Структура машинних команд

Приклади трансляції асемблерних команд

Приклад 1. Програма без використання сегменту даних. Операції з регістрами.

```
1                                page        60, 132
2                                TITLE       EXASM1   (EXE)   Приклад регістрових операцій
3                                ;-------------------------------------------------------------------------
4   0000                         STACKSG   SEGMENT   PARA    STACK   'Stack'
5   0000   0C [                             DB          12 DUP ( 'STACKSEG' )
6              53  54  41  43
7              48  53  45  47
8                               ]
9
10  0060                         STACKSG    ENDS
11                               ;-------------------------------------------------------------------------
12  0000                         CODESG    SEGMENT    PARA    'Code'
13  0000                         BEGIN      PROC       FAR
14                                          ASSUME     SS:STACKSG, CS:CODESG, DS:NOTHING
15  0000   lE                               PUSH       DS                  ; Записати DS в стек
16  0001   2B  C0                           SUB        AX,AX               ; Записати  нуль
17  0003   50                               PUSH       AX                  ;       в стек
18
19  0004   B8  0123                        MOV        AX, 0123H           ; Записати шіст. 0123 в AX
20  0007   05  0025                        ADD        AX, 0025H           ; Додати шіст. 25 до AX
21  000A   8B  D8                          MOV        BX, AX              ; Переслати AX в BX
22  000C   03   D8                         ADD        BX, AX              ; Додати BX до AX
23  000E   8B  CB                           MOV        CX, BX              ; Переслати BX в CX
24  0010   2B   C8                         SUB        CX, AX              ; Відняти AX від CX
25  0012   2B   C0                         SUB         AX, AX             ; Очистити AX
26  0014   90                              NOP
27  0015   CB                              RET                            ; Повернення в DOS
28  0016                          BEGIN     ENDP                          ; Кінець процедури
29
30  0016                         CODESG    ENDS                           ; Кінець сегмента
31                                          END        BEGIN               ; Кінець програми
-------------------------------------------------------------------------------------------------------------------
    Segments and Groups:
                        N a m e          Size          Align      Combine     Class
    CODESG . . . . . . . . . . . . . . . 0016          PARA       NONE        'CODE'
    STACKSG . . . . . . . . . . . . . . . 0060         PARA       STACK       'STACK'

   SymboIs:
                        N a m e          Type          Value          Attr
   BEGIN . . . . . . . . . . . . . . . . F  PROC       0000         CODESG     Length=0016
```

Сегмент коду починається з відносної адреси 0000. Він завантажується в пам’ять у відповідності до адреси в регістрі CS та з нульовим зміщенням відносно цієї адреси. Оскільки ASSUME є директивою асемблеру, то перша команда, що генерує машинный код, - це PUSH DS, однобайтова команда (1E), що знаходиться в сегменті з нульовим зміщенням.

Приклад 2. Програма з використанням сегменту даних.

```
1                                       page   60, 132
2                               TITLE   EXASM2   (EXE)    Операції пересилання і додавання
3                               ;-------------------------------------------------------------------------------
4    0000                       STACKSG   SEGMENT    PARA    STACK   'Stack'
5    0000   20 [                        DW          32  DUP ( ? )
6                ????
7                        ]
8
9    0040                       STACKSG   ENDS
10                              ;-------------------------------------------------------------------------------
11   0000                       DATASG    SEGMENT   PARA   'Data'
12   0000     00FA              FLDA      DW        250
13   0002     007D              FLDB      DW        125
14   0004     ????              FLDC      DW        ?
15   0006                       DATASG    ENDS
16                              ;-------------------------------------------------------------------------------
17   0000                       CODESG    SEGMENT   PARA    'Code'
18   0000                       BEGIN     PROC      FAR
19                                  ASSUME    CS:CODESG, DS:DATASG, SS:STACKSG, ES: NOTHING
20   0000     IE                        PUSH      DS                ; Записати DS в стек
21   0001     2B  C0                    SUB       AX, AX            ; Записати в стек
22   0003     50                        PUSH      AX                ;       нульову адресу
23   0004     B8  - - - -  R            MOV       AX, DATASG        ; Помістити адресу DATASG
24   0007     8E  D8                    MOV       DS, AX            ;       у регістр DS
25
26   0009     Al  0000  R               MOV       AX, FLDA          ; Переслати 0250 в AX
27   000C     03 06 0002  R             ADD       AX, FLDB          ; Додати 0125 до AX
28   0010     A3  0004  R               MOV       FLDC, AX          ; Записати суму в FLDC
29   0013     CB                        RET                         ; Повернутися в DOS
30   0014                       BEGIN     ENDP
31   0014                       CODESG    ENDS
32                                       END       BEGIN
```

```
Segments and Groups:
                    Name                Size      A1Ign     Combine     Class
CODESG . . . . . . . . . . . . . . .0014        PARA      NONE        'CODE'
DATASG . . . . . . . . . . . . . . .0006        PARA      NONE        'DATA'
STACKSG. . . . . . . . . . . . . . 0040         PARA      STACK       'STACK'

Symbo1s:
                    Name                Type      Value     Attr
BEGIN. . . . . . . . . . . . . . . . F  PROC      0000      CODESG    Length=0014
FLDA . . . . . . . . . . . . . . . . L  WORD                0000      DATASG
FLDB . . . . . . . . . . . . . . . . L  WORD                0002      DATASG
FLDC . . . . . . . . . . . . . . . . L  WORD                0004      DATASG
```

Поле FLDA визначає слово з десятковим значенням 250, яке транслюється в шістнадцяткове 00FA. Поле FLDB визначає слово з десятковим значенням 125, яке транслюється в шістнадцяткове 007D. Насправді значення цих двох констант у пам’яті є FA00 та 7D00.

