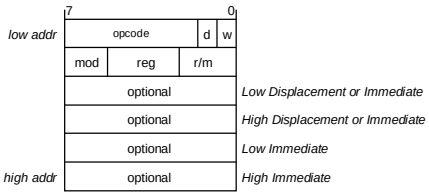
Instruction Format

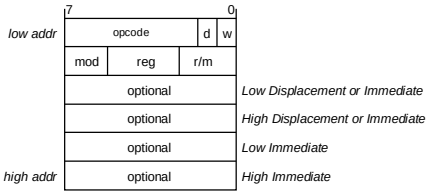


opcode 6-bit value that specifies instruction type

d is 1-bit value that specifies destination
d=0 for memory and *d*=1 for register

w is 1-bit value that specifies if destination is word or byte
w=0 for byte and *w*=1 for word

Instruction Format (Cont.)



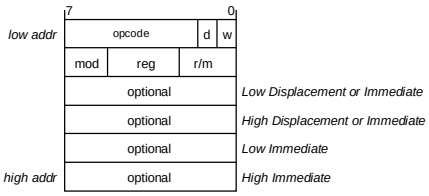
mod is 2-bit value that indicates *addressing mode*
(along with *r/m* value)

reg is 3-bit value that specifies a (destination) register
(see table to right)

r/m is 3-bit value that specifies operand location
(r/m means register/memory)

reg	w=1	w=0
000	ax	al
001	cx	cl
010	dx	dl
011	bx	bl
100	sp	ah
101	bp	ch
110	si	dh
111	di	bh

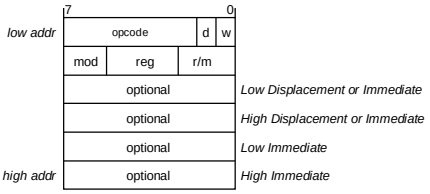
Instruction Format (Cont.)



Displacement may be either 8 or 16 bits
- signed integer encoded into instruction
- used in computation of operand address

Immediate may be 8, 16 or 32 bits
- signed integer
- used as an actual operand

Instruction Format Example



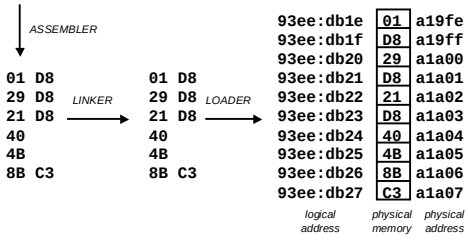
Consider the Instruction: **`mov ax, bx`**

The Assembler translates this into: **`8B C3`**

opcode is:	100010	**mov**
d is:	1	destination is register
w is:	1	destination size = 1 word
mod is:	11	this indicates that r/m specifies a register
reg is:	000	destination register is **ax**
r/m is:	011	source register is **bx**

Assembler versus Machine Code

```
ADD   AX, BX      ;AX gets value AX+BX
SUB   AX, BX      ;AX gets value AX-BX
AND   AX, BX      ;AX gets bitwise AND of AX and BX
INC   AX          ;AX gets its original value plus 1
DEC   BX          ;BX gets its original value minus 1
MOV   AX, BX      ;AX gets values in BX
```



Operand types

1) Register - Encoded in Instruction
   - Fastest Executing
   - No Bus Access (in Instr. Queue)
   - Short Instruction Length

2) Immediate - Constant Encoded in Instruction
   - 8 or 16 bits
   - No Bus Access (in Instr. Queue)
   - Can only be Source Operand

3) Memory - Requires Bus Transfer
   - Can Require Computation of Address
   - Address of Operand *DATA* is Called

EFFECTIVE ADDRESS

Operand in Memory

1) Resident at an Address
   - Fastest Executing
   - No Bus Access (in Instr. Queue)
   - Short Instruction Length

2) Immediate - Constant Encoded in Instruction
   - 8 or 16 bits
   - No Bus Access (in Instr. Queue)
   - Can only be Source Operand

3) Memory - Requires Bus Transfer
   - Can Require Computation of Address
   - Address of Operand *DATA* is Called

EFFECTIVE ADDRESS

Effective Address

- Computed by EU
- In General,

```
displacement + base register + index register
```

- Any Combination of These 3 Values
  - Leads to Several Different Addressing Modes
- Displacement
  - 8 or 16 bit Constant in the Instruction
  - "base register" Must be **BX** or **BP**
  - "index register" Must be **SI** or **DI**

