МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний авіаційний університет

МІКРОПРОЦЕСОРИ В СИСТЕМАХ ТЕХНІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

Лабораторний практикум

для студентів напряму підготовки 6.170102

"Системи технічного захисту інформації"

Київ 2012

ББК З 973. 26Р
УДК 681. 325. 5-181 4(076. 5)

Укладачі: В. А. Швець, Т. В. Мелешко

Лабораторні роботи складені відповідно до програми курсу, містять у собі рекомендації по їхньому виконанню й оформленню, а також стислі теоретичні відомості і необхідний довідковий матеріал.

Призначені для студентів напряму підготовки 6.170102 "Системи технічного захисту інформації"

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Мета лабораторних робіт – ознайомлення з елементами мікропроцесорних пристроїв, одержання навичок упорядкування програм на мові Асемблера, в роботі з інструментальними та налагоджувальними засобами мікропроцесорних пристроїв, а також поглиблення і конкретизація програмного матеріалу, який вивчається в лекційній частині курсу.

На початку циклу лабораторних робіт студенти проходять інструктаж з техніки безпеки, правила якої зобов'язані дотримуватись в ході робіт, а також інструктаж з основних правил роботи на персональній ЕОМ.

Виконання кожної лабораторної роботи складається із доаудиторної підготовки, що включає:

вивчення відповідного теоретичного матеріалу;

ознайомлення з лабораторною роботою;

упорядкування алгоритмів і програм, якщо це потрібно відповідно до опису роботи;

відповіді на контрольні питання;

підготовка звіту з лабораторної роботи;

проведення лабораторних досліджень, опрацювання їх результатів досліджень;

оформлення звіту і захисту роботи.

Перед виконанням чергової лабораторної роботи студент повинний перед’явити викладачеві цілком оформлений звіт по попередній роботі, а також матеріал, необхідний для виконання наступної роботи.

По закінченні роботи, студент пред'являє отримані результати викладачеві. Якщо робота виконана неправильно, студент повинний виконати її вдруге.

Звіт з лабораторної роботи кожний студент складає самостійно. Звіт повинен містити в собі:

назву та мету лабораторної роботи;

основні теоретичні відомості;

алгоритми та вихідні тексти досліджуваних програм;

таблиці експериментальних даних і графіки;

висновки за результатами роботи.

Залік з виконаної роботи відбувається до виконання наступної лабораторної роботи. Якщо попередня робота не виконана, студент не допускається до виконання наступної лабораторної роботи.

ЛАБОРАТОРНА РОБОТА 1

ДОСЛІДЖЕННЯ НА МІКРОПРОЦЕСОРНИХ ПРИСТРОЯХ АЛГОРИТМІВ АРИФМЕТИЧНИХ ОПЕРАЦІЙ З ОДНОБАЙТОВИМИ І МНОГОБАЙТОВИМИ ЦІЛИМИ ЧИСЛАМИ (ДОДАВАННЯ ТА ВІДНІМАННЯ)

Мета роботи: ознайомлення з принципами програмування на мові АСЕМБЛЕР МП 1810, із використанням налагоджених засобів, з дослідженням алгоритмів арифметичних операцій додавання і віднімання однобайтових і многобайтових цілих чисел.

Стислі теоретичні відомості й приклади програм

Арифметичні операції над цілими числами в будь-який позиційній системі числення виконують за тими же правилам, що й у десятковій арифметиці, оскільки всі вони грунтуються на загальних правилах виконання операцій над відповідними поліномами. При цьому використовують ті таблиці додавання (віднімання) і множення, що мають місце в конкретній системі числення. Зокрема, для двійкової системи діють правила:

0+0=0;	0–0=0;	0×0=0;
0+1=1;	1–0=1;	0×1=0;
1+0=1;	1–1=0;	1×0=0;
1+1=10;	10–1=1;	1×1=1.

Ці правила не містять операцій ділення двійкових цифр, оскільки їх виконання на відміну від інших операцій не може бути зведене до дій над окремими цифрами. При додаванні в двійковому розряді двох одиниць відбувається перенос із даного розряду в наступний, більш старший, а при вирахуванні в двійковому розряді одиниці з нуля відбувається позика в даний розряд із більш старшого розряду, у результаті в даному розряді встановлюється одиниця.

При виконанні арифметичних дій виникає проблема ідентифікації від'ємних чисел. Для зображення знака числа приділяється спеціальний знаковий розряд (звичайно старший розряд числа) S. Зображення знака "+" у цьому розряді прийнято кодувати для двійкових чисел цифрою 0, а знака "–" – цифрою 1. При цьому, зображення числа із знаком містять тільки двійкові цифри, але припускається, що число складається з двох частин: знакової та цифрової.

Якщо цифрова частина додатних і від'ємних чисел містить завжди абсолютне значення числа, то такий засіб представлення знакових чисел називають прямим кодом. Обробка поданих у прямому коді чисел потребує окремих операцій над цифровою і знаковою частинами. Виконання операцій додавання і віднімання, призводить до появи двох представлень нуля: +0 і –0 (наприклад, +0=00000000; –0=10000000). Ці недоліки прямого коду стримують його вико-

ристання для обробки даних. У мікропроцесорах для обробки знакових чисел використовують в основному додаткові коди.

Зміст використання додаткового коду полягає в тому, що, по-перше, арифметичні операції віднімання і додавання чисел у додаткових кодах зводяться до операції алгебраїчного підсумовування (віднімання замінюється додаванням зменшуваного з додатковим кодом, що віднімається); по-друге, обробка знакової й цифрової частин чисел при додаванні провадиться за тими самими правилами, причому правильний знак результату формується автоматично.

У мікропроцесорі 1810 розрядність формату даних може бути подана як n=8N, де N– кількість байтів (N≥1), операції додавання (віднімання) чисел для випадків N=1 і N=2 виконуються за допомогою відповідних команд (див. програми 1.1 і 1.2).

Програма 1.1

```
 TITLE   ДОДАВАННЯ ЦІЛИХ БЕЗЗНАКОВИХ ЧИСЕЛ
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      256 DUP(0)
SSEG   ENDS
; Визначення сегмента даних для доданків і результату
DSEG    SEGMENT PARA PUBLIC     'DATA'
SLOG1   DW      1 DUP (0) ; Доданок 1, довжина слово
SLOG2   DW      1 DUP (0) ; Доданок 2, довжина слово
SUM     DD      2 DUP(0)  ; Результат, довжина 3 байта
DSEG    ENDS
; Визначення сегмента коду програми
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
SUMMA   PROC    FAR
       mov     ax,DSEG    ; Визначення адреси сегмента DATA
       mov     ds,ax          ; Пересилання в сегментний регістр DS
Start:  clc                    ; Очищення прапора переносу
       mov     cx,0
       mov     ax,SLOG1   ; Додавання SLOG1 і SLOG2
       add     ax,SLOG2
       mov     bx,OFFSET SUM   ; Одержання зсуву і
       mov     [bx],ax        ; зберігання результату додавання
       jnc     Lmem           ; Був перенос?  Немає - перехід на мітку
       inc     cx             ; І - створити старший байт результату
Lmem:   mov     [bx+2],cx  ; Зберегти старший байт результату
       jmp     Start
SUMMA   ENDP
CSEG    ENDS
       END     SUMMA
```

Програма 1.2

```
TITLE   ВІДНІМАННЯ ЦІЛИХ БЕЗЗНАКОВИХ ЧИСЕЛ
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      256 DUP(0)
SSEG   ENDS
; Визначення сегмента даних для даних і результату
;
DSEG    SEGMENT PARA PUBLIC     'DATA'
UMEN    DW      1 DUP (0) ; зменшуване, довжина слово
VICH     DW      1 DUP (0) ; від'ємник, довжина слово
RAZ      DW      1 DUP(0)  ; результат, довжина слово
DSEG    ENDS
; Визначення сегмента коду програми
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
SUBST   PROC    FAR
       mov     ax,DSEG   ; Визначення адреси сегмента DATA
       mov     ds,ax          ; Пересилання в сегментний регістр DS.
Start:  clc                    ; Очищення прапора переносу.
       mov     cx,0
       mov     ax,UMEN   ; Віднімання.
       sub     ax,VICH
       mov     bx,OFFSET RAZ   ; Одержання зсуву і
       mov     [bx],ax                 ; зберігання результату
       jmp      Start
SUBST    ENDP
CSEG    ENDS
       END     SUBST
```

При N>2 потрібна розробка програм обробки многобайтових чисел (див. програми 1.3 і 1.4). Нехай X=(XN, ..., X1) і Y=(YN, ..., Y1) - N байтове число. В основі програми додавання (віднімання) X±Y=Z лежить обчислювальна схема (рис. 1.1.).

