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).

<table>
<thead>
<tr>
<th>byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>opcode</td>
<td>d</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>mod</td>
<td>reg</td>
<td>r/m</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>[optional]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>[optional]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>[optional]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>[optional]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 "backwards"
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:

\[
\begin{align*}
  d = 1 & \quad \text{data moves from operand specified by R/M field to operand specified by REG field} \\
  d = 0 & \quad \text{data moves from operand specified by REG field to operand specified by R/M field}
\end{align*}
\]

! d position MAY be replaced by "s" bit

\[
\begin{align*}
  s = 1 & \quad \text{one byte of immediate data is present which must be sign-extended to produce a 16-bit operand} \\
  s = 0 & \quad \text{two bytes of immediate are present}
\end{align*}
\]

! 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

\[
\begin{align*}
  W = 1 & \quad \text{data is word} \\
  W = 0 & \quad \text{data is byte}
\end{align*}
\]
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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>16-bit code</th>
<th>32-bit code</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov ax,[bx]</td>
<td>8B 07</td>
<td>67 66 8B 07</td>
</tr>
<tr>
<td>mov eax,[bx]</td>
<td>66 8B 07</td>
<td>67 8B 07</td>
</tr>
<tr>
<td>mov ax,[ebx]</td>
<td>67 8B 03</td>
<td>66 8B 03</td>
</tr>
<tr>
<td>mov eax,[ebx]</td>
<td>67 66 8B 03</td>
<td>8B 03</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>REG</th>
<th>w=0</th>
<th>w=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>AL</td>
<td>AX</td>
</tr>
<tr>
<td>001</td>
<td>CL</td>
<td>CX</td>
</tr>
<tr>
<td>010</td>
<td>DL</td>
<td>DX</td>
</tr>
<tr>
<td>011</td>
<td>BL</td>
<td>BX</td>
</tr>
<tr>
<td>100</td>
<td>AH</td>
<td>SP</td>
</tr>
<tr>
<td>101</td>
<td>CH</td>
<td>BP</td>
</tr>
<tr>
<td>110</td>
<td>DH</td>
<td>SI</td>
</tr>
<tr>
<td>111</td>
<td>BH</td>
<td>DI</td>
</tr>
</tbody>
</table>

**SREG**

<table>
<thead>
<tr>
<th>SREG</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>ES</td>
</tr>
<tr>
<td>001</td>
<td>CS</td>
</tr>
<tr>
<td>010</td>
<td>SS</td>
</tr>
<tr>
<td>110</td>
<td>DS</td>
</tr>
</tbody>
</table>

**R/M Table 1 (Mod = 00)**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>[BX+SI]</td>
<td>010</td>
<td>[BP+SI]</td>
</tr>
<tr>
<td>001</td>
<td>[BX+DI]</td>
<td>011</td>
<td>[BP+DI]</td>
</tr>
<tr>
<td>100</td>
<td>[SI]</td>
<td>110</td>
<td>Direct Add</td>
</tr>
</tbody>
</table>

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>[BX+SI]</td>
<td>010</td>
<td>[BP+SI]</td>
</tr>
<tr>
<td>001</td>
<td>[BX+DI]</td>
<td>011</td>
<td>[BP+DI]</td>
</tr>
<tr>
<td>100</td>
<td>[SI]</td>
<td>110</td>
<td>[BP]</td>
</tr>
<tr>
<td>101</td>
<td>[DI]</td>
<td>111</td>
<td>[BX]</td>
</tr>
</tbody>
</table>
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

\textbf{Specifies R/M Table 1 (with NO displacement)}

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Addressing Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>[BX+SI]</td>
</tr>
<tr>
<td>001</td>
<td>[BX+DI]</td>
</tr>
<tr>
<td>010</td>
<td>[BP+SI]</td>
</tr>
<tr>
<td>011</td>
<td>[BP+DI]</td>
</tr>
<tr>
<td>100</td>
<td>[SI]</td>
</tr>
<tr>
<td>101</td>
<td>[DI]</td>
</tr>
<tr>
<td>110</td>
<td>Direct Add</td>
</tr>
<tr>
<td>111</td>
<td>[BX]</td>
</tr>
</tbody>
</table>

