x86 Instruction Encoding (12.3)

OUTLINES

- Introduction
- Instruction Format
- Single Byte Instructions
- Move Immediate to Register
- Register Mode Instructions
- Processor Operand-Size Prefix
- Memory Mode Instructions

INTRODUCTION

- The Intel 8086 processor was the first in a line of processors using a ***Complex Instruction Set Computer*** (CISC) design.
  - a wide variety of memory-addressing, shifting, arithmetic, data movement, and logical operations.
- To *encode* an instruction means to convert an assembly language instruction and its operands into machine code.
- To *decode* an instruction means to convert a machine code instruction into assembly language.
- We will begin with the 8086/8088 processor as an illustrative example.
  - Later, we will show some of the changes made when Intel introduced 32-bit processors.

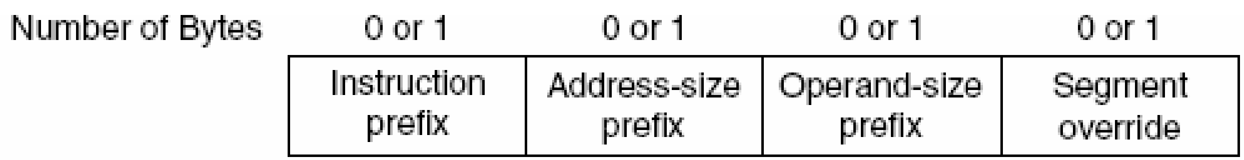
INSTRUCTION FORMAT

Instruction Prefix	Opcode	ModR/M	SIB	Address Displacement	Immediate Data
1 byte	1-3 bytes	1 byte	1 byte	1-4 bytes	1-4 bytes

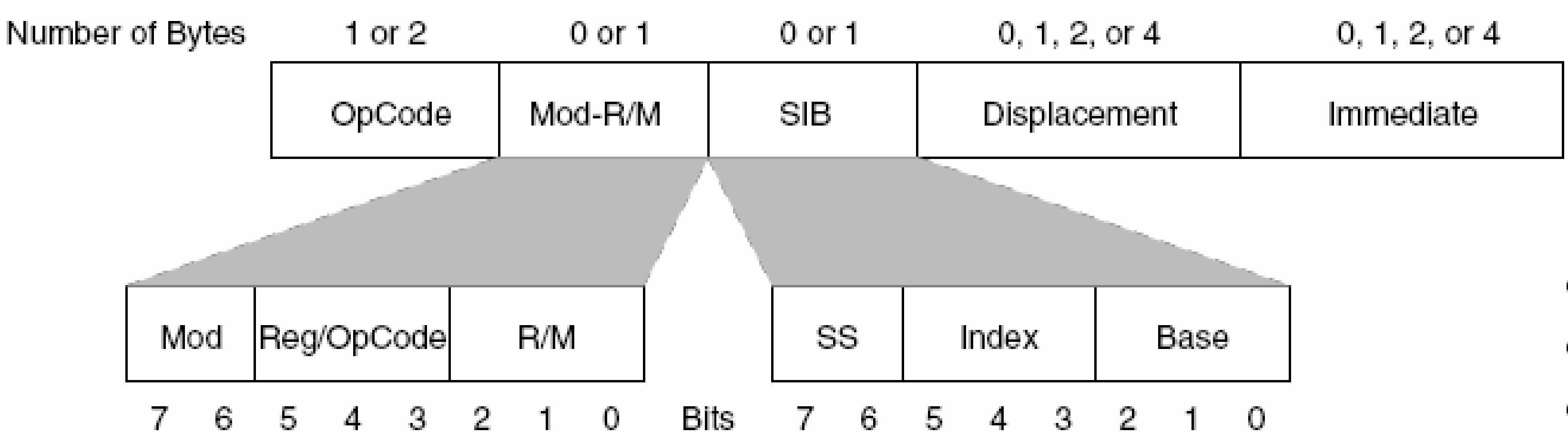
- Instructions are stored in little-endian order, so the prefix byte is located at the instruction's starting address.
- Every instruction has an opcode, but the remaining fields are optional.
- Most instructions are 2 or 3 bytes.

