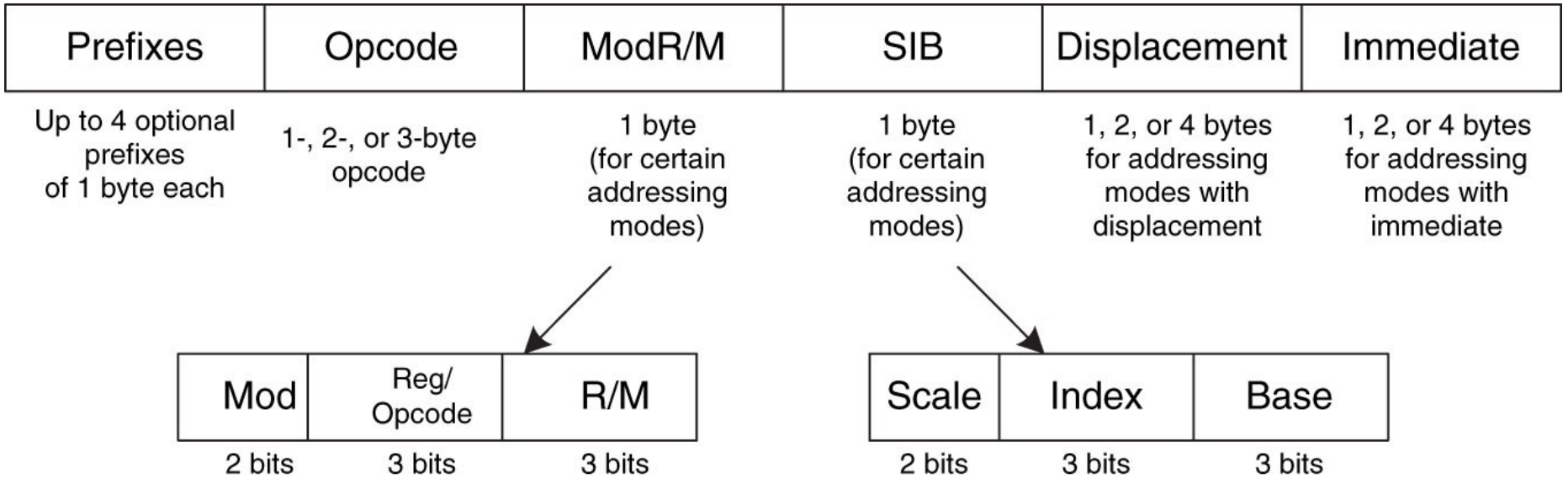
x86 Instruction Encoding

x86 Instruction Encoding

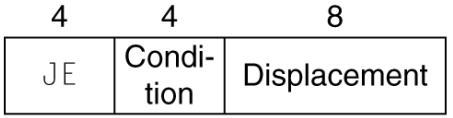
- Variable length encoding
  - Some bytes specify addressing mode
  - Some bits modify operation, e.g., operand size
  - Instruction length varies from 1 to 15 bytes



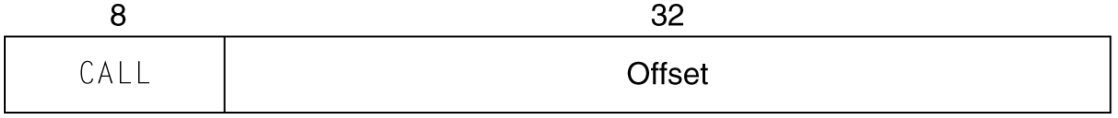
x86 Instruction Encoding Examples

JE = 74h
CALL = E8h
MOV = 8Bh
PUSH = 50h
ADD = 05h

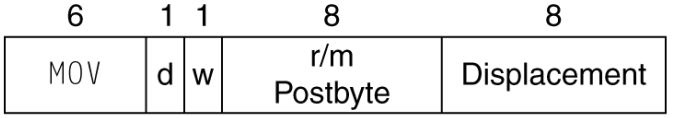
a. JE EIP + displacement



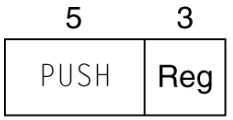
b. CALL



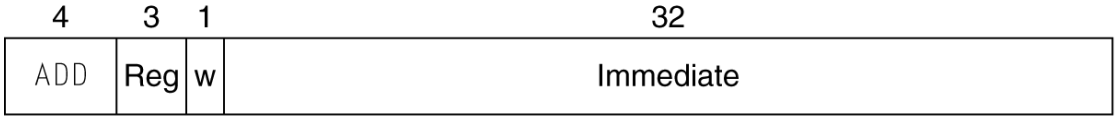
c. MOV EBX, [EDI + 45]



