Лабораторна робота № 5
КОМАНДИ ПЕРЕСИЛАННЯ ДАНИХ. СТЕК

Мета роботи: вивчення режимів адресації, правил адресації при використанні регістрів, директив визначення даних, стеку.

Теоретичні відомості

Команди пересилання даних

Команда `MOV` (від англ. move – рух) – базова команда пересилання даних, яка копіює вміст джерела в приймач (табл. 5.1), при цьому джерело не змінюється.

Таблиця 5.1

Основні характеристики команди **`MOV`**

Команда	Призначення	Процесор
`MOV` `приймач, джерело`	Пересилання (копіювання, присвоєння) даних	8086

Команда `MOV` діє аналогічно операторам присвоєння в мовах високого рівня:

```
приймач: = джерело (або приймач = джерело)
```

Приклад: команда процесора

```
mov ax, bx
```

є еквівалентною виразу мови С

```
ax = bx;
```

Слід зауважити, що на відміну від мов високого рівня команди мови Асемблер дають змогу працювати не тільки зі змінними в пам'яті, а й з усіма регістрами процесора.

Як *джерело* можуть використовуватися число (безпосередній операнд), регістр загального призначення, сегментний регістр або змінна (тобто операнд, що знаходиться в пам'яті). *Приймачем* може бути регістр загального призначення, сегментний регістр (крім `CS`) або змінна. *Обидва операнди мають бути одного й того самого розміру*: байт, слово (2 байти) або подвійне слово (4 байти).

Не можна виконувати пересилання даних з допомогою `MOV` з

однієї змінної в іншу, з одного сегментного регістра в інший і не можна поміщати в сегментний регістр безпосередній операнд. Ці операції виконують двома командами `MOV` або парою команд `PUSH` і `POP`.

Наприклад, скористаємося командами `MOV`, для чого скопіюємо вміст сегментного регістра у звичайний і вже з нього – в інший сегментний. При цьому необхідно мати один вільний регістр:

```
mov eax,cs
mov ds,eax
```

Команда CMOV* (від англ. conditional move – умовний рух) – це набір команд, які копіюють вміст джерела в приймач, якщо задовольняється та чи інша умова (табл. 5.2). Символ * в кінці команди означає від однієї до трьох уточнювальних букв (табл. 5.3).

Таблиця 5.2

Основні характеристики команди **CMOV***

Команда	Призначення	Процесор
CMOV* `приймач, джерело`	Умовне пересилання даних	P6

Джерелом може бути регістр загального призначення або змінна, а *приймачем* – тільки регістр. Умова, яка має задовольнятися, – просто рівність нулю або одиниці тих чи інших прапорців з регістра `FLAGS (PSW)`, але якщо використовувати команди `CMOV*` відразу після команди `CMP` (порівняння) з тими самими операндами, то умови набувають особливого змісту, наприклад:

```
cmp   cx,dx;     порівняти cx і dx
cmovl ax,bx;     якщо cx < dx, скопіювати bx в ax
```

Таблиця 5.3

Різновиди команди `CMOV*`

Код команди	Реальна умова	Умова для CMP
CMOVA CMOVNBE	CF = 0 і ZF = 0	Якщо вище Якщо нижче або дорівнює
CMOVAE CMOVNB CMOVNC	CF = 0	Якщо вище або дорівнює Якщо не нижче Якщо немає переносу
CMOVB CMOVNAE CMOVC	CF = 1	Якщо нижче Якщо вище або дорівнює Якщо перенесення

Закінчення табл. 5.3

Код команди	Реальна умова	Умова для CMP
CMOVB CMOVNAE CMOVC	CF = 1	Якщо нижче Якщо вище або дорівнює Якщо перенесення
CMOVBE CMOVNA	CF = 1 і ZF = 0	Якщо нижче або дорівнює Якщо не вище
CMOVE CMOVZ	ZF = 1	Якщо дорівнює Якщо нуль
CMOVG CMOVNLE	ZF = 0 і SF = OF	Якщо більше Якщо не менше або дорівнює
CMOVGE CMOVNL	SF = OF	Якщо більше або дорівнює Якщо не менше
CMOVL CMOVNGE	SF <> OF	Якщо менше Якщо не більше або дорівнює
CMOVLE CMOVNG	ZF = 1 і SF <> OF	Якщо менше або дорівнює Якщо не більше
CMOVNE CMOVNZ	ZF = 0	Якщо не дорівнює Якщо не нуль
CMOVNO	OF = 0	Якщо немає переповнення
CMOVO	OF = 1	Якщо є переповнення
CMOVNP CMOVPO	PF = 0	Якщо немає парності Якщо є непарним
CMOVP CMOVPE	PF = 1	Якщо є парність Якщо є парним
CMOVNS	SF = 0	Якщо немає знака
CMOVS	SF = 1	Якщо є знак

