



인텔 메뉴얼을 이용하여 OP코드를 어셈블리어 명령으로 변환하기

2006. 01.

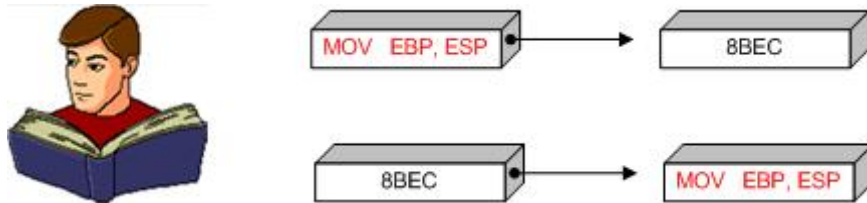
이강석

certlab@certlab.org

어셈블리어 개발자 그룹 :: 어셈러브
www.asmlove.co.kr

#1. 문서의 처음

이 문서는 인텔 기반으로 작성된 문서이며 문서의 흐름은 디어셈블 화면에서 볼수 있는 OPCODE를 보고, IA-32 Instruction Format을 본 후에 인텔 메뉴얼을 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 <- OPCODE가 달라지게 됨.

OPCODE를 어셈블리어 명령으로 변환 하는 일은 컴퓨터가 알아서 해주지만 인텔 메뉴얼을 이용하여 역변환 할 수 있다는 방법을 제시하는 문서입니다.

1 : 문서의 처음	- 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

IDA에서 디어셈블한 모습

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 0B	xor	ebx, ebx ; Logical Exclusive OR
start+28	89 5D FC	mov	[ebp+var_4], ebx
start+2B	6A 02	push	2

OillyDBG에서 디어셈블한 모습

00401110	55	PUSH ESP	
00401111	8B EC	MOV EBP,ESP	
00401113	6A FF	PUSH -1	
00401115	68 90404000	PUSH a_00404030	
0040111A	68 00104000	PUSH a_00401B00	
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	83 EC 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:[<0>KERNEL32.GetVersion	kernel32.GetVersion
0040113C	3302	XOR EDX,EDX	
0040113E	9AD4	MOV DL,AH	
00401140	8915 A4524000	MOV DWORD PTR DS:[4052A4],EDX	
00401146	8BC8	MOV ECX,EAX	
00401148	81E1 FF000000	AND ECX,0FF	

Visual Studio 6.0 에서의 Debug mode

→ 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]
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]

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
  
```

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
  
```

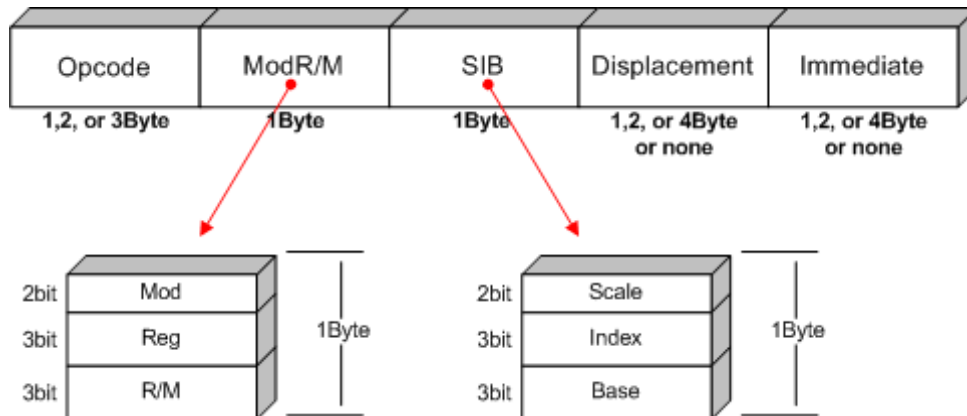
여기까지 서로 다른 모습의 디어셈블 화면들을 보면서 OPCODE가 어셈블리어 명령어와 1:1로 매칭 된다는 것을 볼 수 있습니다.

여기서 조금 의문점이 드는 분들이 있으실 것입니다.

왜 OPCODE 55가 push ebp 인것인가!

#3. IA-32 Instruction Format

다음은 IA-32 Instruction Format 입니다.



인텔 매뉴얼을 보면서 변환을 하기 앞서 인텔 매뉴얼을 다운 받으십시오.

인텔 매뉴얼은 인텔 사이트에 있지만 아래 어셈러브 사이트에서 쉽게 다운 받으실 수 있습니다.

어셈러브 (<http://www.asmlove.co.kr>)

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

<http://asmlove.co.kr/Board/PDS/63530>

위 자료실 링크로 가시면 Developer's Manual 매뉴얼들을 다운 받으실 수 있습니다.