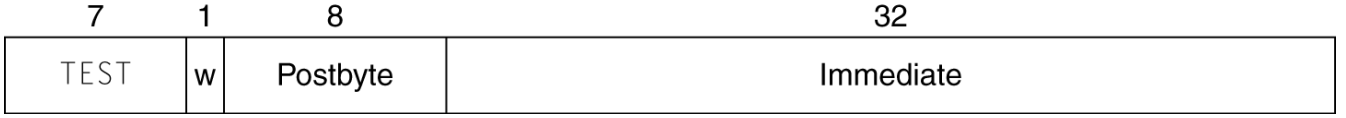
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



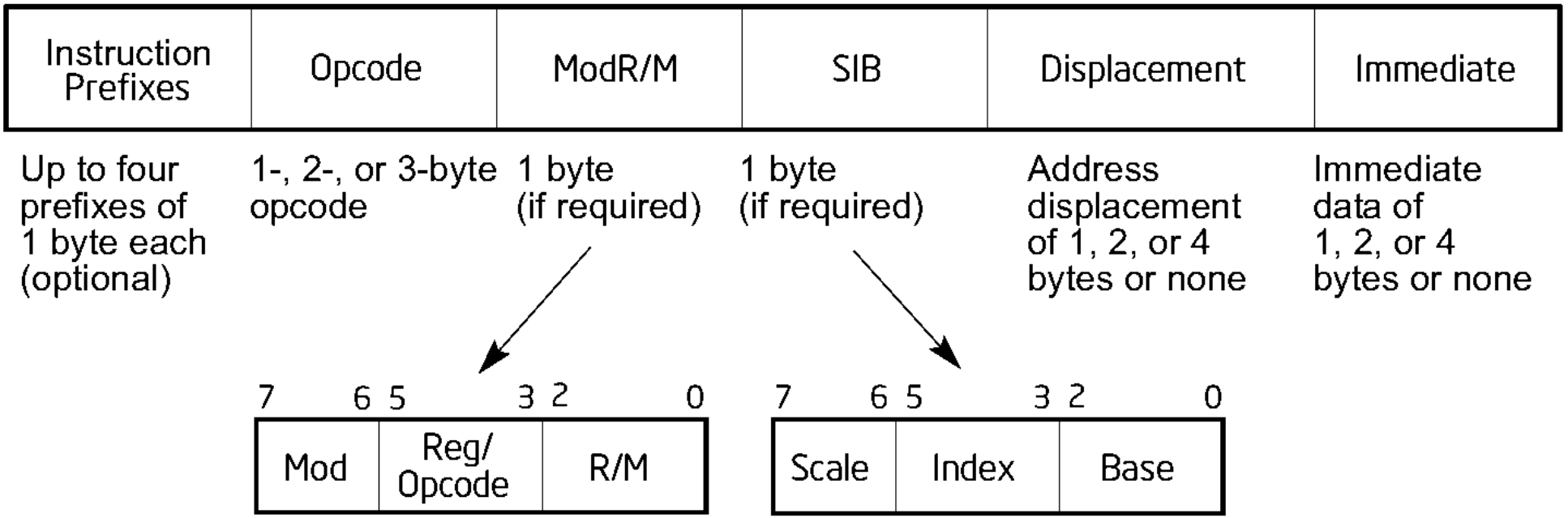

Instruction Prefixes
Opcode
ModR/M
SIB
Displacement
Immediate
Up to four prefixes of 1 byte each (optional)
1-, 2-, or 3-byte opcode
1 byte (if required)
1 byte (if required)
Address displacement of 1, 2, or 4 bytes or none
Immediate data of 1, 2, or 4 bytes or none
7 6 5 3 2 0
Mod
Reg/ Opcode
R/M
7 6 5 3 2 0
Scale
Index
Base

Table 17-3. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r)		AL	CL	DL	BL	AH	CH	DH	BH
r16(/r)		AX	CX	DX	BX	SP	BP	SI	DI
r32(/r)		EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
/digit (Opcode)		0	1	2	3	4	5	6	7
REG =		000	001	010	011	100	101	110	111
Effective Address	Mod	R/M	ModR/M Values in Hexadecimal						
[EAX]		000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--] [--]	00	100	04	0C	14	1C	24	2C	34	3C
disp32		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
disp8[EAX]		000	40	48	50	58	60	68	70	78
disp8[ECX]		001	41	49	51	59	61	69	71	79
disp8[EDX]		010	42	4A	52	5A	62	6A	72	7A
disp8[EPX];		011	43	4B	53	5B	63	6B	73	7B
disp8[--] [--]	01	100	44	4C	54	5C	64	6C	74	7C
disp8[ebp]		101	45	4D	55	5D	65	6D	75	7D
disp8[ESI]		110	46	4E	56	5E	66	6E	76	7E
disp8[EDI]		111	47	4F	57	5F	67	6F	77	7F
disp32[EAX]		000	80	88	90	98	A0	A8	B0	B8
disp32[ECX]		001	81	89	91	99	A1	A9	B1	B9
disp32[EDX]		010	82	8A	92	9A	A2	AA	B2	BA
disp32[EBX]		011	83	8B	93	9B	A3	AB	B3	BB
disp32[--] [--]	10	100	84	8C	94	9C	A4	AC	B4	BC
disp32[EBP]		101	85	8D	95	9D	A5	AD	B5	BD
disp32[ESI]		110	86	8E	96	9E	A6	AE	B6	BE
disp32[EDI]		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL		000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH	11	100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH		111	C7	CF	D7	DF	E7	EF	F7	FF

NOTES:
[--] [--] means a SIB follows the ModR/M byte. disp8 denotes an 8-bit displacement following the SIB byte, to be sign-extended and added to t index. disp32 denotes a 32-bit displacement following the ModR/M byte, be added to the index.