- The **instruction prefix** overrides default operand sizes. The prefix byte **is not** the *opcode expansion prefix* discussed earlier - they are special bytes to modify the behavior of existing instruction
- The **opcode** (operation code) identifies a specific variant of an instruction.
  - E.g. the ADD instruction has nine different opcodes, depending on the parameter types used.
- The **Mod R/M** field identifies the addressing mode and operands. The notation "**R/M**" stands for ***register*** and ***mode***.
- The **scale index byte** (SIB) is used to calculate offsets of array indexes.
- The **address displacement** field holds an operand's offset, or it can be added to base and index registers in addressing modes such as *base-displacement* or *base-index-displacement*
- The **immediate data** field holds constant operands

Another view of the x86 instruction format:



(a) Optional instruction prefixes



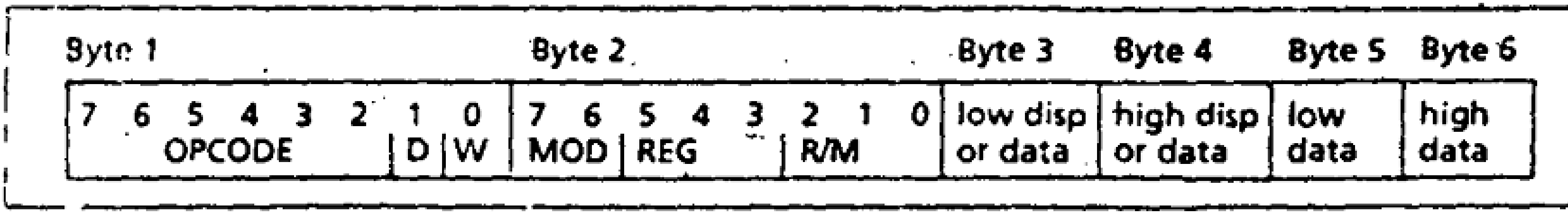
(b) General instruction format

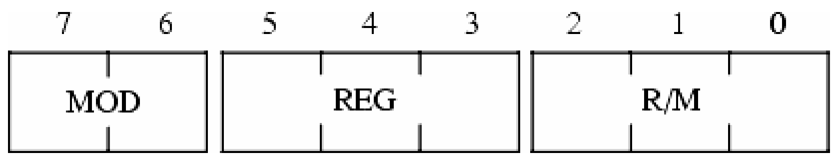
Byte 1								Byte 2								Byte 3	Byte 4	Byte 5	Byte 6
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	low disp or data	high disp or data	low data	high data
OPCODE						D	W	MOD		REG			R/M						

- The six-bit opcode identifies the operation. The same opcode is used for both 8- and 16-bit operations.
- The size of the operands is given by the **W** bit: *W = 0* means 8-bit data and W = 1 means 16-bit (or 32 bits) data.
- Bit number one, marked **D**, specifies the direction of the data transfer:
  - If **d = 0** then the destination operand is a memory location (not in reg mod), e.g.

    e.g. `add [ebx], al`
  - If **d = 1** then the destination operand is a register (not in reg mod), e.g.

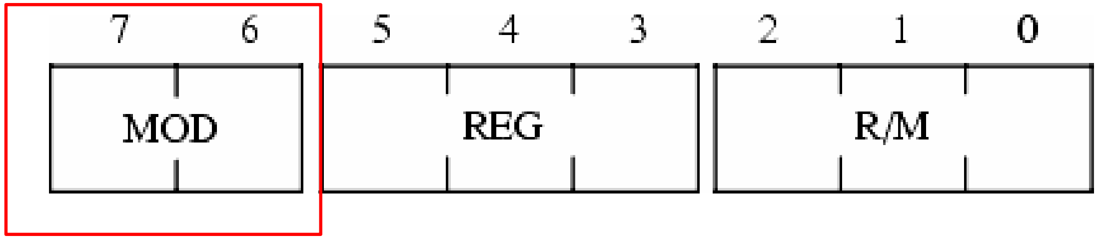
    e.g. `add al, [ebx]`
- the **D** bit specifies whether the register in the **REG** field is a source or destination operand, **D = 0** means source and ***D* = 1** means destination.