Слова «вище» і «нижче» у табл. 5.3 належать до порівняння чисел без знака, слова «більше» і «менше» – зі знаком. Букви і їх комбінації, що підставляють замість знака *, означають таке:

A = Above – вище; B = Below – нижче; G = Greater – більше; L = Less – менше; N = Not – ні; E = Equal – дорівнює; C = CF (Carry Flag) – прапорець перенесення; Z = ZF (Zero Flag) — прапорець нуля; O = OF (Overflow Flag) — прапорець переповнення; P = PF (Parity Flag) — прапорець парності;
S = SF (Sign Flag) — прапорець знака.

Команда `XCHG` (від англ. exchange – обмін) – вміст операнду 2 копіюється в операнд 1, а попередній вміст операнду 1 – в операнд 2 (табл. 5.4).

Команду XCHG можна виконувати над двома регістрами або над регістром і змінною:

```
xchg eax,ebx ; теж саме, що три команди на мові C:
             ; temp = eax; eax = ebx; ebx = temp;

xchg al,al      ; ця команда не робить нічого
```

Таблиця 5.4

Основні характеристики команди XCHG

Команда	Призначення	Процесор
XCHG операнд1, операнд2	Обмін операндів між собою	8086

Цю саму дію можна виконати з допомогою трьох команд MOV, додатково використовуючи регістр EDX:

```
mov edx,eax
mov eax,ebx
mov ebx,edx
```

Однак це забере більше часу й необхідно, щоб був один вільний регістр.

Команда BSWAP (від англ. byte SWAP – обмін байтів) звертає порядок байтів у 32-бітному регістрі (табл. 5.5).

Таблиця 5.5

Основні характеристики команди BSWAP

Команда	Призначення	Процесор
BSWAP регістр32	Обмін байтів усередині регістра	80486

Біти 0–7 (молодший байт молодшого слова) міняються місцями з битами 24–31 (старший байт старшого слова), а біти 8–15 (старший байт молодшого слова) міняються місцями з бітами 16–23 (молодший байт старшого слова), наприклад:

```
mov eax,12345678h  ;поміщаємо в eax число 12345678h
bswap eax  ;тепер в eax знаходиться число 78563412h
```

Команду BSWAP неможливо замінити сукупністю команд XCHG, оскільки немає доступу до окремих байтів старшої половини 32-розрядного регістра. Слід зазначити, що команда BSWAP міняє місцями байти цілком, а не окремі дзеркально протилежні біти. Щоб звернути

порядок байтів у 16-бітному регістрі, слід використовувати команду XCHG:

```
xchg  al,ah    ; звернути порядок байтів у AX
```

У процесорах Intel команду BSWAP можна використовувати і для звернення порядку байтів у 16-бітових регістрах, але в деяких сумісних процесорах інших фірм цей варіант BSWAP не реалізовано.

Команда IN (від англ. in – в (уведення)) копіює число з порту уведення-виведення (номер якого вказано в джерелі) у приймач (табл. 5.6).

Таблиця 5.6

Основні характеристики команди IN

Команда	Призначення	Процесор
IN приймач, джерело	Зчитати дані з порту	8086

Приймачем може бути тільки AL, AX або EAX. *Джерело* – або безпосередній операнд, або DX, причому можна вказувати тільки номери портів не більше 255.

Команда OUT (від англ. out – з (виведення)) копіює число з джерела (AL, AX або EAX) у порт уведення-виведення, номер якого вказано в приймачі (табл. 5.7). *Приймач* може бути або безпосереднім номером порту, або регістром DX.

Таблиця 5.7

Основні характеристики команди OUT

Команда	Призначення	Процесор
OUT приймач, джерело	Записати дані в порт	8086

На командах IN і OUT будується все спілкування процесора з пристроями уведення-виведення (ПУВ): клавіатурою, жорсткими дисками, різними контролерами. Їх використовують насамперед у драйверах пристроїв.

Команда CWD (від англ. convert word in double – конвертувати слово на подвійне) перетворює слово в AX на подвійне слово, молодша половина якого (біти 0–15) залишається в AX, а старша (біти 16–31) розташовується в DX (табл. 5.8).

