FORMATY NIEKTÓRYCH ROZKAZÓW W ARCHITEKTURZE X86 (TRYB 32-BITOWY)

Podstawowy format rozkazu

Pierwszy bajt			Drugi bajt			Trzeci bajt SIB (opcjonalny)			Czwarty bajt	Piąty bajt	Szósty bajt	Siódmy bajt
Kod oper.	d	w	mod	reg	r/m	ss	index	base	Przesunięcie (adres) albo argument bezpośredni (liczba)			

Jeśli bit d = 1, to odbiorcą informacji jest rejestr określony przez pole reg, jeśli d = 0, to r/m.

Liczba bitów	reg	000	001	010	011	100	101	110	111
8	w = 0	AL.	CL	DL	BL	AH	CH	DH	BH
16	w = 1	AX	CX	DX	BX	SP	BP	SI	DI
32	w = 1	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI

Jeśli bit w = 1 i kod rozkazu poprzedzony jest dodatkowym bajtem (przedrostkiem) o wartości 66H, to operacja wykonywana jest na 16 bitach.

r/m	Adres efektywny
000	EAX + przesunięcie
001	ECX + przesunięcie
010	EDX + przesunięcie
011	EBX + przesunięcie
100	Sposób oblicz. określa bajt SIB
101	EBP + przesunięcie
110	ESI + przesunięcie
111	EDI + przesunięcie

Pole mod			
00	01	10	11
Liczba bajtów pola przesunięcia			
0	1	4	*)

*) Argumenty podane są w rejestrach

Współczynnik skali ss (w bajcie SIB)			
00	01	10	11
1	2	4	8

MOV

1. przesłanie liczby do rejestru

1011 w reg	liczba (L)	liczba (..)	liczba (..)	liczba (H)

2. przesłanie liczby do pamięci

1100011 w	mod 000 r/m	liczba (L)	liczba (..)	liczba (..)	liczba (H)

3. przesłanie rejestru do rejestru

1000 10 d w	11 reg r/m

4. przesłanie zawartości komórki pamięci do rejestru i odwrotnie

100010 d w	mod reg r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

5. przesłanie rejestru do rejestru segmentowego (rsgm: ES – 00, CS – 01, SS – 10, DS – 11)

10001110	110 rsgm reg

MOVSX *(przesłanie reg2 do reg1)*

00001111	1011111 w	11 reg1 reg2

Jeśli reg2 jest 8-bitowy, to w=0, a jeśli 16-bitowy, to w =1.

MOVZX *(przesłanie reg2 do reg1)*

00001111	1011011 w	11 reg1 reg2

Jeśli reg2 jest 8-bitowy, to w=0, a jeśli 16-bitowy, to w =1.

XCHG

1. zamiana zawartości rejestrów

1000011 w	11 reg1 reg2

LEA

10001101	mod reg r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

PUSH

01010 reg

POP

01011 reg

IN

11100100	nr portu

OUT

11100110	nr portu

ADD

1. dodawanie liczby do rejestru

100000 s w	11 000 reg	liczba (L)	liczba (..)	liczba (..)	liczba (H)

w przypadku s = w =1 pole *liczba* jest jednobajtowe

2. dodawanie liczby do rejestru AL, AX, EAX

0000010 w	liczba (L)	liczba (..)	liczba (..)	liczba (H)

3. dodawanie zawartości rejestrów

000000 d w	11 reg1 reg2

4. dodawanie zawartości komórki pamięci do rejestru i odwrotnie

000000 d w	mod reg r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

SUB

1. *odejmowanie liczby od rejestru*

100000 s w	11 101 reg	liczba (L)	liczba (..)	liczba (..)	liczba (H)

w przypadku s = w =1 pole *liczba* jest jednobajtowe

2. odejmowanie zawartości rejestrów

001010 d w	11 reg1 reg2

3. odejmowanie zawartości komórki pamięci od rejestru i odwrotnie

001010 d w	mod reg r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

INC

01000 reg

DEC

01001 reg

CMP

1. porównanie zawartości rejestru i liczby

100000 s w	11 111 reg	liczba (L)	liczba (..)	liczba (..)	liczba (H)

w przypadku s = w =1 pole *liczba* jest jednobajtowe

2. porównanie zawartości rejestrów

001110 d w	11 reg1 reg2

MUL

1. mnożenie liczby bez znaku zawartej w rejestrze AL albo AX albo EAX przez liczbę o tej samej długości zawartą w rejestrze reg