Table 17-4. 32-Bit Addressing Forms with the SIB Byte

r32		EAX	ECX	EDX	EBX	ESP	[*]	ESI	EDI
Base =		0	1	2	3	4	5	6	7
Base =		000	001	010	011	100	101	110	111
Scaled Index	SS	Index	ModR/M Values in Hexadecimal						
[EAX]		000	00	01	02	03	04	05	06	07
[ECX]		001	08	09	0A	0B	0C	0D	0E	0F
[EDX]		010	10	11	12	13	14	15	16	17
[EBX]		011	18	19	1A	1B	1C	1D	1E	1F
none	00	100	20	21	22	23	24	25	26	27
[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ESI]		110	30	31	32	33	34	35	36	37
[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EAX*2]		000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[ECX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
none	01	100	60	61	62	63	64	65	66	67
[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F
[ESI*2]		110	70	71	72	73	74	75	76	77
[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F
[EAX*4]		000	80	81	82	83	84	85	86	87
[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX*4]		011	98	89	9A	9B	9C	9D	9E	9F
none	10	100	A0	A1	A2	A3	A4	A5	A6	A7
[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EAX*8]		000	C0	C1	C2	C3	C4	C5	C6	C7
[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
[EBX*8]		011	D8	D9	DA	DB	DC	DD	DE	DF
none	11	100	E0	E1	E2	E3	E4	E5	E6	E7
[EBP*8]		101	E8	E9	EA	EB	EC	ED	EE	EF
[ESI*8]		110	F0	F1	F2	F3	F4	F5	F6	F7
[EDI*8]		111	F8	F9	FA	FB	FC	FD	FE	FF

NOTES:
[*] means a disp32 with no base if MOD is 00, [ESP] otherwise. This provides the following addressing modes:
disp32[index] (MOD=00)
disp8[EBP][index] (MOD=01)
disp32[EBP][index] (MOD=10)

codes are--

rb	rw	rd
AL = 0	AX = 0	EAX = 0
CL = 1	CX = 1	ECX = 1
DL = 2	DX = 2	EDX = 2
BL = 3	BX = 3	EBX = 3
AH = 4	SP = 4	ESP = 4
CH = 5	BP = 5	EBP = 5
DH = 6	SI = 6	ESI = 6
BH = 7	DI = 7	EDI = 7

+rx:

```
05 34 12 34 12          add    eax, 12341234h
81 c0 34 12 34 12       add    eax, 12341234h
```

05 opcode adds literal to EAX
81 opcode adds literal to any register, given in the second byte
Note little-endian ordering of bytes of literal

```
b8 00 00 00 00          mov    eax,0
31 c0                   xor    eax,eax
```

Equivalent instructions, but second (less obvious) form uses fewer bytes

```
b8 00 00 00 00          mov    eax,0
89 d8                   mov    eax,ebx
8b 03                   mov    eax,[ebx]
8b 43 04                mov    eax, [ebx+4]
```

Similar instructions may have very different encodings. MOV has several different opcodes

- Some terminology:
  - Register **direct** mode
    - mov **eax**, 23
  - Register **indirect** mode
    - mov **[eax]**, 23
  - Register indirect mode with **displacement**
    - mov [eax + **0x1024**], 23
  - Register indirect mode with **scale** and displacement
    - mov [eax * **3** + 0x1024], 23
  - Register indirect mode with scale, displacement, and **base**
    - mov [eax * 3 + **edx** + 0x1024], 23
  - Displacement-only mode
    - mov [0xabcd], 23
    - mov [label_name], 23

