Encoding of 8086 Instructions

! 8086 Instructions are represented as binary numbers
Instructions require between 1 and 6 bytes

Note that some architectures have fixed length instructions (particularly RISC architectures)

byte	7	6	5	4	3	2	1	0	
1	opcode						d	w	Opcode byte
2	mod		reg			r/m			Addressing mode byte
3	[optional]								low disp, addr, or data
4	[optional]								high disp, addr, or data
5	[optional]								low data
6	[optional]								high data

! This is the general instruction format used by the majority of 2-operand instructions

There are over a dozen variations of this format

! Note that bytes 1 and 2 are divided up into 6 fields:

opcode	
d	direction (or s = sign extension)
w	word/byte
mod	mode
reg	register
r/m	register/memory

Instruction Format (Cont'd)

! Instruction may also be optionally preceded by one or more prefix bytes for repeat, segment override, or lock prefixes

In 32-bit machines we also have an address size override prefix and an operand size override prefix

! Some instructions are one-byte instructions and lack the addressing mode byte

! Note the order of bytes in an assembled instruction:

[Prefix] Opcode [Addr Mode] [Low Disp] [High Disp]
[Low data] [High data]

- opcode and addressing mode are NOT stored "backwords"

Prefix Bytes

! There are four types of prefix instructions:

- Repetition
- Segment Overrides
- Lock
- Address/Operand size overrides (for 32-bit machines)

Encoded as follows (Each in a single byte)

! Repetition

REP, REPE, REPZ	F3H
REPNE, REPNZ	F2H

Note that REP and REPE and not distinct
Machine (microcode) interpretation of REP and REPE code depends on instruction currently being executed

! Segment override

CS	2EH
DS	3EH
ES	26H
SS	36H

! Lock F0H

Details on Fields
Opcode Byte

! **opcode field specifies the operation performed (mov, xchg, etc)**

! **d (direction) field specifies the direction of data movement:**

d = 1	**data moves from operand specified by R/M field to operand specified by REG field**
d = 0	**data moves from operand specified by REG field to operand specified by R/M field**

! **d position MAY be replaced by "s" bit**

s = 1	**one byte of immediate data is present which muct be sign-extended to produce a 16-bit operand**
s = 0	**two bytes of immediate are present**

! **d position is replaced by "c" bit in Shift and Rotate instructions**
indicates whether CL is used for shift count

! **w (word/byte) specifies operand size**

W = 1	**data is word**
W = 0	**data is byte**

Address and Operand Size Overrides

! Our primary focus is 16-bit instruction encoding so we will not discuss 32-bit encoding beyond this topic

We only have one bit (the w bit) for operand size so only two operand sizes can be directly specified

16-bit machines:	w=0 data is 8 bits; w=1 data is 16 bits
32-bit machines:	w=0 data is 8 bits; w=1 data is 32 bits

! Operand and Address size override prefixes are used to specify 32-registers in 16-bit code and 16-bit registers in 32-bit code

66h = operand size override
67h = address size override

! Interpretation of an instruction depends on whether it is executed in a 16-bit code segment or a 32-bit code segment

Instruction	16-bit code	32-bit code
mov ax,[bx]	8B 07	67 66 8B 07
mov eax,[bx]	66 8B 07	67 8B 07
mov ax,[ebx]	67 8B 03	66 8B 03
mov eax,[ebx]	67 66 8B 03	8B 03

Addressing Mode Byte (Byte 2)

! Contains three fields

Mod	Bits 6-7	(mode; determines how R/M field is interpreted
Reg	Bits 3-5	(register) or SREG (Seg register)
R/M	Bits 0-2	(register/memory)

! Specifies details about operands

! MOD

00	Use R/M Table 1 for R/M operand
01	Use R/M Table 2 with 8-bit displacement
10	Use R/M Table 2 with 16-bit displacement
11	Two register instruction; use REG table

!

REG	w=0	w=1	REG	w=0	w=1
000	AL	AX	100	AH	SP
001	CL	CX	101	CH	BP
010	DL	DX	110	DH	SI
011	BL	BX	111	BH	DI

! SREG

000	ES	001	CS	010	SS	110	DS

! R/M Table 1 (Mod = 00)

000	[BX+SI]	010	[BP+SI]	100	[SI]	110	Drc't Add
001	[BX+DI]	011	[BP+DI]	101	[DI]	111	[BX]

! R/M Table 2 (Mod = 01) Add DISP to register specified:

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

Addressing Mode Byte

! **In general is not present if instruction has no operands**

! **For one-operand instructions the R/M field indicates where the operand is to be found**

! **For two-operand instructions (except those with an immediate operand) one is a register determined by REG (SREG) field and the other may be register or memory and is determined by R/M field.**

Direction bit has meaning only in two-operand instructions

Indicates whether "destination" is specified by REG or by R/M

Note that this allows many instructions to be encoded in two different ways

Addressing Mode 00

! **Specifies R/M Table 1 (with NO displacement)**

000	[BX+SI]	010	[BP+SP]	100	[SI]	110	Drc't Add
001	[BX+DI]	011	[BP+DI]	101	[DI]	111	[BX]