Команда CDQ (від англ. convert double in quadruple – конвертувати подвійне слово на вчетверо більше) виконує аналогічну дію відносно подвійного слова в EAX, розширюючи його до вчетверо більшого слова в EDX: EAX (табл. 5.9).

Таблиця 5.8

Основні характеристики команди CWD

Команда	Призначення	Процесор
CWD	Конвертація слова на подвійне слово	8086

Таблиця 5.9

Основні характеристики команди CDQ

Команда	Призначення	Процесор
CDQ	Конвертація подвійного слова на вчетверо більше	80386

Ці команди всього лише встановлюють усі біти регістра DX або EDX у значення, що дорівнює значенню старшого біта регістра AX або EAX, зберігаючи таким чином його знак.

Команда CBW (від англ. convert byte in word – конвертувати байт у слово) розширює байт, що знаходиться в регістрі AL, до слова в AX (табл. 5.10).

Таблиця 5.10

Основні характеристики команди CBW

Команда	Призначення	Процесор
CBW	Конвертація байта на слово	8086

Команда CWDE (від англ. convert word in double in EAX – конвертувати байт на слово в EAX) розширює слово в AX до подвійного слова в EAX (табл. 5.11).

Таблиця 5.11

Основні характеристики команди CWDE

Команда	Призначення	Процесор
CWDE	Конвертація слова в подвійне слово	80386

Команди CWDE і CWD різняться тим, що CWDE має свій результат в EAX, тоді як CWD (команда, що виконує таку саму дію) має результат у парі регістрів DX:AX. Так само як команди CWD і CDQ, тобто розширення виконується шляхом установлення кожного біта старшої половини результату таким, що дорівнює старшому біту вихідного байта або слова:

```
mov al,0F5h   ; AL = 0F5h = 245 = -11
```

```
cbw             ; зараз AX = 0FFF5h = 65 525 = -11
```

Команда MOVSX (від англ. move with sign extending – рух з розширенням знаку) копіює вміст джерела (реєстр або змінна розміром один байт або одне слово) у приймач (16- або 32-бітний регістр) і розширює знак аналогічно командам CBW і CWDE (табл. 5.12).

Таблиця 5.12

Основні характеристики команди MOVSX

Команда	Призначення	Процесор
MOVSX приймач, джерело	Пересилання з розширенням знака	80386

Команда MOVZX (від англ. move with zeros extending – рух з розширенням нулями) копіює вміст джерела (реєстр або змінна розміром один байт або одне слово) у приймач (16- або 32-бітний регістр) і розширює нулями (табл. 5.13).

Таблиця 5.13

Основні характеристики команди MOVZX

Команда	Призначення	Процесор
MOVZX приймач, джерело	Пересилання з розширенням нулями	80386

Наприклад, команда

```
movzx  ax,bl
```

є еквівалентною парі команд

```
mov  al,bl
mov  ah,0
```

Команда XLAT поміщає в AL один байт із таблиці в пам'яті за адресою ES:BX (або ES:EBX) зі зміщенням відносно початку таблиці, що дорівнює AL. Як аргумент для команди XLAT у мові Асемблер можна вказати ім'я таблиці, але ця інформація ніяк не використовується процесором і є коментарем. Якщо цей коментар не потрібен, можна застосувати **команду XLATB** (табл. 5.14). Як приклад використання XLAT можна написати такий варіант перетворення шістнадцяткового числа на ASCII-код символу, що йому відповідає:

```
mov  al,0Ch
```

```
mov  bx,offset htable
xlatb
```

Якщо в сегменті даних, на який вказує регістр ES, було записано

```
htable   db     "0123456789ABCDEF"
```

то тепер AL містить не число 0Ch, а ASCII-код літери «C».

Таблиця 5.14

Основні характеристики команд XLAT і XLATB

Команда	Призначення	Процесор
XLAT адреса XLATB	Трансляція відповідно до таблиці	8086

Команда LEA (табл. 5.15) обчислює ефективну адресу джерела (змінна) і поміщає його в приймач (реєстр).

Таблиця 5.15

Основні характеристики команди LEA

Команда	Призначення	Процесор
LEA приймач, джерело	Обчислення ефективного адресу	8086

З допомогою LEA можна обчислити адресу змінної, яку описано складним методом адресації, наприклад, за базою з індексуванням. Якщо адреса є 32-бітною, а регістр-приймач – 16-бітним, то старша половина обчисленої адреси втрачається, якщо, навпаки, приймач є 32-бітним, а адреса – 16-бітною, то обчислене зміщення доповнюється нулями.