1111 011 w	11 100 reg

2. mnożenie liczby bez znaku zawartej w rejestrze AL albo AX albo EAX przez liczbę o tej samej długości zawartą w komórce pamięci

1111 011 w	mod 100 r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

DIV

1. dzielenie liczby 64-bitowej (w EDX:EAX) przez liczbę 32-bitową zawartą w rejestrze

1111 0111	11 110 reg

2. dzielenie liczby 64-bitowej (w EDX:EAX) przez liczbę 32-bitową zawartą w komórce pamięci

11110111	mod 110 r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

3. dzielenie liczby 64-bitowej (w EDX:EAX) przez liczbę 32-bitową zawartą w komórce pamięci o podanym adresie (32-bitowym); adres nie jest indeksowany, tj. nie występuje w postaci np. [EBX]

11110111	00110101	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

OR

1. bitowa suma logiczna zawartości rejestrów

000010 d w	11 reg1 reg2

AND

1. bitowy iloczyn logiczny zawartości rejestrów

001000 d w	11 reg1 reg2

2. bitowy iloczyn logiczny zawartości rejestru i argumentu

100000 s w	11 100 reg	liczba (L)	liczba (..)	liczba (..)	liczba (H)

w przypadku s = w =1 pole *liczba* jest jednobajtowe

XOR

1. suma modulo dwa zawartości rejestru i argumentu

100000 s w	11 110 reg	liczba (L)	liczba (..)	liczba (..)	liczba (H)

w przypadku s = w =1 pole *liczba* jest jednobajtowe

2. suma modulo dwa zawartości rejestrów

001100 d w	11 reg1 reg2

SHL	1100000 w	11100 reg	liczba przes.
SHR	1100000 w	11101 reg	liczba przes.
ROL	1100000 w	11000 reg	liczba przes.
ROR	1100000 w	11001 reg	liczba przes.
RCL	1100000 w	11010 reg	liczba przes.
RCR	1100000 w	11011 reg	liczba przes.

SHLD	0000 1111	1010 0100	11 reg2 reg1	liczba_prz.

Format rozkazu: SHLD reg1, reg2, liczba_przesunięć

SHRD	0000 1111	1010 1100	11 reg2 reg1	liczba_prz.

Format rozkazu: SHRD reg1, reg2, liczba_przesunięć

Rozkazy sterujące (skoki)

JMP

1. format dwubajtowy

1110 1011	przesunięcie

2. format 5-bajtowy

11101001	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

JE	01110100	przesunięcie	**JNE**	01110101	przesunięcie
JB (JC)	01110010	przesunięcie	**JA**	01110111	przesunięcie
JBE	01110110	przesunięcie	**JAE (JNC)**	01110011	przesunięcie

JL	01111100	przesunięcie	**JG**	01111111	przesunięcie
JLE	01111110	przesunięcie	**JGE**	01111101	przesunięcie
JO	01110000	przesunięcie	**JNO**	01110001	przesunięcie

LOOP	11100010	przesunięcie

CALL	11101000	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

RET	11000011

RET n	1100 0010	liczba (L)	liczba (H)

Koprocesor arytmetyczny

Wszystkie rozkazy koprocesora zaczynają się od ciągu bitów 11011 (kod ESC).

FMUL

1. *oba argumenty znajdują się na stosie koprocesora*

11011 d 00	11 001 st

Jeśli bit d = 1, to odbiorcą informacji jest rejestr określony przez pole st, jeśli d = 0, to st(0).

