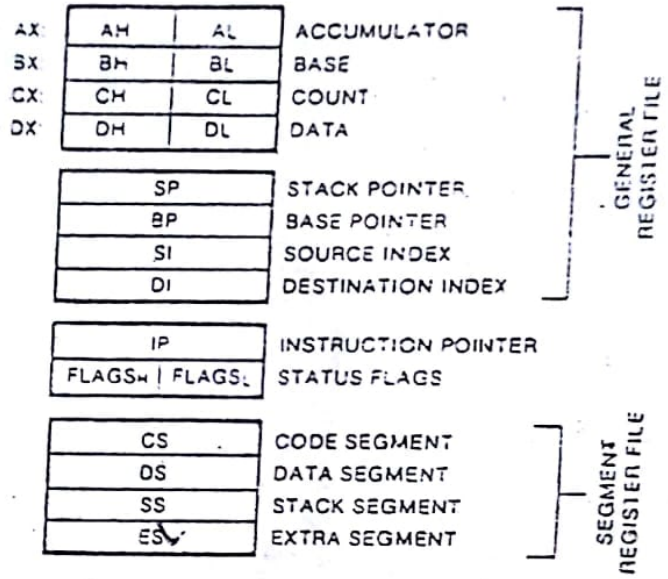
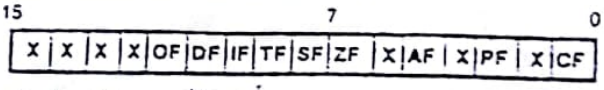
Appendix C

8086/8088 Instruction Set

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

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

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

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 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

Immediate to accumulator

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

Timing: 4 clocks

(Continued)

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.

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

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

(Continued on following page)