

Opcode

어셈명령어로 변환하기

Intel Manual



이강석
certlab@gmail.com

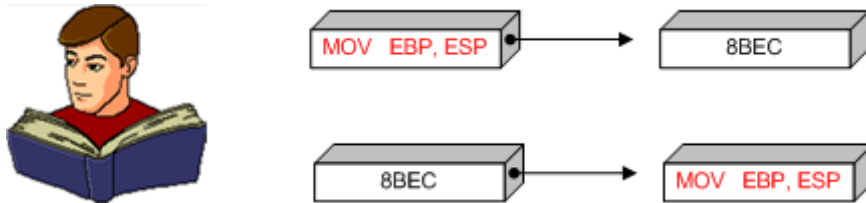
2006. 01.

어셈블리어 개발자 그룹 :: 어셈러브

<http://www.asmlove.co.kr>

#1. 문서의 처음

이 문서에는 다른 플랫폼은 제외하고 인텔 기반위주로 문서를 작성하였고, 문서의 흐름은 디어셈블하면서 볼수 있는 Opcode를 보고, IA-32 Instruction Format을 본 후에 인텔 매뉴얼을 참조하면서 OpCode → 어셈블리어 명령으로 변환하는 순서로 작성했습니다.



OpCode 는 Operation Code의 축약어이며, 프로세서가 해석해서 수행할수 있는 문자 그대로의 명령입니다. 또한, OpCode는 각 어셈블리어 명령에 대해서 1:1로 정확히 대응이 됩니다. ex) OpCode 55는 "push ebp"

단, 같은 add 명령이나 같은 mov 명령뒤의 Operand에 따라 OpCode가 당연히 틀려지겠죠?

ex) mov eax, ebx mov ebx, eax ← OpCode 달라지게 됨.

Opcode를 어셈블리 명령으로 변환하는일은 컴퓨터가 알아서 해주고, 쓸일은 없지만 이렇게도 볼수 있구나 라는 방법을 제시하는 문서입니다.

1 : 문서의 처음&소개	- 2 -
2. 흔히 볼수 있는 OpCode	- 3 -
3. IA-32 Instruction Format	- 5 -
4. opcode → 어셈명령으로 변환	- 6 -
- Opcode map (00H - F7H)	- 7 -
- Opcode map (08H - FFH)	- 8 -
- 32-Bit Addressing Forms with the ModR/M Byte	- 11 -
- 32-Bit Addressing Forms with the SIB Byte	- 12 -

#2. 흔히 볼수 있는 OpCode

다음은 IDA에서 디어셈블한 모습입니다.

```

start      55                push    ebp
start+1    8B EC                mov     ebp, esp
start+3    6A FF                push    0FFFFFFFh
start+5    68 28 DC 48 00        push    offset unk_48DC28
start+A    68 00 6C 47 00        push    offset unk_476C00
start+F    64 A1 00 00 00 00    mov     eax, large fs:0
start+15   50                    push    eax
start+16   64 89 25 00 00 00    mov     large fs:0, esp
start+1D   83 EC 68                sub     esp, 68h           ; Integer Subtraction
start+20   53                    push    ebx
start+21   56                    push    esi
start+22   57                    push    edi
start+23   89 65 E8                mov     [ebp+var_18], esp
start+26   33 DB                xor     ebx, ebx           ; Logical Exclusive OR
start+28   89 5D FC                mov     [ebp+var_4], ebx
start+2B   6A 02                push    2
    
```

다음은 OllyDBG에서 디어셈블한 모습입니다.

