Fike, John L., and George E. Friend, *Understanding Telephone Electronics*, Howard W. Sams, Indianapolis, Ind., 1984. (Good basics.)

Schatt, Stan, *Understanding Local Area Networks*, Howard W. Sams, Indianapolis, Ind., 1989. (Good basics.)

McNamara, John E., *Technical Aspects of Data Communication*, Digital Equipment Corporation, Maynard, Mass., latest edition.

EIA Standard RS-422, Electrical Characteristics of Balanced Voltage Digital Interface Circuits, Electronic Industries Association, Engineering Department, Washington, D.C., 1975.

EIA Standard RS-232-C, Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange, Electronic Industries Association, Engineering Department, Washington, D.C., 1969.

Sterling, Donald J., *Technician's Guide to Fiber Optics*, Delmar Publishers Inc, Albany, NY, 1987. (Good basics with emphasis on cables, connectors, and couplers.)

Chapter 15

Stevens, Al, *Turbo C Memory Resident Utilities, Screen I/O, and Programming Techniques*, MIS Press, Portland, Ore., 1987.

Kaisler, Stephen H., *The Design of Operating Systems for Small Computer Systems*, John Wiley & Sons, Inc., New York, 1983.

Microprocessors, Intel Corporation, Santa Clara, Calif., latest edition (Databook).

ASM386 Assembly Language Reference Manual, Intel Corporation, Santa Clara, Calif., 1986.

80386 System Software Writer's Guide, Intel Corporation, Santa Clara, Calif., 1987.

386 Microprocessor Hardware Reference Manual, Intel Corporation, Santa Clara, Calif., 1988.

80386 Programmer's Reference Manual, Intel Corporation, Santa Clara, Calif., 1986.

Pappas, Chris H., and William H. Murray, *Inside the Model 80*, Osborne/McGraw-Hill, Berkeley, Calif., 1988.

Microsoft Windows User's Guide, Microsoft Corporation, Bellingham, Wash., 1990.

Klimasausskas, Casimir C., *Teaching Your Computer to Learn*, NeuralWare, Inc., Pittsburgh, Penn., 1988 (Booklet).

Periodicals

BYTE. ISSN 0360-5280. Byte Publications, Inc., 70 Main Street, Peterborough, NH 03458.

EDN. ISSN 0012-7515. Cahners Publishing Co., 221 Columbus Avenue, Boston, MA 02116.

Electronic Design. USPS-172-080. Hayden Publishing Co., Inc., 50 Essex Street, Rochelle Park, NJ 07662.

Electronics. ISSN 0013-5070. McGraw-Hill, Inc., 1221 Avenue of the Americas, New York, NY 10020.

Instruments & Control Systems. ISSN 0164-0089. Chilton Company, Chilton Way, Radnor, PA 19089.

Electronic Engineering Times. ISSN 0192-1541. Electronic Engineering Times, 600 Community Drive, Manhasset, NY 11030.

iAPX 86/10
16-BIT HMOS MICROPROCESSOR

8086/8086-2/8086-1

- Direct Addressing Capability 1 MByte of Memory
- Architecture Designed for Powerful Assembly Language and Efficient High Level Languages.
- 14 Word, by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Bit, Byte, Word, and Block Operations
- 8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal Including Multiply and Divide
- Range of Clock Rates:
  5 MHz for 8086,
  8 MHz for 8086-2,
  10 MHz for 8086-1
- MULTIBUS™ System Compatible Interface
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel iAPX 86/10 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CerDIP package. The iAPX 86/10 operates in both single processor and multiple processor configurations to achieve high performance levels.

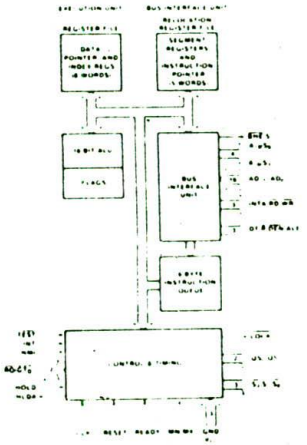
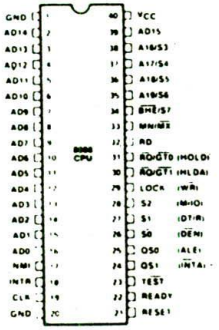


Figure 1. iAPX 86/10 CPU Block Diagram



40 LEAD

Figure 2. iAPX 86/10 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for iAPX 86 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function
AD_{15}-AD_0	2-16, 39	I/O	**Address Data Bus:** These lines constitute the time multiplexed memory/I/O address (T_1) and data (T_2, T_3, T_W, T_4) bus. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7-D_0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A_0 to condition chip select functions. (See $\overline{BHE}$.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge."
A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3	35-38	O	**Address/Status:** During T_1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W, and T_4. The status of the interrupt enable FLAG bit (S_5) is updated at the beginning of each CLK cycle. A_{17}/S_4 and A_{16}/S_3 are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge."
$\overline{BHE}/S_7$	34	O	**Bus High Enable/Status:** During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D_{15}-D_8. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T_2, T_3, and T_4. The signal is active LOW, and floats to 3-state OFF in "hold." It is LOW during T_1 for the first interrupt acknowledge cycle.
$\overline{RD}$	32	O	**Read:** Read strobe indicates that the processor is performing a memory of I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 8086 local bus. $\overline{RD}$ is active LOW during T_2, T_3 and T_W of any read cycle, and is guaranteed to remain HIGH in T_2 until the 8086 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge."
READY	22	I	**READY:** is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	**Interrupt Request:** is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{TEST}$	23	I	**$\overline{TEST}$:** input is examined by the "Wait" instruction. If the $\overline{TEST}$ input is LOW execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

A_{17}/S_4	A_{16}/S_3	Characteristics
0 (LOW)	0	Alternate Data
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data
S_6 is 0 (LOW)		

$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
NMI	17	I	**Non-maskable interrupt:** an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	**Reset:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**Clock:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		V_{CC}: +5V power supply pin.
GND	1, 20		**Ground**
MN/$\overline{MX}$	33	I	**Minimum/Maximum:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8086/8288 system in maximum mode (i.e., MN/$\overline{MX}$ = V_{SS}). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$	26-28	O	**Status:** active during T_4, T_1, and T_2 and is returned to the passive state (1,1,1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S_2}$, $\overline{S_1}$, or $\overline{S_0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals float to 3-state OFF in "hold acknowledge." These status lines are encoded as shown.
$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$	30, 31	I/O	**Request/Grant** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ has an internal pull up resistor so may be left unconnected. The request/grant sequence is as follows (see Figure 9): 1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1). 2. During a T_4 or T_1 clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW. If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met: 1. Request occurs on or before T_2. 2. Current cycle is not the low byte of a word (on an odd address). 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1 (HIGH)	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
			If the local bus is idle when the request is made the two possible events will follow: 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.
$\overline{LOCK}$	29	O	**$\overline{LOCK}$:** output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge."
QS_1, QS_0	24, 25	O	**Queue Status:** The queue status is valid during the CLK cycle after which the queue operation is performed. QS_1 and QS_0 provide status to allow external tracking of the internal 8086 instruction queue.

QS_1	QS_0	CHARACTERISTICS
0 (LOW)	0	No Operation
0	1	First Byte of Op Code from Queue
1 (HIGH)	0	Empty the Queue
1	1	Subsequent Byte from Queue

The following pin function descriptions are for the 8086 in minimum mode (i.e., MN/$\overline{MX}$ = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

M/$\overline{IO}$	28	O	**Status line:** logically equivalent to S_2 in the maximum mode. It is used to distinguish a memory access from an I/O access. M/$\overline{IO}$ becomes valid in the T_4 preceding a bus cycle and remains valid until the final T_4 of the cycle (M = HIGH, IO = LOW). M/$\overline{IO}$ floats to 3-state OFF in local bus "hold acknowledge."
$\overline{WR}$	29	O	**Write:** indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/$\overline{IO}$ signal. $\overline{WR}$ is active for T_2, T_3 and T_W of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge."
$\overline{INTA}$	24	O	$\overline{INTA}$ is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T_2, T_3 and T_W of each interrupt acknowledge cycle.
ALE	25	O	**Address Latch Enable:** provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during T_1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{R}$	27	O	**Data Transmit/Receive:** needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically DT/$\overline{R}$ is equivalent to $\overline{S_1}$ in the maximum mode, and its timing is the same as for M/$\overline{IO}$. (T = HIGH, R = LOW.) This signal floats to 3-state OFF in local bus "hold acknowledge."
$\overline{DEN}$	26	O	**Data Enable:** provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. $\overline{DEN}$ is active LOW during each memory and I/O access and for INTA cycles. For a read or $\overline{INTA}$ cycle it is active from the middle of T_2 until the middle of T_4, while for a write cycle it is active from the beginning of T_2 until the middle of T_4. $\overline{DEN}$ floats to 3-state OFF in local bus "hold acknowledge."
HOLD, HLDA	31, 30	I/O	**HOLD:** indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T_1 clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWer the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. The same rules as for $\overline{RQ}/\overline{GT}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

FUNCTIONAL DESCRIPTION

GENERAL OPERATION

The internal functions of the iAPX 86/10 processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the block diagram of Figure 1.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 6 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First In First Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

The execution unit receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage. See the Instruction Set description for further register set and architectural descriptions.

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3a.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries and are thus not constrained to even boundaries as is the case in many 16-bit computers. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU automatically performs the proper number of memory accesses, one if the word operand is on an even byte boundary and two if it is on an odd byte boundary. Except for the performance penalty, this double access is transparent to the software. This performance penalty does not occur for instruction fetches, only word operands.

Physically, the memory is organized as a high bank (D_{15}-D_8) and a low bank (D_7-D_0) of 512K 8-bit bytes addressed in parallel by the processor's address lines.

'A_{19} - A_1. Byte data with even addresses is transferred on the D_7-D_0 bus lines while odd addressed byte data (A_0 HIGH) is transferred on the D_{15}-D_8 bus lines. The processor provides two enable signals, $\overline{BHE}$ and A_0, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations. Explicitly selected using a segment override.

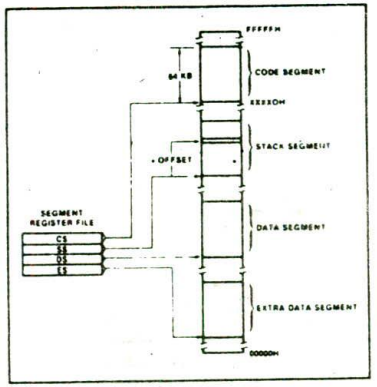


Figure 3a. Memory Organization

In referencing word data the BIU requires one or two memory cycles depending on whether or not the starting byte of the word is on an even or odd address, respectively. Consequently, in referencing word operands performance can be optimized by locating data on even address boundaries. This is an especially useful technique for using the stack, since odd address references to the stack may adversely affect the context switching time for interrupt processing or task multiplexing.

Certain locations in memory are reserved for specific CPU operations (see Figure 3b.) Locations from address FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be. Locations 00000H through 003FFH are reserved for interrupt operations. Each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address. The pointer elements are assumed to have been stored at the respective places in reserved memory prior to occurrence of interrupts.

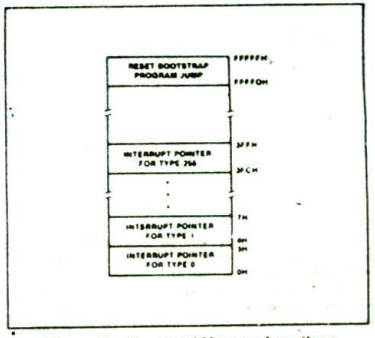


Figure 3b. Reserved Memory Locations

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum iAPX 86/10 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8086 is equipped with a strap pin (MN/$\overline{MX}$) which defines the system configuration. The definition of a certain subset of the pins changes dependent on the condition of the strap pin. When MN/$\overline{MX}$ pin is strapped to GND, the 8086 treats pins 24 through 31 in maximum mode. An 8288 bus controller interprets status information coded into $\overline{S}_0$,$\overline{S}_1$,$\overline{S}_2$ to generate bus timing and control signals compatible with the MULTIBUS™ architecture. When the MN/$\overline{MX}$ pin is strapped to V_{CC}, the 8086 generates bus control signals itself on pins 24 through 31, as shown in parentheses in Figure 2. Examples of minimum mode and maximum mode systems are shown in Figure 4.