Команду LEA часто використовують для швидких арифметичних обчислень, наприклад множення:

```
lea bx,[ebx+ebx*4]  ; BX=EBX*5
```

або складання:

```
lea  ebx,[eax+12]    ; EBX=EAX+12
```

(ці команди є меншими, ніж MOV і ADD, і не змінюють прапорці).

Стек

У ПК є спеціальні команди роботи зі стеком, тобто областю пам’яті, доступ до елементів якої здійснюється за принципом «останнім записаний – першим зчитаний». Однак для того щоб можна було скористатися цими командами, необхідно дотримуватися кількох умов.

Під стек можна відвести область в будь-якому місці пам'яті. Розмір її може бути будь-яким, але не більше 64 Кб, а її початкова адреса має бути кратною 16. Іншими словами, ця область має бути сегментом пам'яті (його ще називають сегментом стеку). Початок цього сегмента (перші 16 бітів початкової адреси) має обов'язково зберігатися в сегментному регістрі SS.

Основні стекові команди. При дотриманні зазначених вимог у програмі можна використовувати команди, призначені для роботи зі стеком. Основні з них наведено в табл. 5.16–5.19.

Команда `PUSH` (від англ. push – штовхати) поміщає вміст джерела в стек (див. табл. 5.16). *Джерелом* може бути регістр загального призначення, сегментний регістр, безпосередній операнд або змінна розміром одне або два слова (два або чотири байти). Фактично ця команда копіює вміст джерела в пам'ять за адресою `SS:ESP` і зменшує `ESP` на розмір джерела в байтах (на два або чотири).

Таблиця 5.16

Основні характеристики команди `PUSH`

Команда	Призначення	Процесор
`PUSH джерело`	Помістити дані в стек	8086

Команда `POP` (від англ. pop – виштовхувати) поміщає в приймач слово або подвійне слово, яке знаходиться у вершині стеку, збільшуючи до цього `ESP` на два або чотири байти відповідно. `POP` виконує дію, повністю протилежну `PUSH` (табл. 5.17). *Приймачем* може бути регістр загального призначення, сегментний регістр, крім `CS` (щоб завантажити `CS` зі стеку, треба скористатися командою `RET`), або змінна. Якщо приймачем є операнд, який використовує `ESP` для непрямої адресації, команда `POP` обчислює адресу операнда вже після того, як вона збільшує `ESP`.

Таблиця 5.17

Основні характеристики команди `POP`

Команда	Призначення	Процесор

POP приймач	Помістити дані в стек	8086

Команда PUSH майже завжди використовується в парі з POP. Так, наприклад, щоб скопіювати вміст одного сегментного регістра в інший (що можна виконати однією командою MOV), необхідно використовувати таку послідовність команд:

```
    push cs
    pop  ds  ; тепер DS вказує на той самий сегмент, що й
CS
```

Інше найчастіше застосування команд PUSH і POP – тимчасове зберігання даних, наприклад:

```
push eax    ; зберігає поточне значення EAX
...         ;тут розташовуються деякі команди,
            ;які використовують і змінюють EAX
pop  eax    ;відновлює попереднє значення EAX
```

Починаючи з 80286, команда push ESP (або SP) поміщає в стек значення ESP до того, як ця ж команда його зменшить, тоді як на 8086 SP містився в стек вже зменшеним на два.

Команда PUSHA (від англ. push all – штовхати все) поміщає в стек регістри в такому порядку: AX, CX, DX, BX, SP, BP, SI і DI. **Команда PUSHAD** (від англ. push all double – штовхати всі подвійні (слова)) поміщає в стек EAX, ECX, EDX, EBX, ESP, EBP, ESI і EDI (табл. 5.18). При роботі з регістрами SP і ESP використовується значення, яке знаходилося в цьому регістрі до початку роботи команди.

Таблиця 5.18

Основні характеристики команд PUSHAD і PUSHA

Команда	Призначення	Процесор
PUSHAD PUSHA	Помістити в стек всі регістри загального призначення	80386 80186

Команди POPAD і **POPA** виконують дії (табл. 5.19), що є повністю оберненими до дій PUSHA і PUSHAD, за винятком того, що поміщене в стек значення SP або ESP ігнорується. POPA завантажує зі стеку DI, SI, BP, збільшує SP на два, завантажує BX, DX, CX, AX, а POPAD завантажує EDI, ESI, EBP, збільшує ESP на чотири байти і завантажує

EBX, EDX, ECX, EAX.