Вибір підмножини команд

Для практичної реалізації виберемо деяку підмножину команд мікропроцесорів I80x86. Слід мати на увазі, що вибір має бути обгрунтованим, зокрема, підмножина команд повинна мати певну функціональну повноту.

Приклад вибору підмножини команд:

MOV	ADD	AND	JMP	LOOP	NOP
XCHG	INC	NOT	CALL	LOOPE	
PUSH	SUB	OR	RET	JE	
POP	DEC	XOR	RET<число>	JG	
PUSHF	IMUL	TEST	INT	JL	
POPF	IDIV	SHL		JNE	
LEA	CBW	SHR			
	CWD				
	CMP				

Кодування регістрів

Основні, базові та індексні регістри:

Біти	w = 0	w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Сегментні регістри:

Біти	Сегм. рег.
00	ES
01	CS
10	SS
11	DS

Наприклад, наступна команда MOV використовує регістр AX і має безпосередній двохбайтовий операнд:

```
MOV  AX,00        10111  000  00000000  00000000
                       |    |
                       w   reg = AX
```

Перший байт машинного коду вказує на розмір в одне слово (w = 1) і на регістр AX (000). Не слід узагальнювати цей приклад, бо вказівка на регістр та біт w можуть бути в різних позиціях коду.

Байт способу адресування

Байт способу адресування, якщо він присутній, займає другий байт машинного коду і складається із таких трьох елементів:

1) mod - двохбітового коду, що має значення 11 для посилання на регістр та 00, 01 і 10 для посилання на пам'ять;
2) reg - трьохбітового вказівника регістра;
3) r/m - трьохбітового вказівника регістра або пам'яті (r - регістр, m - адреса пам'яті).

Крім того, перший байт машинного коду може мати біт "d", який вказує напрям потоку між операндом 1 та операндом 2:

біт "d": 0 - reg → r/m (+ mod)
1 - reg ← r/m (+ mod)

Розглянемо приклад додавання вмісту регістра AX та вмісту регістра BX:

ADD BX, AX 00000011 11 011 000
dw mod reg r/m

У цьому прикладі d=1 означає, що reg (011) описує операнд 1, а r/m (000) і mod (11) описують операнд 2. Так як біт w=1, то розмір дорівнює одному слову. Отже, команда повинна додати AX (000) до BX (011).

Другий байт команди в об'єктному коді визначає більшість способів адресування пам'яті. У подальщому викладенні способи адресування будуть детально розглянуті.

Біти MOD

Два біти mod визначають адресування регістра чи пам'яті. Нижче пояснено їх призначення:

00 біти r/m дають абсолютну адресу, байт зміщення (відносна адреса) відсутній;
01 біти r/m дають абсолютну адресу пам'яті і є один байт зміщення;
10 біти r/m дають абсолютну адресу і є два байти зміщення;
11 біти r/m визначають регістр. Біт w (в байті коду операції) визначає посилання на восьми- або шістнадцятибітовий регістр.

Біти REG

Три біти reg (разом з бітом w) визначають конкретний восьми- або шістнадцятибітовий регістр.

Біти R/M

Три біти r/m (регістр/пам'ять) разом з бітами mod визначають спосіб адресування, як показано в таблиці:

r/m	mod=00	mod=01	mod=10	mod=11 w=0	mod=11 w=1
000	BX+SI	BX+SI+disp	BX+SI+disp	AL	AX
001	BX+DI	BX+DI+disp	BX+DI+disp	CL	CX
010	BP+SI	BP+SI+disp	BP+SI+disp	DL	DX
011	BP+DI	BP+DI+disp	BP+DI+disp	BL	BX
100	SI	SI+disp	SI+disp	AH	SP
101	DI	DI+disp	DI+disp	CH	BP
110	Direct	BP+disp	BP+disp	DH	SI
111	BX	BX+disp	BX+disp	BH	DI

(r/m=110, mod=00) - для прямого адресування типу MOV AX, TABLE

Двобайтові команди

Розглянемо ще раз приклад додавання вмісту регістрів BX та AX:

```
ADD   BX, AX     0000 0011    11   011  000
                                  dw  mod  reg  r/m
```

d 1 означає, що біти reg та w описують операнд 1 (BX), а біти mod, r/m та w - операнд 2 (AX);

w 1 визначає розмір регістрів в одне слово;

mod 11 вказує, що операнд 2 є регістром;

reg 011 вказує, що операнд 1 є регістром BX;

r/m 000 вказує, що операнд 2 є регістром AX.

