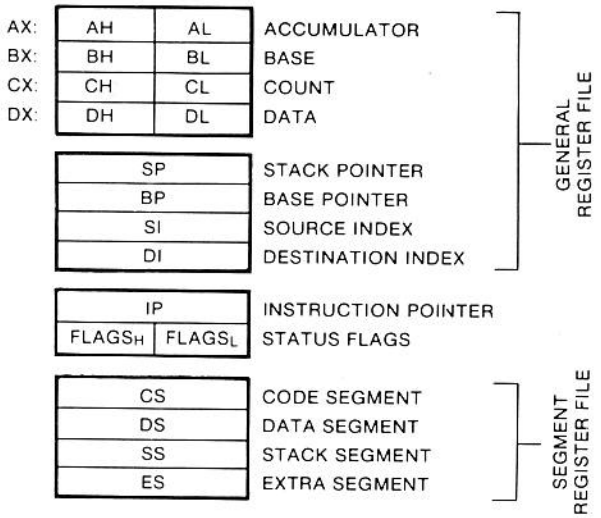
intel

MCS-86™ ASSEMBLY LANGUAGE REFERENCE GUIDE

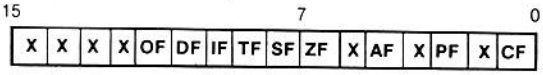
October 1978

9800749A

8086
REGISTER MODEL



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



X = Don't Care

AF: AUXILIARY CARRY — BCD
CF: CARRY FLAG
PF: PARITY FLAG
SF: SIGN FLAG
ZF: ZERO FLAG
— 8080 FLAGS

DF: DIRECTION FLAG (STRINGS)
IF: INTERRUPT ENABLE FLAG
OF: OVERFLOW FLAG (CF ⊕ SF)
TF: TRAP — SINGLE STEP FLAG
— 8086 FLAGS

OPERAND SUMMARY

"reg" field Bit Assignments:

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	

SECOND INSTRUCTION BYTE SUMMARY

mod	xxx	r/m

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

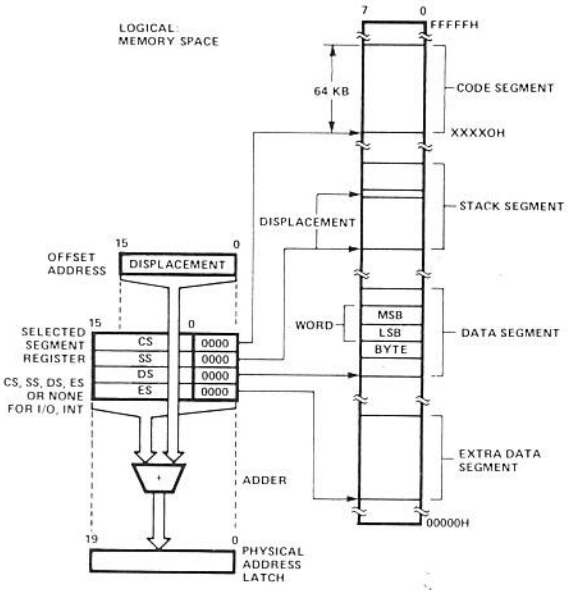
r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP*
111	(BX) + DISP

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

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

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

MEMORY SEGMENTATION MODEL



SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

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

DATA TRANSFER

MOV = Move

Register/memory to/from register

1 0 0 0 1 0 d w	mod reg r/m

Timing (clocks): register to register 2
memory to register 8+EA
register to memory 9+EA

Immediate to register/memory

1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w=1

Timing: 10+EA clocks

Immediate to register

1 0 1 1 w reg	data	data if w=1

Timing: 4 clocks

Memory to accumulator

1 0 1 0 0 0 0 w	addr-low	addr-high

Timing: 10 clocks

Accumulator to memory

1 0 1 0 0 0 1 w	addr-low	addr-high

Timing: 10 clocks

Register/memory to segment register

1 0 0 0 1 1 1 0	mod 0 reg r/m

Timing (clocks): register to register 2
memory to register 8+EA

Segment register to register/memory

1 0 0 0 1 1 0 0	mod 0 reg r/m

Timing (clocks): register to register 2
register to memory 9+EA

PUSH = Push

Register/memory

1 1 1 1 1 1 1 1	mod 1 1 0 r/m

Timing (clocks): register 10
memory 16+EA

Register

0 1 0 1 0 reg

Timing: 10 clocks

(Continued on following page)