\textbf{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:

\begin{itemize}
  \item MOV AX,[2A45]
  \item MOV AX,[DI]
\end{itemize}

Addressing Mode 01

\textbf{Specifies R/M Table 2 with 8-bit signed displacement}

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Addressing Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>[BX+SI+disp]</td>
</tr>
<tr>
<td>001</td>
<td>[BX+DI+disp]</td>
</tr>
<tr>
<td>010</td>
<td>[BP+SI+disp]</td>
</tr>
<tr>
<td>011</td>
<td>[BP+DI+disp]</td>
</tr>
<tr>
<td>100</td>
<td>[SI+disp]</td>
</tr>
<tr>
<td>101</td>
<td>[DI+disp]</td>
</tr>
<tr>
<td>110</td>
<td>[BP+disp]</td>
</tr>
<tr>
<td>111</td>
<td>[BX+disp]</td>
</tr>
</tbody>
</table>

All instructions have the form:

 Opcode Addressing Mode Displacement

Examples

\begin{itemize}
  \item MOV AX,[BP+2]
  \item MOV DX,[BX+DI+4]
  \item MOV [BX-4],AX
\end{itemize}
Addressing Mode 10

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

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Addressing Mode</th>
<th>Disp-Low</th>
<th>Disp-High</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>[BX+SI+disp]</td>
<td>011</td>
<td>[BP+DI+disp]</td>
</tr>
<tr>
<td>001</td>
<td>[BX+DI+disp]</td>
<td>100</td>
<td>[SI+disp]</td>
</tr>
<tr>
<td>010</td>
<td>[BP+SP+disp]</td>
<td>101</td>
<td>[DI+disp]</td>
</tr>
<tr>
<td>011</td>
<td>[BP+disp]</td>
<td>110</td>
<td>[BP+disp]</td>
</tr>
<tr>
<td>111</td>
<td>[BX+disp]</td>
<td>111</td>
<td>[BX+disp]</td>
</tr>
</tbody>
</table>

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:

\[ 8FH \quad \text{MOD} \quad 000 \quad \text{R/M} \]

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

! To POP into AX:
  \begin{align*}
  \text{MOD} &= 11 \quad (\text{Use REG table}) \\
  \text{R/M} &= 000 \\
  \text{Encoding} &= 8FH \quad C0H
  \end{align*}

To POP into BP:
  \begin{align*}
  \text{MOD} &= 11 \\
  \text{R/M} &= 101 \\
  \text{Encoding} &= 8FH \quad C3H
  \end{align*}

To POP into memory location DS:1200H
  \begin{align*}
  \text{MOD} &= 00 \\
  \text{R/M} &= 110 \\
  \text{Encoding} &= 8F \quad 06 \quad 00 \quad 12
  \end{align*}

To POP into memory location CS:1200H
  \begin{align*}
  \text{MOD} &= 00 \\
  \text{R/M} &= 110 \\
  \text{Encoding} &= 2E \quad 8F \quad 06 \quad 00 \quad 12
  \end{align*}
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

8086 Instruction Encoding-12
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**

<table>
<thead>
<tr>
<th>w=0</th>
<th>w=1</th>
<th>w=0</th>
<th>w=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>AL</td>
<td>010</td>
<td>AH</td>
</tr>
<tr>
<td>001</td>
<td>CL</td>
<td>101</td>
<td>CH</td>
</tr>
<tr>
<td>010</td>
<td>DL</td>
<td>110</td>
<td>DH</td>
</tr>
<tr>
<td>011</td>
<td>BL</td>
<td>111</td>
<td>BH</td>
</tr>
</tbody>
</table>

**SegREG**

<table>
<thead>
<tr>
<th>000</th>
<th>010</th>
<th>100</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>SS</td>
<td>DI</td>
<td>Direct Addr</td>
</tr>
</tbody>
</table>

**R/M Table 1 (Mod = 00)**

<table>
<thead>
<tr>
<th>000</th>
<th>010</th>
<th>100</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BX+SI]</td>
<td>[BP+SI]</td>
<td>[SI]</td>
<td>Direct Addr</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>000</th>
<th>010</th>
<th>100</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BX+DI+Disp]</td>
<td>[BP+DI+Disp]</td>
<td>[DI+Disp]</td>
<td>[BX+Disp]</td>
</tr>
</tbody>
</table>

