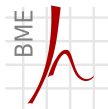


Intel x86 utasításkészlet + disassembler működése

Kód visszafejtés.



Híradástechnikai Tanszék

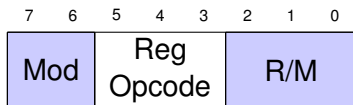
Izsó Tamás

2016. november 10.

Section 1

Intel X86 utasításkészlet

ModR/M



A *ModR/M* mező a műveletben résztvevő regisztereket vagy memória címzés típusát azonosítja.

- A *mod* és az *r/m* mező összevonva 5 bites, azaz 32 lehetséges értéket vehet fel. Ebből 8 érték regisztert címez, 4 pedig a címzési módot adja meg.
- A *reg/opcode* vagy regiszter cím, vagy 3 bit még hozzáadódik az utasításhoz, ami plusz információt hordoz.
- Az *r/m* mező a regiszter típusú operandust adja meg, vagy a *mod* mezővel kombinálva a címzési módot írja le.

Némely *ModR/M* értékhez még egy byte címzési információt kell elhelyezni, ez a *SIB* byte.

SIB mező



- skálafaktor
- index
- bázis

Néhány címzési módnál a *ModR/M* vagy ha van *SIB* byte után találjuk az eltolás (displacement) értéket. Ez az érték 1, 2, vagy 4 byte-os lehet.

Címszámítás a SIB alapján

$$\begin{array}{c} \textit{base} \\ \left[\begin{array}{c} \textit{EAX} \\ \textit{EBX} \\ \textit{ECX} \\ \textit{EDX} \\ \textit{ESP} \\ \textit{EBP} \\ \textit{ESI} \\ \textit{EDI} \end{array} \right] \end{array} + \begin{array}{c} \textit{index} \\ \left(\begin{array}{c} \textit{EAX} \\ \textit{EBX} \\ \textit{ECX} \\ \textit{EDX} \\ \textit{EBP} \\ \textit{ESI} \\ \textit{EDI} \end{array} \right) \end{array} * \begin{array}{c} \textit{scale} \\ \left(\begin{array}{c} 1 \\ 2 \\ 4 \\ 8 \end{array} \right) \end{array} + \begin{array}{c} \textit{displacement} \\ \left[\begin{array}{c} \textit{nincs} \\ 8 - \textit{bit} \\ 16 - \textit{bit} \\ 32 - \textit{bit} \end{array} \right] \end{array}$$

$$\text{offset} = \text{base} + (\text{index} * \text{scale}) + \text{displacement}$$

ModR/M számítása

000	AL	AX	EAX	MM0	XMM0
001	CL	CX	ECX	MM1	XMM1
010	DL	DX	EDX	MM2	XMM2
011	BL	BX	EBX	MM3	XMM3
100	AH	SP	ESP	MM4	XMM4
101	CH	BP	EBP	MM5	XMM5
110	DH	SI	ESI	MM6	XMM6
111	BH	DI	EDI	MM7	XMM7

regiszterek számozása

MOD 11

R/M 000

REG 001

ModR/M 11 001 000 = C8H
ECX EAX

ModR/M értelmezése

r32(/r) (In binary) REG =			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
			000	001	010	011	100	101	110	111
Effective addr	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
[--] (SIB jön)		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
[EAX]+disp8	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 / ...	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX / ...		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX / ...		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX / ...		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP / ...		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP / ...		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI / ...		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI / ...		111	C7	CF	D7	DF	E7	EF	F7	FF

SIB értelmezése

r32 (In binary) Base =	EAX 000	ECX 001	EDX 010	EBX 011	ESP 100	megj(1) 101	ESI 110	EDI 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

ModR/M – SIB byte-hoz megjegyzés

megjegyzés SIB-hez:

- [-][-] azt jelenti, hogy SIB jön a ModR/M byte után.
- Ahol a disp8 szerepel ott a ModR/M és ha van SIB akkor az után 8 bites érték jön.
- Ahol a disp32 szerepel ott a ModR/M és ha van SIB akkor az után 32 bites érték jön.

megjegyzés SIB-hez (megj1):

- Ha a Mod 00_B akkor csak displacement van, bázisregiszter nincs.
- Máskülönben $[EBP + disp8]$ vagy $[EBP+disp32]$ -vel kell számolni.