OPCODE 변환에 관련된 매뉴얼은 아래 두 개의 파일만 있으면 됩니다.

Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 2A: Instruction Set Reference, A-M

Intel® 64 and IA-32 Architectures Software Developer's Manual
Volume 2B: Instruction Set Reference, N-Z

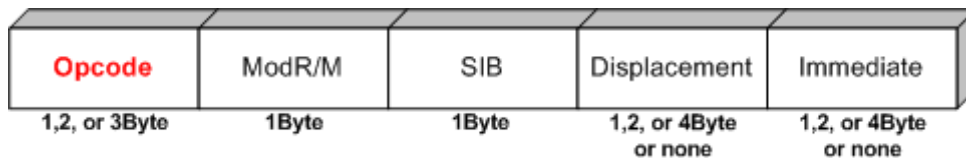
#4. OPCODE -> 어셈블리어 명령 변환

인텔 매뉴얼들은 대부분 페이지 수가 500~700페이지들이 넘는 문서들이기 때문에 찾아가는 부분에 대해서 자세히 설명 하겠습니다.

이제 OPCODE에서 어셈블리어 명령으로 변환 해봅시다.



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>왼쪽 스냅샷과 같이 문서의 하단이나 상단에 나와있는 A-9 페이지로 갑니다.</p>

다음 테이블에서 가로 라인에서 5 세로라인에서 5를 찾아보면
 PUSH rBP/r13 이라고 나와 있는 것을 확인 할 수 있습니다. 정말 쉽죠? :)

Table A-2. One-byte Opcodes Map: (00h ~ FFh) *

	0	1	2	3	4	5	6	7
0	1b, Cb	w, Cw	ADD			rAX, b	PUSH rS ¹⁶	POP ¹⁶ rS ¹⁶
			Cb, b	Cw, W	Al, b			
1	1b, Cb	w, Cw	ADC			rAX, b	PUSH rS ¹⁶	POP ¹⁶ rS ¹⁶
			Cb, b	Cw, W	Al, b			
2	1b, Cb	w, Cw	AND			rAX, b	SHL CL, CL (16bits)	AAA ¹⁶
			Cb, b	Cw, W	Al, b			
3	1b, Cb	w, Cw	XCH			rAX, b	SHR CL, CL (16bits)	AAS ¹⁶
			Cb, b	Cw, W				
4	INC ¹⁶ general register / DEC ¹⁶ rAX							
	rAX 10-X	rCX 10-X, i	rDX 10-X, X	rBX 10-X, X, i	rSI ¹⁶ 10-X, R	rDI ¹⁶ 10-X, R, i	rSI R, X, X	rDI 10-X, X, i
5	PUSH ¹⁶ general register							
	rAX/r0	rCX/r0	rDX/r0	rBX/r1	rSI/r2	rDI/r3	rSI/r4	rDI/r5
6	PUSH IA ¹⁶ / PUSHIA ¹⁶	POPFA ¹⁶ / POPFA ¹⁶	POPN ¹⁶ Ov, Wz	ADPF ¹⁶ Ov, Cw MOVZX ¹⁶ Ov, W	SHL CL, CL (16bits)	SHR CL, CL (16bits)	Openarc Size: (16bits)	Address Size: (16bits)
7	Jcc ¹⁶ , Cb - Short displacement and jump on condition							
	O	NO	OVNA /O	NOVA /NO	Z/O	NZ/NZ	OV/NA	NB/VA
8	Immediate 8bits 1 ¹⁶				11111		XCHG	
	1b, b	1w, W	1b, b ¹⁶	1w, W	1b, Cb	1w, Cw	b, Cb	1w, Cw
9	MOVB XCHG word, double word or quad word register with rAX							
	VALU (3) XCHG r, rAX	rCX/r0	rDX/r0	rBX/r1	rSI/r2	rDI/r3	rSI/r4	rDI/r5
A	MOVB				MOVSB	MOVSW/W/O	CMPSB	CMPSW/O
	Al, Cb	rAX, Cw	Cb, Al	Cw, rAX	Xb, Yb	Xw, Yw	Xb, Yb	Xw, Yw
B	MOVB immediate byte into byte register							
	Al/R0, b	Cb/R0, b	Al/R10, b	Al/R11, b	Al/R12, b	Cb/R13, b	Al/R14, b	Al/R15, b
C	Shift 8bits 2 ¹⁶		10-11 ¹⁶ brr		1-31 ¹⁶ Cw, Wp	111 ¹⁶ Cw, Wp	Cq: 11 ¹⁶ - VCV	
	1b, b	1w, W					1b, b	1w, W
D	Shift 8bits 2 ¹⁶				AAM ¹⁶ b		AA ¹⁶ b	
	1b, b	1w, W	1b, Cb	1w, Cw				
E	LOCK/UNLOCK ¹⁶ b	LOCK/LOCK ¹⁶ b	LOCK ¹⁶ b	JECX ¹⁶ b	IN		OUT	
					Al, b	rAX, b	b, Al	b, rAX
F	LOCK (16bits)	R11/RBP (16bits)		10-11 R11/RP (16bits)	111	CMC	Unary 6bits 3 ¹⁶	
							b	1w

Table A-2. One-byte Opcode Map (C8H - FFH) *

[illegible]

NOTE:

* All blanks in all exposed negative are reserved and must not be used. Do not drop out any film exposures of undeveloped or reserved material.