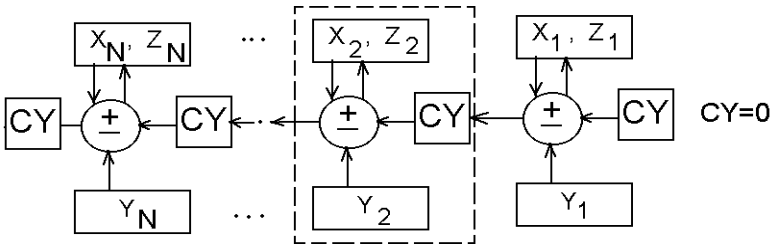

Рис. 1.1

На рис.1.1. X_N, ..., X_1 – байти першого доданка (зменшуваного); Y_N, ..., Y_1 – байти другого доданка (від'ємника); Z_N, ..., Z_2, Z_1 – байти суми (різниці), вони можуть розміщатися на місці першого доданка або пересилатися в інше місце;

C_Y – біт переносу, який можна розглядати перед операцією додавання і-х байтів як молодший розряд додаткового доданка, а після виконання цієї операції – додатковий, старший розряд суми (різниці) і-х байтів. В колах позначений тип операції, стрілки показують напрямок передачі інформації (інформаційні потоки) між елементами схеми. Штриховими лініями виділений повторюваний процес, що використовується для циклічної побудови програм. Можливість створення програм на основі приведеної схеми визначається наявністю в наборі команд додавання (віднімання) із переносом (позикою).

При додаванні двійкових беззнакових чисел може виникнути помилка переповнювання. Її ознакою є значення біта переносу, рівне CY=1 після обчислення старшого байта доданків. При відніманні беззнакових чисел різниця може стати негативним числом. Ознакою цього порушення також є значення біта переносу CY=1 після віднімання старшого байта, що віднімається зі старшого байта зменшуваного (CY=1 унаслідок позики з неіснуючого (N+1) байта зменшуваного)[1]. Прийоми фіксації цих помилок можуть бути різноманітні по розсуду програміста.

Програма 1.3

```
TITLE       ДОДАВАННЯ ЦІЛИХ БЕЗЗНАКОВИХ ЧИСЕЛ
TITLE       ПІДВИЩЕНОЇ ТОЧНОСТІ
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
      DB      256 DUP(0)
SSEG   ENDS
; Визначення сегмента даних для доданків і результату
DSEG    SEGMENT PARA PUBLIC     'DATA'
SLOG1   DD      1 DUP (0) ; Доданок, довжина подвійне слово
SLOG2   DD      1 DUP (0) ; Доданок, довжина подвійне слово
SUM     DQ      1 DUP(0)  ; Результат, довжина 5 байтів
DSEG    ENDS
; Визначення сегмента коду програми
CSEG    SEGMENT PARA PUBLIC     'CODE'
      ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
SUMMA   PROC    FAR
      mov     ax,DSEG   ; Визначення адреси сегмента DATA
      mov     ds,ax          ; Пересилка в сегментний регістр DS
Start:  lea     si,SLOG1   ; Одержання зсуву SLOG1 у SI
      lea     di,SLOG2      ; Одержання зсуву SLOG2 у DI
      lea     bx,SUM         ; Одержання зсуву SUM у BX
      clc                      ; Очистка прапора переносу.
      mov     cx,0
      mov     ax,[si]          ; Додавання SLOG1 і SLOG2 з урахуванням
      adc     ax,[di]          ; біта переносу.
```

[1] Процесор 1810 у цьому випадку подає результат у додатковому коді.

```
        mov     [bx],ax          ; Зберігання результату
        inc     si               ; Обчислення зсуву наступного
        inc     si               ; слова доданків без зміни
        inc     di               ; біта переносу попереднього додавання
        inc     di
        mov     ax,[si]
        adc     ax,[di]
        mov     [bx+2],ax
        jnc     Lmem             ; Був перенос?  Немає - перехід не мітку
        inc     cx               ; I - створити старший байт.
Lmem:   mov     [bx+4],cx  ; Зберегти старший байт у пам'яті
        jmp     Start
SUMMA   ENDP
CSEG    ENDS
        END     SUMMA
```

Програма 1.4

```
TITLE ВІДНІМАННЯ ЦІЛИХ БЕЗЗНАКОВИХ ЧИСЕЛ
TITLE ПІДВИЩЕНОЇ ТОЧНОСТІ
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      256 DUP(0)
SSEG   ENDS
; Визначення сегмента даних для даних і результату
DSEG    SEGMENT PARA PUBLIC     'DATA'
UMEN    DD      1 DUP (0) ; зменшуване, довжина подвійне слово
VICH     DD      1 DUP (0) ; від'ємник, довжина подвійне слово
RAZ      DD      1 DUP(0)  ; різниця, довжина подвійне слово
DSEG    ENDS
; Визначення сегмента коду програми
CSEG    SEGMENT PARA PUBLIC     'CODE'
        ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
SUBST   PROC    FAR
        mov     ax,DSEG  ; Визначення адреси сегмента DATA
        mov     ds,ax          ; Пересилання в сегментний регістр DS
Start:  lea     si,UMEN ; Одержання зсуву UMEN у SI
        lea     di,VICH       ; Одержання зсуву VICH у DI
        lea     bx,RAZ        ; Одержання зсуву RAZ у BX
        clc                     ; Очистка прапора переносу
        mov     cx,0
        mov     ax,[si]     ; Віднімання UMEN і VICH з урахуванням
        sbb     ax,[di]      ; біта переносу
        mov     [bx],ax     ; Зберігання результату
        inc     si              ; Обчислення зсуву наступного
        inc     si              ; слова даних без зміни
```

```
        inc     di              ; біта переносу попереднього віднімання.
        inc     di
        mov     ax,[si]
        sbb     ax,[di]
        mov     [bx+2],ax
        jmp     Start
SUBST   ENDP
CSEG    ENDS
        END     SUBST
```

Домашнє завдання

1. Вивчіть систему команд мікропроцесора 1810ВМ86 (дод. 8).
2. Вивчіть стислий опис програми TURBO ASSEMBLER (дод. 1).
3. Вивчіть стислий опис налагоджувача TURBO DEBUGGER (дод. 2).
4. Вивчіть стислий опис програми NORTON COMMANDER (дод. 6).
5. Намалюйте в протоколі схему алгоритму програм 1.1, 1.2, 1.3, 1.4.

Порядок виконання роботи

1. Наберіть текст програми 1.1 за допомогою вмонтованого редактора NORTON COMMANDER (Виклик Shift-F4 або F4) і збережіть під ім'ям st???.asm (??? – останні три цифри номера залікової книжки).

2. Проведіть асемблювання програми, набрав у командному рядку: tasm /ZI /L st???.asm, (не забувай, що замість ??? треба набирати цифри).

3. Якщо помилок немає (з'явився файл із розширенням *.obj), то треба переходити до п.4 виконання роботи.

При наявності помилок перегляньте файл із розширенням *.lst (F3), у якому вони відзначені, виправіть їх у файлі з розширенням *.asm (F4) і повторіть п.п. 2 і 3 виконання роботи.

4. Створіть завантажувальний файл, наберіть у командному рядку: tlink /v st???.obj, (тут теж замість ??? треба набирати потрібні цифри).

5. Запустіть налагоджувач, набираючи у командному рядку: td st???.exe, (знову замість ??? треба набирати ті ж цифри).

Примітка. Якщо ви не набрали імені програми (тобто запустили налагоджувач командним рядком td), то уважно вивчіть розділ "Завантаження нової програми для налагодження" із стислого опису налагоджувача TD.

6. У налагоджувачі, після появи на екрані вихідного тексту програми у вікні Module, викликайте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Registers (натисніть R). Перемістить вікно Registers (Регістри) у правий верхній кут екрана (натисніть послідовно клавіші Ctrl-F5, End, PgUp, Enter).

7. Трасування програми виконуйте клавішею F7. Після кожного натискання клавіші F7 заносить вміст регістрів і регістра прапорів у табл. 1.1 і 1.2.

Таблиця 1.1

Коман-да	Вміст регістрів													
	ax	bx	cx	dx	cs	ds	di	si	Прапори					
									c	z	s	p	a	d

Таблиця 1.2

Вікно Dump	
Адреса	Значення

Примітки: 1. У вікні Module покажчик у вигляді зафарбованого трикутника встановлюється перед командою Асемблера, що повинна виконуватися наступною.

2. Після занесення адреси сегмента даних у регістр DS відчиніть вікно Dump, викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Dump (натисніть D). Перемістіть вікно Dump (Дані) у правий нижній кут екрана (натисніть послідовно клавіші Ctrl-F5, End, PgDn, Enter).

3. У вікні Dump за відповідними адресами занесіть доданки (зменшуване і від'ємник).

8. Проведіть обчислення для значень наданих викладачем.

Примітка. У програмі організований нескінчений цикл, тому її не треба перевантажити. Після команди jmp Start, програма починає виконуватися спочатку.

9. Вийдіть з програми TD.EXE (натисніть сполучення клавіш Alt-X).

10. Виконайте п.п. 1 – 8 для програм 1.2, 1.3, 1.4.

Контрольні питання

1. Які групи команд МП 1810ВМ86 ви знаєте?

2. Яка розрядність шини адреси і шини даних МП 1810ВМ86?

3. Призначення регістрів AX, BX, CX, DX, SP, BP, DI, SI, IP, F, CS, DS, SS, ES?