00401110	55	PUSH EBP	
00401111	8BEC	MOV EBP,ESP	
00401113	6AFF	PUSH -1	
00401115	68 90404000	PUSH a.00404090	
0040111A	68 D81B4000	PUSH a.00401BD8	SE handler installation
0040111F	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
00401125	50	PUSH EAX	
00401126	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
0040112D	83EC 10	SUB ESP,10	
00401130	53	PUSH EBX	
00401131	56	PUSH ESI	
00401132	57	PUSH EDI	
00401133	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
00401136	FF15 04404000	CALL DWORD PTR DS:[<KERNEL32.GetVersion	kernel32.GetVersion
0040113C	33D2	XOR EDX,EDX	
0040113E	8AD4	MOV DL,AH	
00401140	8915 A4524000	MOV DWORD PTR DS:[405244],EDX	
00401146	8BC8	MOV ECX,EAX	
00401148	81E1 FF000000	AND ECX,0FF	

다음은 Visual Studio 6.0 에서의 Debug mode

```

2:      {
  00401010 55                push    ebp
  00401011 8B EC                mov     ebp,esp
  00401013 81 EC A4 00 00 00    sub     esp,0A4h
  00401019 53                    push    ebx
  0040101A 56                    push    esi
  0040101B 57                    push    edi
  0040101C 8D BD 5C FF FF FF    lea    edi,[ebp-0A4h]
  00401022 B9 29 00 00 00        mov     ecx,29h
  00401027 B8 CC CC CC CC        mov     eax,0CCCCCCCCh
  0040102C F3 AB                rep stos dword ptr [edi]

3:      char buffer[100];
4:      strcpy(buffer, argv[1]);
  0040102E 8B 45 0C                mov     eax,dword ptr [ebp+0Ch]
  00401031 8B 48 04                mov     ecx,dword ptr [eax+4]
  00401034 51                    push    ecx
  00401035 8D 55 9C                lea    edx,[ebp-64h]
    
```

다음은 PE Explorer 에서의 디어셈블 모습

```

00401110                               EntryPoint:
00401110 55                                     push    ebp
00401111 8BEC                                  mov     ebp,esp
00401113 6AFF                                  push    FFFFFFFFh
00401115 6890404000                            push    L00404090
0040111A 68D81B4000                            push    L00401BD8
0040111F 64A100000000                          mov     eax,fs:[00000000h]
00401125 50                                     push    eax
00401126 64892500000000                        mov     fs:[00000000h],esp
0040112D 83EC10                                  sub     esp,00000010h
00401130 53                                     push    ebx
00401131 56                                     push    esi
00401132 57                                     push    edi
00401133 8965E8                                  mov     [ebp-18h],esp
00401136 FF1504404000                            call   [KERNEL32.dll!*GetVersion]
0040113C 33D2                                  xor     edx,edx
  
```

다음은 W32Dasm 디어셈블 모습

```

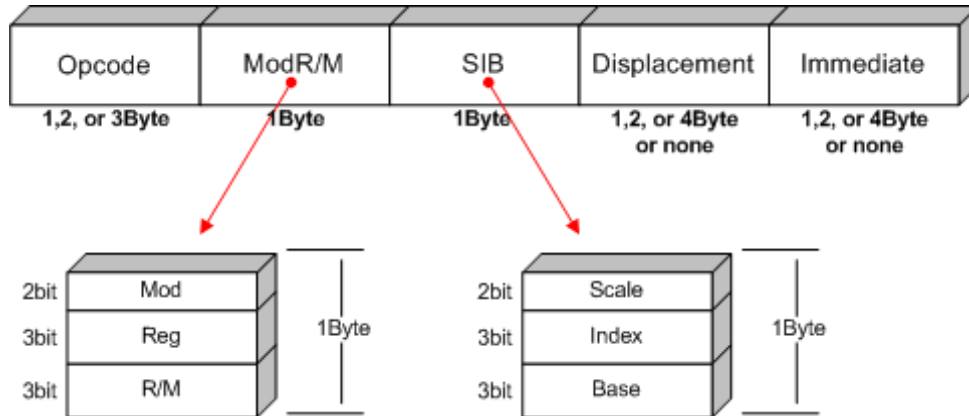
//***** Program Entry Point *****
:00401110 55                                     push ebp
:00401111 8BEC                                  mov ebp, esp
:00401113 6AFF                                  push FFFFFFFF
:00401115 6890404000                            push 00404090
:0040111A 68D81B4000                            push 00401BD8
:0040111F 64A100000000                          mov eax, dword ptr fs:[00000000]
:00401125 50                                     push eax
:00401126 64892500000000                        mov dword ptr fs:[00000000], esp
:0040112D 83EC10                                  sub esp, 00000010
:00401130 53                                     push ebx
:00401131 56                                     push esi
:00401132 57                                     push edi
:00401133 8965E8                                  mov dword ptr [ebp-18], esp
  
```

여기까지 서로 다른 모습의 디어셈블 화면들을 보면서 OpCode가 어셈 명령어와 1:1로 대응된것을 볼수 있습니다.