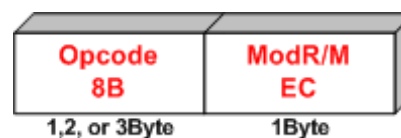
다른 것을 해볼까요?

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 변환 할 것입니다. 그럼 다음 스냅샷과 같이 되겠군요.

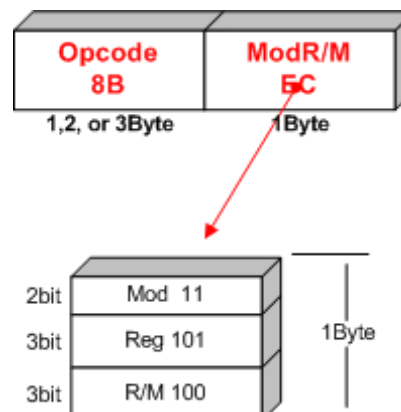
8B를 8Page의 테이블에서 찾아보면 **MOV Gv, Ev** 라고 나오는것을 볼수가 있습니다.



이제 **EC**를 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 메뉴얼을 열어보겠습니다.

25366619.pdf 파일을 열어주세요. -> **Volume 2A : Instruction Set Reference, A-M**

	왼쪽 스냅샷과 같이 37페이지로 가시거나
	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 나오는 것을 볼 수 있습니다.

1 32/32/11A/100M/100M/100 100

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 **101**을 찾고 나오는 정보들중에 **r32(r)** 부분을 찾으면 **EBP**가 나오는 것을 확인 할 수 있습니다.

r32(r)	AX	CH	DI	SI	EDI	EBP	ESI	EDI	ESI
r16(r)	AX	CH	DI	SI	EDI	EBP	ESI	EDI	ESI
r32(r)	AX	CH	DI	SI	EDI	EBP	ESI	EDI	ESI
r16(r)	AX	CH	DI	SI	EDI	EBP	ESI	EDI	ESI
r32(r)	AX	CH	DI	SI	EDI	EBP	ESI	EDI	ESI
r16(r)	AX	CH	DI	SI	EDI	EBP	ESI	EDI	ESI
(In decimal) / Shift (Operation)	0	1	2	3	4	5	6	7	8
(In binary) / Shift (Operation)	000	001	010	011	100	101	110	111	111

지금까지 구한 정보를 종합해보면 다음과 같습니다.

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 만들어 진것이다.

start+1 8B EC mov ebp, esp

Table 2-2. 32-bit Addressing Forms with the Mod/Reg Byte

$Rn(r)$ $11R(r)$ $02R(r)$ $imm(r)$ $xttr(r)$ {in decimal} / {in C-syntax} {in binary} / {in C-syntax}	AL AX EAX MM0 XMM0	CL CX ECX MM1 XMM1	DL DX EDX MM2 XMM2	BL BX EBX MM3 XMM3	AL AX EAX MM4 XMM4	CL CX ECX MM5 XMM5	DL DX EDX MM6 XMM6	BL BX EBX MM7 XMM7	
	000	001	010	011	100	101	110	111	
16-bit Address	16-bit Index	Values of Mod/Reg Bytes (in hexadecimal)							
[AX]	00	000	00	00	10	10	20	20	30
[CX]	00	001	01	00	11	10	21	20	31
[BX]	010	010	02	0A	12	1A	22	2A	32
[BX]	011	010	03	0B	13	1B	23	2B	33
[-] ¹	100	01	0C	0C	14	1C	24	2C	34
disp32 ²	101	01	0D	0D	15	1D	25	2D	35
[SI]	110	01	0E	0E	16	1E	26	2E	36
[DI]	111	01	0F	0F	17	1F	27	2F	37
[AX]+disp8 ³	01	000	40	40	50	50	60	60	70
[CX]+disp8	01	001	41	40	51	50	61	60	71
[BX]+disp8	010	010	42	4A	52	5A	62	6A	72
[BX]+disp8	011	010	43	4B	53	5B	63	6B	73
[-]+disp8	100	01	44	4C	54	5C	64	6C	74
[SI]+disp8	101	01	45	4D	55	5D	65	6D	75
[SI]+disp8	110	01	46	4E	56	5E	66	6E	76
[DI]+disp8	111	01	47	4F	57	5F	67	6F	77
[AX]+disp32	10	000	60	60	90	90	A0	A0	B0
[CX]+disp32	10	001	61	60	91	90	A1	A0	B1
[BX]+disp32	010	010	62	6A	92	9A	A2	AA	B2
[BX]+disp32	011	010	63	6B	93	9B	A3	AB	B3
[-]+disp32	100	01	64	6C	94	9C	A4	AC	B4
[SI]+disp32	101	01	65	6D	95	9D	A5	AD	B5
[SI]+disp32	110	01	66	6E	96	9E	A6	AE	B6
[DI]+disp32	111	01	67	6F	97	9F	A7	AF	B7
AX/AXI/MM0/XMM0	11	000	00	00	100	100	110	110	110
CX/CXI/MM1/XMM1	11	001	01	00	101	100	111	110	111
BX/BXI/MM2/XMM2	010	010	02	0A	102	10A	112	11A	112
BX/BXI/MM3/XMM3	011	010	03	0B	103	10B	113	11B	113
SI/SXI/MM4/XMM4	100	01	0C	0C	104	10C	114	11C	114
SI/SXI/MM5/XMM5	101	01	0D	0D	105	10D	115	11D	115
SI/SXI/MM6/XMM6	110	01	0E	0E	106	10E	116	11E	116
DI/DXI/MM7/XMM7	111	01	0F	0F	107	10F	117	11F	117