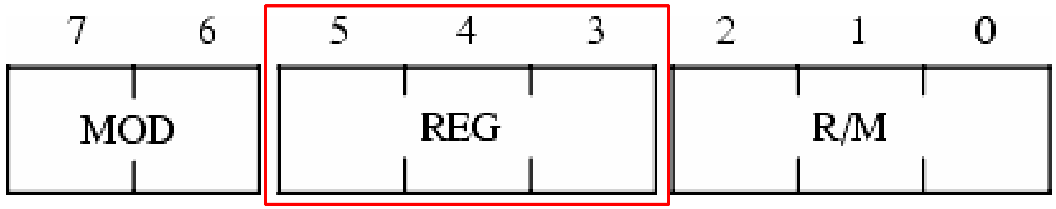


- The **MODR/M** byte (byte 2 above) specifies instruction operands and their addressing mode

- The **R/M field**, combined with **MOD**, specifies either
  - the second operand in a two-operand instruction, or
  - the only operand in a single-operand instruction like NOT or NEG.
  - These two are used to let the CPU know if there is some memory operand, if yes, then how to calculate its offset address.



- The **MOD** field specifies x86 addressing mode
- **MOD = 11** means register mode.
- **MOD = 00** means memory mode with no displacement.
  - (Except when R/M= 110, then a 16-bit displacement follows).
- **MOD= 01** means memory, with 8 bit displacement following (**D8**).
- **MOD = 10** means memory mode with 16-bit displacement following (D16).



- The **REG** field specifies source or destination **register**.
- For certain (often single-operand or immediate-operand) instructions, the **REG** field may contain an *opcode extension* rather than the register bits.
- The **R/M** field will specify the operand in such case.
- Depending on the instruction, this can be either the source or the destination operand

REG Value	Register if data size is eight bits (W = 0)	Register if data size is 16-bits (W =1)
000	al	ax
001	cl	cx
010	dl	dx
011	bl	bx
100	ah	sp
101	ch	bp
110	dh	si
111	bh	di

MOD AND R/M FIELDS

MOD=11			Effective Address Calculation			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

REGISTER-MODE INSTRUCTIONS

- In instructions using register operands, the **Mod R/M** byte contains a 3-bit identifier for each register operand.
- Bits 6 to 7 are the *mod* field, which identifies the addressing mode.
- Bits 3 to 5 are the *reg* field, which identifies the source operand.
- Bits 0 to 2 are the *r/m* field, which identifies the destination operand.

	7	6	5	4	3	2	1	0
opcode byte	11		second register			first register		

E.G.

ADD CX, AX

= 00000001 11 000 001

= **01C1** h

MEMORY MODE INSTRUCTIONS

- Intel assembly language has a wide variety of memory addressing modes, causing the encoding of the Mod R/M byte to be fairly complex.

- Exactly 256 different combinations of operands can be specified by the Mod R/M byte.

MEMORY MODE INSTRUCTIONS

- The two bits in the **Mod** column indicate groups of addressing modes.
  - Mod 00, for example, has eight possible **R/M** values (000 to 111 binary) that identify operand types listed in the **Effective Address** column.

- Encode :

1. **MOV AX, [SI] ;**

2. **MOV [SI], AL**

Encoding of registers used in indirect addressing modes

Codes	*registers*
000	[BX + SI]
001	[BX + DI]
010	[BP + SI]
011	[BP + DI]
100	[SI]
101	[DI]
110	[BP]
111	[BX]

Table 12-19 16-Bit R/M Field Values (for Mod = 10).

R/M	Effective Address
000	[BX + SI] + D16[a]
001	[BX + DI] + D16
010	[BP + SI] + D16
011	[BP + DI] + D16
100	[SI] + D16
101	[DI] + D16
110	[BP] + D16
111	[BX] + D16

[a] D16 indicates a 16-bit displacement.

Effective Address Calculation			
R/M	MOD = 00	MOD = 01	MOD = 10
000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	(SI)	(SI) + D8	(SI) + D16
101	(DI)	(DI) + D8	(DI) + D16
110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	(BX)	(BX) + D8	(BX) + D16

1. Provide opcodes for the following MOV instructions:

```
.data
myByte BYTE ?
myWord WORD ?
.code
mov  ax,@data
mov  ds,ax                                  ; a.
mov  ax,bx                                  ; b.
mov  bl,al                                  ; c.
mov  al,[si]                                ; d.
mov  myByte,al                              ; e.
mov  myWord,ax                              ; f.
```

2. Provide Mod R/M bytes for the following MOV instructions:

```
.data
array WORD 5 DUP(?)
.code
mov  ax,@data
mov  ds,ax                              ; a.
mov  dl,bl                              ; b.
mov  bl,[di]                            ; c.
mov  ax,[si+2]                          ; d.
mov  ax,array[si]                       ; e.
mov  array[di],ax                       ; f.
```

‘Mod r/m’ Byte	Mod Register r/m	Remarks
D1	**11 010 001**	**No operand in memory.**
8E	**10 001 110**	**One operand in memory; offset = one word displacement + contents of register BP.**
0C		
1E		
18		
C2		
17		
91		

SINGLE-BYTE INSTRUCTIONS

- The simplest type of instruction is one with either no operand or an implied operand. Such instructions require only the opcode field, the value of which is predetermined by the processor's instruction set.