Addressing Modes

CLASSIFICATION I

- Register/Register*
- Immediate*
- Direct
- Register Indirect
- Based
- Indexed
- Based Indexed
- String
- I/O Port

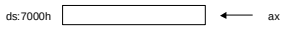
CLASSIFICATION II

- Register/Register*
- Immediate*
- Direct
- Indirect
  - Register Indirect
  - Based
  - Indexed
  - Based Indexed
- String
- I/O Port

*Considered to be Addressing Modes by Some People, although the Technically Involve No Memory Accesses

Direct Addressing

```
mov  [7000h],  ax
```

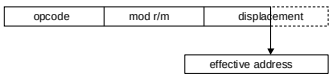


A3 00 70

```
mov  es:[7000h], ax
```

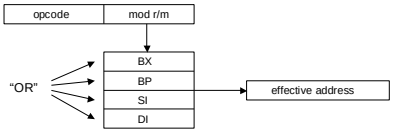
26 A3 00 70

prefix byte
- longer instruction
- more fetch time



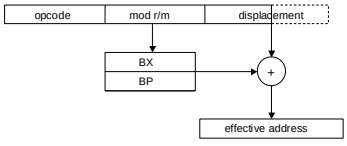
Register Indirect Addressing

```
mov  al,  [bp]      ;al gets 8 bits at SS:BP
mov  ah,  [bx]      ;ah gets 8 bits at DS:BX
mov  ax,  [di]      ;ax gets 16 bits at DS:SI
mov  eax, [si]      ;eax gets 32 bits at DS:SI
```



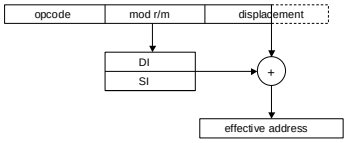
Based Indirect Addressing

```
mov   al,   [bp+2]      ;al gets 8 bits at SS:BP+2
mov   ah,   [bx-4]      ;ah gets 8 bits at DS:BX-4
```



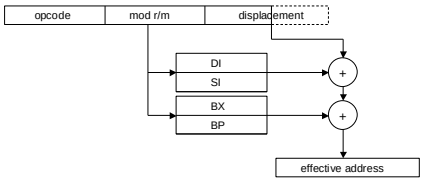
Indexed Indirect Addressing

```
mov    ax,    [di+1000h]    ;ax gets 16 bits at DS:SI+1000h
mov    eax,   [si+300h]     ;eax gets 32 bits at DS:SI+300h
```



Based Indexed Indirect Addressing

```
mov    ax,    [bp+di]       ;ax gets 16 bits at SS:BP+DI
mov    ax,    [di+bp]       ;ax gets 16 bits at DS:BP+DI
mov    eax,   [bx+si+10h]   ;eax gets 32 bits at DS:BX+SI+10h
mov    cx,    LIST[bp+si-7] ;cx gets 16 bits at SS:BP+SI-7
```



Addressing Mode Examples

`mov al, bl ;8-bit register addressing`
`mov di, bp ;16-bit register addressing`
`mov eax, eax ;32-bit register addressing`
	Register

`mov al, 12 ;8-bit immediate, al<-0ch`
`mov cx, faceh ;16-bit immediate, cx<-64,206`
`mov ebx, 2h ;32-bit immediate, ebx<-00000002h`
	Immediate

`mov al, LIST ;al<-8 bits stored at label LIST`
`mov ch, DATA ;ch<-8 bits stored at label DATA`
`mov ds, DATA2 ;ds<-16 bits stored at label DATA2`
	Direct

`mov al, [bp] ;al<-8 bits stored at SS:BP`
`mov ah, [bx] ;ah<-8 bits stored at DS:BX`
`mov ax, [bp] ;ax<-16 bits stored at SS:BP`
`mov eax, [bx] ;eax<-32 bits stored at DS:BX`
`mov ax, es:[bp] ;ax<-16 bits stored at SS:DI`
`mov al, [bp+2] ;al<-8 bits stored at SS:(BP+2)`
`mov ax, [bx-4] ;ax<-16 bits stored at DS:(BX-4)`
`mov al, LIST[bp] ;al<-8 bits stored at SS:(BP+LIST)`
`mov bx, LIST[bx] ;bx<-16 bits stored at DS:(BX+LIST)`
`mov al, LIST[bp+2] ;al<-8 bits stored at SS:(BP+2+LIST)`
`mov ax, LIST[bx-12h] ;ax<-16 bits stored at DS:(BX-18+LIST)`
	Based

