

2.6.5. Reprezentarea instrucțiunilor mașină

O instrucțiune mașină x86 reprezintă o secvență de 1 până la 15 octeți, care prin valorile lor specifică o operație de executat, operanții asupra cărora va fi aplicată, precum și modificatori suplimentari care controlează modul în care aceasta va fi executată.

O instrucțiune mașină x86 are maximum doi operanzi. Pentru cele mai multe dintre instrucțiuni, cei doi operanzi poartă numele de *sursă*, respectiv *destinație*. Dintre cei doi operanzi, maximum unul se poate afla în memoria RAM. Celălalt se află fie într-un registru al **EU**, fie este o constantă întreagă. Astfel, o instrucțiune are forma:

numeinstrucțiune destinație, sursă

Formatul intern al unei instrucțiuni este variabil, el putând ocupa între 1 și 15 octeți, având următoarea formă generală de reprezentare (*Instructions byte-codes from OllyDbg*):

[prefixe] + cod + [ModR/M] + [SIB] + [deplasament] + [imediat]

Prefixele controlează modul în care o instrucțiune se execută. Acestea sunt optionale (0 până la maximum 4) și ocupă câte un octet fiecare. De exemplu, acestea pot solicita execuția repetată (în buclă) a instrucțiunii curente sau pot bloca magistrala de adrese pe parcursul execuției pentru a nu permite accesul concurrent la operanți și rezultate.

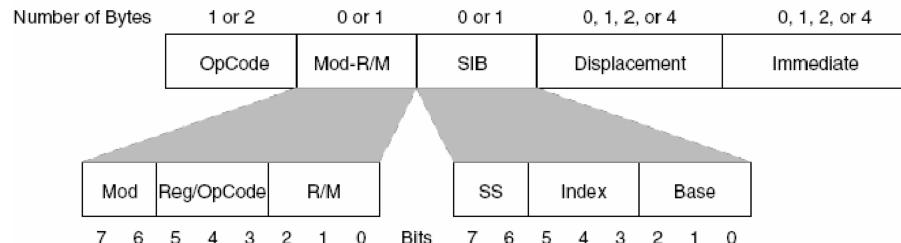
Operația care se va efectua este identificată prin intermediul a 1 sau 2 octeți de *cod* (opcode), aceștia fiind singurii octeți obligatoriu prezenti, indiferent de instrucțiune.

Reprezentarea_instr_masina_Formula_offset

Page 2 of 6

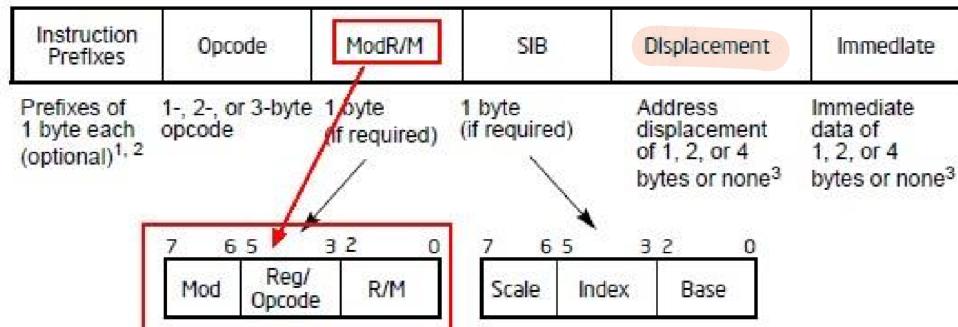
Number of Bytes	0 or 1	0 or 1	0 or 1	0 or 1
	Instruction prefix	Address-size prefix	Operand-size prefix	Segment override

(a) Optional instruction prefixes



(b) General instruction format

Although the diagram seems to imply that instructions can be up to 16 bytes long, in actuality the x86 will not allow instructions greater than 15 bytes in length.