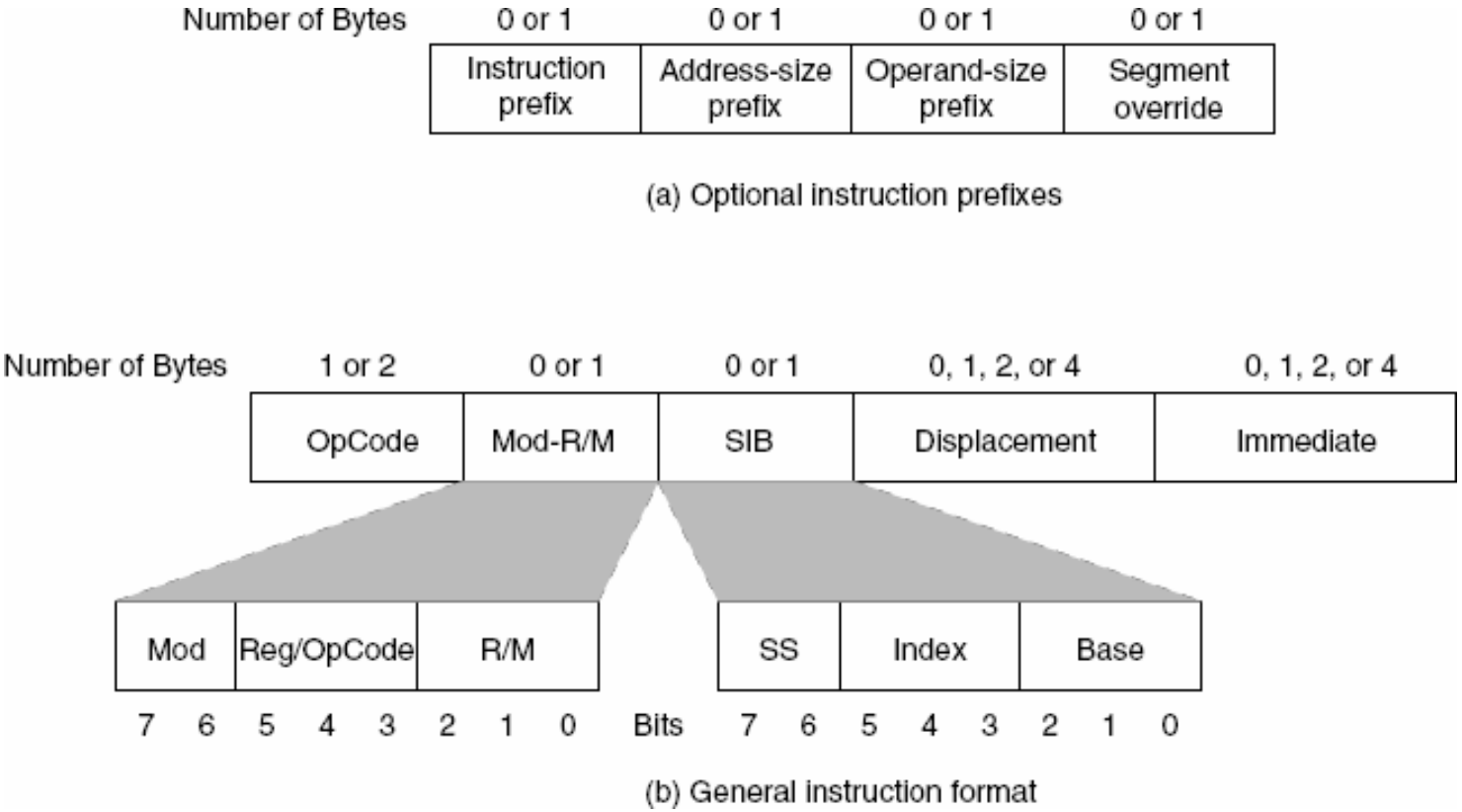
```
MOD R/M Addressing Mode
=== === ================================
 00 000 [ eax ]
 01 000 [ eax + disp8 ]                (1)
 10 000 [ eax + disp32 ]
 11 000 register  ( al / ax / eax )    (2)
 00 001 [ ecx ]
 01 001 [ ecx + disp8 ]
 10 001 [ ecx + disp32 ]
 11 001 register  ( cl / cx / ecx )
 00 010 [ edx ]
 01 010 [ edx + disp8 ]
 10 010 [ edx + disp32 ]
 11 010 register  ( dl / dx / edx )
 00 011 [ ebx ]
 01 011 [ ebx + disp8 ]
 10 011 [ ebx + disp32 ]
 11 011 register  ( bl / bx / ebx )
 00 100 SIB  Mode                      (3)
 01 100 SIB  +  disp8  Mode
 10 100 SIB  +  disp32  Mode
 11 100 register  ( ah / sp / esp )
 00 101 32-bit Displacement-Only Mode (4)
 01 101 [ ebp + disp8 ]
 10 101 [ ebp + disp32 ]
 11 101 register  ( ch / bp / ebp )
 00 110 [ esi ]
 01 110 [ esi + disp8 ]
 10 110 [ esi + disp32 ]
 11 110 register  ( dh / si / esi )
 00 111 [ edi ]
 01 111 [ edi + disp8 ]
 10 111 [ edi + disp32 ]
 11 111 register  ( bh / di / edi )
```

```
[ reg32 + eax*n ] MOD = 00
[ reg32 + ebx*n ]
[ reg32 + ecx*n ]
[ reg32 + edx*n ]
[ reg32 + ebp*n ]
[ reg32 + esi*n ]
[ reg32 + edi*n ]

[ disp + reg8 + eax*n ] MOD = 01
[ disp + reg8 + ebx*n ]
[ disp + reg8 + ecx*n ]
[ disp + reg8 + edx*n ]
[ disp + reg8 + ebp*n ]
[ disp + reg8 + esi*n ]
[ disp + reg8 + edi*n ]

[ disp + reg32 + eax*n ] MOD = 10
[ disp + reg32 + ebx*n ]
[ disp + reg32 + ecx*n ]
[ disp + reg32 + edx*n ]
[ disp + reg32 + ebp*n ]
[ disp + reg32 + esi*n ]
[ disp + reg32 + edi*n ]

[ disp + eax*n ] MOD = 00, and
[ disp + ebx*n ] BASE field = 101
[ disp + ecx*n ]
[ disp + edx*n ]
[ disp + ebp*n ]
[ disp + esi*n ]
[ disp + edi*n ]
```