Segment register

0 0 0 reg 1 1 0

Timing: 10 clocks

POP = Pop

Register/memory

1 0 0 0 1 1 1 1	mod 0 0 0 r/m

Timing (clocks): register 8
memory 17+EA

Register

0 1 0 1 1 reg

Timing: 8 clocks

Segment register

0 0 0 reg 1 1 1

Timing: 8 clocks

XCHG = Exchange

Register/memory with register

1 0 0 0 0 1 1 w	mod reg r/m

Timing (clocks): register with register 4
memory with register 17+EA

Register with accumulator

1 0 0 1 0 reg

Timing: 3 clocks

IN = Input to AL/AX from

Fixed port

1 1 1 0 0 1 0 w	port

Timing: 10 clocks

Variable port (DX)

1 1 1 0 1 1 0 w

Timing: 8 clocks

OUT = Output from AL/AX to

Fixed port

1 1 1 0 0 1 1 w	port

Timing: 10 clocks

Variable port (DX)

1 1 1 0 1 1 1 w

Timing: 8 clocks

XLAT = Translate byte to AL

1 1 0 1 0 1 1 1

Timing: 11 clocks

LEA = Load EA to register

1 0 0 0 1 1 0 1	mod reg r/m

Timing: 2+EA clocks

LDS = Load pointer to DS

1 1 0 0 0 1 0 1	mod reg r/m

Timing: 16+EA clocks

LES = Load pointer to ES

1 1 0 0 0 1 0 0	mod reg r/m

Timing: 16+EA clocks

LAHF = Load AH with flags

1 0 0 1 1 1 1 1

Timing: 4 clocks

SAHF = Store AH into flags

1 0 0 1 1 1 1 0

Timing: 4 clocks

PUSHF = Push flags

1 0 0 1 1 1 0 0

Timing: 10 clocks

POPF = Pop flags

1 0 0 1 1 1 0 1

Timing: 8 clocks

ARITHMETIC

ADD = Add

Reg./memory with register to either

0 0 0 0 0 0 d w	mod reg r/m

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w=01

Timing (clocks): immediate to register 4
immediate to memory 17+EA

(Continued on following page)

Immediate to accumulator

0 0 0 0 0 1 0 w	data	data if w=1

Timing: 4 clocks

ADC = Add with carry
Reg./memory with register to either

0 0 0 1 0 0 d w	mod reg r/m

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w=01

Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate to accumulator

0 0 0 1 0 1 0 w	data	data if w=1

Timing: 4 clocks

INC = Increment
Register/memory

1 1 1 1 1 1 1 w	mod 0 0 0 r/m

Timing (clocks): register 2
memory 15+EA

Register

0 1 0 0 0 reg

Timing: 2 clocks

AAA = ASCII adjust for add

0 0 1 1 0 1 1 1

Timing: 4 clocks

DAA = Decimal adjust for add

0 0 1 0 0 1 1 1

Timing: 4 clocks

SUB = Subtract
Reg./memory and register to either

0 0 1 0 1 0 d w	mod reg r/m

Timing (clocks): register from register 3
memory from register 9+EA
register from memory 16+EA

(Continued of following page)

Immediate from register/memory

1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w=01

Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate from accumulator

0 0 1 0 1 1 0 w	data	data if w=1

Timing: 4 clocks

SBB = Subtract with borrow
Reg./memory and register to either

0 0 0 1 1 0 d w	mod reg r/m

Timing (clocks): register from register 3
memory from register 9+EA
register from memory 16+EA

Immediate from register/memory

1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w=01

Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate from accumulator

0 0 0 1 1 1 0 w	data	data if w=1

Timing: 4 clocks

DEC = Decrement
Register/memory

1 1 1 1 1 1 1 w	mod 0 0 1 r/m

Timing (clocks): register 2
memory 15+EA

Register

0 1 0 0 1 reg

Timing: 2 clocks

NEG = Change sign

1 1 1 1 0 1 1 w	mod 0 1 1 r/m

Timing (clocks): register 3
memory 16+EA

CMP = Compare
Register/memory and register

0 0 1 1 1 0 d w	mod reg r/m

Timing (clocks): register with register 3
memory with register 9+EA
register with memory 9+EA

(Continued on following page)