ЛАБОРАТОРНА РОБОТА 2

ДОСЛІДЖЕННЯ НА МІКРОПРОЦЕСОРНИХ ПРИСТРОЯХ АЛГОРИТМІВ АРИФМЕТИЧНИХ ОПЕРАЦІЙ З ОДНОБАЙТОВИМИ І МНОГОБАЙТОВИМИ ЦІЛИМИ ЧИСЛАМИ (МНОЖЕННЯ ТА ДІЛЕННЯ)

Мета роботи: ознайомлення з принципами програмування на мові АСЕМБЛЕР МП 1810 із використанням налагоджених засобів, з дослідженням алгоритмів арифметичних операцій множення та ділення одно- та многобайтових цілих чисел.

Стислі теоретичні відомості й приклади програм

Двійкове множення

Двійкове і десяткове множення також, як і двійкове і десяткове додавання або віднімання, виконують за схожими алгоритмами обчислення.

Множення – це швидкий засіб додавання декількох однобайтових чисел. Наприклад, множення 7 на 5 зводиться до додавання п'ятьох однакових однобайтових чисел, кожне з котрих дорівнює семи (програма 2.1).

Програма 2.1

```
TITLE МНОЖЕННЯ ОДНОБАЙТОВИХ ЦІЛИХ
TITLE БЕЗ ЗНАКОВИХ ЧИСЕЛ
TITLE МЕТОДОМ ДОДАВАННЯ
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      128 DUP(0)
SSEG   ENDS
;
; Визначення сегмента даних
;
DSEG    SEGMENT PARA PUBLIC     'DATA'
MNOG1   DB      1 DUP (0) ; Множене, довжина байтів
MNOG2   DB      1 DUP (0) ; Множник, довжина байтів
REZULT  DW    1 DUP (0) ; Результат, довжина слово
DSEG    ENDS
;
; Визначення сегмента коду програми
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
;
MULT    PROC    FAR
```

```
      mov     ax,DSEG    ; Визначення адреси сегмента DATA
      mov     ds,ax       ; Пересилка в сегментний регістр DS
Start: mov    ax,0        ; Очистка регістра AX, CX і
      mov     cx,0        ; прапора переносу.
      clc
      mov     bx,OFFSET MNOG1 ; Одержання зсуву  множеного
      mov     cl,MNOG2          ; Скільки разів складати?
@@2: add     al,[bx]
      jnc     @@1
      inc     ah
@@1: loop    @@2          ; Цикл
      mov     REZULT,ax    ; Зберігання результату додавання.
      jmp     Start
MULT    ENDP
CSEG    ENDS
      END     MULT
```

Недолік такого засобу множення – час обчислення добутку залежить від значення множника, чим більше множник, тим більше час обчислення.

Однією з ідей зменшення часу обчислення добутку є виняток множень на нуль. При множенні одного числа на інше одне з чисел називають множене, інше – множником. Множення виконують поразрядно. Часто виникає необхідність переносу до наступного за старшинством розряду. Перемножуючи десяткові числа, звичайно "вирішуємо в розумі" виникаючі при цьому проблеми переносу. По завершенні множення множеного на значення молодшого розряду множника утворюється перший частковий добуток. В результаті множення множеного на значення наступного за старшинством розряду множника формується другий частковий добуток. Подібна процедура повторюється з метою одержання всіх необхідних часткових добутків. Оскільки черговий частковий добуток – результат перемножування множеного і розряду множника, значимість якого в 10 разів більше значимості розряду, використаного в попередній операції множення, то всі цифри отриманого добутку зсовують вліво на одну позицію (розряд). Для одержання результату, усунуті щодо один одного часткові добутки складаються. Виникаючі при додаванні переноси повинні бути обчислені при формуванні остаточного результату.

Розглянемо як приклад множення числа 17 на число 12:

```
  17    множене
* 12    множник
  34    перший частковий добуток
 17     другий частковий добуток
 100    перенос
204  результуючий добуток
```

Оскільки множник складається з двох розрядів, одержуєм два часткових добутки, додавання яких викликає перенос у розряд сотень.

Тепер перейдемо до розгляду двійкового множення, схематично поданого за допомогою наступних правил:

$0 \times 0 = 0;$
$0 \times 1 = 0;$
$1 \times 0 = 0;$
$1 \times 1 = 1.$

Скористаємося цими правилами для обчислення добутку 8-розрядних двійкових еквівалентів десяткових чисел 17 і 12:

```
  00010001  множене 17
  00001100  множник 12
  00000000 перший частковий добуток
 00000000  другий частковий добуток
00010001   третій частковий добуток
00010001   четвертий. частковий добуток
00000000   п'ятий  частковий добуток
00000000   шостий частковий добуток
00000000   сьомий частковий добуток
00000000   восьмий частковий добуток
0000000000000000 перенос
0000000011001100 результат 204
```

Отже в результаті множення отримано вісім часткових добутків, оскільки множник складається з восьми розрядів. Перші два часткових добутки включають тільки нулі, такі множники – значення першого і другого розрядів двійкового еквівалента числа 12 рівні 0. Третій частковий добуток – копія множеного. Різниця між ними полягає в тому, що копія зсунута щодо множеного на два розряди вліво, оскільки для одержання цього часткового добутку в якості множника використовується значення третього розряду. Четвертий частковий добуток також є копією множеного, зсунутою щодо останнього на три двійкових розряди вліво. Частковий добуток із п'ятого по восьме складається тільки з нулів, тому що відповідні множники, що беруть участь у формуванні добутку, – двійкові нулі. Додавання всіх часткових добутків у даному прикладі не супроводжується переносом.

Створений простий засіб виконання двійкового множення, що одержав назву множення шляхом зсуву і додавання.

Перерахуємо основні правила цього засобу.

1. Формування першого часткового добутку. Якщо значення молодшого значущого розряду множника дорівнює 1, то і результат є копією множеного.

2. Правило зсуву. При використанні чергового розряду множника для формування часткового добутку провадиться зсув множеного на один розряд уліво.

3. Правило додавання. Щораз, коли значення розряду множника дорівнює 1, до результату необхідно додати множене, розташоване в визначеній правилом зсуву позиції.

4. Визначення результуючого добутку. Шуканий добуток є результат виконання всіх операцій зсуву і додавання.

Даний спосіб дозволяє робити операції множення не тільки над однобайтовими числами, але і над многобайтовими. Програма 2.2 показує застосування способу для множення однобайтових чисел із фіксованою комою.

Програма 2.2

```
TITLE МНОЖЕННЯ ОДНОБАЙТОВИХ ЦІЛИХ
TITLE БЕЗ ЗНАКОВИХ ЧИСЕЛ
TITLE ЗА ПРАВИЛОМ ЗСУВУ І ДОДАВАННЯ
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      128 DUP(0)
SSEG   ENDS
;
; Визначення сегмента даних
DSEG    SEGMENT PARA PUBLIC     'DATA'
MNOG1   DB      1 DUP (0) ; Множене, довжина байт
MNOG2   DB      1 DUP (0) ; Множник, довжина байт
REZULT  DW      1 DUP (0) ; Результат, довжина слово
DSEG    ENDS
;
; Визначення сегмента коду програми
;
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
;
MULT    PROC    FAR
       mov     ax,DSEG    ; Визначення адреси сегмента DATA
       mov     ds,ax           ; Пересилка в сегментний регістр DS
Start: mov      ax,0           ; Очищення регістра AX, CX і
       mov     cx,0             ; прапора переносу.
       clc
       mov     cl,8      ; Занесення довжини чисел, що перемножуються
       mov     dl,MNOG1   ; Занесення множеного в DL
       mov     ah,MNOG2   ; Занесення множника в AH
Lmem:   add     ax,ax      ; Зсув уліво на один розряд
       jnc     @@1        ; Дорівнює 1.  Ні?  Перехід на @@1
       add     ax,dx      ; Так?  Отримати частковий добуток
@@1:    loop    Lmem       ; Цикл
       mov     REZULT,ax  ; Зберігання результату додавання
       jmp      Start
MULT     ENDP
CSEG    ENDS
       END     MULT
```

Система команд МП 1810 містить вмонтовані команди множення, що дозволяють значно скоротити час перемноження двох чисел і зменшити код програми (програма 2.3).