mod R/M
deplasament = offset variabilă prezentată în mod direct

Curs 14

Tipuri de prefixe

- 1) Prefix de instrucțiune | prefixe explicite
- 2) Prefix de precizare explicită a segmentului | (programatorului)
- 3) Prefix de precizare dimensiunii operanilor | prefixe implicate
- 4) Prefix de precizare a adresii unui operand | (generat de către asamblator)

Coduri fixe generate de către asamblator pt:

F3h - REP, REPE

F2h - REPNE

F3 : codul imot care urmărește

2) Prefix de precizare explicită a segmentului

CS : 2Eh

DS : 3Eh

SS : 36h

ES : 26h

3) 66h
bits 32

exemplu curs

4) 67h
bits 16

push dword [ebx]; 66:67 : FF

push dword [CS:ebx]; 2E:66:67

dim op formula de calcul prescurtat

rep push dword [CS:ebx]; F3:2E:66:67

Examen: Ce sunt prefixele de instrucții?

Clasificare, 2 exemple

Clasificarea conversiilor

- 3 criterii:
- a) → destructive - cbw, cwd, cwde, cdq
 $\text{mov ah}, \text{0}$
 $\text{mov dx}, \text{0}$
 $\text{mov edx}, \text{0}$
 $\text{movzx d}, \text{s}$
 $\text{movsx d}, \text{s}$
 - instructioni
 - medistructive: operatori de tip: - byte, word, dword, qword
 - b) → conversie cu semn: cbw, cwd, cwde, cdq, movsx d, s
 - conversie fără semn: movzx d, s
 $\text{mov ah}, \text{0}$
 $\text{mov dx}, \text{0}$
 $\text{mov edx}, \text{0}$
byte, dword, word, qword
 - c) → conversie primă lungire - toate cele destructive din assembly
& word, dword, qword
 - primă reprezentare: → byte, word, dword

De întrebător Vlad ee o zis în ultima parte

Subiect examen:

- I Teorie
- little endian
 - EFlags
 - complementul față de 2
 - conceptul de Overflow
 - coduri de apel

or du 23456
add ebx, v
sub ebx, 6
mov eax, ebx

lea eax, [ebx+v-6]

- IV Problema simplă / multi model
primit, scmp

or du 23456
add ebx, v
sub ebx, 6
mov eax, ebx

nu există
instrucțiune echivalentă

Octetul *ModR/M* (mod registru/memorie) specifică pentru unele dintre instrucțiuni natura și locul operanzilor (registru, memorie). Acesta permite specificarea fie a unui registru, fie a unei locații de memorie a cărei adresa este exprimată prin intermediul unui offset (<http://datacadamia.com/intel/modrm>)

Pentru cazuri mai complexe de adresare decât cele codificabile direct prin ModR/M, combinarea acestuia cu octetul SIB (Scale – Index – Base) permite următoarea formulă generală de definire a unui offset:

unde pentru bază și index vor fi folosite valorile a doi registri iar scală este 1, 2, 4 sau 8. Regiștrii permisi ca bază sau / și index sunt: EAX, EBX, ECX, EDX, EBP, ESI, EDI. Registrul ESP este disponibil ca bază însă nu poate fi folosit cu rol de index. (http://www.c-jump.com/CIS77/CPU/x86/lecture.html#X77_0100_sib_byte_layout)

Majoritatea instrucțiunilor folosesc pentru reprezentare fie numai campul de cod, fie cod urmat de ModR/M.

Deplasament (displacement) apare în cazul unor forme de adresare particulare (operanzi din memorie) și urmează direct după ModR/M sau SIB, când SIB este prezent. Acest câmp poate fi codificat fie pe octet, fie pe cuvânt, fie pe dublu cuvânt (32 biți).