Immediate with register/memory

1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w=01

Timing (clocks): immediate with register 4
immediate with memory 17+EA

Immediate with accumulator

0 0 1 1 1 1 0 w	data	data if w=1

Timing: 4 clocks

AAS = ASCII adjust for subtract

0 0 1 1 1 1 1 1

Timing: 4 clocks

DAS = Decimal adjust for subtract

0 0 1 0 1 1 1 1

Timing: 4 clocks

MUL = Multiply (unsigned)

1 1 1 1 0 1 1 w	mod 1 0 0 r/m

Timing (clocks): 8-bit 71+EA
16-bit 124+EA

IMUL = Integer multiply (signed)

1 1 1 1 0 1 1 w	mod 1 0 1 r/m

Timing (clocks): 8-bit 90+EA
16-bit 144+EA

AAM = ASCII adjust for multiply

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0

Timing: 83 clocks

DIV = Divide (unsigned)

1 1 1 1 0 1 1 w	mod 1 1 0 r/m

Timing (clocks): 8-bit 90+EA
16-bit 155+EA

IDIV = Integer divide (signed)

1 1 1 1 0 1 1 w	mod 1 1 1 r/m

Timing (clocks): 8-bit 112+EA
16-bit 177+EA

AAD = ASCII adjust for divide

1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0

Timing: 60 clocks

CBW = Convert byte to word

1 0 0 1 1 0 0 0

Timing: 2 clocks

CWD = Convert word to double word

1 0 0 1 1 0 0 1

Timing: 5 clocks

LOGIC

NOT = Invert

1 1 1 1 0 1 1 w	mod 0 1 0 r/m

Timing (clocks): register 3
memory 16+EA

SHL/SAL = Shift logical/arithmetic left

1 1 0 1 0 0 v w	mod 1 0 0 r/m

Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

SHR = Shift logical right

1 1 0 1 0 0 v w	mod 1 0 1 r/m

Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

SAR = Shift arithmetic right

1 1 0 1 0 0 v w	mod 1 1 1 r/m

Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

ROL = Rotate left

1 1 0 1 0 0 v w	mod 0 0 0 r/m

Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

ROR = Rotate right

1 1 0 1 0 0 v w	mod 0 0 1 r/m

Timing (clocks):

single-bit register	2
single-bit memory	15+EA
variable-bit register	8+4/bit
variable-bit memory	20+EA+4/bit

RCL = Rotate through carry left

1 1 0 1 0 0 v w	mod 0 1 0 r/m

Timing (clocks):

single-bit register	2
single-bit memory	15+EA
variable-bit register	8+4/bit
variable-bit memory	20+EA+4/bit

RCR = Rotate through carry right

1 1 0 1 0 0 v w	mod 0 1 1 r/m

Timing (clocks):

single-bit register	2
single-bit memory	15+EA
variable-bit register	8+4/bit
variable-bit memory	20+EA+4/bit

AND = And
Reg./memory and register to either

0 0 1 0 0 0 d w	mod reg r/m

Timing (clocks):

register to register	3
memory to register	9+EA
register to memory	16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1

Timing (clocks):

immediate to register	4
immediate to memory	17+EA

Immediate to accumulator

0 0 1 0 0 1 0 w	data	data if w=1

Timing: 4 clocks

TEST = And function to flags, no result
Register/memory and register

1 0 0 0 0 1 0 w	mod reg r/m

Timing (clocks):

register to register	3
register with memory	9+EA

(Continued on following page)

Immediate data and register/memory

1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w=1

Timing (clocks):

immediate with register	4
immediate with memory	10+EA

Immediate data and accumulator

1 0 1 0 1 0 0 w	data	data if w=1

Timing: 4 clocks

OR = Or
Reg./memory and register to either

0 0 0 0 1 0 d w	mod reg r/m

Timing (clocks):

register to register	3
memory to register	9+EA
register to memory	16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w=1

Timing (clocks):

immediate to register	4
immediate to memory	17+EA

Immediate to accumulator

0 0 0 0 1 1 0 w	data	data if w=1

Timing: 4 clocks

XOR = Exclusive or
Reg./memory and register to either

0 0 1 1 0 0 d w	mod reg r/m

Timing (clocks):

register to register	3
memory to register	9+EA
register to memory	16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w=1

Timing (clocks):

immediate to register	4
immediate to memory	17+EA

Immediate to accumulator

0 0 1 1 0 1 0 w	data	data if w=1

Timing: 4 clocks

STRING MANIPULATION

REP = Repeat

1 1 1 1 0 0 1 z

Timing: 6 clocks/loop

MOVS = Move String

1 0 1 0 0 1 0 w

Timing: 17 clocks

CMPS = Compare String

1 0 1 0 0 1 1 w

Timing: 22 clocks

SCAS = Scan String

1 0 1 0 1 1 1 w

Timing: 15 clocks

LODS = Load String

1 0 1 0 1 1 0 w

Timing: 12 clocks

STOS = Store String

1 0 1 0 1 0 1 w

Timing: 10 clocks

CONTROL TRANSFER

NOTE: Queue reinitialization is not included in the timing information for transfer operations. To account for instruction loading, add 8 clocks to timing numbers.