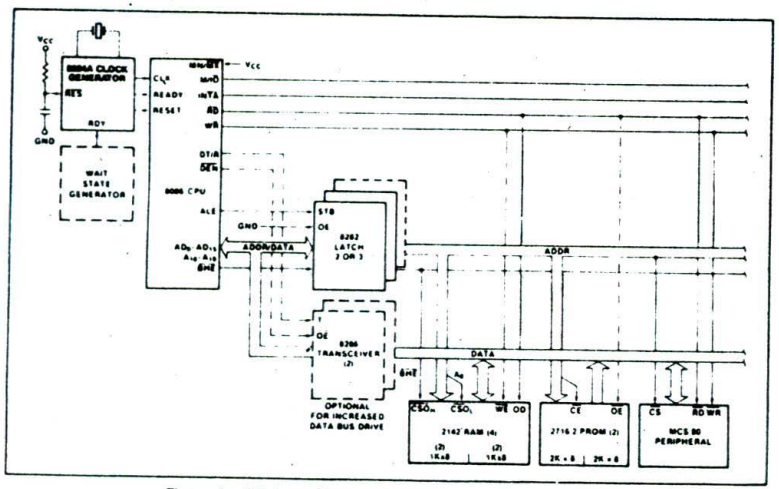


Figure 4a. Minimum Mode iAPX 86/10 Typical Configuration

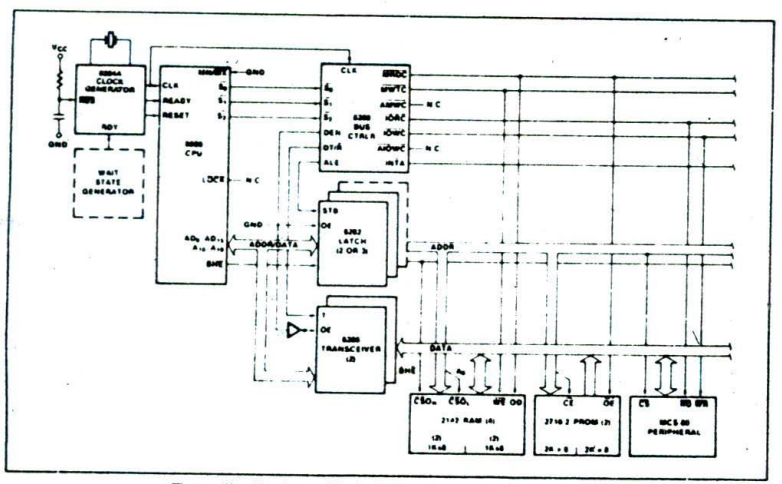


Figure 4b. Maximum Mode iAPX 86/10 Typical Configuration

BUS OPERATION

The 86/10 has a combined address and data bus commonly referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor while permitting the use of a standard 40-lead package. This "local bus" can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. In addition, the bus can also be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T_1, T_2, T_3 and T_4 (see Figure 5). The address is emitted from the processor during T_1 and data transfer occurs on the bus during T_3 and T_4. T_2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "Wait" states (T_W) are inserted between T_3 and T_4. Each inserted "Wait" state is of the same duration as a CLK cycle. Periods can occur between 8086 bus cycles. These are referred to as "Idle" states (T_I) or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T_1 of any bus cycle the ALE (Address Latch Enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S_0}$, $\overline{S_1}$, and $\overline{S_2}$ are used, in maximum mode, by the bus controller to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S_3 through S_7 are multiplexed with high-order address bits and the $\overline{BHE}$ signal, and are therefore valid during T_2 through T_4. S_3 and S_4 indicate which segment register (see Instruction Set description) was used for this bus cycle in forming the address, according to the following table:

S_4	S_3	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S_5 is a reflection of the PSW interrupt enable bit. S_6=0 and S_7 is a spare status bit.

I/O ADDRESSING

In the 86/10, I/O operations can address up to a maximum of 64K I/O byte registers or 32K I/O word registers. The I/O address appears in the same format as the memory address on bus lines A_{19}-A_0. The address lines A_{19}-A_{16} are zero in I/O operations. The variable I/O instructions which use register DX as a pointer have full address capability while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space.

I/O ports are addressed in the same manner as memory locations. Even addressed bytes are transferred on the D_7-D_0 bus lines and odd addressed bytes on D_{15}-D_8. Care must be taken to assure that each register within an 8-bit peripheral located on the lower portion of the bus be addressed as even.

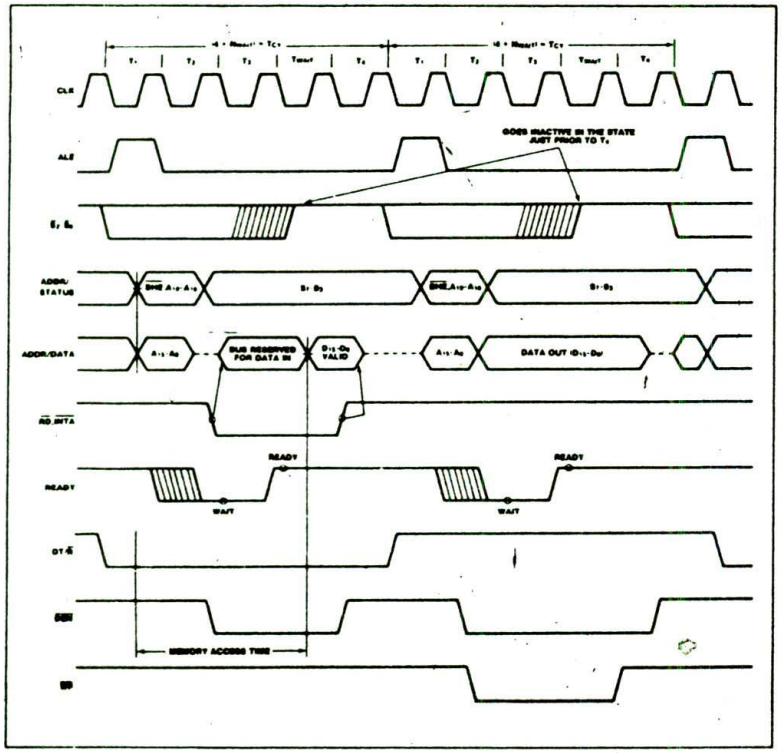


Figure 5. Basic System Timing

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 8086 RESET is required to be HIGH for greater than 4 CLK cycles. The 8086 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 10 CLK cycles. After this interval the 8086 operates normally beginning with the instruction in absolute location FFFF0H (see Figure 3B). The details of this operation are specified in the Instruction Set description of the MCS-86 Family User's Manual. The RESET input is internally synchronized to the processor clock. At initialization the HIGH-to-LOW transition of RESET must occur no sooner than 50 μs after power-up, to allow complete initialization of the 8086.

NMI may not be asserted prior to the 2nd CLK cycle following the end of RESET.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes, software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the Instruction Set description. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256-element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 3b), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type". An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to "vector" through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt. (See Instruction Set description.)

NMI is required to have a duration in the HIGH state of greater than two CLK cycles, but is not required to be synchronized to the clock. Any high-going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves of a block-type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 86/10 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable FLAG status bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block-type instruction. During the interrupt response sequence further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt or single-step), although the

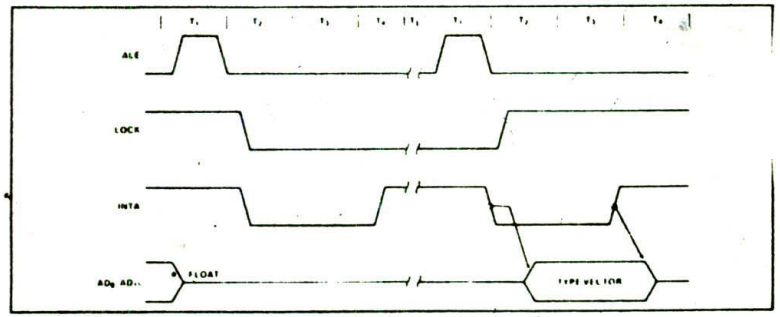


Figure 6. Interrupt Acknowledge Sequence

FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored the enable bit will be zero unless specifically set by an instruction.

During the response sequence (figure 6) the processor executes two successive (back-to-back) interrupt acknowledge cycles. The 8086 emits the LOCK signal from T_2 of the first bus cycle until T_2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle a byte is fetched from the external interrupt system (e.g., 8259A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The INTERRUPT RETURN instruction includes a FLAGS pop which returns the status of the original interrupt enable bit when it restores the FLAGS.

HALT

When a software "HALT" instruction is executed the processor indicates that it is entering the "HALT" state in one of two ways depending upon which mode is strapped. In minimum mode, the processor issues one ALE with no qualifying bus control signals. In Maximum Mode, the processor issues appropriate HALT status on $\overline{S}_2\overline{S}_1\overline{S}_0$ and the 8288 bus controller issues one ALE. The 8086 will not leave the "HALT" state when a local bus "hold" is entered while in "HALT". In this case, the processor reissues the HALT indicator. An interrupt request or RESET will force the 8086 out of the "HALT" state.

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The $\overline{LOCK}$ status information is provided by the processor when directly consecutive bus cycles are required during the execution of an instruction. This provides the processor with the capability of performing read/modify/write operations on memory (via the Exchange Register With Memory instruction, for example) without the possibility of another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{LOCK}$ signal is activated (forced LOW) in the clock cycle following the one in which the software "LOCK" prefix instruction is decoded by the EU. It is deactivated at the end of the last bus cycle of the instruction following the "LOCK" prefix instruction. While $\overline{LOCK}$ is active a request on a RQ/GT pin will be recorded and then honored at the end of the LOCK.

EXTERNAL SYNCHRONIZATION VIA TEST

As an alternative to the interrupts and general I/O capabilities, the 8086 provides a single software-testable input known as the $\overline{TEST}$ signal. At any time the program may execute a WAIT instruction. If at that time the $\overline{TEST}$ signal is inactive (HIGH), program execution becomes suspended while the processor waits for $\overline{TEST}$ to become active. It must remain active for at least 5 CLK cycles. The WAIT instruction is re-executed repeatedly until that time. This activity does not consume bus cycles. The processor remains in an idle state while waiting. All 8086 drivers go to 3-state OFF if bus "Hold" is entered. If interrupts are enabled, they may occur while the processor is waiting. When this occurs the processor fetches the WAIT instruction one extra time, processes the interrupt, and then re-fetches and re-executes the WAIT instruction upon returning from the interrupt.

BASIC SYSTEM TIMING

Typical system configurations for the processor operating in minimum mode and in maximum mode are shown in Figures 4a and 4b, respectively. In minimum mode, the MN/$\overline{MX}$ pin is strapped to V_{CC} and the processor emits bus control signals in a manner similar to the 8085. In maximum mode, the MN/$\overline{MX}$ pin is strapped to V_{SS} and the processor emits coded status information which the 8288 bus controller uses to generate MULTIBUS compatible bus control signals. Figure 5 illustrates the signal timing relationships.

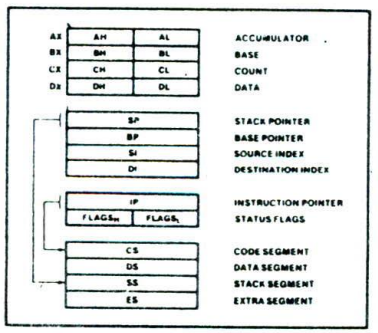


Figure 7. IAPX 86/10 Register Model

SYSTEM TIMING — MINIMUM SYSTEM

The read cycle begins in T_1 with the assertion of the Address Latch Enable (ALE) signal. The trailing (low-going) edge of this signal is used to latch the address information, which is valid on the local bus at this time, into the 8282/8283 latch. The $\overline{BHE}$ and A_0 signals address the low, high, or both bytes. From T_1 to T_4 the M/$\overline{IO}$ signal indicates a memory or I/O operation. At T_2 the address is removed from the local bus and the bus goes to a high impedance state. The read control signal is also asserted at T_2. The read ($\overline{RD}$) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver (8286/8287) is required to buffer the 8086 local bus, signals DT/$\overline{R}$ and $\overline{DEN}$ are provided by the 8086.

A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{IO}$ signal is again asserted to indicate a memory or I/O write operation. In the T_2 immediately following the address emission the processor emits the data to be written into the addressed location. This data remains valid until the middle of T_4. During T_2, T_3, and T_W the processor asserts the write control signal. The write ($\overline{WR}$) signal becomes active at the beginning of T_2 as opposed to the read which is delayed somewhat into T_2 to provide time for the bus to float.

The $\overline{BHE}$ and A_0 signals are used to select the proper byte(s) of the memory/IO word to be read or written according to the following table.

$\overline{BHE}$	A0	CHARACTERISTICS
0	0	Whole word
0	1	Upper byte from/ to odd address
1	0	Lower byte from/ to even address
1	1	None

I/O ports are addressed in the same manner as memory location. Even addressed bytes are transferred on the D_7–D_0 bus lines and odd addressed bytes on D_{15}–D_8.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge signal ($\overline{INTA}$) is asserted in place of the read ($\overline{RD}$) signal and the address bus is floated. (See Figure 6.) In the second of two successive INTA cycles, a byte of information is read from bus lines D_7–D_0 as supplied by the interrupt system logic (i.e., 8259A Priority Interrupt Controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into an interrupt vector lookup table, as described earlier.