Розглянемо інший приклад множення регістра AL на регістр BL:

```
MUL  BL     11110110   11   100  011
                       w  mod  reg  r/m
```

Команда MUL передбачає, що регістр AL містить множник. Розмір регістра дорівнює одному байтові (w=0), mod вказує на регістрову операцію, r/m=011 вказує на регістр BL. У даному випадку reg=100 завжди має таке значення, бо є частиною коду операції.

Трибайтові команди

Наступна команда MOV регенерує три байти машинного коду:

```
MOV  mem, AX     10100001  dddddddd  dddddddd
```

Для команди пересилання з регістра AX чи AL необхідно знати, скільки байтів приймає участь в операції: один чи два. В даному прикладі w=1 означає слово, отже, передбачається 16-бітовий регістр AX. Використання в другому операнді регістра AL приведе до значення біта w=0. Байти 2 і 3 містять відносну адресу пам'яті. Команди, які використовують регістри AX чи AL, часто генерують більш ефективний (короткий) машинний код.

Чотирибайтові команди

Розглянемо приклад множення регістра AL на значення в пам'яті. Процесор припускає, що множник знаходиться в регістрі AL для однобайтового множення і в регістрі AX для двохбайтового множення:

```
MUL  mem_byte     11110110   00   100  110
                             w  mod  reg  r/m
```

Для даної команди reg завжди має значення 100, mod=00 вказує на операцію з пам'яттю, а r/m=110 – на прямий спосіб адресування. Машинна команда має також ще два наступні байти (тут не показані), що визначають відносну адресу пам'яті.

Розглянемо ще приклад для команди LEA, яка завжди визначає двобайтову адресу:

```
LEA  DX,mem      10001101   00   010  110
                          LEA  mod  reg  r/m
```

Reg=010 означає регістр DX. Mod=00 та r/m=110 визначають прямий спосіб адресування пам'яті. У наступних двох байтах (тут не показаних) є відносна адреса.

Поле знака (розширення)

Деякі команди мають однобітове поле s – ознака розширення безпосереднього операнда за його знаковим розрядом. Ефект поля s (біта s) є таким. Якщо s=0, то немає

ніякого ефекту – операнд закодований в самій команді у повній формі без скорочення. Значення s=1 дійсне лише для 8-бітових безпосередніх операндів: 8 бітів операнда доповнюються старшими розрядами, які мають таке ж саме значення, що і знаковий розряд операнда, в кількості, необхідній для заповнення 16/32-бітового приймача.

Таблиці кодів команд та їх швидкодії

У таблицях подано лише коди самих команд та байтів способу адресування, якщо вони присутні. Адресна частина команди, якщо вона повинна бути, містить адресу операнда в пам'яті і розташовується за байтом способу адресування.

Таблиця 1. Команди пересилання

Команда	Формат	Число тактів	
MOV – пересилання			
(R/M) ← (R)	1000 100w mod reg r/m	2/2	2/2
(R) ← (R/M)	1000 101w mod reg r/m	2/4	2/4
(R/M) ← (im)	1100 011w mod 000 r/m	2/2	2/2
(R) ← (im)	1011w reg im8, 16, 32	2	2
(A) ← (M)	1010 000w d8, 16, 32	4	4
(M) ← (A)	1010 001w d8, 16, 32	2	2
(SR) ← (R/M)	1000 1110 mod sreg r/m	2/5	18/19
(R/M) ← (SR)	1000 1100 mod sreg r/m	2/2	2/2
PUSH – запис в стек			
S ← (M)	1111 1111 mod 110 r/m	5	5
S ← (R)	01010 reg	2	2
S ← (ES, CS, SS, DS)	000 sreg 110	2	2
S ← (FS або GS)	0000 1111 10 sreg 000	2	2
S ← (im)	0110 10s0 im8, 16, 32	2	2
PUSHA – запис в стек всіх РЗП	0110 0000	18	18
POP – читання з стеку			
(M) ← S	1000 1111 mod 000 r/m	5	5
(E) ← S	01011 reg	4	4
(ES, CS, SS, DS) ← S	000 sreg 111	7	21
(FS або GS) ← S	0000 1111 10 sreg 001	7	21
POPA – читання із стеку всіх РЗП	0110 0001	24	24
XCHG – обмін			
R/M ↔ (R)	1000 011w mod reg r/m	3/5	3/5
(A) ↔ (R)	1001 0 reg	3	3
XLAT – перетворення кодів	1101 0111	5	5
MOVSX – пересилання з розширенням знаку	0000 1111 1011 111w mod reg r/m	3/6	3/6
MOVZX – пересилання з розширенням нулями	0000 1111 1011 011w mod reg r/m	3/6	3/6
IN – введення з фіксованого порту	1110 010w p8	5	5
IN – введення із змінного порту	1110 110w	6	6
OUT – виведення у фіксований порт	1110 011w p8	3	3
OUT – виведення у змінний порт	1110 111w	4	4
LEA – завантаження адреси EA в	1000 1101 mod reg r/m	2	2