Mod bits	base
00	$[disp32]$
01	$[EBP+disp8]$
10	$[EBP+disp32]$

Utasítás dekódolásra 1. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
89 19	00	011	001				mov [ecx],ebx

r32(/r) (In binary) REG =			EAX 000	ECX 001	EDX 010	EBX 011	ESP 100	EBP 101	ESI 110	EDI 111
Effective addr	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
[-][-] (SIB jön)		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
[EAX]+disp8	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

Utasítás dekódolásra 2. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 35 F4 8C 12 10	00	110	101				mov esi, [10128CF4]

r32(r) (In binary) REG =			EAX 000	ECX 001	EDX 010	EBX 011	ESP 100	EBP 101	ESI 110	EDI 111
Effective addr	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
[-][-] (SIB jön)		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
[EAX]+disp8	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

Utasítás dekódolásra 3. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 91 A8 00 00 00	10	010	001				mov edx, [ecx+0A8h]

r32(/r) (In binary) REG =			EAX 000	ECX 001	EDX 010	EBX 011	ESP 100	EBP 101	ESI 110	EDI 111
Effective addr	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
.....
[EAX]+disp8	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
.....

Utasítás dekódolásra 4. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 14 81	00	010	100				mov edx, [ecx+eax*4]

r32(/r) (In binary) REG =			EAX 000	ECX 001	EDX 010	EBX 011	ESP 100	EBP 101	ESI 110	EDI 111	
Effective addr	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	
[-][-] (SIB jön)		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	
[EAX]+disp8		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	
.....

Utasítás dekódolásra 4. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 14 81	00	010	100				mov edx, [ecx+eax*4]

r32(r) (In binary) REG =	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI				
	000	001	010	011	100	101	110	111				
Effective addr	r32 (In binary) Base =				EAX	ECX	EDX	EBX	ESP	megj(1)	ESI	EDI
[EAX]					000	001	010	011	100	101	110	111
[ECX]	Scaled Index	SS	Index	Value of SIB Byte (in Hexadecimal)								
[EDX]
[EBX]	[EAX*2]	01	000	40	41	42	43	44	45	46	47	47
[] [] (SIB jön)	[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F	4F
disp32	[EDX*2]		010	50	51	52	53	54	55	56	57	57
[ESI]	[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F	5F
[EDI]	none		100	60	61	62	63	64	65	66	67	67
[EAX]+disp8	[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F	6F
[ECX]+disp8	[ESI*2]		110	70	71	72	73	74	75	76	77	77
[EDX]+disp8	[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F	7F
[EBX]+disp8	[EAX*4]	10	000	80	81	82	83	84	85	86	87	87
[] [] +disp8	[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F	8F
[EBP]+disp8	[EDX*4]		010	90	91	92	93	94	95	96	97	97
.....	[EBX*4]		011	98	99	9A	9B	9C	9D	9E	9F	9F
	none		100	A0	A1	A2	A3	A4	A5	A6	A7	A7

Utasítás dekódolásra 5. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 8C 24 80 00 00 00	10	001	100	00	100	100	mov ecx, [esp+80h]

r32(/r) (In binary) REG =			EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
			000	001	010	011	100	101	110	111
Effective addr	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
.....
[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
.....

Utasítás dekódolásra 5. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 8C 24 80 00 00 00	10	001	100	00	100	100	mov ecx, [esp+80h]

r32(/r) (In binary) REG =	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI			
	000	001	010	011	100	101	110	111			
Effective addr	r32 (In binary) Base =			EAX	ECX	EDX	EBX	ESP	megj(1)	ESI	EDI
.....				000	001	010	011	100	101	110	111
[ECX]+disp8	Scaled Index	SS	Index	Value of SIB Byte (in Hexadecimal)							
[EDX]+disp8
[EBX]+disp8	[EAX]	00	000	00	01	02	03	04	05	06	07
[−][−]+disp8	[ECX]		001	08	09	0A	0B	0C	0D	0E	0F
[EBP]+disp8	[EDX]		010	10	11	12	13	14	15	16	17
[ESI]+disp8	[EBX]		011	18	19	1A	1B	1C	1D	1E	1F
[EDI]+disp8	none		100	20	21	22	23	24	25	26	27
[EAX]+disp32	[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ECX]+disp32	[ESI]		110	30	31	32	33	34	35	36	37
[EDX]+disp32	[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EBX]+disp32	[EAX*2]	01	000	40	41	42	43	44	45	46	47
[−][−]+disp32	[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[EBP]+disp32	[EDX*2]		010	50	51	52	53	54	55	56	57
[ESI]+disp32	[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
.....	none		100	60	61	62	63	64	65	66	67

Utasítás dekódolásra 6. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 54 B5 8C	01	010	100	10	110	101	mov edx, [ebp+esi*4 - 74h]

r32(/r) (In binary) REG =			EAX 000	ECX 001	EDX 010	EBX 011	ESP 100	EBP 101	ESI 110	EDI 111
Effective addr	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)							
.....
[-][-] (SIB jön)		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
[EAX]+disp8	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]+displ8		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
.....