NOTES:

- The [-] memorandums means a 32-bit follows the Mod/Reg byte.
- The disp32 memorandums denotes a 32-bit displacement that follows the Mod/Reg byte (or the 32-bit byte if one is present) and that is added to the index.
- The disp8 memorandums denotes an 8-bit displacement that follows the Mod/Reg byte (or the 32-bit 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

32-bit (In decimal) Base (In binary) Base	RAX 0 000	RCX 1 001	RDX 2 010	RBX 3 011	RSP 4 100	R* 5 101	RSI 6 110	RDI 7 111		
Scaled Index	000	Index	Values of SIB Bytes (in Hexadecimal)							
0: RAX	00	000	00	01	02	03	04	05	06	07
1: RCX		001	08	09	0A	0B	0C	0D	0E	0F
2: RDX		010	10	11	12	13	14	15	16	17
3: RBX		011	18	19	1A	1B	1C	1D	1E	1F
4: RSP		100	20	21	22	23	24	25	26	27
5: R*		101	28	29	2A	2B	2C	2D	2E	2F
6: RSI		110	30	31	32	33	34	35	36	37
7: RDI		111	38	39	3A	3B	3C	3D	3E	3F
8: RAX*2	01	000	40	41	42	43	44	45	46	47
9: RCX*2		001	48	49	4A	4B	4C	4D	4E	4F
10: RDX*2		010	50	51	52	53	54	55	56	57
11: RBX*2		011	58	59	5A	5B	5C	5D	5E	5F
12: RSP*2		100	60	61	62	63	64	65	66	67
13: R**2		101	68	69	6A	6B	6C	6D	6E	6F
14: SI*2		110	70	71	72	73	74	75	76	77
15: DI*2		111	78	79	7A	7B	7C	7D	7E	7F
16: RAX*4	10	000	80	81	82	83	84	85	86	87
17: RCX*4		001	88	89	8A	8B	8C	8D	8E	8F
18: RDX*4		010	90	91	92	93	94	95	96	97
19: RBX*4		011	98	99	9A	9B	9C	9D	9E	9F
20: RSP*4		100	A0	A1	A2	A3	A4	A5	A6	A7
21: R**4		101	AB	AC	AD	AE	AF	AF	AF	AF
22: SI*4		110	B0	B1	B2	B3	B4	B5	B6	B7
23: DI*4		111	B8	B9	BA	BB	BC	BD	BE	BF
24: RAX*8	11	000	C0	C1	C2	C3	C4	C5	C6	C7
25: RCX*8		001	C8	C9	CA	CB	CC	CD	CE	CF
26: RDX*8		010	D0	D1	D2	D3	D4	D5	D6	D7
27: RBX*8		011	D8	D9	DA	DB	DC	DD	DE	DF
28: RSP*8		100	E0	E1	E2	E3	E4	E5	E6	E7
29: R**8		101	F0	F1	FA	FB	FC	FD	FE	FF
30: SI*8		110	F8	F9	FA	FB	FC	FD	FE	FF
31: DI*8		111	F8	F9	FA	FB	FC	FD	FE	FF

NOTES:

- The [*] memorandum means a disp32 with no base if the MOD is 001. Otherwise, [*] means disp8 or disp32 + [RIP]. This provides the following address modes:

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

= Reference =

<http://www.intel.com>