Mod-R/M = Mode Register/Memory
SIB = Scaled Index Byte
We are going to ignore most prefixes

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
04 *ib*	ADD AL, *imm8*	I	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	I	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	I	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	MI	Valid	Valid	Add *imm8* to *r/m8.*
REX + 80 /0 *ib*	ADD *r/m8**, *imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m8.*
81 /0 *iw*	ADD *r/m16, imm16*	MI	Valid	Valid	Add *imm16* to *r/m16.*
81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32.*
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	MI	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64.*
83 /0 *ib*	ADD *r/m16, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m16.*
83 /0 *ib*	ADD *r/m32, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m32.*
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64.*
00 /*r*	ADD *r/m8, r8*	MR	Valid	Valid	Add *r8* to *r/m8.*
REX + 00 /*r*	ADD *r/m8**, *r8**	MR	Valid	N.E.	Add *r8* to *r/m8.*
01 /*r*	ADD *r/m16, r16*	MR	Valid	Valid	Add *r16* to *r/m16.*
01 /*r*	ADD *r/m32, r32*	MR	Valid	Valid	Add r32 to *r/m32.*
REX.W + 01 /*r*	ADD *r/m64, r64*	MR	Valid	N.E.	Add r64 to *r/m64.*
02 /*r*	ADD *r8, r/m8*	RM	Valid	Valid	Add *r/m8* to *r8.*
REX + 02 /*r*	ADD *r8**, *r/m8**	RM	Valid	N.E.	Add *r/m8* to *r8.*
03 /*r*	ADD *r16, r/m16*	RM	Valid	Valid	Add *r/m16* to *r16.*
03 /*r*	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32.*
REX.W + 03 /*r*	ADD *r64, r/m64*	RM	Valid	N.E.	Add *r/m64* to *r64.*

Interpreting encoding rules

81 /0 id — ADD r/m32, imm32

This opcode is for storing a 32-bit immediate value into a 32-bit register *or* into a memory address.
The first byte must be 81
The "/0" means "use a Mod-R/M byte, where the Reg field is 0."
A Mod-R/M byte has three fields:
2-bit mode; 3-bit Reg; 3-bit R/M
The "id" means "immediate dword."
You'll also see "/r" which means "use a Mod-R/M byte, where the Reg field indicates a register operand" (according to the table at right)

Mod	Meaning

00	if R/M==100:
	Register indirect mode, SIB and displacement follows after
	Mod-R/M
	elif R/M==101:
	displacement-only mode
	address follows Mod-R/M
	else:
	Register indirect mode, no
	displacement, no SIB

01	Register indirect mode, 1-byte displacement after Mod-R/M (and SIB if R/M==100)
10	Register indirect mode, 4-byte displacement after Mod-R/M (and SIB if R/M==100)
11	Register direct mode

B8+ rd id — MOV r32, imm32

This opcode is for storing a 32-bit immediatevalue into a 32-bit register.
The "+" means that the first byte is the sum of B8 and a 3-bit dword register indicator (see table on right).
That byte is followed by an immediate dword.

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.

Only for adding an immediate value into EAX. For example:

```
05 33 33 ef be              add    eax,0xbeef3333
```

81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.

For adding an immediate value into any register or memory location.

```
81 c3 33 33 ef be                     add  ebx,0xbeef3333
81 c0 33 33 ef be                     add  eax,0xbeef3333
81 05 22 00 00 00 33 33 ef be         add  [my_var],0xbeef3333
81 02 33 33 ef be                     add  [edx],0xbeef3333
```

03 /*r*	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32*.

For adding a register or memory value into another register.

```
03 05 22 00 00 00        add    eax,[my_var]
03 03                    add    eax,[ebx]
03 d3                    add    edx,ebx
```

Memory accesses using a register can also be modified arithmetically within the instruction, in certain limited situations:

- the address can be multiplied by 1, 2, 4, or 8 (the **Scale**)
- the address can be added to an 8- or 32-bit value (the **Displacement**)

This is only valid within memory dereferencing operands.

81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.