Програма 2.3

```
TITLE МНОЖЕННЯ ОДНОБАЙТОВИХ ЦІЛИХ
TITLE БЕЗ ЗНАКОВИХ ЧИСЕЛ З ВИКОРИСТАННЯМ
TITLE КОМАНД МНОЖЕННЯ МП 1810
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      128 DUP(0)
SSEG   ENDS
;
; Визначення сегмента даних
;
DSEG    SEGMENT PARA PUBLIC     'DATA'
MNOG1   DB      1 DUP (0) ; Множене, довжина байтів
MNOG2   DB      1 DUP (0) ; Множник, довжина байтів
REZULT  DW       1 DUP (0) ; Результат, довжина слово
DSEG    ENDS
;
; Визначення сегмента коду програми
;
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
;
MULT    PROC    FAR
       mov  ax,DSEG        ; Визначення адреси сегмента DATA
       mov  ds,ax           ; Пересилка в сегментний регістр DS
Start:  mov     ax,0         ; Очищення регістра AX
       mov   al,MNOG1       ; Занесення множеного в AL
       mul    BYTE PTR MNOG2  ; Множення двох чисел
                                  ; (AL×MNOG2=AX)
       mov    REZULT,ax            ; Зберігання результату.
       jmp     Start
MULT    ENDP
CSEG    ENDS
       END     MULT
```

Команда MUL може множити тільки 8- або 16-бітові значення, але нею можна скористатися і для множення чисел підвищеної точності без знака. Наприклад, за її допомогою можна перемножити два 32-бітових числа. Для цього треба обчислити серію 32-бітових перехресних добутків, а потім скомбінувати з них 64-бітовий результат з використанням такої схеми (програма 2.4):

```
        AB
      ¯ CD
        BD     (B×D) Перший частковий результат
    +   AD     (A×D)
    +   BC     (B×C) Зсув на одну позицію вліво
    +AC        (A×C)
      MNOD     (кінцевий результат)
```

Програма 2.4

```
TITLE МНОЖЕННЯ ЦІЛИХ БЕЗЗНАКОВИХ ЧИСЕЛ
TITLE ПІДВИЩЕНОЇ ТОЧНОСТІ
TITLE З ВИКОРИСТАННЯМ КОМАНД МНОЖЕННЯ 1810
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      256 DUP(0)
SSEG   ENDS
;
; Визначення сегмента даних
DSEG    SEGMENT PARA PUBLIC     'DATA'
MNOG1   DD      1 DUP (0) ; Множене, довжина 4 байта
MNOG2   DD      1 DUP (0) ; Множник, довжина 4 байта
REZULT  DQ      1 DUP (0) ; Результат, довжина 8 байтів
DSEG    ENDS
;
; Визначення сегмента коду програми
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
;
MULT    PROC    FAR
       mov     ax,DSEG         ; Визначення адреси сегмента DATA
       mov     ds,ax           ; Пересилка в сегментний регістр DS
Start:  lea     si,MNOG1        ; SI вказує перше число
       lea     bx,MNOG2        ; BX вказує друге число
       lea     di,REZULT       ; DI вказує результат
       mov     ax,[si]
       mul     WORD PTR[bx]    ; Обчислити B×D
       mov     [di],ax         ; Молодше слово 1-го часткового результату
       mov     [di+2],dx       ; Старше слово 1-го часткового результату
       mov     ax,[si+2]
       mul     WORD PTR[bx]    ; Обчислити A×D
       add     [di+2],ax       ; Зсув 2-го часткового результату
       adc     [di+4],dx       ; Старше слово 2-го часткового результату
       mov     ax,[si]
       mul     WORD PTR[bx+2]  ; Обчислити B×C
```

```
        add     [di+2],ax       ; Частковий результат множення на
                                ;другу цифру множника.
        adc     [di+4],dx       ; Старше слово 3-го часткового результату
        mov     ax,[si+2]
        mul     WORD PTR[bx+2]  ; Обчислити A×C
        add     [di+4],ax       ; Зсув 4-го часткового результату
        adc     [di+5],dx       ; Старше слово 4-го часткового результату
        jmp     Start
MULT    ENDP
CSEG    ENDS
        END     MULT
```

Двійкове ділення

Ділення – це операція, обернена множенню. При діленні операцію віднімання повторюють доти, поки зменшуване не стане менш за від'ємник. Число цих повторювань показує, скільки разів від'ємник укладається в зменшуваному. Фрагмент програми, що використовує цей метод, приведений нижче:

```
        SUB   CX,CX
@@1:    CMP   AX,BX
        JB    @@2
        SUB   AX,BX
        INC   CX
        JMP   @@1
@@2:    RET
```

Процедура ділення декілька складніше процедури множення.

Роздивимося, наприклад, ділення числа 204 на число 12, користуючись десятковою та двійковою арифметикою:

```
204 | 12          1100 1100 | 1100
12  |  17         1100      | 10001
84                  0 1
84                    0
 0                  0 11
                      0
                   0 110
                     0
                   0 1100
                   1100
                      0
```

Двійкове ділення починається з аналізу діленого 11001100 і дільника 1100. Відразу ж виявляємо, що дільник 1100 точно вкладає в 1100, і тому записуємо цифру 1 у подане для формування частки поле. Множимо дільник на 1 і віднімаємо результат із 1100. Різниця дорівнює нулю, тобто менше дільника, а тому процес ділення можна продовжувати. Об'єднуємо нуль залишку із значен-

ням наступного розряду діленого, рівним 1. Оскільки число 1100 укладається 0 разів у числі 1, записуємо 0 у частці, а число 1 об'єднуємо з наступною цифрою діленого і т.д., процедура, що описується, продовжується доти, поки ділене не виявляється вичерпаним.

Таким чином, реалізувати операцію ділення на мікропроцесорі (МП) не настільки просто, як операцію множення. Занадто багато зусиль потрібно для з'ясовування того, скільки разів дільник укладається у визначеному числі.

Процедура двійкового ділення в дійсності проста, тому що кожний біт частки приймає одне з двох можливих значень: 1 або 0. Як у випадку двійкового множення, зручно виявляється використовувати операції зсуву.

Продемонструємо засіб двійкового ділення на розглянутому вище прикладі, попередньо представивши дільник 1100 у додатковому коді. Це дозволить обмежитися двійковим додаванням в усіх випадках, коли потрібно виконати віднімання.

Як і у випадку вище розглянутого так названого довгого двійкового ділення, необхідно визначити скільки разів дільник укладається в числі, утвореному відповідною кількістю старших значущих бітів діленого. Мікропроцесор не може будувати здогадок стосовно того, скільки разів дільник укладається в указаному числі. У дійсності МП починає віднімати дільник із цього числа. Якщо дільник не буде укладатися в згаданій частині діленого, завжди можна повернути біти, які віднімали, назад діленому. Про те, що дільник не укладається, свідчить поява від'ємного результату віднімання (біт знака дорівнює 1).

Почнемо спробу виконати віднімання перший раз:

01100 1100 ділене
10100 0000 від'ємник число 12
00000 1100 перший результат
↑---------- наявність тут нуля означає, що 1-й біт шуканої частки дорівнює 1 частка 1XXXX.

Якщо дільник укладається у відповідну частину діленого, біт знака дорівнює 0. Це значить, що результат ділення є позитивне число. У даному прикладі це так. А тому перший біт шуканого результату (частки) дорівнює 1.

Другий крок процедури ділення. Слід ще раз спробувати виконати віднімання дільника. Але попередньо необхідно зсунути перший результат. Зсув повинний бути таким, щоб при останній операції віднімання був сформований другий біт частки. У результаті зсуву отримуємо:

000001100 перший результат до зсуву
00001100 перший результат після зсуву

Тепер можна виконати другу операцію віднімання:

00001100 зсунутий перший результат
10100000 від'ємник 12
10101100 другий результат
↑---------- наявність тут 1 означає, що 2-й біт шуканої частки дорівнює 0, частка: 10XXX.

Результат цього віднімання містить 1 у позиції знака, тобто отримано від'ємне число. Отже, дільник не вкладається у відповідному числі. Тому, на-

самперед у другу (по старшинству) позицію поля представлення частки варто записати 0. Крім того, оскільки віднімання не відбулося, необхідно повернути всі біти дільника назад першому результату:

10101100 другий результат
01100000 що повертається число 12
00001100 зсунутий перший результат

Тепер настає черга наступного зсуву:

0000100 зсунутий перший результат
0001100 перший результат після другого зсуву

Виконує третю операцію віднімання:

0001100 двічі зсунутий перший результат
1010000 від'ємник число 12
1011100 третій результат

↑---------- наявність тут 1 означає, що 3-й біт шуканого дорівнює 0, частка: 100XX.

Третій результат – негативне число, тобто підданий подвійному зсуву перший результат виявився менше дільника, отже, третій біт шуканої частки дорівнює 0. Тому число 12 (дільник) слід повернути зсунутому двічі першому результату:

1011100 третій результат
0110000 дільник 12
0001100 двічі зсунутий перший результат

Виконуємо черговий зсув:

0001100 двічі зсунутий перший результат
001100 перший результат після третього зсуву

Проведемо віднімання

001100
101000
110100 четвертий результат

↑------- наявність тут одиниці означає, що 4-й біт частки дорівнює 0, частка: 1000X.

Повернемо дільник четвертому результату:

110100
011000
001100

Виконуємо четвертий зсув

001100 до зсуву
01100 після зсуву

Проведемо віднімання:

01100 результат після четвертого зсуву
10100 число 12, що віднімається
00000 п'ятий результат

↑------- наявність тут нуля означає, що 5-й битий частки дорівнює 1, частка: 10001.

На цьому виконання процедури закінчується. Таким чином, при діленні числа 11001100 на число 1100 результат дорівнює 10001.

Розглянута процедура двійкового ділення трохи складніша процедури множення. Проте її неважко здійснити на практиці, якщо послідовно виконувати пропоновані операції. Оскільки правила ділення чітко сформульовані, вони можуть бути реалізовані в МП.

У МП 1810 для реалізації процедури ділення використовуються вмонтовані команди. Програма 2.5 показує використання цих команд для ділення 32-бітного числа на 16-бітне число. Числа розташовуються в DX: AX ділене в регістрі або в пам'яті дільника. Результат ділення: у AX частка, у DX залишок.