Таблиця 5.19

Основні характеристики команд POPAD і POPA

Команда	Призначення	Процесор
POPAD POPA	Завантажити зі стеку всі регістри загального призначення	80386 80186

Команди PUSHA і PUSHAD разом з командами POPA і POPAD дають змогу писати підпрограми (зазвичай обробники переривань), які не повинні змінювати значення **регістрів** після закінчення своєї роботи. На початку такої підпрограми викликають команду PUSHA або PUSHAD, а в кінці – POPA або POPAD.

Завдання

1. Написати наведену нижче програму з ім'ям Lab5.asm, зробити виконуваний файл.

```
SSEG     segment para stack 'stack'
Db       1, 2, 3, 4, 5, 6, 7, 8, 9, 128 dup(0Ah)
SSEG     ends
DSEG     segment para public 'data'
B_TAB    db 1Ah, 2Bh, 3Ch, 4Dh, 5Eh, 6Fh, 7Ah, 8Bh
W_TAB    dw 1A2Bh, 3C4Dh, 5E6Fh, 7A8Bh
B_TAB1   db 0Ah,8 dup(1)
W_TAB1   dw 8 dup(1)
DSEG     ends
ESEG     segment
W_TAB2   dw 11h,12h,13h,14h,15h,16h,17h,18h
ESEG     ends
CSEG     segment para public 'code'
PROG     proc far
         assume ds:DSEG, cs:CSEG, ss:SSEG, es:ESEG
         push ds
         mov ax,0
         push ax
;ініціалізація сегментних регістрів
         ov ax,dseg
         mov ds,ax
         mov ax,eseg
         mov es,ax
; безпосередня (операнд-джерело)
         mov al,-3            ; розширення знака
         movax,3
```

```
          mov B_TAB,-3
          mov W_TAB,-3
          mov ax,2A1Bh
;регістрова
          mov bl,al
          mov bh,al
          sub ax,bx
          sub ax,ax
;пряма
          mov ax,W_TAB
          mov ax,W_TAB+3
          mov ax,W_TAB+5
          mov al,byte ptr W_TAB+6
          mov al,B_TAB
          mov al,B_TAB+2
          mov ax,word ptr B_TAB
          mov es:W_TAB2+4,ax
;непряма
          mov bx,offset B_TAB
          mov si,offset B_TAB+1
          mov di,offset B_TAB+2
          mov dl,[bx]
          mov dl,[si]
          mov dl,[di]
          mov ax,[di]
          mov bp,bx
          mov al,[bp]
          mov al,ds:[bp]
          mov al,es:[bx]
          mov ax,cs:[bx]
;базова
          mov ax,[bx]+2    ;основна форма
          mov ax,[bx]+4
          mov ax,[bx+2]
          mov ax,[4+bx]
          mov ax,2+[bx]
          mov ax,4+[bx]
          mov al,[bx]+2
          mov bp,bx             ;інший базовий регістр
          mov ax,[bp+2]
          mov ax,ds:[bp]+2      ;спроба переназначити
                                ;сегментний регістр
          mov ax,ss:[bx+2]
;індексна
          mov si,2                  ;завантаження
```

```
індексу
            mov ah,B_TAB[si]          ;основна форма
            mov al,[B_TAB+si]
            mov bh,[si+B_TAB]
            mov bl,[si]+B_TAB
            mov bx,es:W_TAB2[si]
            mov di,4
            mov bl,byte ptr es:W_TAB2[di]
            mov bl,B_TAB[si]
;базова індексна
            mov bx,offset B_TAB       ;завантаження бази
            mov al,3[bx][si]          ;основна форма
            mov ah,[bx+3][si]
            mov al,[bx][si+2]
            mov ah,[bx+si+2]
            mov bp,bx
            mov ah,3[bp][si]
            mov ax,ds:3[bp][si]
            mov ax,word ptr ds:2[bp][si]
            ret
PROG endp
CSEG ends
end PROG
```

2. На основі роботи програми в графи 2 і 3 таблиці 5.20 записати очікувані значення операндів.

Таблиця 5.20

Оператор	Операнд-приймач	
	До виконання	Після виконання

3. На основі виконаної роботи написати звіт, занести до нього короткі висновки.