Utasítás dekódolásra 6. példa

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	skála (bin)	index (bin)	bázis (bin)	utasítás
8B 54 B5 8C	01	010	100	10	110	101	mov edx, [ebp+esi*4 - 74h]

r32(/r) (In binary) REG =	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI					
000	001	010	011	100	101	110	111						
Effective addr	Mod	R/M	Value of ModR/M Byte (in Hexadecimal)										
..... [-][-] (SIB jön) disp32 [ESI] [EDI]			r32 (In binary) Base =			EAX	ECX	EDX	EBX	ESP	megj(1)	ESI	EDI
			Scaled Index	SS	Index	Value of SIB Byte (in Hexadecimal)							
		
[EAX]+disp8			[EAX*4]	10	000	80	81	82	83	84	85	86	87
[ECX]+disp8			[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX]+disp8			[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX]+disp8			[EBX*4]		011	98	99	9A	9B	9C	9D	9E	9F
none			none		100	A0	A1	A2	A3	A4	A5	A6	A7
[-][-]+disp8			[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[EBP]+disp8			[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[ESI]+disp8			[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EDI]+disp8			[EAX*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7
[EAX]+disp32			[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[ECX]+disp32			[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
.....		

Utasítás értelmezése a Mod bitek alapján

opcode (hex)	opcode extension Mod/R-ből (bin)	utasítás	op1	op2
83	000	ADD	r/m16/32	imm8
83	001	OR	r/m16/32	imm8
83	010	ADC	r/m16/32	imm8
83	011	SBB	r/m16/32	imm8
83	100	AND	r/m16/32	imm8
83	101	SUB	r/m16/32	imm8
83	110	XOR	r/m16/32	imm8
83	111	CMP	r/m16/32	imm8

opcode (hex)	Mod (bin)	opcode vagy Reg (bin)	r/m (bin)	utasítás
83 C4 04	11	000	100	add esp,4

Regiszter kiosztás 1 byte-os utasításkód esetén

INC r16/32 utasítás kódja $40+r$,

INC EAX \rightarrow 40,

INC EBX \rightarrow 41

Speciális bitek az utasításokban

mező neve	leírás	bitek száma
reg	általános regiszterek	3
w	adat szélessége (16 vagy 32 bit)	1
s	immediate adat sign vagy unsigned	1
sreg2	szegmens regiszter megadás (ES,DS,CS, SS)	2
sreg3	szegmens regiszter megadás (ES,DS,CS, SS, FS, GS)	3
eee	speciális regiszterek,control vagy debug reg.	3
ttn	feltételes utasításoknál a feltételt és annak negálását írja le	4
d	adatok irányának a leírása	1

Példa adatírányra

```

8B 75 0C      mov     esi , dword ptr [ebp+0Ch]
89 75 0C      mov     dword ptr [ebp+0Ch] , esi
  
```


Vezérlésátadó utasítások

Utasítás	Feltétel	Leírás
jmp	1	feltétel nélkül
je	ZF = 1	= vagy 0
jne	ZF = 0	≠
js	SF = 1	< 0
jns	SF = 0	≥ 0
jg	((SF XOR OF) OR ZF) = 0	>
jge	(SF XOR OF) = 0	≥
jl	(SF XOR OF) = 1	<
jle	((SF XOR OF) OR ZF) = 1	≤
ja	(CF OR ZF) = 0	>
jb	CF = 1	<

Vezérlésátadó utasítások

Utasítás	Kód (bináris)	Feltétel 3-2-1 bitek	Negálás 0 bit
JZ	01110100	010	0
JNZ	01110101	010	1
JB	01110010	001	0
JNB	01110011	001	1
JG	01111111	111	1
JNG	01111110	111	0

Intel X86 utasításkészlet:

<http://ref.x86asm.net/index.html>

Section 2

Disassembler

Lineárisan pásztázó disassembler

startAddr, endAddr: pointer of byte;

```
proc DisasmLinear(addr)
```

```
begin
```

```
    while startAddr ≤ addr and addr ≤ endAddr do
```

```
        I := decode_instruction(addr)
```

```
        addr += length( I )
```

```
    od
```

```
end
```

```
proc main()
```

```
begin
```

```
    startAddr := address of the first code segment byte
```

```
    endAddr := address of the last code segment byte
```

```
    DisasmLinear( startAddr )
```

```
end
```

Lineárisan pásztázó disassembler

- előnye
 - egyszerű;
 - gyors;
 - bejárja az összes utasítást.
- hátránya
 - nem különbözteti meg a kódot az adattól (pl. switch utasítás után címtábla található, vagy a visszafejtés megnehezítésére junk (szemét) byte-ok kerülnek a kódban).

C program

```
int func(int a, int b)
{
    int c;
    c= a + b;
    return c-6;
}
```

Disassembler output (dumpbin)

```

_func:
00000000: 55          push  ebp
00000001: 8B EC      mov   ebp, esp
00000003: 51        push  ecx
00000004: 8B 45 08   mov   eax, dword ptr [ebp+8]
00000007: 03 45 0C   add   eax, dword ptr [ebp+0Ch]
0000000A: 89 45 FC   mov   dword ptr [ebp-4], eax
0000000D: 8B 45 FC   mov   eax, dword ptr [ebp-4]
00000010: 83 E8 06   sub   eax, 6
00000013: 8B E5     mov   esp, ebp
00000015: 5D        pop   ebp
00000016: C3       ret

```

Lineárisan pásztázó disassembler összezavarása (obfuscation)

```
int func(int a, int b)
{
    int c;
    goto L;
    _asm {
        _emit 0x0f;
    }
L:
    c = a + b;
    return c - 6;
}
```



Szemét byte

Disassembler output (dumpbin)

```

_func :
00000000: 55                push     ebp
00000001: 8B EC            mov     ebp,esp
00000003: 51              push     ecx
00000004: EB 01          jmp     00000007
00000006: 0F 8B 45 08 03 45 jnp     45030851
0000000C: 0C 89          or     al,89h
0000000E: 45            inc     ebp
0000000F: FC          cld
00000010: 8B 45 FC      mov     eax,dword ptr [ebp-4]
00000013: 83 E8 06      sub     eax,6
00000016: 8B E5        mov     esp,ebp
00000018: 5D          pop     ebp
00000019: C3          ret

```

Disassembler output (dumpbin)

```

_func :
00000000: 55                push     ebp
00000001: 8B EC            mov     ebp,esp
00000003: 51              push     ecx
00000004: EB 01            jmp     00000007
00000006: 0F 8B 45 08 03 45 → jnp     45030851
0000000C: 0C 89            or      al,89h
0000000E: 45              inc     ebp
0000000F: FC              cld
00000010: 8B 45 FC        mov     eax,dword ptr [ebp-4]
00000013: 83 E8 06        sub     eax,6
00000016: 8B E5            mov     esp,ebp
00000018: 5D              pop     ebp
00000019: C3              ret

```

■ hibás utasítás;

Disassembler output (dumpbin)

```

_func :
00000000: 55                push    ebp
00000001: 8B EC            mov     ebp,esp
00000003: 51              push    ecx
00000004: EB 01            jmp     00000007
00000006: 0F 8B 45 08 03 45 → jnp    45030851
0000000C: 0C 89            or     al,89h
0000000E: 45              inc    ebp
0000000F: FC              cld
00000010: 8B 45 FC        mov     eax,dword ptr [ebp-4]
00000013: 83 E8 06        sub     eax,6
00000016: 8B E5            mov     esp,ebp
00000018: 5D              pop    ebp
00000019: C3              ret

```

- hibás utasítás;
- szinkronizálás jó utasítás határra.

Rekurzívan pásztázó disassembler utasításfeldolgozás sorrendje

- nem vezérlésátadó utasítás esetén a következő utasításon folytatódik;
- feltételes ugrásnál az ugrás címén folytatódik az adatok feldolgozása, és később visszatér az utasítást követő részekhez;
- függvényhívásnál az eljárás címétől folytatódik a disassemblálás, és azt befejezve a következő utasításra kell rátérni;
- feltétel nélküli ugrás esetén az ugrás címén lévő adatokat kell feldolgozni, és ha az eljárás visszatért, akkor a hívó is befejezi az adott részt;
- return esetén a utasításfeldolgozás végére értünk;
- ha azt tapasztaljuk, hogy az adott címen lévő adatokat feldolgoztuk, akkor vissza kell térni a hívó rutinhoz.