여기서 조금 의문점이 드는 분들이 있으실 것입니다.
왜 OpCode 55가 push ebp 인것이지?

#3. IA-32 Instruction Format

다음은 IA-32 Instruction Format입니다.










지금부터 인텔 매뉴얼을 보면서 변환을 할것입니다.

인텔 매뉴얼은 인텔 사이트에 있지만 아래 어셈러브에서 다운받으시는게 편하실 것 같네요.

어셈러브(<http://www.asmlove.co.kr>)

자료실 -> IA-32 Intel® Architecture Software Developer's Manual

<http://asmlove.co.kr/zBdC7/viewtopic.php?t=487&sid=1e263759ff268eeff62afd3b4921c2fa>

<ul style="list-style-type: none">  24896613.pdf  25204616.pdf  25366519.pdf  25366619.pdf  25366719.pdf  25366819.pdf  25366919.pdf 	<p>위 자료실 링크로 가시면 왼쪽 스냅샷과 같이 7개의 파일을 다운로드 할수 있습니다. OpCode 변환에 관련된 매뉴얼은 아래 두 개의 파일만 있으면 됩니다.</p> <p>25366619.pdf -> Volume 2A : Instruction Set Reference, A-M</p> <p>25366719.pdf -> Volume 2A : Instruction Set Reference, N-Z</p>
--	--

#4. OpCode -> 어셈명령 변환

인텔 매뉴얼 찾는 부분은 페이지 수가 거의 500~700페이지들이 넘는 문서들이기 때문에 찾아가는 부분을 자세히 설명하겠습니다.

이제 드디어 궁금증을 푸는 시간입니다.

우선 간단히 Opcode “55” 를 어셈블리어 명령으로 변환해보도록 하겠습니다.

우선 55 하나만 있는 상태 이니 다음 구조에서 Opcode 부분만 해당이 되니 인텔 매뉴얼의 Opcode map을 보면 되겠습니다.



55가 과연 무엇인지 인텔 매뉴얼을 찾아보겠습니다.

25366719.pdf 파일을 열어주세요. -> [Volume 2A : Instruction Set Reference, N-Z](#)

	<p>왼쪽 스냅샷과 같이 423페이지로 가시거나</p>
<p>Vol. 2B A-9</p>	<p>왼쪽 스냅샷과 같이 문서의 하단이나 상단에 나와있는 A-9 페이지로 갑니다.</p>

다음 테이블에서 가로 라인에서 5 세로라인에서 5를 찾아보면
PUSH rBP/r13 이라고 나와있는것을 확인할수 있습니다. 끝입니다. 정말 쉽죠?