For adding an immediate value into any register or memory location.

```
81 02 33 33 ef be                   add [edx],0xbeef3333
81 82 cd ab cd 00 33 33 ef be       add [edx+0xcdabcd], 0xbeef3333
81 04 95 00 00 00 00 33 33 ef be    add [edx*4], 0xbeef3333
```

03 /r	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32*.

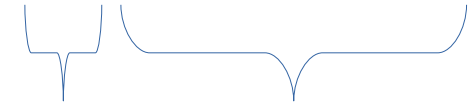
For adding a register or memory value into another register.

```
03 05 22 00 00 00          add    eax,[my_var]
03 43 01                   add    eax,[ebx+1]
03 04 5d 00 00 00 00       add    eax,[ebx*2]
03 04 5d 01 00 00 00       add    eax,[ebx*2+0x1]
```

05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.

Only for adding an immediate value into EAX. For example:

```
05 33 33 ef be          add    eax,0xbeef3333
```



opcode immediate

The encoding is just the opcode (0x05) followed by the immediate data in little-endian.

81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.

For adding an immediate value into any register or memory location.

```
81 c3 33 33 ef be                    add  ebx,0xbeef3333
```

opcode Mod-R/M immediate

The encoding is the opcode (0x81) followed by the Mod-R/M byte (0xc3), then the immediate data in little-endian. The instruction format (specifically, the "/0") tells us that the Reg field of the Mod-R/M byte must be zero.

```
0xc3 = 11   000   011
       Mod  Reg   R/M
```

We're storing into a register

Required by opcode

Target is ebx

Mod	Meaning

00	if R/M==100:
	  Register indirect mode, SIB and displacement follows after Mod-R/M
	elif R/M==101:
	  displacement-only mode address follows Mod-R/M
	else:
	  Register indirect mode, no displacement, no SIB

01	Register indirect mode, 1-byte displacement after Mod-R/M (and SIB if R/M==100)
10	Register indirect mode, 4-byte displacement after Mod-R/M (and SIB if R/M==100)
11	Register direct mode

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.

For adding an immediate value into any register or memory location.

```
81 82 cd ab cd 00 33 33 ef be    add [edx+0xcdabcd], 0xbeef3333
```

opcode Mod-R/M displacement immediate

```
0x82 = 10  000  010
       Mod Reg  R/M
```

We're storing into an address given by a register and displacement

Required by opcode

Target is edx

The encoding is the opcode (0x81) followed by the Mod-R/M byte (0x82), then the 4-byte displacement (as specified in the Mod field), then the immediate data in little-endian. The instruction format (specifically, the "/0") tells us that the Reg field of the Mod-R/M byte must be zero.

Mod	Meaning

00	if R/M==100:
	  Register indirect mode, SIB and displacement follows after Mod-R/M
	elif R/M==101:
	  displacement-only mode address follows Mod-R/M
	else:
	  Register indirect mode, no displacement, no SIB

01	Register indirect mode, 1-byte displacement after Mod-R/M (and SIB if R/M==100)
10	Register indirect mode, 4-byte displacement after Mod-R/M (and SIB if R/M==100)
11	Register direct mode

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.

For adding an immediate value into any register or memory location.

```
81 05 cd ab cd 00 33 33 ef be    add [0xcdabcd], 0xbeef3333
```

opcode Mod-R/M displacement immediate

The encoding is the opcode (0x81) followed by the Mod-R/M byte (0x05), then the 4-byte displacement (as specified in the Mod field), then the immediate data in little-endian. The instruction format (specifically, the "/0") tells us that the Reg field of the Mod-R/M byte must be zero.

```
0x05 = 00  000  101
       Mod Reg  R/M
```

Displacement only

Required by opcode

Displacement only

Mod	Meaning

00	if R/M==100:
	  Register indirect mode, SIB and displacement follows after Mod-R/M
	elif R/M==101:
	  displacement-only mode address follows Mod-R/M
	else:
	  Register indirect mode, no displacement, no SIB

01	Register indirect mode, 1-byte displacement after Mod-R/M (and SIB if R/M==100)
10	Register indirect mode, 4-byte displacement after Mod-R/M (and SIB if R/M==100)

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.

For adding an immediate value into any register or memory location.

```
81 04 95 aa 00 00 00 33 33 ef be     add [edx*4+0xaa], 0xbeef3333
```

opcode | Mod-R/M | SIB | displacement | immediate

```
0x04 = 00  000  100
       Mod Reg  R/M
```

```
0x95 = 10     010    101
       Scale  Index  Base
```

4 * edx + 0

SIB follows after Mod-R/M

Required by opcode

SIB follows after Mod-R/M

Scale	Meaning
00	Index * 1
01	Index * 2
10	Index * 4
11	Index * 8

Base	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	displacement if Mod=0, else ebp
110	esi
111	edi

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.