BUS TIMING—MEDIUM SIZE SYSTEMS

For medium size systems the MN/$\overline{MX}$ pin is connected to V_{SS} and the 8288 Bus Controller is added to the system as well as an 8282/8283 latch for latching the system address, and a 8286/8287 transceiver to allow for bus loading greater than the 8086 is capable of handling. Signals ALE, DEN, and DT/$\overline{R}$ are generated by the 8288 instead of the processor in this configuration although their timing remains relatively the same. The 8086 status outputs ($\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$) provide type-of-cycle information and become 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence data isn't valid at the leading edge of write. The 8286/8287 transceiver receives the usual T and OE inputs from the 8288's DT/$\overline{R}$ and DEN.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an 8259A located on either the local bus or the system bus. If the master 8259A Priority Interrupt Controller is positioned on the local bus, a TTL gate is required to disable the 8286/8287 transceiver when reading from the master 8259A during the interrupt acknowledge sequence and software "poll".

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . – 65°C to + 150°C
Voltage on Any Pin with
Respect to Ground . . . . . – 1.0 to + 7V
Power Dissipation . . . . . 2.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

(8086: T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)
(8086-1: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)
(8086-2: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	– 0.5	+ 0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = – 400 µA
I_{CC}	Power Supply Current: 8086 8086-1 8086-2		340 360 350	mA	T_A = 25°C
I_{LI}	Input Leakage Current		± 10	µA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		± 10	µA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	– 0.5	+ 0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0 – AD_{15}, $\overline{RQ}$/GT)		15	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD_0 – AD_{15}, $\overline{RQ}$/GT)		15	pF	fc = 1 MHz

A.C. CHARACTERISTICS

(8086: T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)
(8086-1: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)
(8086-2: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (See Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	-8		-10		-8		ns	
THVCH	HOLD Setup Time	35		20		20		ns	
TINVCH	INTR, NMI, $\overline{TEST}$ Setup Time (See Note 2)	30		15		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

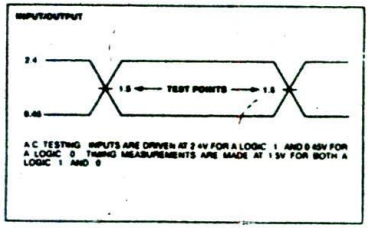
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

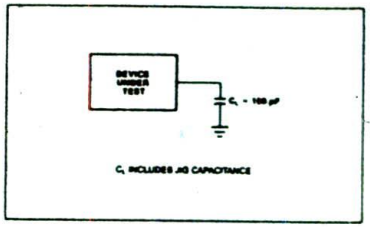
Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH - 20		TCLCH - 10		TCLCH-10		ns	
TCLLH	ALE Active Delay		80		40		50	ns	
TCHLL	ALE Inactive Delay		85		45		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL - 10		TCHCL - 10		TCHCL - 10		ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	*C_L = 20-100 pF for all 8086 Outputs (In addition to 8086 self-load)
TCHDX	Data Hold Time	10		10		10		ns	
TWHDX	Data Hold Time After WR	TCLCH - 30		TCLCH - 25		TCLCH - 30		ns	
TCVCTV	Control Active Delay 1	10	110	10	50	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	45	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	50	10	70	ns	
TAZRL	Address Float to READ Active	0		0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	70	10	100	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	80	10	80	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL - 45		TCLCL - 35		TCLCL - 40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	80	10	100	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL - 75		2TCLCL - 40		2TCLCL - 50		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL - 60		2TCLCL - 35		2TCLCL - 40		ns	
TAVAL	Address Valid to ALE Low	TCLCH - 60		TCLCH - 35		TCLCH - 40		ns	
TOLOH	Output Rise Time		20		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

NOTES:
1 Signal at 8284A shown for reference only
2 Setup requirement for asynchronous signal only to guarantee recognition at next CLK
3 Applies only to T2 state (8 ns into T3)

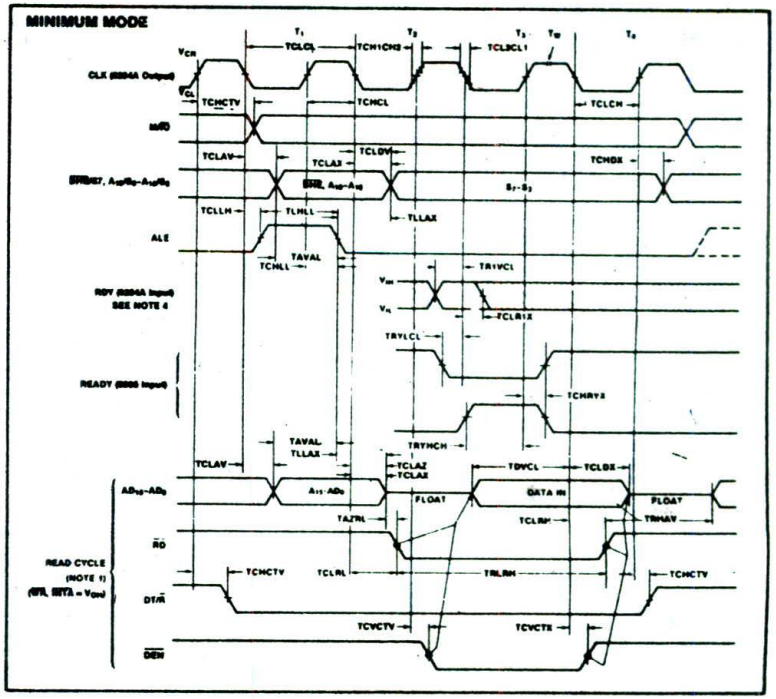
A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT

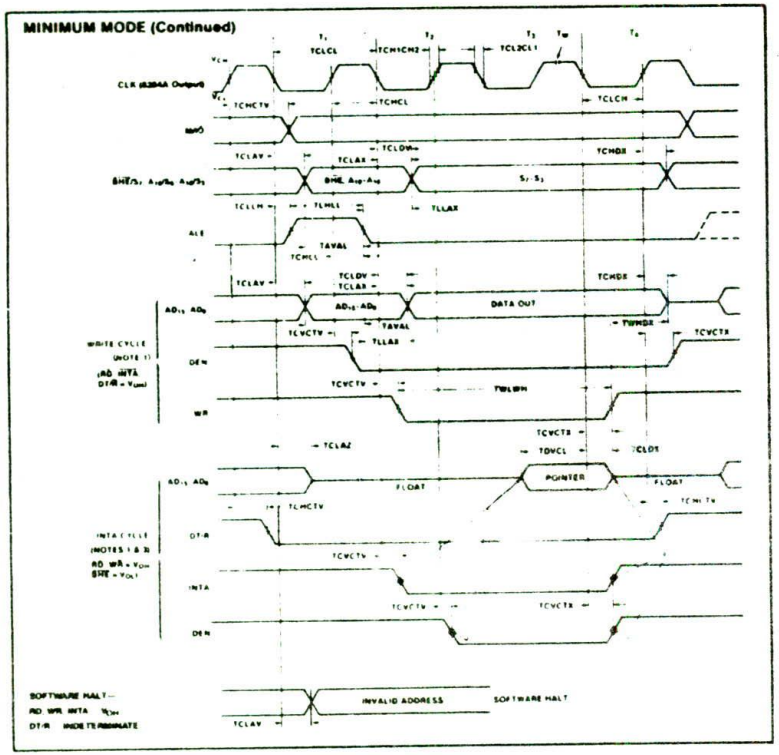


WAVEFORMS



WAVEFORMS (Continued)

MINIMUM MODE (Continued)



NOTES

1. All signals switch between V_{OH} and V_{OL} unless otherwise specified
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted
3. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle
4. Signals at 8284A are shown for reference only
5. All timing measurements are made at 1.5V unless otherwise noted

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)
TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2 (Preliminary)		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	68		30		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.8V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (See Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 4)	-8		-10		-8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (See Note 2)	30		15		15		ns	
TGVCH	AD/DT Setup Time	30		12		15		ns	
TCHGX	AD Hold Time into 8086	40		20		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

NOTES:

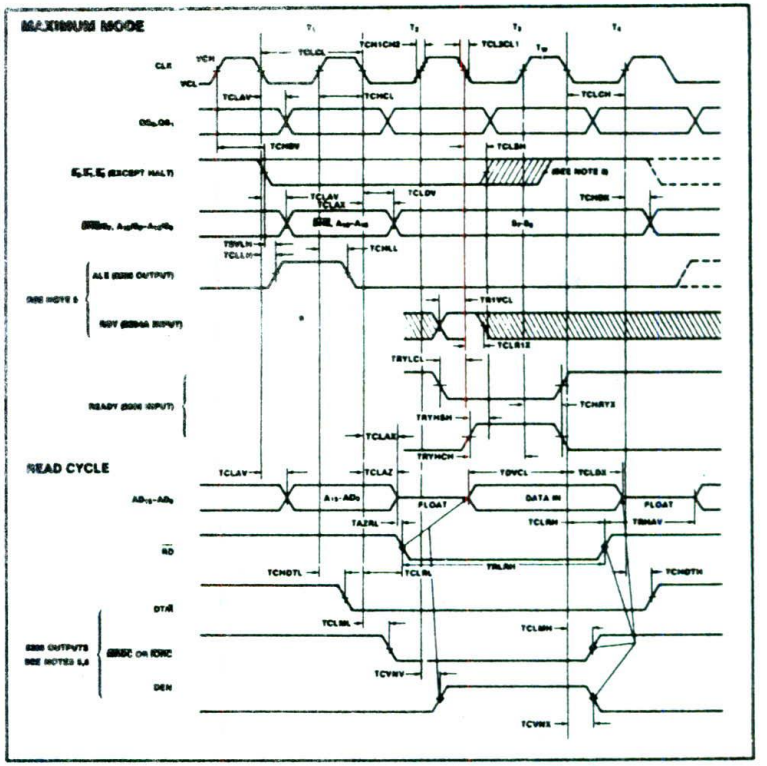
1. Signal at 8284A or 8288 shown for reference only
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK
3. Applies only to T3 and wait states.
4. Applies only to T2 state (8 ns into T3)

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

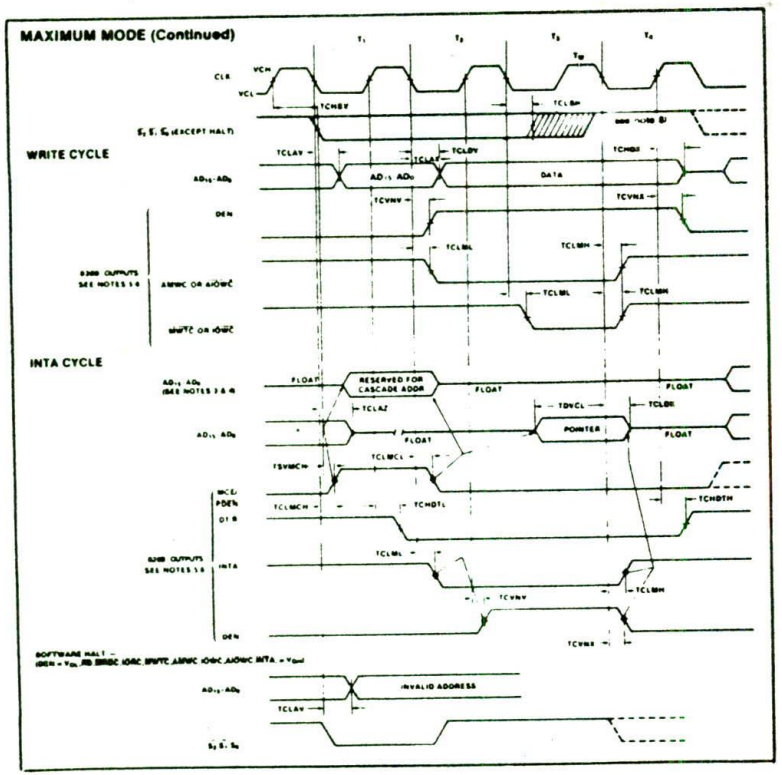
Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2 (Preliminary)		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLML	Command Active Delay (See Note 1)	10	35	10	35	10	35	ns	
TCLMH	Command Inactive Delay (See Note 1)	10	35	10	35	10	35	ns	
TRYHSH	READY Active to Status Passive (See Note 3)		110		45		53	ns	
TCHSV	Status Active Delay	10	110	10	45	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	55	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15		15		15	ns	
TSVMCH	Status Valid to MCE High (See Note 1)		15		15		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15		15	ns	
TCLMCH	CLK Low to MCE High (See Note 1)		15		15		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15		15		15	ns	C_L = 20-150 pF for all 8086 Outputs (in addition to 8086 self-load)
TCLMCL	MCE Inactive Delay (See Note 1)		15		15		15	ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	
TCHDX	Data Hold Time	10		10		10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	10	45	ns	
TAZRL	Address Float to Read Active	0		0		0		ns	
TCLRL	RD Active Delay	10	165	10	70	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	80	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL - 45		TCLCL - 35		TCLCL - 40		ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30		30		30	ns	
TCLGL	GT Active Delay	0	85	0	45	0	50	ns	
TCLGH	GT Inactive Delay	0	85	0	45	0	50	ns	
TRLRH	RD Width	2TCLCL - 75		2TCLCL - 40		2TCLCL - 50		ns	
TOLOH	Output Rise Time		20		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