регістр			
LDS – завантаження селектора в регістр DS	1100 0101 mod reg r/m	7	22
LES – завантаження селектора в регістр ES	1100 0100 mod reg r/m	7	22
LFS – завантаження селектора в регістр FS	0000 1111 1011 0100 mod reg r/m	7	25
LGS – завантаження селектора в регістр GS	0000 1111 1011 0101 mod reg r/m	7	25
LSS – завантаження селектора в регістр SS	0000 1111 1011 0010 mod reg r/m	7	22

Таблиця 2. Команди арифметичних операцій

Команда	Формат	Число тактів	
ADD – додавання			
(R) ← (R) + (R)	0000 00dw mod reg r/m	2	2
(M) ← (R) + (M)	0000 000w mod reg r/m	7	7
(R) ← (R) + (M)	0000 001w mod reg r/m	6	6
(R/M) ← (R/M) + (im)	1000 00sw mod 000 r/m im8,16,32	2/7	2/7
(A) ← (A) + (im)	0000 010w im 8, 16, 32	2	2
ADC – додавання з переносом (CF)			
(R) ← (R) + (R) + (CF)	0001 00dw mod reg r/m	2	2
(M) ← (R) + (M) + (CF)	0001 000w mod reg r/m	7	7
(R) ← (R) + (M) + (CF)	0001 001w mod reg r/m	6	6
(R/M) ← (R/M) + (im) + (CF)	1000 00sw mod 010 r/m im8,16,32	2/7	2/7
(A) ← (A) + (im) + (CF)	0001 010w im 8,16, 32	2	2
INC – інкремент			
(R/M) ← (R/M) + 1	1111 111w mod 000 r/m	2/6	2/6
(R) ← (R) + 1	0100 0 reg	2	2
AAA – ASCII корекція додавання	0011 1111	4	4
DAA – десяткова корекція додавання	0010 0111	4	4
SUB – віднімання			
(R) ← (R) - (R)	0010 10dw mod reg r/m	2	2
(M) ← (M) - (R)	0010 100w mod reg r/m	7	7
(R) ← (R) - (M)	0010 101w mod reg r/m	6	6
(R/M) ← (R/M) - (im)	1000 00sw mod 101 r/m im8,16,32	2/7	2/7
(A) ← (A) - (im)	0010 110w im8,16,32	2	2
SBB – віднімання з запозиченням (CF)			
(R) ← (R) - (R) - (CF)	0001 10dw mod reg r/m	2	2
(M) ← (M) - (R) - (CF)	0001 100w mod reg r/m	7	7
(R) ← (R) - (M) - (CF)	0001 101w mod reg r/m	6	6
(R/M) ← (R/M) - (im) - (CF)	1000 00sw mod 011 r/m im8,16,32	2/7	2/7
(A) ← (A) - (im) - (CF)	0001 110w im8,16,32	2	2
DEC – декремент			
(R/M) ← (R/M) - 1	1111 111w reg 001 r/m	2/6	2/6
(R) ← (R) - 1	0100 1 reg	2	2

CMP – порівняння			
(R) - (R)	0011 10dw mod reg r/m	2	2
(M) - (R)	0011 100w mod reg r/m	5	5
(R) - (M)	0011 101w mod reg r/m	6	6
(R/M) - (im)	1000 00sw mod 111 r/m im8,16,32	2/5	2/5
(A) - (im)	0011 110w im8,16,32	2	2
NEG – зміна знаку	1111 011w mod 011 r/m	2/6	2/6
AAS – ASCII корекція віднімання	0011 1111	4	4
DAS – десяткова корекція	0010 1111	4	4
MUL – беззнакове множення			
(A) ← (A) * (R/M)	1111 011w mod 100 r/m		
- 8-розрядне		9-14 / 12-17	9-14 / 12-17
- 16-розрядне		9-22 / 12-25	9-22 / 12-25
- 32-розрядне		9-38 / 12-41	9-38 / 12-41
IMUL - множення ціле із знаком			
(A) ← (A) * (R/M)	1111 011w mod 101 r/m		
- 8-розрядне		9-14 / 12-17	9-14 / 12-17
- 16-розрядне		9-22 / 12-25	9-22 / 12-25
- 32-розрядне		9-38 / 12-41	9-38 / 12-41
(R) ← (R) * (R/M)	0000 1111 1010 1111 mod reg r/m		
- 8-розрядне		9-14 / 12-17	9-14 / 12-17
- 16-розрядне		9-22 / 12-25	9-22 / 12-25
- 32-розрядне		9-38 / 12-41	9-38 / 12-41
(R) ← (R/M) * im	0110 10s1 mod reg r/m im8,16,32		
- 8-розрядне		9-14 / 12-17	9-14 / 12-17
- 16-розрядне		9-22 / 12-25	9-22 / 12-25
- 32-розрядне		9-38 / 12-41	9-38 / 12-41
AAM – ASCII корекція	1101 0100 0000 1010	17	17
DIV – беззнакове ділення			
(A) ← (A) / (R/M)	1111 011w mod 110 r/m		
- 8-розрядне		14/17	14/17
- 16-розрядне		22/25	22/25
- 32-розрядне		38/41	38/41
IDIV – ділення ціле знакове			
(A) ← (A) / (R/M)	1111 011w mod 111 r/m		
- 8-розрядне		19/22	19/22
- 16-розрядне		27/30	27/30