For adding an immediate value into any register or memory location.

```
81 84 13 aa 00 00 00 33 33 ef be  add [edx+ebx+0xaa], 0xbeef3333
```

opcode | Mod-R/M | SIB | displacement | immediate

```
0x84 = 10  000  100
       Mod Reg  R/M
```

SIB and displacement follows after Mod-R/M

Required by opcode

SIB follows after Mod-R/M

```
0x13 = 00     010    011
       Scale  Index  Base
```

1 * edx + ebx

Scale	Meaning
00	Index * 1
01	Index * 2
10	Index * 4
11	Index * 8

Base	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	displacement if Mod=0, else ebp
110	esi
111	edi

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

MOV — Move

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
88 /r	MOV *r/m8,r8*	MR	Valid	Valid	Move *r8* to *r/m8*.
REX + 88 /r	MOV *r/m8****,*r8****	MR	Valid	N.E.	Move *r8* to *r/m8*.
89 /r	MOV *r/m16,r16*	MR	Valid	Valid	Move *r16* to *r/m16*.
89 /r	MOV *r/m32,r32*	MR	Valid	Valid	Move *r32* to *r/m32*.
REX.W + 89 /r	MOV *r/m64,r64*	MR	Valid	N.E.	Move *r64* to *r/m64*.
8A /r	MOV *r8,r/m8*	RM	Valid	Valid	Move *r/m8* to *r8*.
REX + 8A /r	MOV *r8***,*r/m8****	RM	Valid	N.E.	Move *r/m8* to *r8*.
8B /r	MOV *r16,r/m16*	RM	Valid	Valid	Move *r/m16* to *r16*.
8B /r	MOV *r32,r/m32*	RM	Valid	Valid	Move *r/m32* to *r32*.
REX.W + 8B /r	MOV *r64,r/m64*	RM	Valid	N.E.	Move *r/m64* to *r64*.
8C /r	MOV *r/m16,Sreg***	MR	Valid	Valid	Move segment register to *r/m16*.
REX.W + 8C /r	MOV *r16/r32/m16, Sreg***	MR	Valid	Valid	Move zero extended 16-bit segment register to *r16/r32/r64/m16*.
REX.W + 8C /r	MOV *r64/m16, Sreg***	MR	Valid	Valid	Move zero extended 16-bit segment register to *r64/m16*.
8E /r	MOV *Sreg,r/m16***	RM	Valid	Valid	Move *r/m16* to segment register.
REX.W + 8E /r	MOV *Sreg,r/m64***	RM	Valid	Valid	Move *lower 16 bits of r/m64* to segment register.
A0	MOV AL,*moffs8**	FD	Valid	Valid	Move byte at (*seg:offset*) to AL.
REX.W + A0	MOV AL,*moffs8**	FD	Valid	N.E.	Move byte at (*offset*) to AL.
A1	MOV AX,*moffs16**	FD	Valid	Valid	Move word at (*seg:offset*) to AX.
A1	MOV EAX,*moffs32**	FD	Valid	Valid	Move doubleword at (*seg:offset*) to EAX.
REX.W + A1	MOV RAX,*moffs64**	FD	Valid	N.E.	Move quadword at (*offset*) to RAX.
A2	MOV *moffs8*,AL	TD	Valid	Valid	Move AL to (*seg:offset*).
REX.W + A2	MOV *moffs8****,AL	TD	Valid	N.E.	Move AL to (*offset*).
A3	MOV *moffs16**,AX	TD	Valid	Valid	Move AX to (*seg:offset*).
A3	MOV *moffs32**,EAX	TD	Valid	Valid	Move EAX to (*seg:offset*).
REX.W + A3	MOV *moffs64**,RAX	TD	Valid	N.E.	Move RAX to (*offset*).
B0+ *rb ib*	MOV *r8, imm8*	OI	Valid	Valid	Move *imm8* to *r8*.
REX + B0+ *rb ib*	MOV *r8****, *imm8*	OI	Valid	N.E.	Move *imm8* to *r8*.
B8+ *rw iw*	MOV *r16, imm16*	OI	Valid	Valid	Move *imm16* to *r16*.
B8+ *rd id*	MOV *r32, imm32*	OI	Valid	Valid	Move *imm32* to *r32*.
REX.W + B8+ *rd io*	MOV *r64, imm64*	OI	Valid	N.E.	Move *imm64* to *r64*.
C6 */0 ib*	MOV *r/m8, imm8*	MI	Valid	Valid	Move *imm8* to *r/m8*.
REX + C6 */0 ib*	MOV *r/m8***, imm8*	MI	Valid	N.E.	Move *imm8* to *r/m8*.
C7 */0 iw*	MOV *r/m16, imm16*	MI	Valid	Valid	Move *imm16* to *r/m16*.
C7 */0 id*	MOV *r/m32, imm32*	MI	Valid	Valid	Move *imm32* to *r/m32*.
REX.W + C7 */0 id*	MOV *r/m64, imm32*	MI	Valid	N.E.	Move *imm32 sign extended to 64-bits* to *r/m64*.