WAVEFORMS



WAVEFORMS (Continued)

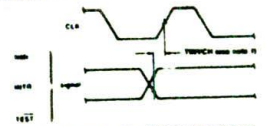
MAXIMUM MODE (Continued)



NOTES:
1. All signals switch between V_{OH} and V_{OL} unless otherwise specified
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted
3. Cascade address is valid between first and second INTA cycle
4. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle
5. Signals at 8284A or 8288 are shown for reference only
6. The issuance of the 8288 command and control signals ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{INTA}$ and DEN) lags the active high 8288 CEN
7. All timing measurements are made at 1.5V unless otherwise noted
8. Status inactive in state just prior to T_4

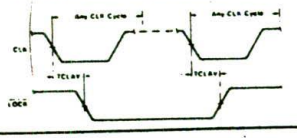
WAVEFORMS (Continued)

ASYNCHRONOUS SIGNAL RECOGNITION

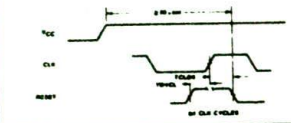


NOTE 1: SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNAL IS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

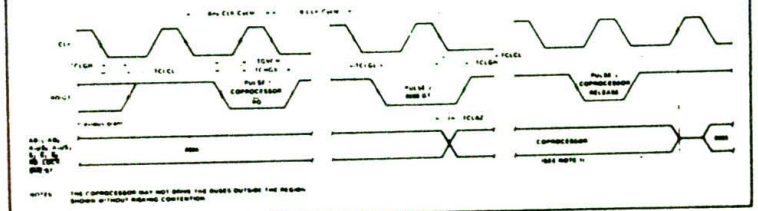
BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



RESET TIMING



REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)

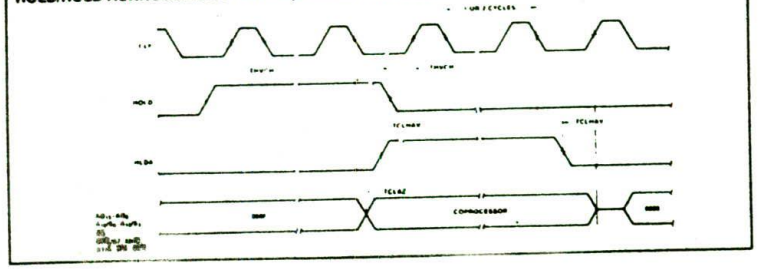


Table 2. Instruction Set Summary

1 Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010dw	mod reg r/m		
Immediate to Register/Memory	1100011w	mod 000 r/m	data	data if w = 1
Immediate to Register	1011w reg	data	data if w = 1	
Memory to Accumulator	1010000w	addr-low	addr-high	
Accumulator to Memory	1010001w	addr-low	addr-high	
Register/Memory to Segment Register	10001110	mod 0 reg r/m		
Segment Register to Register/Memory	10001100	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	11111111	mod 110 r/m		
Register	01010 reg			
Segment Register	000 reg 110			
POP = Pop:				
Register/Memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment Register	000 reg 111			
XCHG = Exchange:				
Register/Memory with Register	1000011w	mod reg r/m		
Register with Accumulator	10010 reg			
IN = Input from:				
Fixed Port	1110010w	port		
Variable Port	1110110w			
OUT = Output to:				
Fixed Port	1110011w	port		
Variable Port	1110111w			
XLAT = Translate Byte to AL	11010111			
LEA = Load EA to Register	10001101	mod reg r/m		
LDS = Load Pointer to DS	11000101	mod reg r/m		
LES = Load Pointer to ES	11000100	mod reg r/m		
LAHF = Load AH with Flags	10011111			
SAHF = Store AH into Flags	10011110			
PUSHF = Push Flags	10011100			
POPF = Pop Flags	10011101			
ARITHMETIC				
ADD = Add:				
Reg./Memory with Register to Either	000000dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod 000 r/m	data	data if s w = 01
Immediate to Accumulator	0000010w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	000100dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod 010 r/m	data	data if s w = 01
Immediate to Accumulator	0001010w	data	data if w = 1	
INC = Increment:				
Register/Memory	1111111w	mod 000 r/m		
Register	01000 reg			

2 Mnemonic and Description	Instruction Code			
	76543210	76543210	76543210	76543210
AAA = ASCII Adjust for Add	00110111			
DAA = Decimal Adjust for Add	00100111			
SUB = Subtract:				
Reg./Memory and Register to Either	001010dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod 101 r/m	data	data if s w = 01
Immediate from Accumulator	0010110w	data	data if w = 1	
SSB = Subtract with Borrow				
Reg./Memory and Register to Either	000110dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod 011 r/m	data	data if s w = 01
Immediate from Accumulator	0001111w	data	data if w = 1	
DEC = Decrement:				
Register/memory	1111111w	mod 001 r/m		
Register	01001 reg			
NEG = Change sign	1111011w	mod 011 r/m		
CMP = Compare:				
Register/Memory and Register	001110dw	mod reg r/m		
Immediate with Register/Memory	100000sw	mod 111 r/m	data	data if s w = 01
Immediate with Accumulator	0011110w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	00111111			
DAS = Decimal Adjust for Subtract	00101111			
MUL = Multiply (Unsigned)	1111011w	mod 100 r/m		
IMUL = Integer Multiply (Signed)	1111011w	mod 101 r/m		
AAM = ASCII Adjust for Multiply	11010100	00001010		
DIV = Divide (Unsigned)	1111011w	mod 110 r/m		
IDIV = Integer Divide (Signed)	1111011w	mod 111 r/m		
AAD = ASCII Adjust for Divide	11010101	00001010		
CBW = Convert Byte to Word	10011000			
CWD = Convert Word to Double Word	10011001			
LOGIC				
NOT = Invert	1111011w	mod 010 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	110100vw	mod 100 r/m		
SHR = Shift Logical Right	110100vw	mod 101 r/m		
SAR = Shift Arithmetic Right	110100vw	mod 111 r/m		
ROL = Rotate Left	110100vw	mod 000 r/m		
ROR = Rotate Right	110100vw	mod 001 r/m		
RCL = Rotate Through Carry Flag Left	110100vw	mod 010 r/m		
RCR = Rotate Through Carry Right	110100vw	mod 011 r/m		
AND = And:				
Reg./Memory and Register to Either	001000dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 100 r/m	data	data if w = 1
Immediate to Accumulator	0010010w	data	data if w = 1	
TEST = And Function to Flags, No Result:				
Register/Memory and Register	1000010w	mod reg r/m		
Immediate Data and Register/Memory	1111011w	mod 000 r/m	data	data if w = 1
Immediate Data and Accumulator	1010100w	data	data if w = 1	

3 Mnemonic and Description	Instruction Code			
	76543210	76543210	76543210	76543210
OR = Or:				
Reg./Memory and Register to Either	000010dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 001 r/m	data	data if w = 1
Immediate to Accumulator	0000110w	data	data if w = 1	
XOR = Exclusive or:				
Reg./Memory and Register to Either	001100dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 110 r/m	data	data if w = 1
Immediate to Accumulator	0011010w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1111001z			
MOVS = Move Byte/Word	1010010w			
CMPS = Compare Byte/Word	1010011w			
SCAS = Scan Byte/Word	1010111w			
LODS = Load Byte/Wd to AL/AX	1010110w			
STOS = Stor Byte/Wd from AL/A	1010101w			
CONTROL TRANSFER				
CALL = Call:				
Direct within Segment	11101000	disp-low	disp-high	
Indirect within Segment	11111111	mod 010 r/m		
Direct Intersegment	10011010	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	11111111	mod 011 r/m		
JMP = Unconditional Jump:				
Direct within Segment	11101001	disp-low	disp-high	
Direct within Segment-Short	11101011	disp		
Indirect within Segment	11111111	mod 100 r/m		
Direct Intersegment	11101010	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	11111111	mod 101 r/m		
RET = Return from CALL:				
Within Segment	11000011			
Within Seg Adding Immed to SP	11000010	data-low	data-high	
Intersegment	11001011			
Intersegment Adding Immediate to SP	11001010	data-low	data-high	
JE/JZ = Jump on Equal/Zero	01110100	disp		
JL/JNGE = Jump on Less/Not Greater or Equal	01111100	disp		
JLE/JNG = Jump on Less or Equal/ Not Greater	01111110	disp		
JB/JNAE = Jump on Below/Not Above or Equal	01110010	disp		
JBE/JNA = Jump on Below or Equal/ Not Above	01110110	disp		
JP/JPE = Jump on Parity/Parity Even	01111010	disp		
JO = Jump on Overflow	01110000	disp		
JS = Jump on Sign	01111000	disp		
JNE/JNZ = Jump on Not Equal/Not Zero	01110101	disp		
JNL/JGE = Jump on Not Less/Greater or Equal	01111101	disp		
JNLE/JG = Jump on Not Less or Equal/ Greater	01111111	disp		

4 Mnemonic and Description	Instruction Code			
	76543210	76543210	76543210	76543210
JNB/JAE = Jump on Not Below/Above or Equal	01110011	disp		
JNBE/JA = Jump on Not Below or Equal/Above	01110111	disp		
JNP/JPO = Jump on Not Par/Par Odd	01111011	disp		
JNO = Jump on Not Overflow	01110001	disp		
JNS = Jump on Not Sign	01111001	disp		
LOOP = Loop CX Times	11100010	disp		
LOOPZ/LOOPE = Loop While Zero/Equal	11100001	disp		
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	11100000	disp		
JCXZ = Jump on CX Zero	11100011	disp		
INT = Interrupt				
Type Specified	11001101	type		
Type 3	11001100			
INTO = Interrupt on Overflow	11001110			
IRET = Interrupt Return	11001111			
PROCESSOR CONTROL				
CLC = Clear Carry	11111000			
CMC = Complement Carry	11110101			
STC = Set Carry	11111001			
CLD = Clear Direction	11111100			
STD = Set Direction	11111101			
CLI = Clear Interrupt	11111010			
STI = Set Interrupt	11111011			
HLT = Halt	11110100			
WAIT = Wait	10011011			
ESC = Escape (to External Device)	11011xxx	mod x x x r/m		
LOCK = Bus Lock Prefix	11110000			

NOTES:
AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction
if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
if mod = 10 then DISP = disp-high; disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)
*except if mod = 00 and r/m = 110 then EA = disp-high; disp-low.

if s w = 01 then 16 bits of immediate data form the operand
if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

001 reg 110

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:
FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

APPENDIX B

8086/8088 Instructions

Notes for 8086/8088 Instructions

The individual instruction descriptions are shown by a format box such as the following:

These are byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- Opcode is the 8-bit opcode for the instruction. The actual opcode generated is defined in the "Opcode" column of the instruction table that follows each format box.
- m/op/r/m is the byte that specifies the operands of the instruction. It contains a 2-bit mode field (m), a 3-bit register field (op), and a 3-bit register or memory (r/m) field.
- Dashed blank boxes following the m/op/r/m box are for any displacement required by the mode field.
- Data is for a byte of immediate data.
- A dashed blank box following a Data box is used whenever the immediate operand is a word quantity.