The most common addressing mode, and the one that's easiest to understand, is the *displacement-only* (or *direct*) addressing mode. **The displacement-only addressing mode consists of a 32-bit constant that specifies the address of the target location.** The displacement-only addressing mode is perfect for accessing simple scalar variables. Intel named this the displacement-only addressing mode because a 32-bit constant (displacement) follows the MOV opcode in memory. **On the 80x86 processors, this displacement is an offset from the beginning of memory (that is, address zero).**

Displacement mode, the operand's offset is contained as part of the instruction as an 8-, 16-, or 32-bit displacement. The displacement addressing mode is found on few machines because, as mentioned earlier, it leads to long instructions. In the case of the x86, the displacement value can be as long as 32 bits, making for a 6-byte instruction. **Displacement addressing can be useful for referencing global variables.**

Ca și consecință a imposibilității prezenței mai multor câmpuri de ModR/M, SIB și deplasament într-o instrucțiune, arhitectura 80x86 NU permite codificarea a două adrese de memorie în aceeași instrucțiune.

Valoare imediată oferă posibilitatea definirii unui operand ca fiind o constantă numerică pe 1, 2 sau 4 octeți. Când este prezent, acest câmp apare întotdeauna la sfârșitul instrucțiunii.

X *WW*

2.6.6. Adrese FAR și NEAR

Pentru a adresa o locație din memoria RAM sunt necesare două valori: una care să indice segmentul, alta care să indice offsetul în cadrul segmentului. Pentru a simplifica referirea la memorie, microprocesorul derivă, în lipsă unei alte specificări, adresa segmentului din **unul dintre registrii de segment CS, DS, SS sau ES**. Alegerea implicită a unui registru de segment se face după niște reguli proprii instrucțiunii folosite.

Prin definiție, o adresă în care se specifică doar offsetul, urmând ca segmentul să fie preluat implicit dintr-un registru de segment poartă numele de *adresă NEAR* (adresă apropiată). O adresă NEAR se află întotdeauna în interiorul unuia din cele patru segmente active.

O adresă în care programatorul indică explicit un selector de segment poartă numele de *adresă FAR* (adresă îndepărtată). O adresă FAR este deci o SPECIFICARE COMPLETA DE ADRESA și ea se poate exprima în trei moduri:

- $s_3s_2s_1s_0$: specificare_offset unde $s_3s_2s_1s_0$ este o constantă;
- registru_segment : specificare_offset, registru segment fiind CS, DS, SS, ES, FS sau GS;
- FAR [variabilă], unde variabilă este de tip QWORD și conține cei 6 octeți constituind adresa FAR. (ceea ce numim variabila pointer în limbajele de nivel înalt)

Formatul intern al unei adrese FAR este: la adresa mai mică se află offsetul, iar la adresa mai mare cu 4 (cuvântul care urmează după dublucuvântul curent) se află cuvântul ce conține selectorul care indică segmentul.

Reprezentarea adreselor respectă principiul reprezentării little-endian expus în capitolul 1, paragraf 1.3.2.3: partea cea mai puțin semnificativă are adresa cea mai mică, iar partea cea mai semnificativă are adresa cea mai mare.

2.6.7. Calculul offsetului unui operand. Moduri de adresare

În cadrul unei instrucțiuni există 3 moduri de a specifica un operand pe care aceasta îl solicită:

- *modul registru*, dacă pe post de operand se află un registru al mașinii; mov eax, 17
- *modul imediat*, atunci când în instrucțiune se află chiar valoarea operandului (nu adresa lui și nici un registru în care să fie conținut); mov eax, 17
- *modul adresare la memorie*, dacă operandul se află efectiv undeva în memorie. În acest caz, offsetul lui se calculează după următoarea formulă:

$$\text{adresa_offset} = [\text{bază}] + [\text{index} \times \text{scală}] + [\text{constanta}]$$

Deci *adresa_offset* se obține din următoarele (maxim) patru elemente:

- conținutul unuia dintre regiștri EAX, EBX, ECX, EDX, EBP, ESI, EDI sau ESP ca bază;
- conținutul unuia dintre regiștri EAX, EBX, ECX, EDX, EBP, ESI sau EDI drept index;
- factor numeric (scală) pentru a înmulți valoarea registratorului index cu 1, 2, 4 sau 8
- valoarea unei constante numerice, pe octet, cuvant sau dublucuvânt.

