Selected 8086 Instructions

Quick Reference List

adc	Add with carry flag
add	Add two numbers
and	Bitwise logical AND
call	Call procedure or function
cbw	Convert byte to word (signed)
cli	Clear interrupt flag (disable interrupts)
cwd	Convert word to doubleword (signed)
cmp	Compare two operands
dec	Decrement by 1
div	Unsigned divide
idiv	Signed divide
imul	Signed multiply
in	Input (read) from port
inc	Increment by 1
int	Call to interrupt procedure
iret	Interrupt return
j??	Jump if ?? condition met
jmp	Unconditional jump
lea	Load effective address offset
mov	Move data
mul	Unsigned multiply
neg	Two's complement negate
nop	No operation
not	One's complement negate
or	Bitwise logical OR
out	Output (write) to port
pop	Pop word from stack
popf	Pop flags from stack
push	Push word onto stack
pushf	Push flags onto stack
ret	Return from procedure or function
sal	Bitwise arithmetic left shift (same as shl)
sar	Bitwise arithmetic right shift (signed)
sbb	Subtract with borrow
shl	Bitwise left shift (same as sal)
shr	Bitwise right shift (unsigned)
sti	Set interrupt flag (enable interrupts)
sub	Subtract two numbers
test	Bitwise logical compare
xor	Bitwise logical XOR

Important Usage Notes:

1. The first operand of an instruction is also the destination if there is a resulting value. Divide and multiply instructions are common exceptions to this rule.
2. There can be *at most* one memory operand per instruction.
3. There can be *at most* one immediate operand per instruction.
4. Operands generally must be of the same size (i.e., byte or word).
5. Using a label is the same as using an immediate or constant value.

6. When BP is used in a memory reference, SS is assumed as the segment. Otherwise DS is assumed.
7. While an instruction is executing, IP refers to the next instruction.
8. Many instructions are smaller if you use the appropriate registers (usually AX or AL).
9. In NASM, all labels are case sensitive but instruction and register names are not.

Terminology Used:

- **memory** - Refers to an 8 or 16-bit memory location determined by an effective address.
- **register** - AX, BX, CX, DX, SI, DI, BP, or SP as well as the 8-bit derivatives of AX, BX, CX, and DX (other registers or flags are not allowed).
- **immediate** - A numeric constant or label.
- **REG1::REG2** - The concatenation of two registers (e.g., the 32-bit value DX::AX) A single colon is used for memory addresses.
- **XF** or **XF=b** - A flag's value after an instruction can be 0 or 1 and usually depends on the result of the instruction. A flag being set to '?' by an instruction indicates that the flag is undefined after the operation.

Instructions:

adc   Add with carry flag

```
  Syntax:      adc    dest, src
  dest: memory or register
  src:  memory, register, or immediate
  Action: dest = dest + src + CF
  Flags Affected: OF, SF, ZF, AF, PF, CF
  Notes: This instruction is used to perform 32-bit addition.
```

add   Add two numbers

```
  Syntax:      add    dest, src
  dest: register or memory
  src: register, memory, or immediate
  Action: dest = dest + src
  Flags Affected: OF, SF, ZF, AF, PF, CF
  Notes: Works for both signed and unsigned numbers.
```

and   Bitwise logical AND

```
  Syntax:      and    dest, src
  dest: register or memory
  src: register, memory, or immediate
  Action: dest = dest & src
  Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0
```

call   Call procedure or function

```
  Syntax:      call   addr
  addr: register, memory, or immediate
  Action: Push IP onto stack, set IP to addr.
  Flags Affected: None
```

cbw Convert byte to word (signed)

Syntax: cbw
Action: Sign extend AL to create a word in AX.
Flags Affected: None
Notes: For unsigned numbers use "mov ah, 0".

cli Clear interrupt flag (disable interrupts)

Syntax: cli
Action: Clear IF
Flags Affected: IF=0

cmp Compare two operands

Syntax: cmp op1, op2
op1: register or memory
op2: register, memory, or immediate
Action: Perform op1-op2, discarding the result but setting the flags.
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: Usually used before a conditional jump instruction.

cwd Convert word to doubleword (signed)

Syntax: cwd
Action: Sign extend AX to fill DX, creating a dword contained in DX::AX.
Flags Affected: None
Notes: For unsigned numbers use "xor dx, dx" to clear DX.