Rekurzívan pásztázó disassembler

```

startAddr, endAddr: pointer of byte;
proc DisasmRec(addr)
begin
  while startAddr ≤ addr and addr ≤ endAddr do
    if ( Visited(addr) return
    Visited(addr) := true
    I := decode_instruction(addr)
    if ( I == jmp || I == conditional jmp || I == call )
      newAddr= DecodeOperandAddr( I )
      DisasmRec( newAddr )
    fi
    if ( I == jmp || I == ret )
      return
    fi
    addr += length( I )
  od
end

proc main()
begin
  startAddr := address of the first code segment byte
  endAddr := address of the last code segment byte
  addr := program entry point
  DisasmRec( addr )
end

```

Rekurzívan pásztázó disassembler

- előnye
 - Megkülönbözteti a kódot az adattól.
- hátránya
 - Számított vezérlésátadás esetén (`jmp EAX`) nem biztos, hogy bejárja az összes kódrészt, mivel a statikus disassembler nem ismeri az EAX értékét.

Ollydebug output

```

00401000  55          push ebp
00401001  8BEC       mov ebp,esp
00401003  51         push ecx
00401004  EB 01     jmp short 00401007
00401006  0F        db 0F
00401007  8B45 08   mov eax,dword ptr ss:[arg.1]
0040100A  0345 0C   add eax,dword ptr ss:[arg.2]
0040100D  8945 FC   mov dword ptr ss:[local.1],eax
00401010  8B45 FC   mov eax,dword ptr ss:[local.1]
00401013  83E8 06   sub eax,6
00401016  8BE5     mov esp,ebp
00401018  5D        pop ebp
00401019  C3        retn

```

IDA disassembler output

```

00000000 55          push    ebp
00000001 8B EC      mov     ebp, esp
00000003 51        push    ecx
00000004 EB 01      jmp     short loc_7
00000006 0F        db 0Fh
00000007                                loc_7:
00000007 8B 45 08   mov     eax, [ebp+arg_0]
0000000A 03 45 0C   add     eax, [ebp+arg_4]
0000000D 89 45 FC   mov     [ebp+var_4], eax
00000010 8B 45 FC   mov     eax, [ebp+var_4]
00000013 83 E8 06   sub     eax, 6
00000016 8B E5     mov     esp, ebp
00000018 5D        pop     ebp
00000019 C3        retn

```

Rekurzívan pásztázó disassembler összezavarása (obfuscation)

```
int func(int a, int b)
{
    int c=a;
    int d = ~0 ^ c;
    if( ~ c != d ) goto Junk;
    goto L;
Junk:
    _asm {
        _emit 0x0f;
    }
L:
    c= a + b;
    return c-6;
}
```

Átlátszó feltétel

Ollydebug output

```

00401000 55          push ebp
00401001 8B EC      mov ebp,esp
00401003 83 EC 08   sub esp,8
00401006 8B 45 08   mov eax,dword ptr ss:[ebp+8]
00401009 89 45 F8   mov dword ptr ss:[ebp-8],eax
0040100C 8B 4D F8   mov ecx,dword ptr ss:[ebp-8]
0040100F 83 F1 FF   xor ecx,FFFFFFFF
00401012 89 4D FC   mov dword ptr ss:[ebp-4],ecx
00401015 8B 55 F8   mov edx,dword ptr ss:[ebp-8]
00401018 F7 D2     not edx
0040101A 3B 55 FC   cmp edx,dword ptr ss:[ebp-4]
0040101D 74 02     je short 00401021
0040101F EB 02     jmp short 00401023
00401021 EB 01     jmp short 00401024
00401023 0F       db 0F
00401024 8B 45 08   mov eax,dword ptr ss:[ebp+8]
00401027 03 45 0C   add eax,dword ptr ss:[ebp+0C]
0040102A 89 45 F8   mov dword ptr ss:[ebp-8],eax
0040102D 8B 45 F8   mov dword ptr ss:[ebp-8]
00401030 83 E8 06   sub eax,6
00401033 8B E5     mov esp,ebp
00401035 5D       pop ebp
00401036 C3       retn

```

jól elkülönítet-
te az adatot a
kódtól.