- 32-розрядне		43/46	43/46
AAD – ASCII корекція	1101 0101 0000 1010	19	19
CBW / CWDE - перетворення з байта в слово / слова в подвійне слово	1001 1000	3	3
CWD / CDO - подвоєння слова / подвійного слова	1001 1001	2	2

Таблиця 3. Встановлення ознак при виконанні команд арифметичних, логічних операцій і зсувів

Команди	Ознаки					
	OF	CF	AF	SF	ZF	PF
ADD, ADC, SUB, SBB, CMP, NEG	+	+	+	+	+	+
INC, DEC	+	-	+	+	+	+
MUL, IMUL	+	+	н	н	н	н
DIV, IDIV	н	н	н	н	н	н
DAA, AAS	н	+	+	+	+	+
AAA, AAS	н	+	+	н	н	н
AAM, AAD	н	н	н	+	+	+
AND, OR, XOR, TEST	0	0	н	+	+	+
SHL, SHR (один розряд)	+	+	н	+	+	+
SHL, SHLD, SHR, SHRD (перем. розряд)	н	+	н	+	+	+
SAR	0	+	н	+	+	+
ROL, ROR, RCL, RCR (один розряд)	+	+	-	-	-	-
ROL, ROR, RCL, RCR (перем. розряд)	н	+	-	-	-	-

+ – встановлення ознаки за результатом операції
- – збереження раніше встановленої ознаки
н – невизначений стан ознаки
0 – встановлення нульового значення ознаки

Таблиця 4. Команди логічних операцій і зсувів

Команда	Формат	Число тактів	
NOT – інверсія (логічне НЕ)	1111 011w mod 010 r/m	2/6	2/6
AND – коньюнкція (логічне І)			
(R) ← (R) ∧ (R)	0010 00dw mod reg r/m	2	2
(M) ← (R) ∧ (M)	0010 000w mod reg r/m	7	7
(R) ← (R) ∧ (M)	0010 001w mod reg r/m	6	6
(R/M) ← (R/M) ∧ (im)	1000 00sw mod 100 r/m im8,16,32	2/7	2/7
(A) ← (A) ∧ (im)	0010 010w im8,16,32		
OR – диз'юнкція (логічне АБО)			
(R) ← (R) ∨ (R)	0000 10dw mod reg r/m	2	2
(M) ← (R) ∨ (M)	0000 100w mod reg r/m	7	7
(R) ← (R) ∨ (M)	0000 101w mod reg r/m	6	6
(R/M) ← (R/M) ∨ (im)	1000 00sw mod 001 r/m im8,16,32	2/7	2/7
(A) ← (A) ∨ (im)	0000 110w im8,16,32	2	2
XOR – виключне АБО			
(R) ← (R) + (R)	0011 00dw mod reg r/m	2	2
(M) ← (R) + (M)	0011 000w mod reg r/m	7	7
(R) ← (R) + (M)	0011 001w mod reg r/m	6	6

(R/M) ← (R/M) + (im)	1000 00sw mod 110 r/m im8,16,32	2/7	2/7
(A) ← (A) + (im)	0011 010w im8,16,32	2	2
TEST – логічне порівняння (встановлення ознак ZF, SF, PF)			
(R/M) ∧ (R)	1000 010w mod reg r/m	2/5	2/5
(R/M) ∧ (im)	1111 011w mod 011 r/m im8,16,32	2/7	2/7
(A) ∧ (im)	1010 100w im8,16,32		
SHL/SAL, SHR, SAR, ROL, ROR – циклічні зсуви			
(R/M) на 1 розряд	1101 000w mod TTT r/m	3/7	3/7
(R/M) на (CL) розрядів	1101 001w mod TTT r/m	3/7	3/7
(R/M) на (im) розрядів	1100 000w mod TTT r/m im8	3/7	3/7
RCL, RCR – циклічні зсуви через CF			
(R/M) на 1 розряд	1101 000w mod TTT r/m	9/10	9/10
(R/M) на (CL) розрядів	1101 001w mod TTT r/m	9/10	9/10
(R/M) на (im) розрядів	1100 000w mod TTT r/m im8	9/10	9/10
SHLD – подвійний зсув вліво			
(R/M) на (im) розрядів	0000 1111 1010 0100 mod reg r/m im8	3/7	3/7
(R/M) на (CL) розрядів	0000 1111 1010 0101 mod reg r/m	3/7	3/7
SHLD – подвійний зсув вправо			
(R/M) на (im) розрядів	0000 1111 1010 1100 mod reg r/m im8	3/7	3/7
(R/M) на (CL) розрядів	0000 1111 1010 1101 mod reg r/m	3/7	3/7