Програма 2.5

```
TITLE ДІЛЕННЯ ЦІЛИХ БЕЗЗНАКОВИХ ЧИСЕЛ
TITLE З ВИКОРИСТАННЯМ КОМАНД ДІЛЕННЯ МП 1810
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      128 DUP(0)
SSEG   ENDS
;
; Визначення сегмента даних
DSEG    SEGMENT PARA PUBLIC     'DATA'
DELIM   DD      1 DUP (0) ; Ділене, довжина подвійне слово
DELIT   DW      1 DUP (0) ; Дільник, довжина слово
REZLT   DW      1 DUP (0) ; Результат, довжина слово
OSTAT   DW      1 DUP (0) ; Залишок, довжина слово
DSEG    ENDS
;
; Визначення сегмента коду програми
;
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
;
DIVD    PROC    FAR
       mov   ax,DSEG      ; Визначення адреси сегмента DATA
     mov   ds,ax             ; Пересилка в сегментний регістр DS Start:
     lea       bx,DELIM  ; Занесення адреси діленого в регістр BX
       mov     ax,[bx]
       mov     dx,[bx+2]
       div     DELIT     ; Ділення двох чисел (DXAX:DELIT=AX, DX)
       mov     REZLT,ax        ; Зберігання результату
       mov     OSTAT,dx
       jmp     Start
DIVD    ENDP
CSEG    ENDS
       END     DIVD
```

Домашнє завдання

1. Вивчіть систему команд мікропроцесора 1810ВМ86 (дод. 8).
2. Вивчіть стислий опис програми TURBO ASSEMBLER (дод. 1).
3. Вивчіть стислий опис налагоджувача TURBO DEBUGGER (дод. 2).
4. Вивчіть стислий опис програми NORTON COMMANDER (дод. 6).
5. Намалюйте в протоколі схему алгоритму програм 2.1, 2.2, 2.3, 2.4, 2.5.
6. Напишіть програму ділення 16-бітного числа на 8-бітне число за описаним методом.

Порядок виконання роботи

1. Наберіть текст програми 2.4 за допомогою вмонтованого редактора NORTON COMMANDER (виклик Shift-F4 або F4) і збережіть під ім'ям st???.asm (??? – останні три цифри номера залікової книжки).
2. Проведіть асемблювання програми, набрав у командному рядку: tasm /zi /l st???.asm, (не забудьте, що замість ??? треба набирати цифри).
3. Якщо помилок немає (з'явився файл із розширенням *.obj), то переходьте до п. 3.4 виконання роботи.

При наявності помилок перегляньте файл із розширенням *.lst (F3), у якому вони відзначені, виправте їх у файлі з розширенням *.asm (F4) і повторіть п.п. 3.2 і 3.3 виконання роботи.

4. Створіть завантажувальний файл, набрав у командному рядку: tlink /v st???.obj, (тут теж замість ??? треба набирати потрібні цифри).
5. Запустіть налагоджувач, набрав у командному рядку td st???.exe, (знову замість ??? треба набирати ті ж цифри).

Примітка. Якщо ви не набрали імені програми (тобто запустили налагоджувач командним рядком td), то уважно вивчіть розділ "Завантаження нової програми для налагодження" із стислого опису налагоджувача TD.

6. У налагоджувачі, після появи на екрані вихідного тексту програми у вікні Module, викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Registers (натисніть R). Перемістіть вікно Registers (Регістри) у правий верхній кут екрану (натисніть послідовно клавіші Ctrl-F5, End, PgUp, Enter).
7. Трасування програми виконуйте клавішею F7. Після кожного натискання клавіші F7 заносіть вміст регістрів і регістра прапорів у табл. 2.1 і 2.2.

Таблиця 2.1

Команда	Вміст регістрів													
	ax	bx	cx	dx	cs	ds	di	si	Прапори					
									c	z	s	p	a	d

Таблиця 2.2

Вікно Dump	
Адреса	значення

Примітки: 1. У вікні Module покажчик у вигляді зафарбованого трикутника встановлюється перед командою Асемблера, що повинна виконуватися наступною.

2. Після занесення адреси сегмента даних у регістр DS відкрийте вікно Dump, викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Dump (натисніть D). Перемістіть вікно Dump (Дані) у правий нижній кут екрана (натисніть послідовно клавіші Ctrl-F5, End, PgDn, Enter).

3. У вікно Dump за відповідними адресами занесіть числа.

8. Проведіть обчислення для значень указаних викладачем.

Примітка. У програмі організований безконечний цикл, тому її не треба перевантажувати. Після команди jmp Start, програма починає виконуватися спочатку.

9. Вийдіть з програми TD.EXE (натисніть сполучення клавіш Alt-X).

10. Виконайте п.п. 1 – 8 для програми 2.5.

4. Контрольні питання

1. Які відмінності в команд без знакового і знакового множення і ділення даних МП 1810ВМ86?

2. Які засоби адресації даних використовуються в програмах?

3. Які функції виконують регістри AX, BX, CX, DX, SP, BP, DI, SI, IP, F, CS, DS, SS, ES у програмах 1 – 5?

ЛАБОРАТОРНА РОБОТА 3

ДОСЛІДЖЕННЯ НА МІКРОПРОЦЕСОРНИХ ПРИСТРОЯХ АЛГОРИТМУ ВИТЯГУ КВАДРАТНОГО КОРЕНЯ

Мета роботи: ознайомлення з принципами програмування на мові АСЕМБЛЕР МП 1810 із використанням налагоджених засобів, дослідження алгоритму витягу квадратного кореня з многобайтового цілого числа.

Стислі теоретичні відомості і приклади програм

Витяг квадратного кореня з використанням МП засновано на застосуванні класичного методу послідовних наближень, відомого за назвою методу Нью-

тона. Відповідно до цього методу наближене значення A квадратного кореня з числа N можна одержати за допомогою формули

$$A1 = \frac{\left(\frac{N}{A} + A\right)}{2}$$

Проілюструємо це на прикладі. Припустимо, що необхідно одержати квадратний корінь із числа, що має значення N. В якості першого наближення візьмемо значення (N/200)+2. Щоб одержати друге наближення, поділимо N на перше наближення, і знайдемо середнє цих значень. Для одержання третього наближення поділимо N на друге наближення і знову одержимо середнє значення і т.д. Наприклад, при обчисленні квадратного кореня з 10000 знадобиться виконати наступні дії:

N=10000; перше наближення дорівнює (10000/200)+2=52.

10000/52=192, (192+52)/2=122,
10000/122=81, (122+81)/2=101,
10000/101=99, (101+99)/2=100,
10000/100=100.

Таким чином, квадратний корінь із 10000 дорівнює 100. Визначаємо, що квадратним коренем є саме 100, а не інші проміжні наближення тому, що добуток 100 на 100 дорівнює 10000.

Обране число мало цілий квадратний корінь, але далеко не всі числа такі. Наприклад, квадратний корінь із 9999 не є цілим числом. Якщо запрограмувати метод Ньютона і використовувати співвідношення корінь×корінь=число в якості критерію завершення обчислень, то процесор буде повторювати команди обчислення наближень до нескінченності, оскільки квадрат цілого наближення ніколи не може стати 9999. Найкраще зупинити мікропроцесор (МП) після того, як він знайде найближчий, або "кращий", квадратний корінь із числа.

Можна користуватися декількома різноманітними методами для завершення процедури обчислення наближень. Який із них підійде більше всього, залежить від бажаної точності відповіді й обмежень на час обчислення.

Дозволити МП виконати цикл обчислення наближення 10 разів, припускаючи, що відповідь буде достатньо точною. Хоча в багатьох додатках цей засіб цілком придатний, але він досить довільний. Для одержання більш точного рішення можна дозволити МП повторювати цикл, поки два послідовних наближення не будуть ідентичні або відрізнятися усього на 1.

Програма 3.1 реалізує процедуру SQRT32, що використовує метод послідовних наближень для обчислення квадратного кореня з 32-бітового числа. Вона одержує вихідне число з регістрів DX (старше слово) і AX (молодше слово) і повертає 16-бітовий квадратний корінь у регістрі BX.

Спочатку програма зберігає значення регістрів BP, DX, AX у стеку, потім копіює покажчик стека SP у регістр BP. Тепер регістр BP вказує на значення регістра AX, поміщене в стек. Після цього обчислюється перше наближення по формулі (N/200)+2.

Команда з міткою NXT_APP відкриває цикл, що закінчується міткою DONE. При кожному проході цього циклу МП обчисляє нове наближення за

допомогою ділення 32-бітового вихідного числа (яке зчитується зі стека) на попереднє наближення, а потім усереднює результат. При усередненні значення регістра AX зсовує вправо на один біт, що відповідає діленню на два. Застосування команди SHR замість команди DIV зменшує час обчислення.

Програма порівнює кожне нове наближення з попереднім, щоб визначити момент, коли вони стануть однакові або будуть відрізнятися усього лише на 1 (+1 або –1). Виявивши це, МП передає керування мітці DONE; у противному випадку він повертає керування мітці NXT_APP для обчислення нового наближення. Після виходу на мітку DONE МП поміщає остаточне значення квадратного кореня в регістр BX, а потім витягає зі стека вихідне число (регістри AX і DX) і початкове значення регістра BP.