De aici rezultă următoarele moduri de adresare la memorie:

- **directă**, atunci când apare numai *constanta*;
- **bazată**, dacă în calcul apare unul dintre registrii bază;
- **scalat-indexată**, dacă în calcul apare unul dintre registrii index;

Cele trei moduri de adresare a memoriei pot fi combinate. De exemplu, poate să apară adresare directă bazată, adresare bazată și scalat-indexată etc

Adresarea care NU este directă se numește adresare indirectă (bazată și/sau indexată). Deci o adresare indirectă este cea pt care avem specificat cel puțin un registru intre parantezele drepte.

La instrucțiunile de salt mai apare și un alt tip de adresare numit adresare *relativă*.

Adresa relativă indică poziția următoarei instrucțiuni de executat, în raport cu poziția curentă. Poziția este indicată prin numărul de octeți de cod peste care se va sări. Arhitectura x86 permite atât adrese relative scurte (SHORT Address), reprezentate pe octet și având valori între -128 și 127, cât și adrese relative apropriate (NEAR Address), pe dublucuvânt cu valori între -2147483648 și 2147483647.

Jmp MaiJos ; aceasta instructiune se traduce (vezi OllyDbg) de obicei în Jmp [0084]↓
.....
.....

MaiJos:

Mov eax, ebx

Intel x86 Assembler Instruction Set Opcode Table

ADD Eb Gb 00	ADD Ev Gv 01	ADD Gb Eb 02	ADD Gv Ev 03	ADD AL Ib 04	ADD eAX Iv 05	PUSH ES 06	POP ES 07	OR Eb Gb 08	OR Ev Gv 09	OR Gb Eb 0A	OR Gv Ev 0B	OR AL Ib 0C	OR eAX Iv 0D	PUSH CS 0E	TWOBYT E 0F
ADC Eb Gb 10	ADC Ev Gv 11	ADC Gb Eb 12	ADC Gv Ev 13	ADC AL Ib 14	ADC eAX Iv 15	PUSH SS 16	POP SS 17	SBB Eb Gb 18	SBB Ev Gv 19	SBB Gb Eb 1A	SBB Gv Ev 1B	SBB AL Ib 1C	SBB eAX Iv 1D	PUSH DS 1E	POP DS 1F
AND Eb Gb 20	AND Ev Gv 21	AND Gb Eb 22	AND Gv Ev 23	AND AL Ib 24	AND eAX Iv 25	ES:	DAA	SUB Eb Gb 28	SUB Ev Gv 29	SUB Gb Eb 2A	SUB Gv Ev 2B	SUB AL Ib 2C	SUB eAX Iv 2D	CS:	DAS
XOR Eb Gb 30	XOR Ev Gv 31	XOR Gb Eb 32	XOR Gv Ev 33	XOR AL Ib 34	XOR eAX Iv 35	SS:	AAA	CMP Eb Gb 38	CMP Ev Gv 39	CMP Gb Eb 3A	CMP Gv Ev 3B	CMP AL Ib 3C	CMP eAX Iv 3D	DS:	AAS
INC eAX 40	INC eCX 41	INC eDX 42	INC eBX 43	INC eSP 44	INC eBP 45	INC eSI 46	INC eDI 47	DEC eAX 48	DEC eCX 49	DEC eDX 4A	DEC eBX 4B	DEC eSP 4C	DEC eBP 4D	DEC eSI 4E	DEC eDI 4F
PUSH eAX 50	PUSH eCX 51	PUSH eDX 52	PUSH H eBX 53	PUSH eSP 54	PUSH eBP 55	PUSH eSI 56	PUSH eDI 57	POP eAX 58	POP eCX 59	POP eDX 5A	POP eBX 5B	POP eSP 5C	POP eBP 5D	POP eSI 5E	POP eDI 5F
PUSHA 60	POPA 61	BOUN D Gv Ma 62	ARP L Ew Gw 63	FS:	GS:	OPSIZE: 64	ADSIZE: 65	PUSH Iv 66	IMUL Gv Ev Iv 68	PUSH lb 6A	IMUL Gv Ev lb 6B	INSB Yz DX 6C	INSW Yz DX 6D	OUTSB DX Xb 6E	OUTSW DX Xv 6F
JO Jb 70	JNO Jb 71	JB Jb 72	JNB Jb 73	JZ Jb 74	JNZ Jb 75	JBE Jb 76	JA Jb 77	JS Jb 78	JNS Jb 79	JP Jb 7A	JNP Jb 7B	JL Jb 7C	JNL Jb 7D	JLE Jb 7E	JNLE Jb 7F
ADD Eb Ib	ADD Ev Iv	SUB Eb Ib	SUB Ev Ib	TEST Eb Gb	TEST Ev Gv	XCHG Eb Gb	XCHG Ev Gv	MOV Eb Gb	MOV Ev Gv	MOV Gb Eb	MOV Gv Ev	MOV Ew Sw	LEA Gv M	MOV Sw Ew	POP Ev