Instruction	Opcode
AAA	37
AAS	3F
CBW	98
LODSB	AC
XLAT	D7
INC DX	42

- register increments are optimized for code size and execution speed

SINGLE IMMEDIATE DATA

- When the only operand of an instruction is an immediate data the machine language code is the opcode followed by that immediate data.

- E.g. **RET 8** is **C2 08 00,** where C2 is the opcode and 0008 is the immediate data (appended in little endian order).

REGISTER, IMMEDIATE INTRUCTIONS

- Immediate operands (constants) are appended to instructions in little-endian order (lowest byte first).
- The encoding format of a MOV instruction that moves an immediate word into a register is

B8 + *rw* *dw*

- where the opcode byte value is **B8 + *rw***, indicating that a register number (0 through 7) is added to B8
- *dw* is the immediate word operand, low byte first.

- **Example:** `MOV AX,1` The machine instruction is **B8 01 00** (hexadecimal). Here's how it is encoded:

1. The opcode for moving an immediate value to a 16-bit register is **B8**.
2. The register number for AX is 0, so 0 is added to B8
3. The immediate operand (0001) is appended to the instruction in little-endian order (01 , 00)

- **Example**: `MOV BX, 1234h` The machine instruction is **BB 34 12**. The encoding steps are as follows:

1. The opcode for moving an immediate value to a 16-bit register is **B8**.
2. The register number for BX is 3, so add 3 to B8, producing opcode **BB**.
3. The immediate operand bytes are **34 12**.

SINGLE OPERAND (IN A REGISTER) INSTRUCTIONS

- The machine language instruction can be obtained by adding to the register number to the opcode byte.
- **Example:** ***PUSH* CX** The machine instruction is **51**. The encoding steps are as follows:

1. The opcode for PUSH with a 16-bit register operand is **50.**
2. The register number for CX is 1, so add 1 to 50, producing opcode **51**.

SINGLE OPERAND (IN A MEMORY) INSTRUCTIONS

Assembly Language	Opcode	mod-register-r/m			Machine Language
INC BYTE PTR [BX][SI]	FE	00	000	**000**	FE 00

Assembly Language	Opcode	mod-register-r/m			Machine Language
IDIV WORD PTR [DI] +1A2Bh	F7	10	111	101	F7 BD 2B 1A
			= BD		
IDIV WORD PTR [SI + 2Bh]	F7	01	111	100	F7 7C 2B
			= 7C		

REG bits of R/M byte hold **opcode extension** in these instructions

MEMORY, IMMEDIATE INSTRUCTIONS

Assembly Language: SUB WORD PTR [DS:200h], 1A2Bh

Opcode	mod-register-r/m			Machine Language
81	00	101	110	81 2E 00 02 2B 1A
		= 2Eh		

Assembly Language: SUB WORD PTR [BX], 5

Opcode	mod-register-r/m			Machine Language
83	00	101	111	83 2F 05
		= 2Fh		

REG bits of R/M byte hold **opcode extension** in these instructions

MEMORY, IMMEDIATE INSTRUCTIONS

Assembly Language: SUB WORD PTR [BX] + 0E0Fh, 1A2Bh

Opcode	mod-register-r/m			Machine Language
81	10	101	111	81 AF 0F 0E 2B 1A
		= AFh		

REG bits of R/M byte hold **opcode extension** in these instructions

SUMMARY (FORMATS)

- ✓ CBW
- ✓ INT 21h
- ✓ PUSH AX
- ✓ SUB CX, 15
- ✓ ADD CL, CH
- ✓ SUB VAR, BX
- ✓ SUB BX, [101Fh]
- ✓ XOR [DI+07h], DX
- ✓ OR [SI+347Ch], Bh
- ✓ POP mem16
- ✓ INC mem8
- ✓ INC WORD PTR [100Fh]
- ✓ DEC WORD PTR [BX+02h]
- ✓ DEC WORD PTR [DI+767Fh]
- ✓ SUB VAR, 15
- ✓ SUB [SI+1fh], 15
- ✓ SUB [SI+1f1fh],15