Таблиця 5. Коди операцій зсуву

TTT (КОП)	Команда
000	ROL
001	ROR
010	RCL
011	RCR
100	SHL / SAL
101	SHR
111	SAR

Таблиця 6. Команди бітових операцій

Команда	Формат	Число тактів	
BT – перевірка біта			
NB ← (im)	0000 1111 1011 1010 mod 100 r/m im8	3/6	3/6
NB ← (R)	0000 1111 1010 0011 mod reg r/m	3/12	3/12
BTS – перевірка і встановлення біту			
NB ← (im)	0000 1111 1011 1010 mod 010 r/m im8	6/8	6/8
NB ← (R)	0000 1111 1010 1011 mod reg r/m	6/13	6/13
BTR – перевірка і скидання біту			
NB ← (im8)	0000 1111 1011 1010 mod 110 r/m im8	6/8	6/8
NB ← (R)	0000 1111 1011 0011 mod reg r/m	6/13	6/13
BTC – перевірка та інверсія біту			
NB ← (im8)	0000 1111 1011 1010 mod 111 r/m im8	6/8	6/8

NB ← (R)	0000 1111 1011 1011 mod reg r/m	6/13	6/13
BTF – пряме сканування бітів	0000 1111 1011 1100 mod reg r/m	10+3n	10+3n
BSR – обернене сканування бітів	0000 1111 1011 1101 mod reg r/m	10+3n	10+3n

Таблиця 7. Команди операцій з рядками символів

Команда	Формат	Число тактів	
LODS – завантаження символа в акумулятор	1010 110w	5	5
STOS – запис символа з акумулятора	1010 101w	4	4
MOVS – пересилання символа	1010 010w	7	7
INS – введення символа	0110 110w	8	8
OUTS – виведення символа	0110 111w	7	7
SCAS – сканування символів	1010 111w	7	7
CMPS – порівняння символів	1010 011w	10	10
REP LODS – завантажити в аккумулятор	1111 0010 1010 110w	5+6n	5+6n
REP STOS – запам′ятати з акумулятора	1111 0010 1010 101w	5+5n	5+5n
REP MOVS – пересилання рядка	1111 0010 1010 010w	7+4n	7+4n
REP OUTS – вивести рядок в DX-порт	1111 0010 0110 111w	5+5n	5+5n
REP INS – ввести рядок з DX-порту	1111 0010 0110 110w	6+6n	6+6n
REPE CMPS – порівняння рядків	1111 0011 1010 011w	5+9n	5+9n
REPNE CMPS – порівняння рядків	1111 0010 1010 011w	5+9n	5+9n
REPE SCAS – сканування рядка за акумулятором	1111 0011 1010 111w	5+8n	5+8n
REPNE SCAS – сканування рядка за акумулятором	1111 0010 1010 111w	5+8n	5+8n

Таблиця 8. Команди безумовної передачі керування

Команда	Формат	Число тактів	
JMP – безумовний перехід в середині сегмента			
короткий відносний	1110 1011 d8	7+m	7+m
довгий відносний	1110 1001 d16,32	7+m	7+m
неявний	1111 1111 mod 100 r/m	7+m / 10+m	7+m / 10+m
JMP – безумовний перехід між сегментами			
прямий	1110 1010 ip16,eip32 sel16	12+m	23
неявний в середині сегмента	1111 1111 mod 101 r/m	17+m	28
CALL – виклик підпрограми в середині сегмента			
відносний	1110 1000 d16,32	7+m	7+m
неявний	1111 1111 mod 010 r/m	7+m / 10+m	7+m / 10+m
CALL – виклик підпрограми між сегментами			
прямий	1001 1010 ip16,eip32 sel16	17+m	35
неявний	1111 1111 mod 011 r/m	22+m	40
RET – повернення з підпрограми в середині сегмента			

без збільшення (SP)	1100 0011	10+m	10+m
із збільшенням (SP)	1100 0010 d16	10+m	10+m
RET – повернення із підпрограми між сегментами			
без збільшення (SP)	1100 1011	18+m	35
із збільшенням (SP)	1100 1010 d16	18+m	35

Таблиця 9. Команди умовних переходів, умовного встановлення байтів і організації циклів

Команда	Формат	Число тактів	
Jxx – умовний перехід			
короткий	0111 cond d8	7+m або 3	7+m або 3
довгий	0000 1111 1000 cond d16,32	7+m або 3	7+m або 3
SETxx – встановлення байтів	0000 1111 1001 cond mod000r/m	4/5	4/5
JCXZ, JECXZ – повторення циклу, якщо CX, ECX=0	1110 0011 d8	9+m або 5	9+m або 5
LOOP – цикл поки (CX)<>0; (CX) ← (CX)-1	1110 0010 d8	11+m	11+m
LOOPZ / LOOPE – цикл поки (CX)<>0 і ZF=1; (CX) ← (CX)-1	1110 0001 d8	11+m	11+m
LOOPNZ / LOOPNE – цикл поки (CX)<>0 і ZF=0; (CX) ← (CX)-1	1110 0000 d8	11+m	11+m