Operand Summary

"reg" field Bit Assignments:

Word Operand	Byte Operand	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Second Instruction Byte Summary

mod	Displacement
00	DISP = 0*, disp-low and disp-high are absent
01	DISP = disp-low sign-extended to 16-bits, disp-high is absent
10	DISP = disp-high: disp-low
11	r/m is treated as a "reg" field

r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP*
111	(BX) + DISP

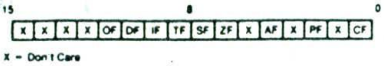
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low

Flags

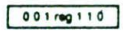
AF: AUXILIARY CARRY — BCD
CF: CARRY FLAG
DF: DIRECTION FLAG (STRINGS)
IF: INTERRUPT ENABLE FLAG
OF: OVERFLOW FLAG (CF ≠ SF)
PF: PARITY FLAG
SF: SIGN FLAG
TF: TRAP (SINGLE STEP FLAG)
ZF: ZERO FLAG

Instructions that reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file



X = Don't Care

Segment Override Prefix



Timing: 2 clocks

Use of Segment Override

Operand Register	Default	With Override Prefix
IP (code address)	CS	Never
SP (stack address)	SS	Never
BP (stack address or stack marker)	SS	BP + DS or ES, or CS
SI or DI (not incl. strings)	DS	ES, SS, or CS
SI (implicit source addr for strings)	DS	ES, SS, or CS
DI (implicit dest addr for strings)	ES	Never

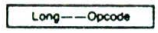
Operand Address (EA) Timing (Clocks):

Add 4 clocks for word operands at ODD ADDRESSES
Immed Offset = 6
Base (BX, BP, SI, DI) = 5
Base + DISP = 9
Base + Index (BP + DI, BX + SI) = 7
Base + Index (BP + SI, BX + DI) = 8
Base + Index (BP + DI, BX + SI) + DISP = 11
Base + Index (BP + SI, BX + DI) + DISP = 12

AAA = ASCII Adjust for Addition

Opcode	Clocks	Operation
37	4	adjust AL, flags, AH

AAD = ASCII Adjust for Division



Opcode	Clocks	Operation
D5.0A	60	Adjust AL, AH prior to division

AAM = ASCII Adjust for Multiplication

Opcode	Clocks	Operation
D4.0A	83	Adjust AL, AH after multiplication

AAS = ASCII Adjust for Subtraction

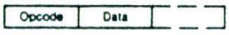
Opcode	Clocks	Operation
3F	4	adjust AL, flags, AH

ADC = Integer Add with Carry

Memory/Reg + Reg

	Opcode	Clocks	Operation
Byte	12	3	Reg8 ← CF + Reg8 + Reg8
	12	9 + EA	Reg8 ← CF + Reg8 + Mem8
	10	16 + EA	Mem8 ← CF + Mem8 + Reg8
Word	13	3	Reg16 ← CF + Reg16 + Reg16
	13	9 + EA	Reg16 ← CF + Reg16 + Mem16
	11	16 + EA	Mem16 ← CF + Mem16 + Reg16

Immed to AX/AL



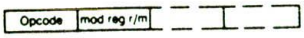
	Opcode	Clocks	Operation
Byte	14	4	AL ← CF + AL + Immed8
Word	15	4	AX ← CF + AX + Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← CF + Reg8 + Immed8
	80	17 + EA	Mem8 ← CF + Mem8 + Immed8
Word	81	4	Reg16 ← CF + Reg16 + Immed16
	81	17 + EA	Mem16 ← CF + Mem16 + Immed16
	83	4	Reg16 ← CF + Reg16 + Immed8
	83	17 + EA	Mem16 ← CF + Mem16 + Immed8

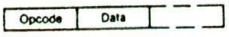
ADD = Integer Addition

Memory/Reg + Reg



	Opcode	Clocks	Operation
Byte	02	3	Reg8 ← Reg8 + Reg8
	02	9 + EA	Reg8 ← Reg8 + Mem8
	00	16 + EA	Mem8 ← Mem8 + Reg8
Word	03	3	Reg16 ← Reg16 + Reg16
	03	9 + EA	Reg16 ← Reg16 + Mem16
	01	16 + EA	Mem16 ← Mem16 + Reg16

Immed to AX/AL



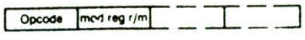
Opcode	Clocks	Operation
04	4	AL ← AL + Immed8
05	4	AX ← AX + Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 + Immed8
	80	17 + EA	Mem8 ← Mem8 + Immed8
Word	81	4	Reg16 ← Reg16 + Immed16
	81	17 + EA	Mem16 ← Mem16 + Immed16
	83	4	Reg16 ← Reg16 + Immed8
	83	17 + EA	Mem16 ← Mem16 + Immed8

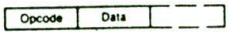
AND = Logical AND

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	22	3	Reg8 ← Reg8 AND Reg8
	22	9 + EA	Reg8 ← Reg8 AND Mem8
	20	16 + EA	Mem8 ← Mem8 AND Reg8
Word	23	3	Reg16 ← Reg16 AND Reg16
	23	9 + EA	Reg16 ← Reg16 AND Mem16
	21	16 + EA	Mem16 ← Mem16 AND Reg16

Immed to AX/AL



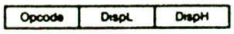
	Opcode	Clocks	Operation
Byte	24	4	AL ← AL AND Immed8
Word	25	4	AX ← AX AND Immed16

Immed to Memory Reg

	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 AND Immed8
	80	17 + EA	Mem8 ← Mem8 AND Immed8
Word	81	4	Reg16 ← Reg16 AND Immed16
	81	17 + EA	Mem16 ← Mem16 AND Immed16

CALL = Call

Within segment or group, IP relative

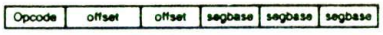


Opcode	Clocks	Operation
E8	19	IP ← IP + Disp16—(SP) ← return link

Within segment or group, Indirect

	Opcode	Clocks	Operation
	FF	16	IP ← Reg16—(SP) ← return link
	FF	21 + EA	IP ← Mem16—(SP) ← return link

Inter-segment or group, Direct



Opcode	Clocks	Operation
9A	28	CS ← segbase IP ← offset

Inter-segment or group, Indirect

Opcode	Clocks	Operation
FF	37 + EA	CS ← segbase IP ← offset

CBW = Convert Byte to Word

Opcode	Clocks	Operation
98	2	convert byte in AL to word in AX

CLC = Clear Carry Flag

Opcode	Clocks	Operation
F8	2	clear the carry flag

CLD = Clear Direction Flag

Opcode	Clocks	Operation
FC	2	clear direction flag

CLI = Clear Interrupt Enable Flag

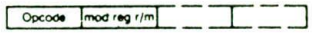
Opcode	Clocks	Operation
FA	2	clear interrupt flag

CMC = Complement Carry Flag

Opcode	Clocks	Operation
F5	2	complement carry flag

CMP = Compare Two Operands

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	38	3	flags ← Reg8 - Reg8
	38	9 + EA	flags ← Reg8 - Mem8
	3A	9 + EA	flags ← Mem8 - Reg8
Word	39	3	flags ← Reg16 - Reg16
	39	9 + EA	flags ← Reg16 - Mem16
	3B	9 + EA	flags ← Mem16 - Reg16

Immed to AX/AL

	Opcode	Clocks	Operation
Byte	3C	4	flags AL - Immed8
Word	3D	4	flags AX - Immed16

Immed to Memory/Reg

	Opcode	Clocks	Operation
Byte	80	4	flags ← Reg8 - immed8
	80	10 + EA	flags ← Mem8 - immed8
Word	81	4	flags ← Reg16 - immed16
	81	10 + EA	flags ← Mem16 - immed16
	83	4	flags ← Reg16 - immed8
	83	10 + EA	flags ← Mem16 - immed8

CWD = Convert Word to Doubleword

Opcode	Clocks	Operation
99	5	coovert word in AX to doubleword in DX AX

DAA = Decimal Adjust for Addition

Opcode	Clocks	Operation
27	4	adjust AL, flags, AH

DAS = Decimal Adjust for Subtraction

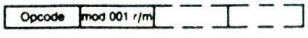
Opcode	Clocks	Operation
2F	4	adjust AL, flags, AH

DEC = Decrement by 1

Word Register

Opcode	Clocks	Operation
48 + reg	2	Reg16 ← Reg16 - 1

Memory/Byte Register



	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 - 1
	FE	15 + EA	Mem8 ← Mem8 - 1
Word	FF	15 + EA	Mem16 ← Mem16 - 1

DIV = Unsigned Division

Memory/Reg with AX or DX:AX

	Opcode	Clocks	Operation
Byte	F6	80-90	AH,AL ← AX / Reg8
	F6	(86-96) + EA	AH,AL ← AX / Mem8
Word	F7	144-162	DX,AX ← DX AX / Reg16
	F7	(150-168) + EA	DX,AX ← DX AX / Mem16

ESC = Escape

Opcode	Clocks	Operation
D8 + i	8 + EA	data bus ← (EA)
D8 + i	2	data bus ← (EA)

HLT = Halt

Opcode	Clocks	Operation
F4	2	halt operation

IDIV = Signed Division

Memory/Reg with AX or DX:AX

	Opcode	Clocks	Operation
Byte	F6	101-112	AH,AL ← AX / Reg8
	F6	(107-118) + EA	AH,AL ← AX / Mem8
Word	F7	165-184	DX,AX ← DX AX / Reg16
	F7	(171-190) + EA	DX,AX ← DX AX / Mem16

IMUL = Signed Multiplication

Memory/Reg with AL or AX

	Opcode	Clocks	Operation
Byte	F6	80-98	AX ← AL*Reg8
	F6	(86-104) + EA	AX ← AL*Mem8
Word	F7	128-154	DX AX ← AX*Reg16
	F7	(134-160) + EA	DX AX ← AX*Mem16

IN = Input Byte, Word

Fixed port

	Opcode	Clocks	Operation
Byte	E4	10	AL ← Port8
	E5	10	AX ← Port8

Variable port

	Opcode	Clocks	Operation
Word	EC	8	AL ← Port16(in DX)
	ED	8	AX ← Port16(in DX)

INC = Increment by 1

Word Register

Opcode	Clocks	Operation
40 + reg	2	Reg16 ← Reg16 + 1

Memory/Byte Register

	Opcode	Clocks	Operation
Byte	FE	3	Reg8 ← Reg8 + 1
	FE	15 + EA	Mem8 ← Mem8 + 1
Word	FF	15 + EA	Mem16 ← Mem16 + 1

INT INTO = Interrupt

Opcode	Clocks	Operation
CC	52	Interrupt 3
CD	51	Interrupt type
CE	53 or 4	Interrupt4 if FLAGS OF = 1 else NOP

IRET = Return from Interrupt

Opcode	Clocks	Operation
CF	24	Return from interrupt

Jcond = Jump on Condition

Operation

```
if condition is true then do
  sign-extend displacement to 16 bits;
  IP ← IP + sign-extended displacement;
end if;
```

Format

Opcode	Clocks	Operation	cond =
77	16 or 4	jump if above	JA
73	16 or 4	jump if above or equal	JAE
72	16 or 4	jump if below	JB
76	16 or 4	jump if below or equal	JBE
72	16 or 4	jump if carry set	JC
74	16 or 4	jump if equal	JE
7F	16 or 4	jump if greater	JG
7D	16 or 4	jump if greater or equal	JGE
7C	16 or 4	jump if less	JL
7E	16 or 4	jump if less or equal	JLE
76	16 or 4	jump if not above	JNA
72	16 or 4	jump if neither above nor equal	JNAE
73	16 or 4	jump if not below	JNB
77	16 or 4	jump if neither below nor equal	JNBE
73	16 or 4	jump if no carry	JNC
75	16 or 4	jump if not equal	JNE
7E	16 or 4	jump if not greater	JNG
7C	16 or 4	jump if neither greater nor equal	JNGE
7D	16 or 4	jump if not less	JNL
7F	16 or 4	jump if neither less nor equal	JNLE
71	16 or 4	jump if no overflow	JNO
7B	16 or 4	jump if no parity	JNP
79	16 or 4	jump if positive	JNS
75	16 or 4	jump if not zero	JNZ
70	16 or 4	jump if overflow	JO
7A	16 or 4	jump if parity	JP
7A	16 or 4	jump if parity even	JPE
7B	16 or 4	jump if parity odd	JPO
78	16 or 4	jump if sign	JS
74	18 or 6	jump if zero	JZ
E3	18 or 6	jump if CX is zero (does not test flags)	JCXZ

JMP = Jump

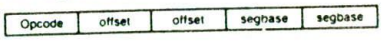
Within segment or group, IP relative

Opcode	Clocks	Operation
E9	15	IP ← IP + Disp16
EB	15	IP ← IP + Disp8 (Disp8 sign-extended)

Within segment or group, Indirect

Opcode	Clocks	Operation
FF	11	IP ← Reg16
FF	18 + EA	IP ← Mem16

Inter-segment or group, Direct



Opcode	Clocks	Operation
EA	15	CS ← segbase IP ← offset

Inter-segment or group, Indirect

Opcode	Clocks	Operation
FF	24 + EA	CS ← segbase IP ← offset

LAHF = Load AH from Flags

Opcode	Clocks	Operation
9F	4	copy low byte of flags word to AH

LDS/LES = Load Pointer to DS/ES and Register

Opcode	Clocks	Operation
C4	16 + EA	dword pointer at EA goes to reg16 (1st word) and ES (2nd word)
C5	16 + EA	dword pointer at EA goes to reg16 (1st word) and DS (2nd word)

LEA = Load Effective Address

Opcode	Clocks	Operation
8D	2 + EA	Reg16 ← EA

LOCK = Assert Bus Lock

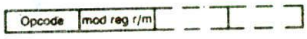
Opcode	Clocks	Operation
F0	2	assert the bus lock next instruction

LOOPxx = Loop Control

Opcode	Clocks	Operation	xx =
E1	18 or 6	dec CX, loop if equal and CX not 0	LOOPE
E0	19 or 5	dec CX, loop if not equal and CX not 0	LOOPNE
E1	18 or 6	dec CX, loop if zero and CX not 0	LOOPZ
E0	19 or 5	dec CX, loop if not zero and CX not 0	LOOPNZ
E2	17 or 5	dec CX, loop if CX not 0	LOOP