Intel x86 Assembler Instruction Set Opcode Table

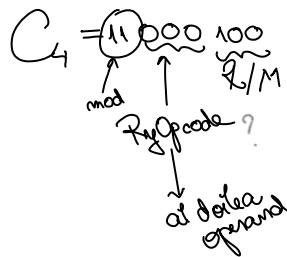
10

Page 2 of 4

80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
NOP 90	XCHG eAX eCX 91	XCHG eAX eDX 92	XCHG G eAX eBX 93	XCHG XCHG eAX eSP 94	XCHG eAX eBP 95	XCHG eAX eSI 96	XCHG eAX eDI 97	CBW 98	CWD 99	CALL Ap 9A	WAIT 9B	PUSH F Fv 9C	POPF Fv 9D	SAHF 9E	LAHF 9F
MOV AL Ob A0	MOV eAX Ov A1	MOV Ob AL A2	MOV Ov eAX A3	MOVS B Xb Yb A4	MOVS W Xv Yv A5	CMPSB Xb Yb A6	CMPS W Xv Yv A7	TEST AL Ib A8	TEST eAX Iv A9	STOS B Yb AL AA	STOS W Yv eAX AB	LODS B AL Xb AC	LODS W eAX Xv AD	SCASB AL Yb AE	SCASW eAX Yv AF
MOV AL Ib B0	MOV CL Ib B1	MOV DL Ib B2	MOV BL Ib B3	MOV AH Ib B4	MOV CH Ib B5	MOV DH Ib B6	MOV BH Ib B7	MOV eAX Iv B8	MOV eCX Iv B9	MOV eDX Iv BA	MOV eBX Iv BB	MOV eSP Iv BC	MOV eBP Iv BD	MOV eSI Iv BE	MOV eDI Iv BF
#2 Eb Ib C0	#2 Ev Ib C1	RETN Iw C2	RET N C3	LES Gv Mp C4	LDS Gv Mp C5	MOV Eb Ib C6	MOV Ev Iv C7	ENTR Iw Ib C8	LEAV E C9	RETF Iw CA	RETF CB	INT3 CC	INT Ib CD	INTO CE	IRET CF
#2 Eb 1 D0	#2 Ev 1 D1	#2 Eb CL D2	#2 Ev CL D3	AAM Ib D4	AAD Ib D5	SALC D6	XLAT D7	ESC 0 D8	ESC 1 D9	ESC 2 DA	ESC 3 DB	ESC 4 DC	ESC 5 DD	ESC 6 DE	ESC 7 DF
LOOPNZ Jb E0	LOOPZ Jb E1	LOOP Jb E2	JCXZ Jb E3	IN AL Ib E4	IN eAX Ib E5	OUT Ib AL E6	OUT Ib eAX E7	CALL Jz E8	JMP Jz E9	JMP Ap EA	JMP Jb EB	IN AL DX EC	IN eAX DX ED	OUT DX AL EE	OUT DX eAX EF
LOCK: F0	INT1 F1	REPNE: F2	REP: F3	HLT F4	CMC F5	#3 Eb F6	#3 Ev F7	CLC F8	STC F9	CLI FA	STI FB	CLD FC	STD FD	#4 INC/DEC C FE	#5 INC/DEC FF