Table A-2. One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
0	ADD Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAX, Iz						PUSH ES ⁱ⁶⁴	POP ES ⁱ⁶⁴
1	ADC Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAX, Iz						PUSH SS ⁱ⁶⁴	POP SS ⁱ⁶⁴
2	AND Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAX, Iz						SEG=ES (Prefix)	DAA ⁱ⁶⁴
3	XOR Eb, Gb Ev, Gv Gb, Eb Gv, Ev AL, Ib rAX, Iz						SEG=SS (Prefix)	AAA ⁱ⁶⁴
4	INC ⁱ⁶⁴ general register / REX ^{o64} Prefixes eAX REX eCX REX.B eDX REX.X eBX REX.XB eSP REX.R eBP REX.RB eSI REX.RX eDI REX.RXB							
5	PUSH ^{o64} general register rAX/r8 rCX/r9 rDX/r10 rBX/r11 rSP/r12 rBP/r13 rSI/r14 rDI/r15							
6	PUSHA ⁱ⁶⁴ PUSHAD ⁱ⁶⁴	POPA ⁱ⁶⁴ POPAD ⁱ⁶⁴	BOUND ⁱ⁶⁴ Gv, Ma	ARPL ⁱ⁶⁴ Ew, Gw MOVSD ^{o64} Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
7	Jcc ⁱ⁶⁴ , Jb - Short-displacement jump on condition O NO B/NAE/C NB/AE/NC Z/E NZ/NE BE/NA NBE/A							
8	Immediate Grp 1 ^{1A} Eb, Ib Ev, Iz Eb, Ib ⁱ⁶⁴ Ev, Ib				TEST Eb, Gb Ev, Gv		XCHG Eb, Gb Ev, Gv	
9	NOP PAUSE(F3) XCHG r8, rAX	XCHG word, double-word or quad-word register with rAX rCX/r9 rDX/r10 rBX/r11 rSP/r12 rBP/r13 rSI/r14 rDI/r15						
A	MOV AL, Ob rAX, Ov Ob, AL Ov, rAX				MOVS/B Xb, Yb	MOVS/W/D/Q Xv, Yv	CMPS/B Xb, Yb	CMPS/W/D Xv, Yv
B	MOV immediate byte into byte register AL/R8L, Ib CL/R8L, Ib DL/R10L, Ib BL/R11L, Ib AH/R12L, Ib CH/R13L, Ib DH/R14L, Ib BH/R15L, Ib							
C	Shift Grp 2 ^{1A} Eb, Ib Ev, Ib		RETN ⁱ⁶⁴ lw	RETN ⁱ⁶⁴	LES ⁱ⁶⁴ Gz, Mp	LDS ⁱ⁶⁴ Gz, Mp	Grp 11 ^{1A} - MOV Eb, Ib Ev, Iz	
D	Shift Grp 2 ^{1A} Eb, 1 Ev, 1 Eb, CL Ev, CL				AAM ⁱ⁶⁴ Ib	AAD ⁱ⁶⁴ Ib	XLAT/ XLATB	
E	LOOPNE ⁱ⁶⁴ LOOPNZ ⁱ⁶⁴ Jb	LOOPE ⁱ⁶⁴ LOOPZ ⁱ⁶⁴ Jb	LOOP ⁱ⁶⁴ Jb	JrCXZ ⁱ⁶⁴ Jb	IN AL, Ib eAX, Ib		OUT Ib, AL Ib, eAX	
F	LOCK (Prefix)	REPNE (Prefix)		REP/REPE (Prefix)	HLT	CMC	Unary Grp 3 ^{1A} Eb Ev	

Table A-2. One-byte Opcode Map: (08H — FFH) *

	8	9	A	B	C	D	E	F
0	OR						PUSH CS ⁱ⁶⁴	2-byte escape (Table A-3)
1	SBB						PUSH DS ⁱ⁶⁴	POP DS ⁱ⁶⁴
2	SUB						SEG=CS (Prefix)	DAS ⁱ⁶⁴
3	CMP						SEG=DS (Prefix)	AAS ⁱ⁶⁴
4	DEC ⁱ⁶⁴ general register / REX ^{o64} Prefixes							
	eAX REX.W	eCX REX.WB	eDX REX.WX	eBX REX.WXB	eSP REX.WR	eBP REX.WRB	eSI REX.WRX	eDI REX.WRXB
5	POP ^{d64} into general register							
	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSH ^{d64} lz	IMUL Gv, Ev, lz	PUSH ^{d64} lb	IMUL Gv, Ev, lb	INS/INSB Yb, DX	INS/INSW/INSD Yz, DX	OUTS/OUTSB DX, Xb	OUTS/OUTSW/OUTSD DX, Xz
7	Jcc ⁱ⁶⁴ , Jb- Short displacement jump on condition							
	S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
8	MOV				MOV Ev, Sw	LEA Gv, M	MOV Sw, Ew	Grp 1A ^{1A} POP ^{d64} Ev
	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev				
9	CBW/CWDE/CDQE	CWD/CDQ/CQO	CALLF ⁱ⁶⁴ Ap	FWAIT/WAIT	PUSHF/D/Q ^{d64} Fv	POPF/D/Q ^{d64} Fv	SAHF	LAHF
A	TEST		STOS/B Yb, AL	STOS/W/D/Q Yv, rAX	LODS/B AL, Xb	LODS/W/D/Q rAX, Xv	SCAS/B AL, Yb	SCAS/W/D/Q rAX, Xv
	AL, lb	rAX, lz						
B	MOV immediate word or double into word, double, or quad register							
	rAX/r8, lv	rCX/r9, lv	rDX/r10, lv	rBX/r11, lv	rSP/r12, lv	rBP/r13, lv	rSI/r14, lv	rDI/r15, lv
C	ENTER lw, lb	LEAVE ^{d64}	RETF lw	RETF	INT 3	INT lb	INTO ⁱ⁶⁴	IRET/D/Q
D	ESC (Escape to coprocessor instruction set)							
E	CALL ⁱ⁶⁴ Jz	near ⁱ⁶⁴ Jz	JMP far ⁱ⁶⁴ AP	short ⁱ⁶⁴ Jb	IN AL, DX eAX, DX		OUT DX, AL DX, eAX	
F	CLC	STC	CLI	STI	CLD	STD	INC/DEC Grp 4 ^{1A}	INC/DEC Grp 5 ^{1A}