MOV = Move Data

Memory/Reg to or from Reg

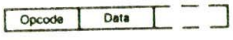


	Opcode	Clocks	Operation
Byte	88	9 + EA	Mem8 ← Reg8
	88	2	Reg8 ← Reg8
	8A	8 + EA	Reg8 ← Mem8
Word	89	9 + EA	Mem16 ← Reg16
	89	2	Reg16 ← Reg16
	8B	8 + EA	Reg16 ← Mem16

Direct-Addressed Memory to or from AX/AL

	Opcode	Clocks	Operation
Byte	A0	10	AL ← Mem8
	A2	10	Mem8 ← AL
Word	A1	10	AX ← Mem16
	A3	10	Mem16 ← AX

Immed to Reg



	Opcode	Clocks	Operation
Byte	B0 + reg	4	Reg 8 ← Immed8
Word	B8 + reg	4	Reg16 ← Immed16

Immed to Memory/Reg

Opcode	Clocks	Operation
C6	4	Reg8 ← Immed8
C6	10 + EA	Mem8 ← Immed8
C7	4	Reg16 ← Immed16
C7	10 + EA	Mem16 ← Immed16

Memory/Reg to or from SReg

	Opcode	Clocks	Operation
Word	8C	9 + EA	Mem16 ← SReg
	8C	2	Reg16 ← SReg
	8E	8 + EA	SReg ← Mem16
	8E	2	SReg ← Reg16

MUL = Unsigned Multiplication

Memory/Reg with AL or AX

	Opcode	Clocks	Operation
Byte	F6	70-77	AX ← AL*Reg8
	F6	(76-83) + EA	AX ← AL*Mem8
Word	F7	118-133	DX AX ← AX*Reg16
	F7	(124-139) + EA	DX AX ← AX*Mem16

NEG = Negate an Integer

Memory/Reg

	Opcode	Clocks	Operation
	F6	3	Reg8 ← 00H - Reg 8
	F7	3	Reg16 ← 0000H - Reg16
	F6	13 + EA	Mem8 ← 00H - Mem8
	F7	16 + EA	Mem16 ← 0000H - Mem16

NOP = No Operation

Opcode	Clocks	Operation
90	3	no operation

NOT = Form One's Complement

Memory/Reg

	Opcode	Clocks	Operation
Byte	F6	3	Reg8 ← 0FFH - Reg8
	F6	16 + EA	Mem8 ← 0FFH - Mem8
Word	F7	3	Reg16 ← 0FFFFH - Reg16
	F7	16 + EA	Mem16 ← 0FFFFH - Mem16

OR = Logical Inclusive OR

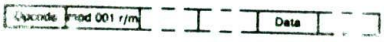
Memory/Reg with Reg

	Opcode	Clocks	Operation
Byte	0A	3	Reg8 ← Reg8 OR Reg8
	0A	9 + EA	Reg8 ← Reg8 OR Mem8
	08	16 + EA	Mem8 ← Mem8 OR Reg8
Word	0B	3	Reg16 ← Reg16 OR Reg 16
	0B	9 + EA	Reg16 ← Reg16 OR Mem16
	09	16 + EA	Mem16 ← Mem16 OR Reg16

Immed to AX/AL

Opcode	Clocks	Operation
0C	4	AL ← AL OR immed8
0D	4	AX ← AX OR immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 OR immed8
	80	17 + EA	Mem8 ← Mem8 OR immed8
Word	81	4	Reg16 ← Reg16 OR immed16
	81	17 + EA	Mem16 ← Mem16 OR immed16

OUT = Output Byte, Word

Fixed port

	Opcode	Clocks	Operation
Byte	E6	10	Port8 ← AL
	E7	10	Port8 ← AX

Variable port

	Opcode	Clocks	Operation
Word	EE	8	Port16 (in DX) ← AL
	EF	8	Port16 (in DX) ← AX

POP = Pop a Word from the Stack

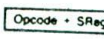
Word Memory

Opcode	Clocks	Operation
8F	17 + EA	Mem16 ← (SP) + +

Word Register

Opcode	Clocks	Operation
58 + reg	8	Reg16 ← (SP) + +

Segment Register



Opcode	Clocks	Operation
07 + SReg	8	SReg ← (SP) + +

POPF = Pop the TOS into the Flags

Opcode	Clocks	Operation
9D	8	FLAGS ← (SP) + +

PUSH = Push a Word onto the Stack

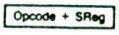
Memory/Reg

Opcode	Clocks	Operation
FF	16 + EA	—(SP) ← Mem16

Word Register

Opcode	Clocks	Operation
50 + reg	11	—(SP) ← Reg16

Segment Register



Opcode	Clocks	Operation
06 + SReg	10	—(SP) ← SReg

PUSHF = Push the Flags to the Stack

Opcode	Clocks	Operation
9C	10	—(SP) ← FLAGS

RCL = Rotate Left Through Carry

Memory or Reg by 1

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg 8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg 16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + EA + 4/bit	rotate Mem8 by CL
Word	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

RCR = Rotate Right Through Carry

Memory or Reg by 1

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + EA + 4/bit	rotate Mem8 by CL
Word	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

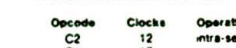
REP*x* = Repeat Prefix

Opcode	Clocks	Operation	REP*x* =
F3	2	repeat next instruction until CX = 0	REP
F3	2	repeat next instruction until CX = 0 or ZF = 0	REPE
F2	2	repeat next instruction until CX = 0 or ZF = 1	REPZ REPNE REPNZ

RET = Return from Subroutine

	Opcode	Clocks	Operation
	C3	8	intra-segment return
	CB	18	inter-segment return

Return and add constant to SP



	Opcode	Clocks	Operation
	C2	12	intra-segment ret and add
	CA	17	inter-segment ret and add

ROL = Rotate Left

Memory or Reg by 1

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL

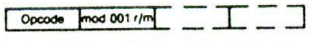
	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + Ea + 4/bit	rotate Mem8 by CL
Word	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

ROR = Rotate Right

Memory or Reg by 1

	Opcode	Clocks	Operation
Byte	D0	2	rotate Reg8 by 1
	D0	15 + EA	rotate Mem8 by 1
Word	D1	2	rotate Reg16 by 1
	D1	15 + EA	rotate Mem16 by 1

Memory or Reg by count in CL



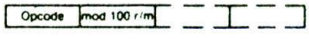
	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	rotate Reg8 by CL
	D2	20 + EA + 4/bit	rotate Mem8 by CL
	D3	8 + 4/bit	rotate Reg16 by CL
	D3	20 + EA + 4/bit	rotate Mem16 by CL

SAHF = Store AH in Flags

Opcode	Clocks	Operation
9E	4	copy AH to low byte of flags word

SAL/SHL = Arithmetic/Logical Left Shift

Memory or Reg by 1



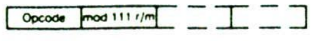
	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 + EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 + EA	shift Mem16 by 1

Memory or Reg by count in CL

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	shift Reg8 by CL
	D2	20 + EA + 4/bit	shift Mem8 by CL
Word	D3	8 + 4/bit	shift Reg16 by CL
	D3	20 + EA + 4/bit	shift Mem16 by CL

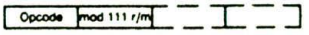
SAR = Arithmetic Right Shift

Memory or Reg by 1



	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 + EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 + EA	shift Mem16 by 1

Memory or Reg by count in CL



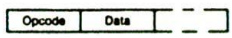
	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	shift Reg8 by CL
	D2	20 + EA + 4/bit	shift Mem8 by CL
Word	D3	8 + 4/bit	shift Reg16 by CL
	D3	20 + EA + 4/bit	shift Mem16 by CL

SBB = Integer Subtraction with Borrow

Memory/Reg with Reg

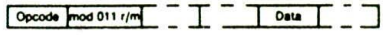
	Opcode	Clocks	Operation
Byte	1A	3	Reg8 ← Reg8 - Reg8 - CF
	1A	9 + EA	Reg8 ← Reg8 - Mem8 - CF
	18	16 + EA	Mem8 ← Mem8 - Reg8 - CF
Word	1B	3	Reg16 ← Reg16 - Reg16 - CF
	1B	9 + EA	Reg16 ← Reg16 - Mem16 - CF
	19	16 + EA	Mem16 ← Mem16 - Reg16 - CF

Immed from AX/AL



Opcode	Clocks	Operation
1C	4	AL ← AL - Immed8 - CF
1D	4	AX ← AX - Immed16 - CF

Immed from Memory/Reg



Opcode	Clocks	Operation
80	4	Reg8 ← Reg8 - Immed8 - CF
80	17 + EA	Mem8 ← Mem8 - Immed8 - CF
81	4	Reg16 ← Reg16 - Immed16 - CF
81	17 + EA	Mem16 ← Mem16 - Immed16 - CF
83	4	Reg16 ← Reg16 - Immed8 - CF
83	17 + EA	Mem16 ← Mem16 - Immed8 - CF (immed8 is sign-extended before subtract)

SHR = Logical Right Shift

Memory or Reg by 1

	Opcode	Clocks	Operation
Byte	D0	2	shift Reg8 by 1
	D0	15 + EA	shift Mem8 by 1
Word	D1	2	shift Reg16 by 1
	D1	15 + EA	shift Mem16 by 1

Memory or Reg by count in CL

	Opcode	Clocks	Operation
Byte	D2	8 + 4/bit	shift Reg8 by CL
	D2	20 + Ea + 4/bit	shift Mem8 by CL
Word	D3	8 + 4/bit	shift Reg16 by CL
	D3	20 + EA + 4/bit	shift Mem16 by CL

STC = Set Carry Flag

Opcode	Clocks	Operation
F9	2	set the carry flag

STD = Set Direction Flags

Opcode	Clocks	Operation
FD	2	set direction flag

STI = Set Interrupt Enable Flag

Opcode	Clocks	Operation
FB	2	set interrupt flag

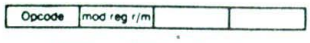
String = String Operations

Opcode	Clocks	Operation	*String* =
A6	22	flags ← (SI) - (DI)	CMPS

A7	22	flags ← (SI) - (DI)	CMPS
A4	18	(DI) ← (SI)	MOVS
A5	18	(DI) ← (SI)	MOVS
AE	15	flags ← (DI) - AL	SCAS
AF	15	flags ← (DI) - AX	SCAS
AC	12	AL ← (SI)	LODS
AD	12	AX ← (SI)	LODS
AA	11	(DI) ← AL	STOS
AB	11	(DI) ← AX	STOS

SUB = Integer Subtraction

Memory/Reg with Reg



	Opcode	Clocks	Operation
Byte	2A	3	Reg8 ← Reg8 - Reg8
	2A	9 + EA	Reg8 ← Reg8 - Mem8
	28	16 + EA	Mem8 ← Mem8 - Reg8
Word	2B	3	Reg16 ← Reg16 - Reg16
	2B	9 + EA	Reg16 ← Reg16 - Mem16
	29	16 + EA	Mem16 ← Mem16 - Reg16

Immed to AX/AL

	Opcode	Clocks	Operation
Byte	2C	4	AL ← AL - Immed8
Word	2D	4	AX ← AX - Immed16

Immed to Memory/Reg

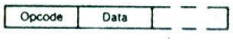
	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 - Immed8
	80	17 + EA	Mem8 ← Mem8 - Immed8
Word	81	4	Reg16 ← Reg16 - Immed16
	81	17 + EA	Mem16 ← Mem16 - Immed16
	83	4	Reg16 ← Reg16 - Immed8
	83	17 + EA	Mem16 ← Mem16 - Immed8

TEST = Logical Compare

Memory/Reg with Reg

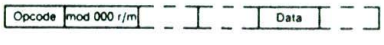
	Opcode	Clocks	Operation
Byte	84	3	flags ← Reg8 AND Reg8
	84	9 + EA	flags ← Reg8 AND Mem8
Word	85	3	flags ← Reg16 AND Reg16
	85	9 + EA	flags ← Reg16 AND Mem16

Immed to AX/AL



	Opcode	Clocks	Operation
Byte	A8	4	flags ← AL AND Immed8
Word	A9	4	flags ← AX AND Immed16

Immed to Memory/Reg



	Opcode	Clocks	Operation
Byte	F6	5	flags ← Reg8 AND Immed8
	F6	11 + EA	flags ← Mem8 AND Immed8
Word	F7	5	flags ← Reg16 AND Immed16
	F7	11 + EA	flags ← Mem16 AND Immed16

WAIT = Wait While TEST Pin Not Asserted

Opcode	Clocks	Operation
9B	3 + 5n	none

XCHG = Exchange Memory/Register with Register

Memory/Reg with Reg

	Opcode	Clocks	Operation
Byte	86	4	Reg8 ↔ Reg8
	86	17 + EA	Mem8 ↔ Reg8
Word	87	4	Reg16 ↔ Reg16
	87	17 + EA	Mem16 ↔ Reg16

Word Register with AX

Opcode	Clocks	Operation
90 + Reg	3	AX ↔ Reg16

XLAT = Table Look-up Translation
XLATB = Table Look-up Translation

Opcode	Clocks	Operation
D7	11	replace AL with table entry

XOR = Logical Exclusive OR

Memory/Reg with Reg

	Opcode	Clocks	Operation
Byte	32	3	Reg8 ← Reg8 XOR Reg8
	32	9 + EA	Reg8 ← Reg8 XOR Mem8
	30	16 + EA	Mem8 ← Mem8 XOR Reg8
Word	33	3	Reg16 ← Reg16 XOR Reg16
	33	9 + EA	Reg16 ← Reg16 XOR Mem16
	31	16 + EA	Mem16 ← Mem16 XOR Reg16

Immed to AX/AL

	Opcode	Clocks	Operation
	34	4	AL ← AL XOR Immed8
	35	4	AX ← AX XOR Immed16

Immed to Memory/Reg

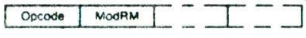
	Opcode	Clocks	Operation
Byte	80	4	Reg8 ← Reg8 XOR Immed8
	80	17 + EA	Mem8 ← Mem8 XOR Immed8
Word	81	4	Reg16 ← Reg16 XOR Immed16
	81	17 + EA	Mem16 ← Mem16 XOR Immed16

186 INSTRUCTIONS

Notes for iAPX 186 Instructions

These instructions can be used only if the MOD186 control is specified. When MOD186 is specified, clocks for all instructions are as stated under "Clocks for MOD186 Operation."