Legend
HAS MOD R/M
LENGTH = 1
OTHER

codificări metode de adresare:
sparks and flames



80x86 Instruction Format

Prefix

INSTRUCTION PREFIX	ADDRESS SIZE PREFIX	OPERAND SIZE PREFIX	SEGMENT OVERRIDE
0 OR 1	0 OR 1	0 OR 1	0 OR 1
NUMBER OF BYTES			

Required

OPCODE	MOD R/M	SIB	DISPLACEMENT	IMMEDIATE
1 OR 2	0 OR 1	0 OR 1	0,1,2 OR 4	0,1,2 OR 4
NUMBER OF BYTES				

MOD R/M BYTE

7	6	5	4	3	2	1	0
MOD	REG/OPCODE			R/M			

SIB BYTE

7	6	5	4	3	2	1	0
SCALE	INDEX			BASE			

MOD R/M 16

	0	1	2	3	4	5	6	7
0	[BX+SI] +1	[BX+DI] +1	[BP+SI] +1	[BP+DI] +1	[SI] +1	[DI] +1	[Iw] +3	[BX] +1
1	[BX+SI+Ib] +2	[BX+DI+Ib] +2	[BP+SI+Ib] +2	[BP+DI+Ib] +2	[SI+Ib] +2	[DI+Ib] +2	[BP+Ib] +2	[BX+Ib] +2
2	[BX+SI+Iw] +3	[BX+DI+Iw] +3	[BP+SI+Iw] +3	[BP+DI+Iw] +3	[SI+Iw] +3	[DI+Iw] +3	[BP+Iw] +3	[BX+Iw] +3
3	AX +1	CX +1	DX +1	BX +1	SP +1	BP +1	SI +1	DI +1

MOD R/M 32

	0	1	2	3	4	5	6	7
0	[eAX] +1	[eCX] +1	[eDX] +1	[eBX] +1	[SIB] +2	[Iv] +5	[eSI] +1	[eDI] +1
1	[eAX+Ib] +2	[eCX+Ib] +2	[eDX+Ib] +2	[eBX+Ib] +2	[SIB+Ib] +2	[eBP+Ib] +2	[eSI+Ib] +2	[eDI+Ib] +2
2	[eAX+Iv] +5	[eCX+Iv] +5	[eDX+Iv] +5	[eBX+Iv] +5	[SIB+Iv] +5	[eBP+Iv] +5	[eSI+Iv] +5	[eDI+Iv] +5
3	eAX +1	eCX +1	eDX +1	eBX +1	eSP +1	eBP +1	eSI +1	eDI +1

REGISTERS

	0	1	2	3	4	5	6	7
Reg 8	AL	CL	DL	BL	AH	CH	DH	BH
Reg 16	AX	CX	DX	BX	SP	BP	SI	DI
Reg 32	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
Segments	DS	ES	FS	GS	SS	CS	IP	

equivalent
 instrukcji
 maszynowej

OllyDbg - lan3.exe - [CPU - main thread, module lan3]

C File View Debug Trace Plugins Options Window

[File] [View] [Debug] [Trace] [Plugins] [Options] [Window]