CALL = Call

Direct within segment

1 1 1 0 1 0 0 0	disp-low	disp-high

Timing: 11 clocks

Indirect within segment

1 1 1 1 1 1 1 1	mod 0 1 0 r/m

Timing: 13+EA clocks

Direct intersegment

1 0 0 1 1 0 1 0	offset-low	offset-high
	seg-low	seg-high

Timing: 20 clocks

Indirect intersegment

1 1 1 1 1 1 1 1	mod 0 1 1 r/m

Timing: 29+EA clocks

JMP = Unconditional Jump

Direct within segment

1 1 1 0 1 0 0 1	disp-low	disp-high

Timing: 7 clocks

Direct within segment-short

1 1 1 0 1 0 1 1	disp

Timing: 7 clocks

Indirect within segment

1 1 1 1 1 1 1 1	mod 1 0 0 r/m

Timing: 7+EA clocks

Direct intersegment

1 1 1 0 1 0 1 0	offset-low	offset-high
	seg-low	seg-high

Timing: 7 clocks

Indirect intersegment

1 1 1 1 1 1 1 1	mod 1 0 1 r/m

Timing: 16+EA clocks

RET = Return from CALL

Within segment

1 1 0 0 0 0 1 1

Timing: 8 clocks

Within seg. adding immed to SP

1 1 0 0 0 0 1 0	data-low	data-high

Timing: 12 clocks

Intersegment

1 1 0 0 1 0 1 1

Timing: 18 clocks

Intersegment, adding immediate to SP

1 1 0 0 1 0 1 0	data-low	data-high

Timing: 17 clocks

JE/JZ = Jump on equal/zero

0 1 1 1 0 1 0 0	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

Mnemonics © Intel, 1978.

JL/JNGE = Jump on less/not greater or equal

0 1 1 1 1 1 0 0	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JLE/JNG = Jump on less or equal/not greater

0 1 1 1 1 1 1 0	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JB/JNAE = Jump on below/ not above or equal

0 1 1 1 0 0 1 0	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JBE/JNA = Jump on below or equal/not above

0 1 1 1 0 1 1 0	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JP/JPE = Jump on parity/parity even

0 1 1 1 1 0 1 0	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JO = Jump on overflow

0 1 1 1 0 0 0 0	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JS = Jump on sign

0 1 1 1 1 0 0 0	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JNE/JNZ = Jump on not equal/not zero

0 1 1 1 0 1 0 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JNL/JGE = Jump on not less/greater or equal

0 1 1 1 1 1 0 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

(Continued on following page)

JNLE/JG = Jump on not less or equal/greater

0 1 1 1 1 1 1 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JNB/JAE = Jump on not below/above or equal

0 1 1 1 0 0 1 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JNBE/JA = Jump on not below or equal/above

0 1 1 1 0 1 1 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JNP/JPO = Jump on not parity/parity odd

0 1 1 1 1 0 1 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JNO = Jump on not overflow