dec Decrement by 1

Syntax: dec op
op: register or memory
Action: op = op - 1
Flags Affected: OF, SF, ZF, AF, PF

div Unsigned divide

Syntax: div op8
div op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, unsigned AL = AX / op8 and AH = AX % op8
If operand is op16, unsigned AX = DX::AX / op16 and DX = DX::AX % op16
Flags Affected: OF=?, SF=?, ZF=?, AF=?, PF=?, CF=?
Notes: Performs both division and modulus operations in one instruction.

idiv Signed divide

Syntax: idiv op8
idiv op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, signed AL = AX / op8 and AH = AX % op8
If operand is op16, signed AX = DX::AX / op16 and DX = DX::AX % op16

```
  Flags Affected: OF=?, SF=?, ZF=?, AF=?, PF=?, CF=?
  Notes: Performs both division and modulus operations in one
instruction.
```

imul   Signed multiply

```
  Syntax:      imul   op8
               imul   op16
  op8: 8-bit register or memory
  op16: 16-bit register or memory
  Action: If operand is op8, signed AX = AL * op8
          If operand is op16, signed DX::AX = AX * op16
  Flags Affected: OF, SF=?, ZF=?, AF=?, PF=?, CF
```

in   Input (read) from port

```
  Syntax:      in     AL, op8
               in     AX, op8
  op8: 8-bit immediate or DX
  Action: If destination is AL, read byte from 8-bit port op8.
          If destination is AX, read word from 16-bit port op8.
  Flags Affected: None
```

inc   Increment by 1

```
  Syntax:      inc    op
  op: register or memory
  Action: op = op + 1
  Flags Affected: OF, SF, ZF, AF, PF
```

int   Call to interrupt procedure

```
  Syntax:      int    imm8
  imm8: 8-bit unsigned immediate
  Action: Push flags, CS, and IP; clear IF and TF (disabling
interrupts); load
          word at address (imm8*4) into IP and word at (imm8*4 + 2)
into CS.
  Flags Affected: IF=0, TF=0
  Notes: This instruction is usually used to call system routines.
```

iret   Interrupt return

```
  Syntax:      iret
  Action: Pop IP, CS, and flags (in that order).
  Flags Affected: All
  Notes: This instruction is used at the end of ISRs.
```

j??   Jump if ?? condition met

```
  Syntax:      j??    rel8
  rel8: 8-bit signed immediate
  Action: If condition ?? met, IP = IP + rel8 (sign extends rel8)
  Flags Affected: None
  Notes: Use the cmp instruction to compare two operands then j?? to
jump
         conditionally. The ?? of the instruction name represents the
jump
         condition, allowing for following instructions:

         ja    jump if above, unsigned >
```

jae	jump if above or equal, unsigned >=
jb	jump if below, unsigned <
jbe	jump if below or equal, unsigned <=
je	jump if equal, ==
jne	jump if not equal, !=
jg	jump if greater than, signed >
jge	jump if greater than or equal, signed >=
jl	jump if less than, signed <
jle	jump if less than or equal, signed <=

All of the ?? suffixes can also be of the form n?? (e.g., jna for jump if not above). See 8086 documentation for many more ?? conditions.

An assembler label should be used in place of the rel8 operand. The assembler will then calculate the relative distance to jump.

Note also that rel8 operand greatly limits conditional jump distance (-127 to +128 bytes from IP). Use the jmp instruction in combination with j?? to overcome this barrier.

jmp   Unconditional jump

Syntax: jump   rel
             jump   op16
             jump   seg:off

rel: 8 or 16-bit signed immediate

op16: 16-bit register or memory

seg:off: Immediate 16-bit segment and 16-bit offset

Action: If operand is rel, IP = IP + rel
        If operand is op16, IP = op16
        If operand is seg:off, CS = seg, IP = off

Flags Affected: None

Notes: An assembler label should be used in place of the rel8 operand. The assembler will then calculate the relative distance to jump.

lea   Load effective address offset

Syntax: lea   reg16, memref

reg16: 16-bit register

memref: An effective memory address (e.g., [bx+2])

Action: reg16 = address offset of memref

Flags Affected: None

Notes: This instruction is used to easily calculate the address of data in memory. It does not actually access memory.