IDA disassembler output

```

00000000 55                push    ebp
00000001 8B EC            mov     ebp, esp
00000003 83 EC 08         sub     esp, 8
00000006 8B 45 08         mov     eax, [ebp+arg_0]
00000009 89 45 F8         mov     [ebp+var_8], eax
0000000C 8B 4D F8         mov     ecx, [ebp+var_8]
0000000F 83 F1 FF         xor     ecx, 0FFFFFFFh
00000012 89 4D FC         mov     [ebp+var_4], ecx
00000015 8B 55 F8         mov     edx, [ebp+var_8]
00000018 F7 D2           not     edx
0000001A 3B 55 FC         cmp     edx, [ebp+var_4]
0000001D 74 02           jz      short loc_21
0000001F EB 02           jmp     short loc_23
00000021                loc_21:
00000021 EB 01           jmp     short near ptr loc_23+1
00000023                loc_23:
00000023 0F 8B 45 08+    jnp     near ptr 4503086Eh
00000029 0C 89           or      al, 89h
0000002B 45             inc     ebp
0000002C F8             cld
0000002D 8B 45 F8         mov     eax, [ebp+var_8]
00000030 83 E8 06         sub     eax, 6
00000033 8B E5           mov     esi, ebp
00000035 5D             pop     ebp
00000036 C3             retn

```

hibás visszafejtés

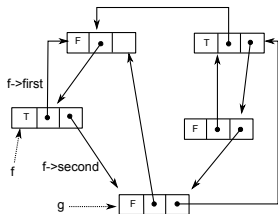
Átlátszó feltétel

Ha feltételes elágazásnál tudjuk, hogy az egyik ág sohasem következhet be, átlátszó feltételről (opaque predicate) beszélünk.

```
if ( i+1 < i ) goto Label ;
```

Az előző feltételt könnyű felismerni, és a felesleges kódot kiszűrni. A jó *átlátszó feltétel* értéke fordítási időben ismert, de nehéz észrevenni, hogy a feltétel értéke konstans, annak ellenére, hogy a benne szerepet játszó tagok változnak. Ezzel a rekurzív disassemblert megzavarhatjuk, és a visszafejtés idejét megnövelhetjük.

Átlátszó feltétel gráffal megvalósítva



- fordítási időben ismert, hogy a program adott pontján lévő feltétel igaz vagy hamis;
- fordítási időben ismert, hogy a pointerok hova mutatnak;
- a gráf menetközben változhat.

```
if ( f->first ->value ) goto Label;
if ( f->second == g ) goto Label;
```

Visszatérési cím manipulálása

- A függvény a stack-en lévő visszatérési címet megváltoztatja, így nem a hívás után folytatódik a program.
- A visszatérési érték kiszámításához használhatunk hash táblát, ami a stackre letett visszatérési cím alapján megadja, hogy mennyivel odébb kell folytatni a programot.
- a jump utasításokat is helyettesíthetjük ilyen trükkös függvényhívással.

Visszatérési cím manipulálására példa

```

#include <stdio.h>
void __stdcall1 f(int k) {
    int* pk=&k; /* stack-en lévő paraméter címe */
    pk--;      /* visszatérési cím */
    *pk+=9;    /* ret addr += 9 */
}
int main() {
    int a=1;
    f(4);
    a++;2 /* eax ← a, eax ← eax+1, a ← eax; 9 byte */
    /* ide térünk vissza */
    printf("%d\n", a);
    return 0;
}

```

¹stack kezelés következő órán

²optimalizálás nélkül fordítva

Miért olyan okos az Ollydebug

Különböző heurisztikákat alkalmaz.

- 12 menetben analizálja a kódot;
- utasítás kezdet tesztelése, nem lehet olyan helyen, ami a program memóriában elfoglalt helyétől függ, nem lehet olyan ugrás, aminek a címe kimutat a címtartományból, stb.
- kettőnél többször történik ugrás egy területre, az 0.99 valószínűséggel kód,
- függvényhívásnál a paraméterek elmentése, lokális változók helyfoglalása, stb. árulkodó lehet.
- 2200 rendszerhívást ismer;
- 7800 konstans szimbolikus nevét felismeri;
- for, while, stb vezérlésszerkezetet felismeri;