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

다른것을 해볼까요?

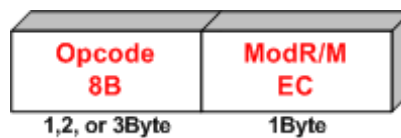
이제 다음 스냅샷에서 두 번째 라인 8B EC 를 변환해보도록 하겠습니다.

```

start      55          push    ebp
start+1    8B EC      mov     ebp, esp
start+3    6A FF      push    0FFFFFFFh
start+5    68 28 DC 48 00  push    offset unk_48DC28
start+A    68 00 6C 47 00  push    offset unk_476C00
start+F    64 A1 00 00 00 00  mov     eax, large fs:0
start+15   50          push    eax
start+16   64 89 25 00 00 00 00  mov     large fs:0, esp
start+1D   83 EC 68      sub     esp, 68h      ; Integer Subtraction
start+20   53          push    ebx
start+21   56          push    esi
start+22   57          push    edi
start+23   89 65 E8      mov     [ebp+var_18], esp
start+26   33 DB      xor     ebx, ebx      ; Logical Exclusive OR
start+28   89 5D FC      mov     [ebp+var_4], ebx
start+2B   6A 02      push    2
    
```

8B EC를 변환할것입니다. 그럼 다음 스냅샷과 같이 되겠군요.

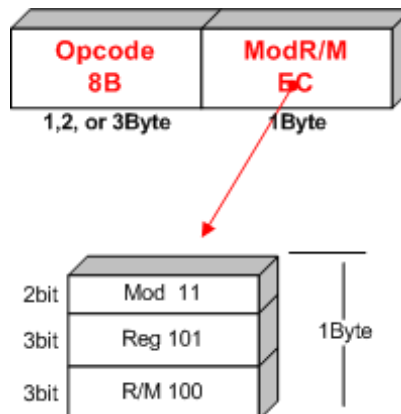
8B를 8Page의 테이블에서 찾아보면 MOV Gv, Ev 라고 나오는것을 볼수가 있습니다.



이제 EC를 2진수로 바꿔보면 11101100입니다.

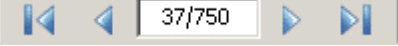
11101100 이것이 다음과 같이 박스에 들어간다고 생각하시면 쉬울것 같네요.

Mod 가 11 Reg가 101 R/M이 100 이 정보를 가지고 테이블에서 찾을것입니다.



이제 다른 매뉴얼을 열어보겠습니다.

25366619.pdf 파일을 열어주세요. -> [Volume 2A : Instruction Set Reference, A-M](#)

	왼쪽 스냅샷과 같이 37페이지로 가시거나
Vol. 2A 2-7	왼쪽 스냅샷과 같이 문서의 하단이나 상단에 나와있는 2-7 페이지로 갑니다.

그럼 11Page와 같이 **32-Bit Addressing Forms with the ModR/M Byte** 테이블이 나옵니다.
우선 Mod가 11 과 R/M이 100 이 일치하는 부분을 찾습니다.
그럼 다음 스냅샷과 같이 **ESP** 가 나오는것을 볼수 있습니다.