Таблиця 10. Коди умов

Мнемо-код	Умова	Код	Мнемо-код	Умова	Код
O	переповнення: (OF)=1	0000	NO	немає переповнення: (OF)=0	0001
B / NAE	нижче / не вище або дорівнює: (CF) = 1	0010	NB / AE	не нижче / вище або дорівнює: (CF) = 0	0011
E / Z	дорівнює / нуль: (ZF)=1	0100	NE / NZ	не дорівнює / не нуль: (ZF)=0	0101
BE / NA	нижче або дорівнює / не вище: (CF) ∨ (ZF)=1	0110	NBE / A	не нижче або дорівнює / вище: (CF) ∨ (ZF)=0	0111
S	від'ємний знак: (SF)=1	1000	NS	додатній знак: (SF)=0	1001
P / PE	парність: (PF)=1	1010	NP / PO	непарність: (PF)=0	1011
L / NGE	менше / не більше або дорівнює: (SF)+(OF)=1	1100	NL / GE	не менше / більше або дорівнює: (SF)+(OF)=0; ((SF)=(OF))	1101
LE / NG	менше або дорівнює / не більше: ((SF)+(OF))∨(ZF)=1	1110	NLE / G	не менше або дорівнює / більше: ((SF)+(OF))∨(ZF)=0	1111

Таблиця 11. Команди переривань і керування ознаками

Команда	Формат	Число тактів	
INT – переривання з заданим вектором	1100 1101 int8	37	59-287
INT3 – переривання в контрольній точці	1100 1100	33	59-283
INTO – переривання за переповненням: (OF)=1	1100 1110	35	59-285

IRET – повернення з підпрограми переривання	1100 1111	22	38-271
CLI – заборона переривань	1111 1010	3	3
STI – дозвіл переривань	1111 1011	3	3
LAHF – завантаження ознак з регістра EFLAGS в AH	1001 1111	2	2
SAHF – запис вмісту регістра AH в регістр EFLAGS	1001 1110	5	5
PUSHF – запис вмісту регістра ознак EFLAGS в стек	1001 1100	4	4
POPF – читання з стеку вмісту регістра ознак EFLAGS	1001 1101	3	3
CLC – скидання ознаки переносу: (CF)=0	1111 1000	2	2
CLD – скидання ознаки напрямку: (DF)=0	1111 1100	2	2
CMC – інвертування ознаки переносу	1111 0101	2	2
STC – встановлення ознаки переносу: (CF)=1	1111 1001	2	2
STD – встановлення ознаки напрямку: (DF)=1	1111 1101	2	2

Особливості адресування

Всі комірки пам’яті перенумеровані послідовно від 0 - мінімальної адреси пам'яті. Процесор забезпечує доступ до байтів чи слів у пам'яті. Розглянемо десяткове число 1025. Для запису у пам'ять шістнадцяткового зображення цього числа - 0401 потрібно два байти або одне слово. Воно складається з старшої частини - 04 і молодшої частини – 01. Система зберігає в пам'яті байти слова у зворотній послідовності: молодша частина за меншою адресою, а старша - за більшою адресою. Припустимо, що процесор записав шістнадцяткове значення 0401 з регістру в комірки пам'яті 5612 і 5613 таким чином:

```
          |   01   |    04    |
              |          |
  комірка  5612        комірка 5613
  молодший байт        старший байт
```

Процесор припускає, що байти числових даних в пам'яті записані в зворотній послідовності, і опрацювує їх відповідно. Не зважаючи на те, що ця властивість повністю автоматизована, варто завжди пам'ятати про цей факт при програмуванні та налагодженні асемблерних програм.

Приклад машинних кодів: безпосередні дані

Мета цього прикладу - проілюструвати просту програму машинною мовою, її зображення в пам'яті та результати її виконання. Програма показана в шістнадцятковому форматі:

Команда	Призначення
B82301	Переслати значення 0123H в AX
052500	Додати значення 0025H до AX
8BD8	Переслати вміст AX у BX
03D8	Додати вміст AX до BX
8BCB	Переслати вміст BX у CX
2BC8	Відняти вміст AX від AX (очистка AX)
90	Немає операції
CB	Повернення в DOS

Можна зауважити, що машинні команди мають різну довжину: один, два або три байти. Машинні команди знаходяться в пам'яті безпосередньо одна за одною. Виконання програми починається з першої команди і далі послідовно виконуються решта.

Приклад машинних кодів: визначення даних

У попередньому прикладі використовувались безпосередні дані, описані безпосередньо в перших двох командах (MOV і ADD). Тепер розглянемо аналогічний приклад, в якому значення 0123 і 0025 визначені в двох полях сегменту даних. Цей приклад дозволяє зрозуміти, як комп'ютер забезпечує доступ до даних з допомогою регістра DS і адресного зміщення.

У поданому прикладі визначені ділянки даних, які мають відповідно такі значення:

Адреса в DS	*Шістнадцяткові значення*	*Номери байтів*
0000	2301	0 і 1
0002	2500	2 і 3
0004	0000	4 і 5
0006	2A2A2A	6, 7 і 8

Нагадаємо , що шістнадцятковий символ займає половину байта, так, наприклад, 23 знаходиться в байті 0 (у першому байті) сегменту даних, 01 - в байті 1 (у другому байті).