mov   Move data

Syntax: mov   dest, src

dest: register or memory

src: register, memory, or immediate

Action: dest = src

Flags Affected: None

mul   Unsigned multiply

```
  Syntax:      mul     op8
               mul     op16
  op8: 8-bit register or memory
  op16: 16-bit register or memory
  Action: If operand is op8, unsigned AX = AL * op8
          If operand is op16, unsigned DX::AX = AX * op16
  Flags Affected: OF, SF=?, ZF=?, AF=?, PF=?, CF
```

neg   Two's complement negate

```
  Syntax:      neg     op
  op: register or memory
  Action: op = 0 - op
  Flags Affected: OF, SF, ZF, AF, PF, CF
```

nop   No operation

```
  Syntax:      nop
  Action: None
  Flags Affected: None
```

not   One's complement negate

```
  Syntax:      not     op
  op: register or memory
  Action: op = ~op
  Flags Affected: None
```

or   Bitwise logical OR

```
  Syntax:      or      dest, src
  dest: register or memory
  src: register, memory, or immediate
  Action: dest = dest | src
  Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0
```

out   Output (write) to port

```
  Syntax:      out     op, AL
               out     op, AX
  op: 8-bit immediate or DX
  Action: If source is AL, write byte in AL to 8-bit port op.
          If source is AX, write word in AX to 16-bit port op.
  Flags Affected: None
```

pop   Pop word from stack

```
  Syntax:      pop     op16
  reg16: 16-bit register or memory
  Action: Pop word off the stack and place it in op16 (i.e., op16 = [SS:SP]
          then SP = SP + 2).
  Flags Affected: None
  Notes: Pushing and popping of SS and SP are allowed but strongly discouraged.
```

popf   Pop flags from stack

```
  Syntax:      popf
  Action: Pop word from stack and place it in flags register.
```

Flags Affected: All

push   Push word onto stack

```
  Syntax:      push    op16
  op16: 16-bit register or memory
  Action: Push op16 onto the stack (i.e., SP = SP - 2 then [SS:SP] = op16).
  Flags Affected: None
  Notes: Pushing and popping of SS and SP are allowed but strongly discouraged.
```

pushf   Push flags onto stack

```
  Syntax:      pushf
  Action: Push flags onto stack as a word.
  Flags Affected: None
```

ret   Return from procedure or function

```
  Syntax:      ret
  Action: Pop word from stack and place it in IP.
  Flags Affected: None
```

sal   Bitwise arithmetic left shift (same as shl)

```
  Syntax:      sal     op, 1
               sal     op, CL
  op: register or memory
  Action: If operand is 1, op = op << 1
          If operand is CL, op = op << CL
  Flags Affected: OF, SF, ZF, AF=?, PF, CF
```

sar   Bitwise arithmetic right shift (signed)

```
  Syntax:      sar     op, 1
               sar     op, CL
  op: register or memory
  Action: If operand is 1, signed op = op >> 1 (sign extends op)
          If operand is CL, signed op = op >> CL (sign extends op)
  Flags Affected: OF, SF, ZF, AF=?, PF, CF
```

sbb   Subtract with borrow

```
  Syntax:      sbb     dest, src
  dest: register or memory
  src: register, memory, or immediate
  Action: dest = dest - (src + CF)
  Flags Affected: OF, SF, ZF, AF, PF, CF
  Notes: This instruction is used to perform 32-bit subtraction.
```

shl   Bitwise left shift (same as sal)

```
  Syntax:      shl     op, 1
               shl     op, CL
  op: register or memory
  Action: If operand is 1, op = op << 1
          If operand is CL, op = op << CL
  Flags Affected: OF, SF, ZF, AF=?, PF, CF
```

shr   Bitwise right shift (unsigned)

```
  Syntax:      shr     op, 1
               shr     op, CL
  op: register or memory
  Action: If operand is 1, op = (unsigned)op >> 1
          If operand is CL, op = (unsigned)op >> CL
  Flags Affected: OF, SF, ZF, AF=?, PF, CF
```

sti   Set interrupt flag (enable interrupts)

```
  Syntax:      sti
  Action: Set IF
  Flags Affected: IF=1
```

sub   Subtract two numbers

```
  Syntax:      sub     dest, src
  dest: regsiter or memory
  src: register, memory, or immediate
  Action: dest = dest - src
  Flags Affected: OF, SF, ZF, AF, PF, CF
  Notes: Works for both signed and unsigned numbers.
```

test   Bitwise logical compare

```
  Syntax:      test    op1, op2
  op1: register, memory, or immediate
  op2: register, memory, or immediate
  Action: Perform op1 & op2, discarding the result but setting the flags.
  Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0
  Notes: This instruction is used to test if bits of a value are set.
```

xor   Bitwise logical XOR

```
  Syntax:      xor     dest, src
  dest: register or memory
  src: register, memory, or immediate
  Action: dest = dest ^ src
  Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0
```