2. *jeden argument znajduje się na stosie koprocesora w st(0, a drugi w pamięci.*

11011 d 00	mod 001 r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

Jeśli bit d = 1, to argument w pamięci jest 64-bitowy (double), jeśli d = 0, to argument 32-bitowy (float).

FDIV

1. *oba argumenty znajdują się na stosie koprocesora*

11011 d 00	11 110 st

Jeśli bit d = 1, to wykonywana jest operacja st(..) ← st(..) / st(0); jeśli d = 0, to wykonywana jest operacja st(0) ← st(0) / st(..)

2. *jeden argument znajduje się na stosie koprocesora w st(0, a drugi w pamięci.*

11011 d 00	mod 001 r/m	Przes.(L)	Przes. (..)	Przes. (..)	Przes. (H)

Jeśli bit d = 1, to argument w pa mięci jest 64-bitowy (double), jeśli d = 0, to argument 32-bitowy (float).
Wykonywana jest operacja st(0) ← st(0) / M[..]

Przypadki szczególne kodowania rozkazów

1. Jeśli pola mod = 00 i r/m = 101, to adres efektywny określony jest wyłącznie przez zawartość 32-bitowego pola przesunięcie (w tym przypadku nie występuje adresowanie indeksowe, a zawartość rejestru EBP jest ignorowana)

Pierwszy bajt			Drugi bajt			Trzeci bajt	Czwarty bajt	Piąty bajt	Szósty bajt
Kod oper.	d	w	00	reg	101	Przesunięcie (adres)			

2. Kodowanie nietypowe ze współczynnikiem skali – rozkaz

mov ecx, [ebx * 4 + 12]

jest kodowany w postaci ciągu bajtów 8B 0C 9D 0C 00 00 00.

Pierwszy bajt			Drugi bajt			Trzeci bajt SIB			Czwarty bajt	Piąty bajt	Szósty bajt	Siódmy bajt
100010	1	1	00	001	100	10	011	101	00001100	00000000	00000000	00000000
Kod oper.	d	w	mod	reg	r/m	ss	index	base	Przesunięcie (adres)			

Interpretacja bitów „d" i „w" w pierwszym bajcie: d=1 oznacza, że wynik zostaje wysłany do obiektu wskazanego przez pole reg (tu: 001 czyli rejestr ECX); w=1 oznacza, że wykonywana operacja. jest 32-bitowa.

Interpretacja drugiego bajtu: mod=00 (zob. dalszy opis), reg=001 tzn. rejestr ECX, r/m=100 oznacza, że . występuje bajt SIB.

Interpretacja trzeciego bajtu (SIB): scale=10 tzn. mnożenie * 4, index=011 oznacza, że rejestrem indeksowym jest rejestr EBX, base=101 tzn. rejestr EBP, ale występuje tu przypadek specjalny: jeśli pole mod=00 i jednocześnie pole base=101, to rejestr wskazany przez pole base pomija się.
(Dalsze szczegóły można znaleźć np. w pliku IA-32-soft-devel-man-instruction-set-v2A.pdf, str.2-8)

3. Rejestr ESP jako rejestr indeksowy

Sposób kodowania binarnego rozkazu

mov eax, [esp + 8]

może budzić wątpliwości, ponieważ w tabeli (podanej na pierwszej stronie) określającej wartości *adresu efektywnego* w zależności od pola r/m nie występuje rejestr ESP. Możliwe jest jednak wprowadzenie bajtu SIB, choć taki schemat kodowania wymaga podania rejestru bazowego i indeksowego, tak jak pokazano na poniższym rysunku.

kod d w	mod reg r/m	skala index base	przesunięcie
	EAX SIB	ESP ESP	
100010 1 1	01 000 100	00 100 100	00001000

Ze względu na zastrzeżenie, że rejestr ESP nie może występować jako rejestr indeksowy, pole index w bajcie SIB jest ignorowane. Zatem rejestrem indeksowym jest tylko rejestr ESP podany w polu base.