BOUND = Check Array Against Bounds



Opcode	Operation
62	if Reg16 < Mem16 at EA, or Reg16 > Mem16 at EA + 2 then INTERRUPT 5

ENTER = High Level Procedure Entry

Opcode	Operation
C8	build new stack frame

IMUL = Signed Multiplication

Mem/Reg* Immediate to Reg

Opcode	Operation
6B	Reg 16 ← Reg 16 * Immed 8
6B	Reg 16 ← Reg 16 * Immed 8
6B	Reg 16 ← Mem 16 * Immed 8
69	Reg 16 ← Reg 16 * Immed 16
69	Reg 16 ← Reg 16 * Immed 16
69	Reg 16 ← Mem 16 * Immed 16

LEAVE = High Level Procedure Exit

Opcode	Operation
C9	release current stack frame and return to prior frame

POPA = Pop All Registers

Opcode	Operation
61	restore registers from stack

PUSH = Push a Word onto the Stack

Word Immediate

Opcode	Data			

Opcode	Operation
6A	—(SP) ← Immed8 (sign extended)
68	—(SP) ← Immed16

PUSHA = Push All Registers

Opcode

Opcode	Operation
60	save registers on the stack

RCL = Rotate Left Through Carry

Mem or Reg by Immed8

Opcode	ModRM*						count

*—(Reg field = 011)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

RCR = Rotate Right Through Carry

Mem or Reg by Immed8

Opcode	ModRM*						count

*—(Reg field = 011)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

ROL = Rotate Left

Mem or Reg by Immed8

Opcode	ModRM*						count

*—(Reg field = 000)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

ROR = Rotate Right

Mem or Reg by Immed8

Opcode	ModRM*						count

*—(Reg field = 001)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SAL/SHL = Arithmetic/Logical Left Shift

Mem or Reg by immediate count

Opcode	ModRM*						count

*—(Reg field = 100)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SAR = Arithmetic Right Shift

Mem or Reg by Immed8

Opcode	ModRM*						count

*—(Reg field = 111)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

SHR = Logical Right Shift

Mem or Reg by Immed8

Opcode	ModRM*						count

*—(Reg field = 101)

Opcode	Operation
C0	rotate Reg8 by Immed8
C0	rotate Mem8 by Immed8
C1	rotate Reg16 by Immed8
C1	rotate Mem16 by Immed8

String = String Operations (INS/OUTS)

Opcode

Opcode	Clocks	Operation
6E	INS	(DI) ← port(DX)
6F	INS	(DI) ← port(DX DX + 1)
6C	OUTS	port(DX) ← (SI)
6D	OUTS	port(DX DX + 1) ← (SI)

8087 INSTRUCTIONS

Notes for 8087 Instructions

The individual instruction descriptions are shown by a format box such as the following:

WAIT	op1	m/op/r/m	addr1	addr2

These are the byte-wise representations of the object code generated by the assembler and are interpreted as follows:

- WAIT is an 8086 wait instruction, NOP or emulator instruction.
- op1 is the opcode, possibly taking two bytes.
- m/op/r/m byte (middle 3-bits is part of the opcode).
- addr1 and addr2 are offsets of either 8 or 16 bits.

For integer functions, m = 0 for short-integer memory operand; 1 for word-integer memory operand.
For real functions, m = 0 for short-real memory operand, 1 for longreal memory operand
i = stack element index
If mod = 00 then DISP = 0 ; disp-lo and disp-hi are absent.
If mod = 01 then DISP = disp-lo sign-extended to 16 bits, disp-hi is absent.
If mod = 10 then DISP = disp-hi; disp-lo.
If mod = 11 then r/m is treated as an ST(i) field.

If r/m = 000 then EA = (BX) + (SI) + DISP
If r/m = 001 then EA = (BX) + (DI) + DISP
If r/m = 010 then EA = (BP) + (SI) + DISP
If r/m = 011 then EA = (BP) + (DI) + DISP
If r/m = 100 then EA = (SI) + DISP
If r/m = 101 then EA = (DI) + DISP
If r/m = 110 then EA (BP) + DISP*
If r/m = 111 then EA = (BX) + DISP
*Except if mod = 000 and r/m = 110 then EA = disp-hi; disp-lo.

ST(0) = Current stack top
ST(i) = i^th register below stack top
d = Destination
0 — Destination is ST(0)
1 — Destination is ST(i)
P = Pop
0 — No pop
1 — Pop ST(0)
R = Reverse
0 — Destination (op) source
1 — Source (op) destination

For FSQRT	$-0 \leq ST(0) \leq +\infty$
For FSCALE	$-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
For F2XM1	$0 \leq ST(0) \leq 2$
For FYL2X	$0 \leq ST(0) < \infty$ $-\infty < ST(1) < +\infty$
For FYL2XP1	$0 < \lvert ST(0) \rvert < (2 - \sqrt{2})/2$ $-\infty < ST(1) < \infty$
For FPTAN	$0 \leq ST(0) < \pi/4$
For FPATAN	$0 \leq ST(0) < ST(1) < +\infty$

F2XMI = Compute $2^x - 1$

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F0	CD 19 F0	500 310-630	ST ← 2^{ST} - 1

FABS = Absolute Value

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E1	CD 19 E1	14 10-17	ST ← |ST|

FADD = Add Real

Stack top ← Stack element

WAIT	op1	op2 ← i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 C0+i	CD 18 C0+i	85 70-100	ST ← ST + ST(i)
9B DC C0+i	CD 1C C0+i	85 70-100	ST(i) ← ST + ST(i)

Stack top + memory operand

WAIT	op1	mod 000 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m0rm	CD 18 m0rm	105 + EA (90-120) + EA	ST ← ST + mem-op (short-real)
9B DC m0rm	CD 1C m0rm	110 + EA (95-125) + EA	ST ← ST + mem-op (long-real)

FADDP = Add Real and Pop

Stack top + Stack Element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE C1	CD 1E C1	90 75-105	ST(1) ← ST + ST(1) pop stack
9B DE C0+i	CD 1E C0+i	90 75-105	ST(i) ← ST + ST(i) pop stack

FBLD = Packed Decimal (BCD) Load

WAIT	op1	mod 100 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DF m4rm	CD 1F m4rm	300 + EA (290-310) + EA	push stack ST ← mem-op

FBSTP = Packed Decimal (BCD) Store and Pop

WAIT	op1	mod 110 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DF m6rm	CD 1F m6rm	530 + EA (520-540) + EA	mem-op ← ST pop stack

FCHS = Change Sign

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E0	CD 19 E0	15 10-17	ST ← −ST

FCLEX FNCLEX = Clear Exceptions

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB E2	CD 1B E2	5 2-8	clear 8087 exceptions
90 DB E2	CD 1B E2	5 2-8	clear 8087 exceptions (no wait)

FCOM = Compare Real

Compare Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 D1	CD 18 D1	45 40-50	ST — ST(1)
9B D8 D0+i	CD 18 D0+i	45 40-50	ST — ST(i)

Compare Stack top and memory operands

WAIT	op1	mod 010 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m2rm	CD 18 m2rm	65 + EA (60-70) + EA	ST — memop (short-real)
9B DC m2rm	CD 1C m2rm	70 + EA (65-75) + EA	ST — memop (long-real)

FCOMP = Compare Real and Pop

Compare Stack top and Stack element and pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 D9	CD 18 D9	47 42-52	ST — ST(1) pop stack
9B D8 D8+i	CD 18 D8+i	47 42-52	ST — ST(i) pop stack

Compare Stack top and memory operand and pop

WAIT	op1	mod 011 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m3rm	CD 18 m3rm	68 + EA (63-73) + EA	ST — mem-op pop stack (short-real)
9B DC m3rm	CD 1C m3rm	72 + EA (67-77) + EA	ST — mem-op pop stack (long-real)

FCOMPP = Compare Real and Pop Twice

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE D9	CD 1E D9	50 45-55	ST — ST(1) pop stack pop stack

FDECSTP = Decrement Stack Pointer

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F6	CD 19 F6	9 6-12	stack pointer ← stack pointer − 1

FDISI FNDISI = Disable Interrupts

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DB E1	CD 1B E1	5 2-8	Set 8087 interrupt mask
90 DB E1	CD 1B E1	5 2-8	Set 8087 interrupt mask (no wait)

FDIV = Divide Real

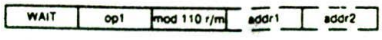
Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 F0+i	CD 18 F0+i	198 193-203	ST ← ST/ST(i)

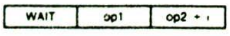
9B DC F8+i	CD 1C F8+i	196 193-203	ST(i) ← ST(i)/ST

Stack top and memory operand



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D8 m6rm	CD 18 m6rm	220 + EA (215-225) + EA	ST ← ST/mem-op (short-real)
9B DC m6rm	CD 1C m6rm	225 + EA (220-230) + EA	ST ← ST/mem-op (long-real)

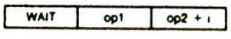
FDIVP = Divide Real and Pop



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DE F9	CD 1E F9	202 197-207	ST(1) ← ST(1)/ST pop stack
9B DE F8+i	CD 1E F8+i	202 197-207	ST(i) ← ST(i)/ST pop stack

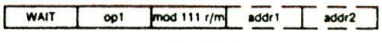
FDIVR = Divide Real Reversed

Stack top and Stack element



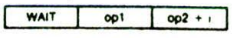
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D8 F8+i	CD 18 F8+i	199 194-204	ST ← ST(i)/ST
9B DC F0+i	CD 1C F0+i	199 194-204	ST(i) ← ST/ST(i)

Stack top and memory operand



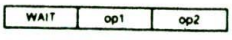
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D8 m7rm	CD 18 m7rm	221 + EA (216-226) + EA	ST ← mem-op/ST (short-real)
9B DC m7rm	CD 1C m7rm	226 + EA (221-231) + EA	ST ← mem-op/ST (long-real)

FDIVRP = Divide Real Reversed and Pop



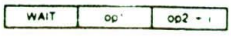
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DE F1	CD 1E F1	203 198-208	ST(1) ← ST/ST(1) pop stack
9B DE F0+i	CD 1E F0+i	203 198-208	ST(i) ← ST/ST(i)

FENI
FNENI = Enable Interrupts



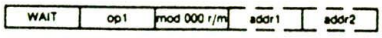
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DB E0	CD 1B E0	5 2-8	clear 8087 interrupt mask
90 DB E0	CD 1B E0	5 2-8	clear 8087 interrupt mask (no wait)

FFREE = Free Register



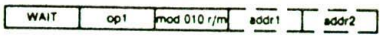
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DD C0+i	CD 1D C0+i	11 9-16	TAG(i) masked empty

FIADD = Integer Add



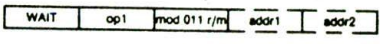
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m0rm	CD 1A m0rm	125 + EA (108-143) + EA	ST ← ST + mem-op (short integer)
9B DE m0rm	CD 1E m0rm	120 + EA (102-137) + EA	ST ← ST + mem-op (word integer)

FICOM = Integer Compare



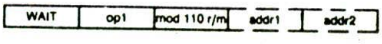
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m2rm	CD 1A m2rm	85 + EA (78-91) + EA	ST — mem-op (short integer)
9B DE m2rm	CD 1E m2rm	80 + EA (72-86) + EA	ST — mem-op (word integer)

FICOMP = Integer Compare and Pop



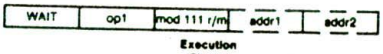
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m3rm	CD 1A m3rm	87 + EA (80-93) + EA	ST — mem-op pop stack (short integer)
9B DE m3rm	CD 1E m3rm	82 + EA (74-88) + EA	ST — mem-op pop stack (word integer)