Програма 3.1

```
TITLE ОБЧИСЛЕННЯ КВАДРАТНОГО КОРЕНЯ
; Визначення сегмента стека
SSEG   SEGMENT PARA    STACK   'STACK'
       DB      128 DUP(0)
SSEG   ENDS
;
; Визначення сегмента даних
;
DSEG    SEGMENT PARA PUBLIC     'DATA'
NUMBER  DD      1 DUP (0) ; Число, довжина 32 біта
REZLT   DW      1 DUP (0) ; Результат, довжина 2 байта (слово)
DSEG    ENDS
;
; Визначення сегмента коду програми
;
CSEG    SEGMENT PARA PUBLIC     'CODE'
       ASSUME  CS:CSEG,DS:DSEG,SS:SSEG
;
SQRT32  PROC    FAR
       mov     ax,DSEG   ; Визначення адреси сегмента DATA
       mov     ds,ax     ; Пересилка в сегментний регістр DS
Start:  lea     bx,NUMBER ; Занесення адреси числа в регістр BX
       mov     ax,[bx]   ; Занести число в регістри МП
       mov     dx,[bx+2] ;
       push    bp        ; Зберегти регістри BP, DX, AX у стеку
       push    dx
       push    ax
       mov     bp,sp     ; Помістити BP на значення AX у стеку
       mov     bx,200    ; В якості початкового наближення
       div     bx        ; розділити вихідне число на 200,
       add     ax,2      ; потім додати 2.
NXT_APP:
```

```
        mov    bx,ax     ; Зберегти отримане наближення у BX.
        mov     ax,[bp]   ; Прочитати вихідне число заново
        mov     dx,[bp+2]
        div     bx        ; Розділити на останнє наближення й
        add     ax,bx     ;  середнє значення результату
        shr     ax,1
        cmp     ax,bx     ; Два останніх наближення ідентичні?
        je      DONE
        sub     bx,ax     ; Ні.  Порівняти їхню різницю з +1 і -1
        cmp       bx,1
        je        DONE
        cmp     bx,-1
        jne     NXT_APP
DONE:   mov     bx,ax     ; Помістити результат у BX
        mov     REZLT,bx  ; Переслати результат у пам'ять
        pop     ax        ; Відновити регістри.
        pop     dx
        pop     bp
        jmp     Start
SQRT32  ENDP
CSEG    ENDS
        END     SQRT32
```

Домашнє завдання

1. Вивчіть систему команд мікропроцесора 1810ВМ86 (дод. 8).
2. Вивчіть стислий опис програми TURBO ASSEMBLER (дод. 1).
3. Вивчіть стислий опис налагоджувача TURBO DEBUGGER (дод. 2).
4. Вивчіть стислий опис програми NORTON COMMANDER (дод. 6).
5. Намалюйте в протоколі схему алгоритму програми 3.1.

Порядок виконання роботи

1. Наберіть текст програми 3.1 за допомогою вмонтованого редактора NORTON COMMANDER (виклик Shift-F4 або F4) і збережіть під ім'ям st???.asm (??? – останні три цифри номера залікової книжки).

2. Проведіть асемблювання програми, набрав у командному рядку: tasm /zi /l st???.asm, (не забудьте, що замість ??? треба набирати цифри).

3. Якщо помилок немає (з'явився файл із розширенням *.obj), те переходіть до п. 4 виконання роботи.

При наявності помилок перегляньте файл із розширенням *.lst (F3), у якому вони відзначені, виправьте їх у файлі з розширенням *.asm (F4) і повторюйте п.п. 2 і 3 виконання роботи.

4. Створіть завантажувальний файл, набрав у командному рядку: tlink /v st???.obj, (тут теж замість ??? треба набирати потрібні цифри).

5. Запустіть налагоджувач, набрав у командному рядку td st???.exe, (знову замість ??? треба набирати ті ж цифри).

Примітка. Якщо ви не набрали імені програми (тобто запустили налагоджувач командним рядком td), то уважно вивчіть розділ "Завантаження нової програми для налагодження" із стислого опису налагоджувача TD.

6. У налагоджувачі, після появи на екрані вихідного тексту програми у вікні Module, викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Registers (натисніть R). Перемістіть вікно Registers (Регістри) у правий верхній кут екрану (натисніть послідовно клавіші Ctrl-F5, End, PgUp, Enter).

7. Трасування програми виконуйте клавішею F7.

Примітки: 1. У вікні Module покажчик у вигляді зафарбованого трикутника встановлюється перед командою Асемблера, що повинна виконуватися наступною.

2. Після занесення адреси сегмента даних у регістр DS відкрийте вікно Dump, викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Dump (натисніть D). Перемістіть вікно Dump (Дані) у правий нижній кут екрана (натисніть послідовно клавіші Ctrl-F5, End, PgDn, Enter).

3. У вікно Dump за відповідними адресами занесіть числа.

8. Проведіть обчислення для значень указаних викладачем.

Заповніть табл. 3.1.

Таблиця 3.1.

Число	Корінь квадратний	Кількість циклів

Примітка. У програмі організований безконечний цикл, тому її не треба перевантажувати. Після команди jmp Start, програма починає виконуватися спочатку.

9. Вийдіть з програми TD.EXE (натисніть сполучення клавіш Alt-X).

Контрольні питання

1. Поняття стекової пам'яті в МП 1810ВМ86.

2. Які способи адресації даних використовуються в програмі 3.1.

3. Які функції виконують регістри AX, BX, CX, DX, SP, BP, DI, SI, IP, F, CS, DS, SS, ES у програмі 3.1.

ЛАБОРАТОРНА РОБОТА 4

ДОСЛІДЖЕННЯ ТАБЛИЧНОГО СПОСОБУ ОБЧИСЛЕННЯ ТРИГОНОМЕТРИЧНИХ ФУНКЦІЙ

Мета роботи: ознайомлення з принципами програмування на мові АСЕМБЛЕР МП 1810 із використанням налагоджених засобів, дослідження алгоритму обчислення значень синуса кута, вираженого в градусах табличним методом.

Стислі теоретичні відомості та приклад програми

При проектуванні мікропроцесорних систем часто необхідно створювати вузли, що вирішують задачі обчислення деяких основних елементарних функцій (синус, косинус, логарифм і т.ін.). Існує багато різноманітних методів обчислення елементарних функцій, що використовують представлення їх у вигляді степеневих рядів, ланцюгових дробів, ітераційних процесів, розкладань за ортогональними багаточленами і ін. Найбільше простим і універсальним є представлення елементарних функцій (за винятком квадратного кореня) у виді степеневих рядів Тейлора.

Щоб чітко уявити, про що йде мова, треба пригадати деякі поняття, які вивчають в математиці. Самі прості елементарні функції належать до класу цілих раціональних функцій: постійні $y=a$, лінійні $y=ax+b$, квадратичні $y=ax^2+bx+c$, функції n-го ступеню, степеневі $y=x^a$.

Додавання до класу раціональних функцій операцій ділення поліномів, породжує клас дрібно-раціональних функцій: оберненої пропорційності $y=a/x$, дрібно-лінійну $y=(ax+b)/(cx+d)$, нелінійну дрібно-раціональну $y=a+(b/x)+(c/x)$, складну $y=F[(x)]$.

Додавання до класу цілих або дрібно-раціональних функцій операції пошуку оберненої функції, тобто $x=\Phi(y)$ для $y=F(x)$, породжує клас ірраціональних функцій: добування кореня, обернена до степеневої функції, степенева функція з раціональним показником.

Розглянуті три класи функцій у цілому утворюють більш широкий клас алгебраїчних функцій.

Функції, що не є алгебраїчними, належать до класу трансцендентних. До них відносяться: показова, логарифмічна, тригонометричні (синус, косинус і т.п.), гіперболічні й обернені до них функції.

Перераховані алгебраїчні і трансцендентні функції утворять безліч основних елементарних функцій. Всі інші функції називають не елементарними. Якщо функція не елементарна, то число операцій над аргументом, або самі операції, або ті й інші змінюються в залежності від значення аргументу. Прикладом не елементарної операції є факторіал: $y=1\times2\times3\times...\times n$. Тут кількість операцій залежить від значення аргументу.

Обчислення елементарних функцій із застосуванням МП здійснюються

на основі точних аналітичних записів цих функцій, що визначають кількість, тип і послідовність виконання арифметичних операцій.

Наприклад, функція y=sin(x) розкладається в ряд Тейлора:

$$\sin(x) = \frac{x}{1!} - \frac{x}{3!} + \frac{x}{5!} - \frac{x}{7!} + \frac{x}{9!} - \ldots \qquad (4.1)$$

Визначимо кількість членів ряду (4.1), необхідну для обчислення функції з точністю до чотирьох значущих десяткових цифр, тобто 0.5×10^{-4} , для діапазону аргументу $x\in[-\pi/4, \pi/4]$. Підставляючи в ряд (4.1) значення $x=\pi/4=0.7854$, знайдемо, що відкидання члена x/7! вносить погрішність менше 0.37×10^{-4} . Тому для обчислення функції y=sin(x) у заданому діапазоні і з необхідною точністю достатньо взяти (із запасом) перші чотири члени ряду (4.1). Для скорочення обчислень можна визначити коефіцієнти перших чотирьох членів:

$1/1!=1$, $-1/3!=-0.1667$, $1/5!=0.8333\times10^{-2}$, $-1/7!=-0.1984\times10^{-3}$.