**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**

- REP, REPE, REPZ F3h
- REPNE, REPNZ F2h

**Segment Override Prefix Codes**

- CS 2Eh  
- DS 3Eh  
- ES 26h  
- SS 36h

**Selected Instruction Formats**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Addr.Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC reg/mem with reg</td>
<td>000100dw</td>
<td>modregr/m [addr]</td>
</tr>
<tr>
<td>ADC immed to reg/mem</td>
<td>100000sw</td>
<td>mod010r/m data</td>
</tr>
<tr>
<td>ADD reg/mem with reg</td>
<td>000000dw</td>
<td>modregr/m [addr]</td>
</tr>
<tr>
<td>ADD immed to accumulator</td>
<td>0000010w</td>
<td>data</td>
</tr>
<tr>
<td>ADD immed to reg/mem</td>
<td>100000sw</td>
<td>mod000r/m [addr] data</td>
</tr>
<tr>
<td>OR reg/mem with reg</td>
<td>000010dw</td>
<td>modregr/m</td>
</tr>
<tr>
<td>OR immed to reg/mem</td>
<td>100000sw</td>
<td>mod001r/m [addr] data</td>
</tr>
<tr>
<td>OR immed to accumulator</td>
<td>0000110w</td>
<td>data</td>
</tr>
<tr>
<td>INC reg16</td>
<td>01000reg</td>
<td></td>
</tr>
<tr>
<td>INC reg/mem</td>
<td>1111111w</td>
<td>mod000r/m [addr]</td>
</tr>
<tr>
<td>MOV reg/mem to/from reg</td>
<td>100010dw</td>
<td>modregr/m [addr]</td>
</tr>
<tr>
<td>MOV reg/mem to segreg</td>
<td>10001110</td>
<td>modsegr/m (seg = segreg)</td>
</tr>
<tr>
<td>MOV immed to reg/mem</td>
<td>1100011w</td>
<td>mod000r/m [addr] data</td>
</tr>
<tr>
<td>MOV immed to reg</td>
<td>1011wreg</td>
<td>data</td>
</tr>
<tr>
<td>MOV direct mem to/from acc</td>
<td>101000dw</td>
<td>addr</td>
</tr>
<tr>
<td>XCHG reg/mem with reg</td>
<td>1000011w</td>
<td>modregr/m [addr]</td>
</tr>
<tr>
<td>XCHG reg16 with accum.</td>
<td>10010reg</td>
<td></td>
</tr>
<tr>
<td>CMP reg/mem with reg</td>
<td>001110dw</td>
<td>modregr/m [addr]</td>
</tr>
<tr>
<td>CMP immed to accumulator</td>
<td>0011110w</td>
<td>data</td>
</tr>
<tr>
<td>CMP immed to reg/mem</td>
<td>100000sw</td>
<td>mod111r/m [addr] data</td>
</tr>
<tr>
<td>POP reg</td>
<td>01011reg</td>
<td></td>
</tr>
<tr>
<td>POP segreg</td>
<td>00reg11</td>
<td></td>
</tr>
<tr>
<td>POP reg/mem</td>
<td>10001111</td>
<td>modxxxr/m (xxx = don’t care)</td>
</tr>
<tr>
<td>RCL reg/mem,CL/Immediate</td>
<td>110100cw</td>
<td>mod010r/m [addr] (if c=0 shift= 1,</td>
</tr>
<tr>
<td>RCR reg/mem,CL/Immediate</td>
<td>110100cw</td>
<td>mod011r/m [addr] if c=1 shift = CL)</td>
</tr>
<tr>
<td>STOS</td>
<td>101011w</td>
<td></td>
</tr>
<tr>
<td>CPFS</td>
<td>1010011w</td>
<td></td>
</tr>
<tr>
<td>MUL reg/mem</td>
<td>1111011w</td>
<td>mod100r/m [addr]</td>
</tr>
</tbody>
</table>