More Addressing Mode Examples

```
mov al,   [si]         ;al<-8 bits stored at DS:SI
mov ah,   [di]         ;ah<-8 bits stored at DS:DI
mov ax,   [si]         ;ax<-16 bits stored at DS:SI
mov eax,  [di]         ;eax<-32 bits stored at DS:DI
mov ax,   es:[di]      ;ax<-16 bits stored at ES:DI
mov al,   [si+2]       ;al<-8 bits stored at DS:(SI+2)
mov ax,   [di-4]       ;ax<-16 bits stored at DS:(DI-4)
mov al,   LIST[si]     ;al<-8 bits stored at DS:(SI+LIST)
mov bx,   LIST[di]     ;bx<-16 bits stored at DS:(DI+LIST)
mov al,   LIST[si+2]   ;al<-8 bits stored at DS:(SI+2+LIST)
mov ax,   LIST[di-12h] ;ax<-16 bits stored at DS:(DI-18+LIST)
```

Indexed

```
mov al,   [bp+di]      ;al<-8 bits from SS:(BP+DI)
mov ah,   ds:[bp+si]   ;ah<-8 bits from DS:(BP+SI)
mov ax,   [bx+si]      ;ax<-16 bits from DS:(BX+SI)
mov eax,  es:[bx+di]   ;eax<-32 bits from ES:(BX+DI)
mov al,   LIST[bp+di]  ;al<-8 bits from SS:(BP+DI+LIST)
mov ax,   LIST[bx+si]  ;ax<-16 bits from DS:(BX+SI+LIST)
mov al,   LIST[bp+di-10h]     ;al<-8 bits from SS:(BP+DI-16+LIST)
mov ax,   LIST[bx+si+1AFH]    ;ax<-16 bits from DS:(BX+SI+431+LIST)
```

Based Indexed

String Addressing

- Implicit Register Use
  - –SI Used for Source EA
  - –DI Used for Destination EA
- SI, DI point to First (or Last) Byte in Strings
- Useful for Repeated String Operations
  - –Another Type of Prefix
- Special Subset of String Instructions

```
movs (movsb, movsw)          ;move a string
cmps (cmpsb, cmpsw)          ;compare two strings
scas  (scasb, scasw)         ;scan (search) string
lods   (lodsb, lodsw)        ;load a string
stos   (stosb, stosw)        ;store a string
```

String Addressing

```
movsb        ;ES:DI gets the byte pointed to by DS:SI
```

```
mov          cx, 10 ;cx gets value 10 decimal (byte counter)
rep movsb           ;ES:DI gets the byte pointed to by DS:SI
                    ;DI gets DI+1 (or DI-1 depending on DF)
                    ;SI gets SI+1 (or SI-1 depending on DF)
                    ;CX gets CX-1
                    ;if CX is not 0, do another movsb
```

```
cld          ;DF gets 0 (clear or reset), means increment
std          ;DF gets 1 (set), means decrement
```

REPEAT PREFIXES

```
rep          ;Repeat (uses CX as counter)
repe         ;Repeat while equal (checks for ZF=1)
repz         ;Same as repe, just different mnemonic
repne        ;Repeat while not equal (checks for ZF=0)
repnz        ;Same as repne
```

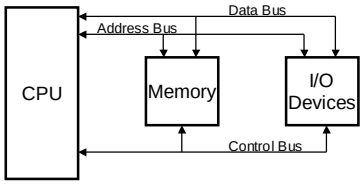
NOTE: These are "*repeat whiles*" NOT "*repeat untils*"

I/O Port Addressing

- x86 Family has 65,536 I/O Ports
- Each Port has Address (like Memory)
  - Referred to as "I/O Memory Space"
- I/O Port is 1 byte or 2 bytes
  - with 386+ also 4 bytes
- Two Addressing Modes
  1) Immediate Port Address
     - Can only be 1 byte
     - Can only Address Ports 00h through ffh
  2) Port Address Present in DX
     - Can Address all Ports 0000h through ffffh
- Can only Use DX for Port Addresses
- Can only Use Al,AX,EAX for Port Data

I/O Port Addressing Examples

```
in   al,    40h    ;al gets 1 byte from port 40h
in   ax,    255    ;ax gets 2 bytes from port ffh
in   al,    dx     ;ax gets 1 byte from port address in dx
in   eax,   dx     ;eax gets 4 bytes from port addr. in dx
out 80h,    al     ;send contents of al to port 80h
out dx,     eax    ;send contents of eax to port addr. in dx
```



Address Bus is Shared - Control Bus Indicates I/O or Memory