Ще більш спростити розрахунки і збільшити швидкість обчислень можна застосуванням табличного засобу обчислення функцій. Існують таблиці значень для багатьох елементарних, зокрема, тригонометричних функцій. Можна записати ці значення в пам'ять мікропроцесорної системи і потім, у залежності від значення аргументу функції, що є індексом масиву значень функції, витягати ці значення в якості результату обчислення. Прикладом застосування такого методу служить програма 4.1 обчислення синуса на мові Асемблера МПК 1810 у діапазоні від 0° до 360°.

Якщо величина аргументу знаходиться не в цьому діапазоні, її легко можна привести до заданого діапазону, тому що

$$y=\sin(x)=\sin(x+2\,\pi\,n)=\sin(x+360\times n)\ ,$$

де n - ціле число.

У приведеній програмі, крім властивості періодичності функції y=sin(x), використовується друга властивість:

$$y=\sin(x)= -\sin(x+\pi)=-\sin(x+180).$$

Значення синуса можуть бути отримані з кроком один градус із точністю чотири десяткові цифри після коми.

Програма 4.1

```
;Програма обчислення синуса кута (цілі значення від 0° до 360°),
;що міститься в регістрі AX. Значення синуса кута
;у прямому коді повертається в регістрі BX
;
01    DOSSEG
02. MODEL TINY
03    STACK 200h
04. DATA
```

```
  ;                Значення синуса              Градуси
  ;
05 Bradees DW      0,175,349,523,698,872  ;  0 -  5
06         DW   1045,1219,1392,1564,1736  ;  6 - 10
07         DW   1908,2079,2250,2419,2588  ; 11 - 15
08         DW   2756,2924,3090,3256,3420  ; 16 - 20
09         DW   3584,3746,3907,4067,4226  ; 21 - 25
10         DW   4384,4540,4695,4848,5000  ; 26 - 30
11         DW   5150,5299,5446,5592,5736  ; 31 - 35
12         DW   5878,6018,6157,6293,6428  ; 36 - 40
13         DW   6561,6691,6820,6947,7071  ; 41 - 45
14         DW   7193,7313,7431,7547,7660  ; 46 - 50
15         DW   7771,7880,7986,8090,8191  ; 51 - 55
16         DW   8290,8387,8480,8572,8660  ; 56 - 60
17         DW   8746,8829,8910,8988,9063  ; 61 - 65
18         DW   9135,9205,9272,9336,9397  ; 66 - 70
19         DW   9455,9511,9563,9613,9659  ; 71 - 75
20         DW   9703,9744,9781,9816,9848  ; 76 - 80
21         DW   9877,9903,9926,9945,9962  ; 81 - 85
22         DW   9976,9986,9994,9998,10000 ; 86 - 90
  ;
23. CODE
24 StartPr: mov    ax,@Data
25     mov  ds,ax
26     mov  ax,0          ; Записати в AX 0
  ;*********************************************************
  ;  Перед виконанням наступної команди  занесіть
  ;  у AX значення кута (у градусах) у HEX коді
  ;*********************************************************
27     sub  cx,cx         ; Установити маску знака "+"
                          ; (старший розряд = 0)
28     cmp  ax,181        ; Кут < 181° ?
29     jb   Sin_pos       ; Так.  Лишити зі знаком "+"
30     mov  cx,8000h      ; Ні.  Змінити знак на "-"
                           ; (старший розряд = 1)
31     sub  ax,180        ; і відняти з кута 180°
32 Sin_pos: cmp    ax,91 ; Кут < 91° ?
33     jb   Get_sin     ;Так.  Витягти з таблиці значення синуса
34     neg  ax           ; Ні.  Відняти кут із 180°
35     add  ax,180
36 Get_sin: mov bx,ax   ; Зробити кут індексом слова (2 байта)
37     shl  bx,1
38     mov  bx,Bradees[bx] ; Витягти значення синуса
39     or   bx,cx              ; за допомогою маски (CX) установить знак
```

```
                    ; у значення синуса (BX)
40    jmp  StartPr
 ;
41    end  StartPr
```

У рядках 1-4, 23 задаються параметри режимів сегментації пам'яті. Рядки 5-22 являють собою масив значень синуса (мантис) для кутів від 0° до 90°. У рядках 24-25 підготовляється регістр сегмента даних DS.

Після занесення нуля в акумулятор, тобто виконання команди MOV AX,0 (рядок 26), передбачається зміна вмісту регістра AX із клавіатури. У п. 3.7 (порядок виконання роботи), примітка, описана ця процедура.

Після занесення значення кута в шестнадцатирічному виді в регістр AX, у програмі визначається знак результату. Кожне значення синуса займає в пам'яті два байти, тому значення кута, що є індексом, збільшується в два рази за допомогою команди: зсув вмісту регістра вліво на один розряд: SHL BX,1 (рядок 37). Це необхідно для визначення адреси комірки пам'яті, у якій зберігається значення синуса. Значення синуса витягається з пам'яті командою MOV BX,BRADEES[BX] (рядок 38), уточнюється його знак (команда OR BX,CX – рядок 39) і результат формується в регістрі BX у прямому коді.

Команда безумовного переходу JMP STARTPR (рядок 40) дозволяє почати повторне виконання описаних вище дій без перевантаження програми.

Домашнє завдання

1. Вивчіть систему команд мікропроцесора 1810ВМ86. (дод. 8).
2. Вивчіть стислий опис налагоджувача TURBO DEBUGGER. (дод. 2).
3. Намалюйте в протоколі схему алгоритму програми 4.1.
4. Напишіть програму обчислення косинуса кута в діапазоні від 0° до 360° табличним засобом на мові Асемблер МПК 1810, використовуючи для обчислень програму 4.1.

Порядок виконання роботи

1. Наберіть текст програми 3.1 за допомогою вмонтованого редактора NORTON COMMANDER (виклик Shift-F4 або F4) і збережіть під ім'ям st???.asm (??? – останні три цифри номера залікової книжки).
2. Проведіть асемблювання програми, набравши у командному рядку: tasm /zi /l st???.asm, (не забувайте, що замість ??? треба набирати цифри).
3. Якщо помилок немає (з'явився файл із розширенням *.obj), то переходіть до п. 4 виконання роботи.

При наявності помилок перегляньте файл із розширенням *.lst (F3), у якому вони відзначені, виправте їх у файлі з розширенням *.asm (F4) і повторюйте п.п. 2 і 3 виконання роботи.

4. Створіть завантажувальний файл, набравши у командному рядку: tlink /v st???.obj, (тут теж замість ??? треба набирати потрібні цифри).
5. Запустіть налагоджувач, набравши у командному рядку td st???.exe, (знову замість ??? треба набирати ті ж цифри).

Примітка. Якщо ви не набрали імені програми (тобто запустили налагоджувач командним рядком td), то уважно вивчіть розділ "Завантаження нової програми для налагодження" із стислого опису налагоджувача TD.

6. У налагоджувачі, після появи на екрані вихідного тексту програми у вікні Module, викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Registers (натисніть R). Перемістіть вікно Registers (Регістри) у правий верхній кут екрана (натисніть послідовно клавіші Ctrl-F5, End, PgUp, Enter).

7. Трасування програми виконуйте клавішею F7. Після кожного натискання клавіші F7 переглядайте вміст регістрів.

Примітки: 1. У вікні Module покажчик у вигляді зафарбованого трикутника встановлюється перед командою Асемблера, що повинна виконуватися наступної.

2. Не забувайте в ПОТРІБНИЙ момент занести значення кута в шістнадцятковому вигляді в регістр AX. Для цього перейдіть у вікно Registers (натисніть F6), стрілками перемістіть підсвічування до регістра AX і наберіть значення кута, що обчислюється (наприкінці числа повинна стояти буква H. Наприклад 010DH).

8. Обчисліть значення синуса для різних кутів за вказівкою викладача.

Примітка. У програмі організований безкінечний цикл, тому її не треба перевантажувати. Після команди jmp StartPr , програма починає виконуватися спочатку.

9. Заповніть табл. 4.1:

Таблиця 4.1

	КУТ		СИНУС КУТА	
	DEC	HEX	DEC	HEX
1				
2				
3				

Примітка. Десяткові значення кута задає викладач, шістнадцяткове значення мантис синуса кута обчисляє програма, шістнадцяткове значення кута і десяткові значення мантис синуса обчисляє студент.

10. Вийдіть з програми TD.EXE (натисніть сполучення клавіш Alt-X).

Контрольні питання

1. Які методи обчислення елементарних функцій ви знаєте?
2. У чому полягає табличний метод обчислення елементарних функцій?
3. Як визначені сегменти в програмі 4.1?
4. Перерахуйте засоби адресації пам'яті даних МП 1810.

ЛАБОРАТОРНА РОБОТА 5

ДОСЛІДЖЕННЯ АРИФМЕТИЧНИХ ОПЕРАЦІЙ ДЛЯ ПОДАНИХ У ФОРМАТІ З ПЛАВАЮЧОЮ КОМОЮ ЧИСЕЛ

Мета роботи: ознайомлення з алгоритмами арифметичних операцій для чисел, поданих у форматі з плаваючою комою на мові Асемблер МПК 1810 із використанням налагоджувальних засобів.