8B /r	MOV r32,r/m32	RM	Valid	Valid	Move r/m32 to r32.

```
8b d9                mov ebx, ecx
```

opcode Mod-R/M

```
0xd9 = 11   011   001
       Mod  Reg   R/M
```

We're storing into a register

Target is ebx

Source is ecx

Mod	Meaning

00	if R/M==100:
	  Register indirect mode, SIB and displacement follows after Mod-R/M
	elif R/M==101:
	  displacement-only mode address follows Mod-R/M
	else:
	  Register indirect mode, no displacement, no SIB

01	Register indirect mode, 1-byte displacement after Mod-R/M (and SIB if R/M==100)
10	Register indirect mode, 4-byte displacement after Mod-R/M (and SIB if R/M==100)
11	Register direct mode

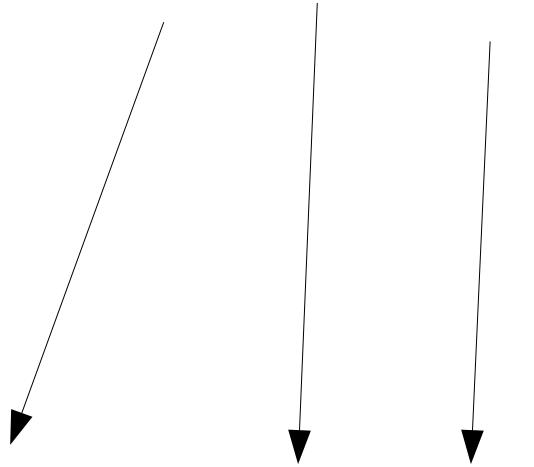
Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

8B /r	MOV r32,r/m32	RM	Valid	Valid	Move r/m32 to r32.

```
8b 19              mov ebx, [ecx]
```

opcode Mod-R/M


```
0x19 = 00  011  001
       Mod Reg  R/M
```



Simple register indirect mode

Target is ebx

Source is ecx

Mod	Meaning

00	if R/M==100:
	  Register indirect mode, SIB and displacement follows after Mod-R/M
	elif R/M==101:
	  displacement-only mode address follows Mod-R/M
	else:
	  Register indirect mode, no displacement, no SIB

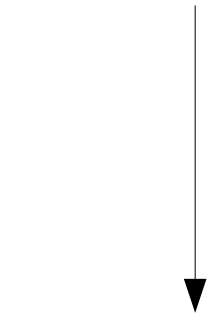
01	Register indirect mode, 1-byte displacement after Mod-R/M (and SIB if R/M==100)
10	Register indirect mode, 4-byte displacement after Mod-R/M (and SIB if R/M==100)
11	Register direct mode

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

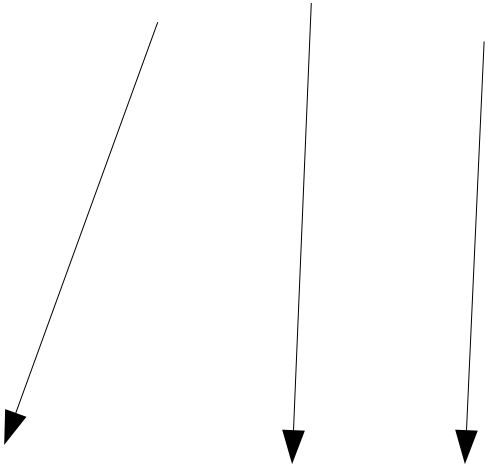
89 /r	MOV r/m32,r32	MR	Valid	Valid	Move r32 to r/m32.

```
89 19                mov [ecx], ebx
```

opcode Mod-R/M



```
0x19 = 00  011  001
       Mod Reg  R/M
```



Simple register indirect mode

Source is ebx

Target is ecx

Mod	Meaning

00	if R/M==100:
	  Register indirect mode, SIB and displacement follows after Mod-R/M
	elif R/M==101:
	  displacement-only mode address follows Mod-R/M
	else:
	  Register indirect mode, no displacement, no SIB

01	Register indirect mode, 1-byte displacement after Mod-R/M (and SIB if R/M==100)
10	Register indirect mode, 4-byte displacement after Mod-R/M (and SIB if R/M==100)
11	Register direct mode

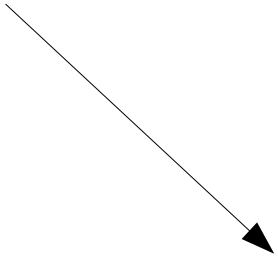
Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi

B8+ *rd id*	MOV *r32, imm32*	OI	Valid	Valid	Move *imm32* to *r32*.

```
B9 BB 32 00 00          mov ecx, 0x32bb
```

opcode + dword register

32-bit dword immediate



```
  B8 = 10111 000
+ 01 =       001
  B9 = 10111 001
```

Reg or R/M	Register
000	eax
001	ecx
010	edx
011	ebx
100	esp
101	ebp
110	esi
111	edi