0 1 1 1 0 0 0 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

JNS = Jump on not sign

0 1 1 1 1 0 0 1	disp

Timing (clocks):		
	Jump is taken	8
	Jump is not taken	4

LOOP = Loop CX times

1 1 1 0 0 0 1 0	disp

Timing (clocks):		
	Jump is taken	9
	Jump is not taken	5

LOOPZ/LOOPE = Loop while zero/equal

1 1 1 0 0 0 0 1	disp

Timing (clocks):		
	Jump is taken	11
	Jump is not taken	5

LOOPNZ/LOOPNE = Loop while not zero/ not equal

1 1 1 0 0 0 0 0	disp

Timing (clocks):		
	Jump is taken	11
	Jump is not taken	5

JCXZ = Jump on CX zero

1 1 1 0 0 0 1 1	disp

Timing (clocks): Jump is taken 9
Jump is not taken 5

8086 CONDITIONAL TRANSFER OPERATIONS

Instruction	Condition	Interpretation
JE or JZ	ZF = 1	"equal" or "zero"
JL or JNGE	(SF xor OF) = 1	"less" or "not greater or equal"
JLE or JNG	((SP xor OF) or ZF) = 1	"less or equal" or "not greater"
JB or JNAE	CF = 1	"below" or "not above or equal"
JBE or JNA	(CF or ZF) = 1	"below or equal" or "not above"
JP or JPE	PF = 1	"parity" or "parity even"
JO	OF = 1	"overflow"
JS	SF= 1	"sign"
JNE or JNZ	ZF = 0	"not equal" or "not zero"
JNL or JGE	(SF xor OF) = 0	"not less" or "greater or equal"
JNLE or JG	((SF xor OF) or ZF) = 0	"not less or equal" or "greater"
JNB or JAE	CF = 0	"not below" or "above or equal"
JNBE or JA	(CF or ZF) = 0	"not below or equal" or "above"
JNP or JPO	PF = 0	"not parity" or "parity odd"
JNO	OF = 0	"not overflow"
JNS	SF = 0	"not sign"

*"Above" and "below" refer to the relation between two unsigned values, while "greater" and "less" refer to the relation between two signed values.

INT = **Interrupt**
Type specified

1 1 0 0 1 1 0 1	type

Timing: 50 clocks

Type 3

1 1 0 0 1 1 0 0

Timing: 51 clocks

INTO = Interrupt on overflow

1 1 0 0 1 1 1 0

Timing: 52 clocks if pass 4 clocks if fail

IRET = Interrupt return

1 1 0 0 1 1 1 1

Timing: 24 clocks

PROCESSOR CONTROL

CLC = Clear carry

1 1 1 1 1 0 0 0

Timing: 2 clocks

CMC = Complement carry

1 1 1 1 0 1 0 1

Timing: 2 clocks

CLD = Clear direction

1 1 1 1 1 1 0 0

Timing: 2 clocks

CLI = Clear interrupt

1 1 1 1 1 0 1 0

Timing: 2 clocks

HLT = Halt

1 1 1 1 0 1 0 0

Timing: 2 clocks

LOCK = Bus lock prefix

1 1 1 1 0 0 0 0

Timing: 2 clocks

STC = Set carry

1 1 1 1 1 0 0 1

Timing: 2 clocks

NOP = No operation

1 0 0 1 0 0 0 0

Timing: 3 clocks

STD = Set direction

1 1 1 1 1 1 0 1

Timing: 2 clocks

STI = Set interrupt

1 1 1 1 1 0 1 1

Timing: 2 clocks

WAIT = Wait

1 0 0 1 1 0 1 1

Timing: 3 clocks

ESC = Escape (to external device)

1 1 0 1 1 x x x	mod x x x r/m

Timing: 7+EA clocks

Footnotes:
if d = 1 then "to"; if d = 0 then "from"
if w = 1 then word instruction; if w = 0 then byte instruction
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 some string primitives to compare with ZF FLAG

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
DX = Variable port register
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
See page 1 for Operand Summary.
See page 2 for Segment Override Summary.

ASSEMBLER DIRECTIVES

Symbol Definition:
- EQU
- LABEL
- PURGE

Memory Reservation and Data Definition:
- DB
- DW
- DD
- RECORD

Location Counter and Segmentation Control:
- SEGMENT/ENDS
- ORG
- GROUP
- ASSUME
- PROC/ENDP
- CODEMACRO/ENDM

Program Linkage:
- NAME
- PUBLIC
- EXTRN
- END

PROCESSOR RESET REGISTER INITIALIZATION

Flags = OOOOH (to disable interrupts and single-stepping)

CS = FFFFH
IP = OOOOH
(to begin execution at FFFFOH)

DS = OOOOH
SS = OOOOH
ES = OOOOH

No other registers are acted upon during reset.

MCS-86™ RESERVED LOCATIONS

Reserved Memory Locations

Intel Corporation reserves the use of memory locations FFFFOH through FFFFFH (with the exception of FFFFOH - FFFF5H for JMP instr.) for Intel hardware and software products. If you use these locations for some other purpose, you may preclude compatibility of your system with certain of these products.