! **Note that the 110 case (direct addressing) requires that the instruction be followed by two address bytes**

There are then two possibilities:

1	Opcode	Addressing Mode		
2	Opcode	Addressing Mode	Offset-Low	Offset-High

Examples:

```
MOV AX,[2A45]
MOV AX,[DI]
```

Addressing Mode 01

! **Specifies R/M Table 2 with 8-bit signed displacement**

000	[BX+SI+disp]	011	[BP+DI+disp]	110	[BP+disp]
001	[BX+DI+disp]	100	[SI+disp]	111	[BX+disp]
010	[BP+SI+disp]	101	[DI+disp]		

All instructions have the form:

Opcode Addressing Mode Displacement

Examples

```
MOV AX,[BP+2]
MOV DX,[BX+DI+4]
MOV [BX-4],AX
```

Addressing Mode 10

! **Specifies R/M Table 2 with 16-bit unsigned displacement**

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

Opcode	**Addressing Mode**	**Disp-Low**	**Disp-High**

Note that we cannot have negative displacements < -128!

Examples:

```
ADD AX,[BX+1000h]
```

Addressing Mode 11

! **Specifies that R/M bits refer to REG table**

All two operand register-to-register instructions use addressing mode 11

EXAMPLES:

```
MOV AX,[BX]
MOV DX,CX
MOV AH,BL
```

Encoding Examples

! POP memory/register has the structure:

    8FH   MOD 000 R/M

! Note that w = 1 always for POP (cannot pop bytes)

! To POP into AX:

    MOD = 11 (Use REG table)
    R/M = 000

Encoding: 8FH C0H

To POP into BP:

    MOD = 11
    R/M = 101

Encoding = 8FH C3H

To POP into memory location DS:1200H

    MOD = 00
    R/M = 110

Encoding = 8F 06 00 12

To POP into memory location CS:1200H

    MOD = 00
    R/M = 110

Encoding = 2E 8F 06 00 12

POP General Register

! This one-byte opcode has the structure:

    01011 REG

  So

    POP AX = 01011000 = 58H
    POP BX = 01001011 = 5BH

! Note that there are two legal encodings of POP REG

  Shorter form exists because POPs are so common

  Most assemblers will use the shorter form

POP Segment Register

! This one-byte opcode has the structure:

    00REG111 07 1f 17

  POP ES = 0000 0111 = 07H
  POP DS = 0001 1111 = 1FH
  POP SS = 0001 0111 = 17H

! Note that both forms of POP REG do not follow the general rules outlined above--registers are coded into the opcode byte

! Note also that even though POP CS is illegal, DEBUG will correctly assemble it as 0F -- but will not unassemble it.

Examples (Cont'd)

! MOV instruction has seven possible formats.
  We will not discuss them all.

MOV reg/mem,reg/mem

! This instruction has the structure:

  100010dw MOD REG R/M Disp1 Disp2

  where displacements are optional depending on the MOD bits

! MOV AX,BX
  - w = 1 because we are dealing with words
  - MOD = 11 because it is register-register

  - if d = 0 then REG = source (BX) and R/M = dest (AX)
      = 1000 1001   1101 1000 (89 D8)

  - if d = 1 then REG = source (AX) and R/M = dest (BX)
      = 1000 1011   1010 0011 (8B C3)

! MOV [BX+10h],CL
  - w = 0 because we are dealing with a byte
  - d = 0 because we need R/M Table 2 to encode [BX+10h]
   therefore first byte is (1000 1000) = 88H

  - since 10H can be encoded as an 8-bit displacement, we can
  use MOD=01 REG=001 and R/M=111 = 0100 1111 = 4FH
  and the last byte is 10H

  result: 88 4F 10

  Note: MOV [BX+10H],CX = 89 4F 10

! Can also encode MOV [BX+10h],CL with a 16-bit displacement, (MOD 10) although there is no reason to do so:

```
88 8F 10 00
```

! Note that there is no way to encode a memory-memory move

MOV reg/mem, immediate

! This instruction has the structure:

```
1100 011w        MOD 000 R/M        disp1      disp2
```

Where displacement bytes optional depending on value of MOD

MOV BYTE PTR [100H],10H
- w = 0 because we have byte operand
- MOD = 00 (R/M Table 1) R/M = 110 (Displacement)
- bytes 3 and 4 are address; byte 5 immediate data

```
C6 06 00 01 10
```

MOV accumulator,mem

- This instruction has the structure:
  1010 000w   disp1 disp2

  MOV AX,[0100]
  - w = 1 because we have word operand

  A1 00 01

- Note special form for accumulator
  Many other instructions have a short form for AX register

- Could also be assembled as:

  1000 1011 0000 0110 0000 0000 0000 0001

  8B 06 00 01

Immediate Operand Instructions

! **Immediate mode instructions have only one register or memory operand; the other is encoded in the instruction itself**

The Reg field is used an "opcode extension"
The addressing mode byte has to be examined to determine which operation is specified

add imm to reg/mem	**1000 00sw**	**mod000r/m**
or imm to reg/mem	**1000 00sw**	**mod001r/m**

! **In many instructions with immediate operands the "d" bit is interpreted as the "s" bit**

When the s bit is set it means that the single byte operand should be sign-extended to 16 bits

Example:

```
     add dx, 3     ;Add imm to reg16

1000 00sw        mod000r/m

w=1 (DX is 16 bits)    mod = 11 (use REG table)     r/m = 010 =DX

With s bit set we have
  1000 0011       11 000 010     operand = 83 C2 03
With s bit clear we have
  1000 0001       11 000 010     operand = 81 C2 03 00
```

Equivalent Machine Instructions

! The short instructions were assembled with debug's A command

The longer instructions were entered with the E command

```
1822:0100 58            POP     AX
1822:0101 8FC0          POP     AX
1822:0103 894F10        MOV     [BX+10],CX
1822:0106 898F1000      MOV     [BX+0010],CX
1822:010A A10001        MOV     AX,[0100]
1822:010D 8B060001      MOV     AX,[0100]
```

! The above examples show inefficient machine language equivalences.

There are also plenty of "efficient" equivalences where the instructions are the same length

! Eric Isaacson claims that the A86 assembler has a unique "footprint" that allows him to detect whether or not a machine language program has been assembled with A86

Instruction Format Reference

Addressing Mode Byte

MOD Field (determines how R/M operand is interpreted)

00	Use R/M Table 1 for R/M operand
01	Use R/M Table 2 with 8-bit signed displacement
10	Use R/M Table 2 with 16-bit unsigned displacement
11	Use REG table for R/M operand

REG Field **SegREG**

	w=0	w=1		w=0	w=1		
000	AL	AX	100	AH	SP	000	ES
001	CL	CX	101	CH	BP	001	CS
010	DL	DX	110	DH	SI	010	SS
011	BL	BX	111	BH	DI	011	DS

R/M Table 1 (Mod = 00)

000	[BX+SI]	010	[BP+SI]	100	[SI]	110	Direct Addr
001	[BX+DI]	011	[BP+DI]	101	[DI]	111	[BX]

R/M Table 2 (Mod = 01 or 10)

000	[BX+SI+Disp]			101	[DI+Disp]
001	[BX+DI+Disp]	011	[BP+DI+Disp]	110	[BP+Disp]
010	[BP+SI+Disp]	100	[SI+Disp]	111	[BX+Disp]

Direction Bit: 0 means data moves from REG operand to R/M operand
1 means data moves from R/M operand to REG operand
(For some instructions with immediate operands, S-bit in place of D bit means if s=1 data is sign extend 8-bit data for word operation)
Word Bit: 1 = word operands, 0 = byte operands

Repetition Prefix Codes		**Segment Override** Prefix Codes			
REP, REPE, REPZ	F3h	CS	2Eh	DS	3Eh
REPNE, REPNZ	F2h	ES	26h	SS	36h

Selected Instruction Formats

Instruction	Opcode	Addr.Mode
ADC reg/mem with reg	000100dw	modregr/m [addr]
ADC immed to reg/mem	100000sw	mod010r/m data
ADD reg/mem with reg	000000dw	modregr/m [addr]
ADD immed to accumulator	0000010w	data
ADD immed to reg/mem	100000sw	mod000r/m [addr] data
OR reg/mem with reg	000010dw	modregr/m
OR immed to reg/mem	100000sw	mod001r/m [addr] data
OR immed to accumlator	0000110w	data
INC reg16	01000reg	
INC reg/mem	1111111w	mod000r/m [addr]
MOV reg/mem to/from reg	100010dw	modregr/m [addr]
MOV reg/mem to segreg	10001110	modsegr/m (seg = segreg)
MOV immed to reg/mem	1100011w	mod000r/m [addr] data
MOV immed to reg	1011wreg	data
MOV direct mem to/from acc	101000dw	addr
XCHG reg/mem with reg	1000011w	modregr/m [addr]
XCHG reg16 with accum.	10010reg	
CMP reg/mem with reg	001110dw	modregr/m [addr]
CMP immed to accumulator	0011110w	data
CMP immed to reg/mem	100000sw	mod111r/m [addr] data
POP reg	01011reg	
POP segreg	00reg111	
POP reg/mem	10001111	modxxxr/m (xxx = don't care)
RCL reg/mem,CL/immediate	110100cw	mod010r/m [addr] (if c=0 shift= 1,
RCR reg/mem,CL/immediate	110100cw	mod011r/m [addr] if c=1 shift = CL)
STOS	1010101w	
CMPS	1010011w	
MUL reg/mem	1111011w	mod100r/m [addr]