ESP/SP/AH/MM4/XMM4 | 100

이제 11Page 테이블 상단에 보면 다음 스냅샷과 같이 나옵니다.
여기서 맨 하단 좌측에 보면 (In Binary) REG = 이라고 나오는데 여기서 아까 구한 **Reg 101** 이 있는곳으로 가봅시다. 그러면 CH, BP, EBP 등등 이 나오는데 **32bit** 계열이니 **r32(/r)** 부분을 봐야겠죠?

정리하자면 REG= 부분에서 101을 찾고 나오는 정보들중에 **r32(/r)** 부분을 찾으면 **EBP**가 나오는것을 확인할수 있습니다.

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
mm(/r)	MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
xmm(/r)	XMM0	XMM1	XMM2	XMM3	XMM4	XMM5	XMM6	XMM7
(In decimal) /digit (Opcode)	0	1	2	3	4	5	6	7
(In binary) REG =	000	001	010	011	100	101	110	111

지금까지 구한 정보를 종합해보면 다음과 같이 구하였습니다.

8B는 → MOV Gv, Ev

Mod가 11 과 R/M이 100 → ESP

REG=101 → EBP

\$ Gv 에는 범용레지스터인 **EBP**를 넣어주면 되고,

\$ Ev 에는 word 혹은 doubleword operand를 넣을수 있으니 **ESP**를 넣어주면 된다.

그럼 다음과 같이 **8B EC**가 → mov ebp, esp 가 만들어진것이다.

```
|start+1| 8B EC | mov ebp, esp
```

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =			AL AX EAX	CL CX ECX	DL DX EDX	BL BX EBX	AH SP ESP	CH BP EBP	DH SI ESI	BH DI EDI
			MM0 0 000	MM1 1 001	MM2 2 010	MM3 3 011	MM4 4 100	MM5 5 101	MM6 6 110	MM7 7 111
Effective Address	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
[EAX]	00	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
[--][--] ¹		100	04	0C	14	1C	24	2C	34	3C
disp32 ²		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
[EAX]+disp8 ³	01	000	40	48	50	58	60	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp8		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp8		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp8		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	10	000	80	88	90	98	A0	A8	B0	B8
[ECX]+disp32		001	81	89	91	99	A1	A9	B1	B9
[EDX]+disp32		010	82	8A	92	9A	A2	AA	B2	BA
[EBX]+disp32		011	83	8B	93	9B	A3	AB	B3	BB
[--][--]+disp32		100	84	8C	94	9C	A4	AC	B4	BC
[EBP]+disp32		101	85	8D	95	9D	A5	AD	B5	BD
[ESI]+disp32		110	86	8E	96	9E	A6	AE	B6	BE
[EDI]+disp32		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0/XMM0	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL/MM1/XMM1		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL/MM2/XMM2		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL/MM3/XMM3		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH/MM4/XMM4		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH/MM5/XMM5		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH/MM6/XMM6		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH/MM7/XMM7		111	C7	CF	D7	DF	E7	EF	F7	FF

NOTES:

1. The [--][--] nomenclature means a SIB follows the ModR/M byte.
2. The disp32 nomenclature denotes a 32-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is added to the index.
3. The disp8 nomenclature denotes an 8-bit displacement that follows the ModR/M byte (or the SIB byte if one is present) and that is sign-extended and added to the index.

Table 2-3. 32-Bit Addressing Forms with the SIB Byte

r32 (In decimal) Base = (In binary) Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
Scaled Index	SS	Index	Value of SIB Byte (in Hexadecimal)							
[EAX]	00	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		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]	01	000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[EDX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
none		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]	10	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	99	9A	9B	9C	9D	9E	9F
none		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]	11	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		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:

- The [*] nomenclature means a disp32 with no base if the MOD is 00B. Otherwise, [*] means disp8 or disp32 + [EBP]. This provides the following address modes:

MOD bits	Effective Address
00	[scaled index] + disp32
01	[scaled index] + disp8 + [EBP]
10	[scaled index] + disp32 + [EBP]

= Reference =

<http://www.intel.com>