00402000	.	B8 04000000 MOV EAX,4
00402005	.	BA 78563412 MOV EDX,12345678
0040200A	.	50 PUSH EAX
0040200B	.	52 PUSH EDX
0040200C	.	FF35 00104000 PUSH DWORD PTR DS:[401000]
00402012	.	FF35 00104000 PUSH DWORD PTR DS:[401000]
00402018	.	B9 20000000 MOV ECX,20
0040201D	.	FF35 04104000 PUSH DWORD PTR DS:[401000]
00402023	.	FF35 05104000 PUSH DWORD PTR DS:[401000]
00402029	.	68 00104000 PUSH OFFSET 0040100A
0040202E	.	FF15 44304000 CALL DWORD PTR DS:[<&msg]
00402034	.	83C4 10 ADD ESP,10
00402037	.	6A 00 PUSH 0
00402039	.	FF15 40304000 CALL DWORD PTR DS:[<&msi]
0040203F	.	0000 ADD BYTE PTR DS:[EAX],AL
00402041	.	0000 ADD BYTE PTR DS:[EAX],AL
00402043	.	0000 ADD BYTE PTR DS:[EAX],AL
00402045	.	0000 ADD BYTE PTR DS:[EAX],AL
00402047	.	0000 ADD BYTE PTR DS:[EAX],AL
00402049	.	0000 ADD BYTE PTR DS:[EAX],AL
0040204B	.	0000 ADD BYTE PTR DS:[EAX],AL
0040204D	.	0000 ADD BYTE PTR DS:[EAX],AL
0040204F	.	0000 ADD BYTE PTR DS:[EAX],AL
00402051	.	0000 ADD BYTE PTR DS:[EAX],AL
00402053	.	0000 ADD BYTE PTR DS:[EAX],AL
00402055	.	0000 ADD BYTE PTR DS:[EAX],AL
00402057	.	0000 ADD BYTE PTR DS:[EAX],AL
00402059	.	0000 ADD BYTE PTR DS:[EAX],AL
0040205B	.	0000 ADD BYTE PTR DS:[EAX],AL
0040205D	.	0000 ADD BYTE PTR DS:[EAX],AL
0040205F	.	0000 ADD BYTE PTR DS:[EAX],AL
00402061	.	0000 ADD BYTE PTR DS:[EAX],AL
00402063	.	0000 ADD BYTE PTR DS:[EAX],AL
00402065	.	0000 ADD BYTE PTR DS:[EAX],AL
00402067	.	0000 ADD BYTE PTR DS:[EAX],AL
00402069	.	0000 ADD BYTE PTR DS:[EAX],AL
0040206B	.	0000 ADD BYTE PTR DS:[EAX],AL
0040206D	.	0000 ADD BYTE PTR DS:[EAX],AL
0040206F	.	0000 ADD BYTE PTR DS:[EAX],AL
00402071	.	0000 ADD BYTE PTR DS:[EAX],AL
00402073	.	0000 ADD BYTE PTR DS:[EAX],AL
00402075	.	0000 ADD BYTE PTR DS:[EAX],AL
00402077	.	0000 ADD BYTE PTR DS:[EAX],AL
00402079	.	0000 ADD BYTE PTR DS:[EAX],AL
0040207B	.	0000 ADD BYTE PTR DS:[EAX],AL
0040207D	.	0000 ADD BYTE PTR DS:[EAX],AL
0040207F	.	0000 ADD BYTE PTR DS:[EAX],AL
00402081	.	0000 ADD BYTE PTR DS:[EAX],AL
00402083	.	0000 ADD BYTE PTR DS:[EAX],AL
00402085	.	0000 ADD BYTE PTR DS:[EAX],AL
00402087	.	0000 ADD BYTE PTR DS:[EAX],AL
00402089	.	0000 ADD BYTE PTR DS:[EAX],AL
0040208B	.	0000 ADD BYTE PTR DS:[EAX],AL
0040208D	.	0000 ADD BYTE PTR DS:[EAX],AL
0040208F	.	0000 ADD BYTE PTR DS:[EAX],AL
00402091	.	0000 ADD BYTE PTR DS:[EAX],AL
00402093	.	0000 ADD BYTE PTR DS:[EAX],AL
00402095	.	0000 ADD BYTE PTR DS:[EAX],AL

Imm=4