Нижче показані команди машинної мови, які опрацювують ці дані:

Команда	*Призначення*
A10000	Переслати слово (два байти), яке починається в DS за адресою 0000, у регістр AX.
03060200	Додати вміст слова (двох байтів), яке починається в DS за адресою 0002, до регістра AX.
A30400	Переслати вміст регістра AX у слово, що починається в DS за адресою 0004.
CB	Повернутися в DOS.

Звернемо увагу, що тут є дві команди MOV з різними машинними кодами: A1 і A3. Фактично машинний код залежить від регістрів, на які маємо посилання, кількості байтів (байт чи слово), напрямку передачі даних (з регістра чи у регістр) і від посилання на безпосередні дані чи на пам'ять.

Машинне адресування

Для доступу до машинної команди процесор визначає її адресу за регістром CS плюс зміщення в регістрі IP. Наприклад, нехай регістр CS містить 04AFH (дійсна адреса 04AF0), а регістр IP містить 0023H:

```
        CS:    04AF0
        IP:     0023
Адреса команди: 04B13
```

Якщо, наприклад, за адресою 04B13 знаходиться команда

```
        A11200      MOV  AX, [ 0012 ]
        |
        адреса 04B13
```

то в пам'яті за адресою 04B13 розташований перший байт команди. Процесор отримує доступ до цього байта і за кодом команди (A1) визначає довжину команди - три байти.

Для доступу до даних за зміщенням [0012] процесор визначає адресу, виходячи з вмісту регістру DS (як правило) плюс зміщення в операнді команди. Якщо DS містить 04B1H (реальна адреса 04B10), то результуюча адреса визначається так:

DS: 04B10
Зміщення: 0012
Адреса даних: 04B22

Припустимо, що за адресами 04B22 і 04B23 знаходяться такі дані:

Вміст: 24 01
| |
Адреса: 04B22 04B23

Процесор вибирає значення 24 з комірки за адресою 04B22 і розміщує його в регістрі AL, а значення 01 за адресою 04B23 – в регістрі AH. Регістр AX буде мати в результаті 0124. У процесі вибирання кожного байта команди процесор збільшує значення регістра IP на одиницю, так що до початку виконання наступної команди в нашому прикладі IP буде містити зміщення 0026. Отже, процесор тепер готовий до виконання наступної команди, яку він отримує за адресою з регістра CS (04AF0) плюс поточне зміщення в регістрі IP (0026), тобто 04B16.

Формат мови асемблера

Кожна асемблерна команда записується в один рядок. Формат рядка є такий:

| поле імені | поле операції | поле операндів | поле коментаря |

або:

| поле (рядок) коментаря |

Будемо вважати для простоти, що поле імені, якщо воно є, повинно починатись в першій позиції рядка. Те ж саме стосується рядка-коментаря. Ім'я не має в кінці знаку '**:**'. Поля відокремлюються між собою одним або більше пропусками. Обов'язковими є лише поле операції та поле операндів, якщо операція вимагає операндів. Поле чи рядок коментаря починається знаком '**;**'. Операнди розділяються між собою комами.

Не допускається вживання пропусків всередині поля імені, поля операції, поля операндів. Однак пропуск може бути літерою-значенням операнда, взятим в лапки.

Будемо вважати, що імена складаються не більше, ніж з восьми латинських букв та цифр, і повинні обов'язково починатися буквою. Великі та малі букви вважатимемо однаковими, наприклад:

Abc12, aBc12, ABC12, abc12 - це все однакові імена.

Константи

Двійкова – послідовність цифр 0 і 1, що закінчується буквою B: 11001010B.

Десяткова – послідовність цифр від 0 до 9, що може закінчуватись буквою D: 419 або 419D.

Шістнадцяткова – послідовність цифр від 0 до 9 і букв від A до F, що закінчується буквою H. Першим символом обов'язкова повинна бути цифра: 105H, 0C04BH.

Літерали – рядок букв, цифр та інших символів, взятих в лапки або апострофи. Дві форми передбачені для того, щоб вставляти в текст самі лапки або апострофи: "Значення функції", 'Значення функції', "Відповідь на запитання 'Ваш вибір?' є такою:".

Від'ємні числа

Якщо число є десяткове, то перед ним ставиться знак мінус: -26.

Якщо число є двійкове або шістнадцяткове, то його треба записувати у доповнювальному коді. Наприклад, для числа –32 записуємо 11100000B або 0E0H.

Директиви асемблера

<ім'я>	SEGMENT	визначає ім'я сегменту (частини) програми
<ім'я>	ENDS	вказує на кінець сегменту
<ім'я>	PROC	фіксує початок процедури
<ім'я>	ENDP	фіксує кінець процедури
	END <ім'я>	вказує кінець тексту програми і задає першу виконувану команду програми
	ORG <число>	зміна лічильника адреси на вказане число і, відповідно, адреси наступного елемента; лічильник набуває значення числа (LOCCTR : = <число>)
[<ім'я>]	DW <число> | ?	формує двобайтову константу – число, або резервує 2 байти (слово) пам'яті
[<ім'я>]	DB <число> | ? | '<рядок>'	а) формує однобайтову константу - число; б) резервує 1 байт пам'яті; в) формує в пам'яті заданий рядок; довжина рядка (кількість байтів) визначається його записом.