FIDIV = Integer Divide



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m6rm	CD 1A m6rm	238 + EA (230-243) + EA	ST ← ST/mem-op (short integer)
9B DE m6rm	CD 1E m6rm	230 + EA (224-238) + EA	ST ← ST/mem-op (word integer)

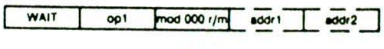
FIDIVR = Integer Divide Reversed



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m7rm	CD 1A m7rm	237 + EA (231-245) + EA	ST ← mem-op/ST (short integer)
9B DE m7rm	CD 1E m7rm	230 + EA (225-239) + EA	ST ← mem-op/ST (word integer)

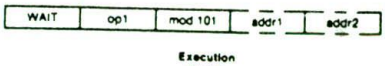
FILD = Integer Load

Word Integer or Short Integer



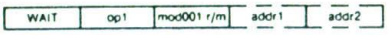
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DB m0rm	CD 1B m0rm	56 + EA (52-60) + EA	push stack ST ← mem-op (short integer)
9B DF m0rm	CD 1F m0rm	50 + EA (46-54) + EA	push stack ST ← mem-op (word integer)

Long Integer



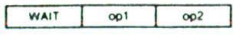
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DF m5rm	CD 1F m5rm	64 + EA (60-68) + EA	push stack ST ← mem-op (long integer)

FIMUL = Integer Multiply



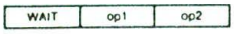
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m1rm	CD 1A m1rm	136 + EA (130-144) + EA	ST ← ST * mem-op (short integer)
9B DE m1rm	CD 1E m1rm	130 + EA (124-138) + EA	ST ← ST * mem-op (word integer)

FINCSTP = Increment Stack Pointer



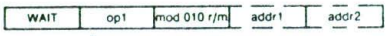
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 F7	CD 19 F7	9 6-12	stack pointer ← stack pointer + 1

FINIT
FNINIT = Initialize Processor



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DB E3	CD 1B E3	5 2-8	initialize 8087
90 DB E3	CD 1B E3	5 2-8	initialize 8087 (no wait)

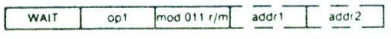
FIST = Integer Store



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DB m2rm	CD 1B m2rm	88 - EA (82-92) - EA	mem-op ← ST (short integer)
9B DF m2rm	CD 1F m2rm	86 - EA (80-90) - EA	mem-op ← ST (word integer)

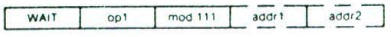
FISTP = Integer Store and Pop

Short Integer or Word Integer



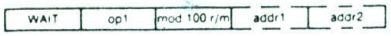
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DB m3rm	CD 1B m3rm	90 - EA (84-94) - EA	mem-op ← ST pop stack (short integer)
9B DF m3rm	CD 1F m3rm	88 - EA (82-92) - EA	mem-op ← ST pop stack (word integer)

Long Integer



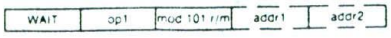
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DF m7rm	CD 1F m7rm	100 - EA (94-105) - EA	mem-op ← ST pop stack (long integer)

FISUB = Integer Subtract



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m4rm	CD 1A m4rm	125 - EA (108-143) - EA	ST ← ST − mem-op (short integer)
9B DE m4rm	CD 1E m4rm	120 - EA (102-137) - EA	ST ← ST − mem-op (word integer)

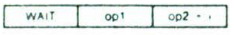
FISUBR = Integer Subtract Reversed



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DA m5rm	CD 1A m5rm	125 - EA (109-144) - EA	ST ← mem-op − ST (short integer)
9B DE m5rm	CD 1E m5rm	120 - EA (103-139) - EA	ST ← mem-op − ST (word integer)

FLD = Load Real

Stack element to Stack top



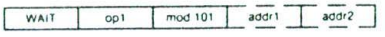
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 C0 + i	CD 19 C0 + i	20 17-22	T ← ST(i) push stack ST ← T

Memory operand to Stack top

Short Integer or Long Integer

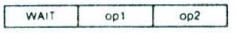
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 m0rm	CD 19 m0rm	43 - EA (38-56) - EA	push stack ST ← mem-op (short integer)
9B DD m0rm	CD 1D m0rm	46 - EA (40-60) - EA	push stack ST ← mem-op (long integer)

Temp Real



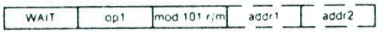
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DB m5rm	CD 1B m5rm	57 - EA (53-65) - EA	push stack ST ← mem-op (temp real)

FLD1 = Load + 1.0



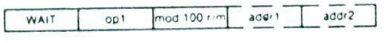
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 E8	CD 19 E8	18 15-21	push stack ST ← 1.0

FLDCW = Load Control Word



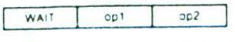
8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 m5rm	CD 19 m5rm	10 - EA (7-14) - EA	processor control word ← mem-op

FLDENV = Load Environment



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 m4rm	CD 19 m4rm	40 - EA (35-45) - EA	8087 environment ← mem-op

FLDL2E = Load $\log_2 e$



8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 EA	CD 19 EA	18 15-21	push stack ST ← $\log_2 e$

FLDL2T = Load $\log_2 10$

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E9	CD 19 E9	19 16-22	push stack ST ← log 10

FLDLG2 = Load $\log_{10}2$

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EC	CD 19 EC	21 18-24	push stack ST ← log ,2

FLDPI = Load π

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EB	CD 19 EB	19 16-22	push stack ST ← π

FLDZ = Load + 0.0

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 EE	CD 19 EE	14 11-17	push stack ST ← 0.0

FMUL = Multiply Real

Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 C8 + i	CD 18 C8 + i	138 130-145	ST ← ST * ST(i)
9B DC C8 + i	CD 1C C8 + i	138 130-145	ST(i) ← ST(i) — ST

Stack top and memory operand

WAIT	op1	mod 001 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D8 m1rm	CD 18 m1rm	118 + EA (110-125) + EA	ST ← ST * mem-op (short real)
9B DC m1rm	CD 1C m1rm	161 + EA (154-168) + EA	ST ← ST * mem-op (long real)

FMULP = Multiply Real and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DE C9 + i	CD 1E C9 + i	142 134-148	ST(i) ← ST(i) * ST pop stack

FNOP = No Operation

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 D0	CD 19 D0	13 10-16	ST ← ST

FPATAN = Partial Arctangent

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F3	CD 19 F3	650 250-800	T ← arctan (ST(1)/ST) pop stack ST ← T

FPREM = Partial Remainder

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F8	CD 19 F8	125 15-190	ST ← REPEAT (ST — ST(1))

FPTAN = Partial Tangent

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F2	CD 19 F2	450 30-540	Y/X ← TAN (ST) ST ← Y push stack ST ← X

FRNDINT = Round to Integer

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FC	CD 19 FC	45 16-50	ST ← nearest integer (ST)

FRSTOR = Restore Saved State

WAIT	op1	mod 100 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD m4rm	CD 1D m4rm	202 + EA (197-207) + EA	8087 state ← mem-op

FSAVE FNSAVE = Save State

WAIT	op1	mod 110 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B DD m6rm	CD 1D m6rm	202 + EA (197-207) + EA	mem-op ← 8087 state
90 DD m6rm	CD 1D m6rm	202 + EA (197-207) + EA	mem-op ← 8087 state (no wait)

FSCALE = Scale

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FD	CD 19 FD	35 32-38	ST ← ST * 2

FSQRT = Square Root

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 FA	CD 19 FA	183 180-186	ST ← √ ST

FST = Store Real

Stack top to Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DD D0+i	CD 1D D0+i	18 15-22	ST(i) ← ST

Stack top to memory operand

WAIT	op1	mod 010 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 m2rm	CD 19 m2rm	87 + EA (84-90) + EA	mem-op ← ST (short-real)
9B D8 m2rm	CD 1D m2rm	100 + EA (96-104) + EA	mem-op ← ST (long-real)

FSTCW
FNSTCW = Store Control Word

WAIT	op1	mod 111 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 m7rm	CD 19 m7rm	15 + EA (12-18) + EA	mem-op ← processor control word
9B D9 m7rm	CD 19 m7rm	15 + EA (12-18) + EA	mem-op ← processor control word (no wait)

FSTENV
FNSTENV = Store Environment

WAIT	op1	mod 110 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 m6rm	CD 19 m6rm	45 + EA (40-50) + EA	mem-op ← 8087 environment
9B D9 r6rm	CD 19 m6rm	45 + EA (40-50) + EA	mem-op ← 8087 environment (no wait)

FSTP = Store Real and Pop

Stack top to Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DD D8+i	CD 1D D8+i	20 17-24	ST(i) ← ST pop stack

Stack top to memory operand

WAIT	op1	mod 011 r/m	addr1	addr2

Long Real or Short Real

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D9 m3rm	CD 19 m3rm	89 + EA (86-92) + EA	mem-op ← ST pop stack (short-real)
9B DB m3rm	CD 1B m3rm	102 + EA (98-106) + EA	mem-op ← ST pop stack (long-real)

Temp Real

WAIT	op1	mod 111 r/m	disp-lo	disp-hi

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DD m7rm	CD 1D m7rm	55 + EA (52-58) + EA	mem-op ← ST pop stack (temp-real)

FSTSW
FNSTSW = Store Status Word

WAIT	op1	mod 111 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DD m7rm	CD 1D m7rm	15 + EA (12-18) + EA	mem-op ← 8087 status word
9B DD m7rm	CD 1D m7rm	15 + EA (12-18) + EA	mem-op ← 8087 status word (no wait)

FSUB = Subtract Real

Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D8 E0+i	CD 18 E0+i	85 70-100	ST ← ST − ST(i)
9B DC E8+i	CD 1C E8+i	85 70-100	ST(i) ← ST(i) − ST

Stack top and memory operand

WAIT	op1	mod 100 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D8 m4rm	CD 18 m4rm	105 + EA (90-120) + EA	ST ← ST − mem-op (short-real)
9B DC m4rm	CD 1C m4rm	110 + EA (95-125) + EA	ST ← ST − mem-op (long-real)

FSUBP = Subtract Real and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B DE E9	CD 1E E9	90 75-105	ST(1) ← ST(1) − ST pop stack
9B DE E8+i	CD 1E E8+i	90 75-105	ST(i) ← ST(i) − ST pop stack

FSUBR = Subtract Real Reversed

Stack top and Stack element

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D8 E8+i	CD D8 E8+i	87 70-100	ST ← ST(i) − ST
9B DC E0+i	CD 1C E0+i	87 70-100	ST(i) ← ST − ST(i)

Stack top and memory operand

WAIT	op1	mod 101 r/m	addr1	addr2

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation
9B D8 m5rm	CD 18 m5rm	105 + EA (90-120) + EA	ST ← mem-op − ST (short-real)
9B DC m5rm	CD 1C m5rm	110 + EA (95-125) + EA	ST ← mem-op − ST (long-real)

FSUBRP = Subtract Real Reversed and Pop

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks: Typical Range	Operation

9B DE E1	CD 1E E1	90 75-105	ST(1) ← ST — ST(1) pop stack
9B DE E0+i	CD 1E E0+i	90 75-105	ST(i) ← ST — ST(i) pop stack

FTST = Test Stack Top Against + 0.0

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E4	CD 19 E4	42 38-48	ST ← ST — 0.0

FWAIT = (CPU) Wait While 8087 Is Busy

WAIT

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B	90	3+5n 3+5n	8086 wait instruction

FXAM = Examine Stack Top

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 E5	CD 19 E5	17 12-23	set condition code

FXCH = Exchange Registers

WAIT	op1	op2 + i

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 C8	CD 19 C8	12 10-15	$T_0 \leftarrow ST(1)$ ST(1) ← ST $ST \leftarrow T_0$
9B D9 C8+i	CD 19 C8+i	12 10-15	$T_0 \leftarrow ST(i)$ ST(i) ← ST $ST \leftarrow T_0$

FXTRACT = Extract Exponent and Significand

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F4	CD 19 F4	50 27-55	$T_0 \leftarrow$ exponent (ST) $T_1 \leftarrow$ significand (ST) $ST \leftarrow T_0$ push stack $ST \leftarrow T_1$

FYL2X = Compute Y * Log_2 X

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F1	CD 19 F1	950 900-1100	$T_0 \leftarrow ST(1) * \log_2(ST)$ pop stack $ST \leftarrow T_0$

FYL2XP1 = Compute Y * Log_2 (X + 1)

WAIT	op1	op2

8087 Encoding	Emulator Encoding	Execution Clocks Typical Range	Operation
9B D9 F9	CD 19 F9	850 700-1000	$T_0 \leftarrow ST + 1$ $T_1 \leftarrow ST(1) * \log_2 T_0$ pop stack $ST \leftarrow T_1$

INDEX