Reserved Input/Output Locations

Intel Corporation reserves the use of input/output locations F8H through FFH for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

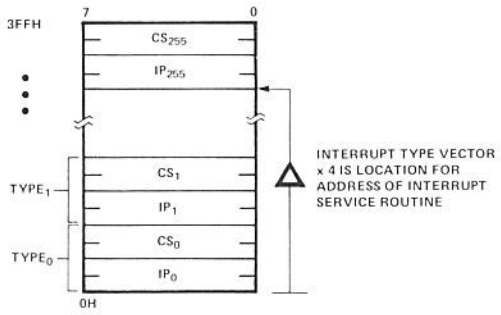
Reserved Interrupt Locations

Intel Corporation reserves the use of interrupts 0-31 (locations 00H through 7FH) for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

Interrupts 0 through 4 (00H-13H) currently have dedicated hardware functions as defined below.

Interrupt	Location	Function
0	00H-03H	Divide by zero
1	04H-07H	Single step
2	08H-0BH	Non-maskable interrupt
3	0CH-0FH	One-byte interrupt instruction
4	10H-13H	Interrupt on overflow

INTERRUPT POINTER TABLE



8086 INSTRUCTION SET MATRIX

Hi \ Lo	0	1	2	3	4	5	6	7
0	ADD b,f,r/m	ADD w,f,r/m	ADD b,t,r/m	ADD w,t,r/m	ADD b,ia	ADD w,ia	PUSH ES	POP ES
1	ADC b,f,r/m	ADC w,f,r/m	ADC b,t,r/m	ADC w,t,r/m	ADC b,i	ADC w,i	PUSH SS	POP SS
2	AND b,f,r/m	AND w,f,r/m	AND b,t,r/m	AND w,t,r/m	AND b,i	AND w,i	SEG =ES	DAA
3	XOR b,f,r/m	XOR w,f,r/m	XOR b,t,r/m	XOR w,t,r/m	XOR b,i	XOR w,i	SEG =SS	AAA
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI
6								
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA
8	Immed b,r/m	Immed w,r/m	Immed b,r/m	Immed is,r/m	TEST b,r/m	TEST w,r/m	XCHG b,r/m	XCHG w,r/m
9	NOP	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI
A	MOV m → AL	MOV m → AX	MOV AL → m	MOV AX → m	MOVS b	MOVS w	CMPS b	CMPS w
B	MOV i → AL	MOV i → CL	MOV i → DL	MOV i → BL	MOV i → AH	MOV i → CH	MOV i → DH	MOV i → BH
C			RET, (i+SP)	RET	LES	LDS	MOV b,i,r/m	MOV w,i,r/m
D	Shift b	Shift w	Shift b,v	Shift w,v	AAM	AAD		XLAT
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b	OUT w
F	LOCK		REP	REP z	HLT	CMC	Grp 1 b,r/m	Grp 1 w,r/m

Hi \ Lo	8	9	A	B	C	D	E	F
0	OR b,f,r/m	OR w,f,r/m	OR b,t,r/m	OR w,t,r/m	OR b,i	OR w,i	PUSH CS	
1	SBB b,f,r/m	SBB w,f,r/m	SBB b,t,r/m	SBB w,t,r/m	SBB b,i	SBB w,i	PUSH DS	POP DS
2	SUB b,f,r/m	SUB w,f,r/m	SUB b,t,r/m	SUB w,t,r/m	SUB b,i	SUB w,i	SEG =CS	DAS
3	CMP b,f,r/m	CMP w,f,r/m	CMP b,t,r/m	CMP w,t,r/m	CMP b,i	CMP w,i	SEG =DS	AAS
4	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6								
7	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	MOV b,f,r/m	MOV w,f,r/m	MOV b,t,r/m	MOV w,t,r/m	MOV sr,t,r/m	LEA	MOV sr,f,r/m	POP r/m
9	CBW	CWD	CALL I,d	WAIT	PUSHF	POPF	SAHF	LAHF
A	TEST b,i	TEST w,i	STOS b	STOS w	LODS b	LODS w	SCAS b	SCAS w
B	MOV i → AX	MOV i → CX	MOV i → DX	MOV i → BX	MOV i → SP	MOV i → BP	MOV i → SI	MOV i → DI
C			RET, I,(i+SP)	RET I	INT Type 3	INT (Any)	INTO	IRET
D	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	CALL d	JMP d	JMP I,d	JMP si,d	IN v,b	IN v,w	OUT v,b	OUT v,w
F	CLC	STC	CLI	STI	CLD	STD	Grp 2 b,r/m	Grp 2 w,r/m

where:

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

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

Mnemonics © Intel, 1978.

MNEMONIC INDEX

Printed in U.S.A./A329/381/20K BL