Стислі теоретичні відомості й приклади програм

Представлення чисел у форматі з плаваючою комою

Для одержання більш точних результатів, при обчисленнях на МП використовують представлення чисел у форматі з плаваючою комою (ФПК). У МПК 1810 арифметичні операції (АО) над числами у ФПК можуть бути виконані за допомогою БИС 1810ВМ87 (математичний сопроцесор) або програмним шляхом. Арифметичні операції із застосуванням математичного сопроцесора виконуються однією командою і тому можуть бути вивчені самостійно. Краще зрозуміти особливості роботи з числами, поданими у ФПК дозволить вивчення програмної реалізації АО над числами у ФПК (без застосування сопроцесора). На рис. 5.1 показаний формат представлення числа з плаваючою комою.

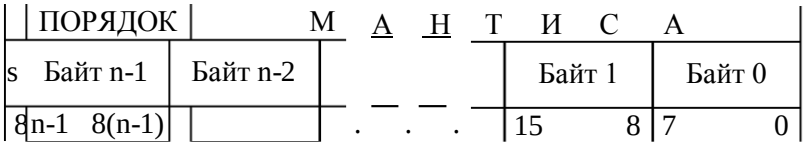

Рис. 5.1

Будь-яке дійсне число може бути записане у вигляді

$$Ч = зн \times М \times О^{п},$$

де Ч – число; зн – знак числа; М – мантиса ($0<M<1$); О – основа системи числення; п – порядок (показник степеня).

Приведемо приклади чисел у ФПК.

десяткова система:	двійкова система:
$1=+0.1\times10^{1}$	$1=+0.1\times2^{1}$
$0.005=+0.5\times10^{-2}$	$0.0001=+0.1\times2^{-3}$
$-20.45=-0.2045\times10^{2}$	$-10100.1=-0.101001\times2^{5}$

У приведених прикладах мантиса нормалізована.

Додавання і віднімання

Результат додавання двох чисел $X = x \cdot 2^{Px}$ і $Y = y \cdot 2^{Py}$, поданих у ФПК, повинний бути теж числом виду $Z = z \cdot 2^{Pz}$ (X + Y = Z). При цьому для доданків і результату повинно виконуватися рівність

$$x \cdot 2^{Px} + y \cdot 2^{Py} = z \cdot 2^{Pz}, \qquad (5.1)$$

де x, y, z – нормалізовані мантиси чисел X, Y і Z; P_x, P_y, P_z – відповідно порядки чисел X, Y і Z.

Тому що числа з різними порядками безпосередньо сумувати не можна, то для додавання двох чисел із нормалізованою мантисою необхідно попередньо привести їх до загального порядку, тобто перетворити один з доданків, наприклад Y, у такий спосіб:

$$Y = y \cdot 2^{Py} = y_{np} \cdot 2^{Px}.$$

Підставивши перетворений доданок $Y = y_{np} \cdot 2^{Px}$ у рівність (5.1), одержемо

$$x \cdot 2^{Px} + y_{np} \cdot 2^{Px} = z \cdot 2^{Pz}.$$

Після винесення загального співмножника за дужки

$$X + Y = x \cdot 2^{Px} + y_{np} \cdot 2^{Px} = (x + y_{np}) \cdot 2^{Px} = z \cdot 2^{Pz}.$$

Перетворена мантиса повинна бути правильним дробом. Тому перетворенню піддається завжди менший доданок, щоб не було переповнювання розрядної сітки мантиси, тобто, щоб виконувалася умова 0<M<1. При цьому частина молодших розрядів мантиси може губитися (при зсуві вправо). Це означає, що арифметичні дії над числами у ФПК, також як вивчені раніше над числами з фіксованою комою, є наближеними.

Звичайно розрахунки провадяться з використанням мов програмування високого рівня. Мова Асемблера дозволяє оперувати практично з будь-якими форматами представлення чисел. Нижче буде приведений приклад програми (програма 5.1) додавання (віднімання) поданих у форматі з плаваючою комою чисел довжиною в 4 байта. Більше число розрядів дозволяє отримати більшу точність. Якщо збільшувати довжину мантиси, то одержимо більше значущих цифр, а при збільшенні довжини порядку отримаємо більший діапазон значень.

Можна приблизно порахувати число точних десяткових розрядів, знаючи кількість виділених для мантиси бітів. $2^{10} = 1024 \approx 10^3$, це означає, що десять двійкових розрядів відповідають приблизно трьом десятковим. Роздивимося докладніше формат представлення числа з плаваючою комою, що займає в пам'яті 4 байта, де s – знак числа; p – порядок; m – мантиса (рис. 5.2).

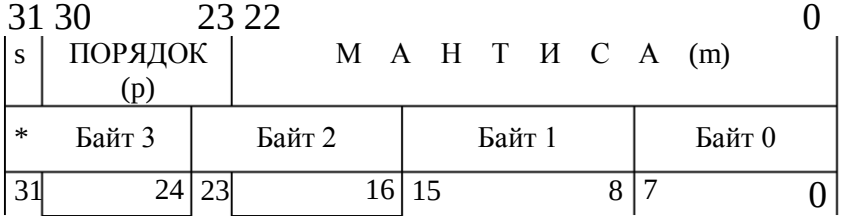

Рис. 5.2

У старшому розряді s (біт 31) знаходиться двійкова цифра, значення якої відповідає знаку числа. Якщо старший розряд s=1, то число від'ємне, якщо s=0 – число додатне.

Наступні 8 біт (із 30 по 23) відведені під порядок числа. Значення порядку p усунуте на 7FH (127). Істинне значення порядку n=p–7FH або n=p–127. Порядок p може змінюватися в межах від 01H до FEH (від –126 до +127). Тобто при значеннях порядку p більше, ніж 7FH, значення числа X>1 (за абсолютним значенням).

Інші 23 розряди (біти з 22 до 0) відведені під нормалізовану мантису, тому що мантиса нормалізована, то її старший розряд завжди дорівнює 1. Він не зберігається в пам'яті, чим досягається збільшення точності представлення числа в два рази (m=M/2). Тому вага біта 22 дорівнює не 0. 1B (0.5), а 0.01B (0.25).

Можна записати

$$M = 1.m,$$

де M – нормалізована мантиса числа (0<M<1); m – мантиса числа, що зберігається в пам'яті комп'ютера (0<m<0.5).

Мантиса подана в прямому коді, тому коди рівних за абсолютним значенням позитивних і негативних чисел збігаються у всіх розрядах, крім знакового (біт 31).

Таким чином, значення чотирибайтового числа X у ФПК можна представити так:

$$X = (-1)^s \cdot M \cdot 2^{p-127} = (-1)^s \cdot 1.m \cdot 2^{p-127}. \qquad (5.2)$$

Крім цього прийнято вважати:

– якщо p = 00000000B (00H), то значення числа X обчислюється як $X = (-1)^s \cdot 0.m \cdot 2^{p-126}$;

– якщо p = 11111111B (FFH), а m = 0, то число вважається нескінченно великим;

– якщо p = 11111111B (FFH), а m не дорівнює 0, то число X вважається неініціалізованим.

Слідство: якщо p = 00000000B (00H) і m = 0, то вважається, що число X = 0.

При переводі чисел із ФПК у десяткову систему і навпаки необхідно пам'ятати, що молодші байти числа в пам'яті розташовуються в молодших адресах, тобто при переводі байти потрібно читати в оберненому порядку.

П р и к л а д 1. Припустимо в пам'яті число X у ФПК розташовується такою уявою:

Adress X:	+00	+01	+02	+03
Data :	00	00	E0	C0

Це означає, що для переведення його в десяткову систему, необхідно спочатку записати байти за старшинством:

X = C0E00000H = 1100 0000 1110 0000 0000 0000 0000 0000B.

s\ p /\ m /

Проаналізуємо число X, записане в двійковому виді.

Знаковий біт 31 (крайній зліва) s = 1, значить число X<0 (від'ємне).

Порядок (біти 30-23) p = 10000001B = 81H = 129. Істинне значення порядку п = p – 127 = 129 – 127 = 2.

Мантиса (біти 22-0)
m = 110 0000 0000 0000 0000 0000B. Істинне значення мантиси

M = 1. m = 1. 110 0000 0000 0000 0000 0000B.

Остаточно одержуємо

$$X = (-1)^s \cdot M \cdot 2^{p-127} = -1 \cdot 1.11 \cdot 2^2 = -111B = -7.$$

П р и к л а д 2. Для пари чисел X = 22.75 і Y =–100.5, здійснити перетворення в шістнадцяткову систему числення і знайти їхню суму.

Число $X = 22.75D = (-1)^0 \cdot 10110.11B \cdot 2^0$.

Після нормалізації $X = (-1)^0 \cdot 1.011011B \cdot 2^4$. Істинний порядок п = 4, отже p = п + 7FH = 83H. Число X у ФПК буде виглядати так:

X = 0100 0001 1011 0110 0000 0000 0000 0000 B = 41B60000H.

s\ p /\ m /

У пам'яті: 00 00 B6 41.

Число $Y = -100.5D = (-1)^1 \cdot 1100100.1B \cdot 2^0$.

Після нормалізації $Y = (-1)^1 \cdot 1.1001001B \cdot 2^6$. істинний порядок п = 6, значить p = п + 7FH = 85H. Число Y у ФПК буде виглядати так:

Y = 1100 0010 1100 1001 0000 0000 0000 0000 B = C2C90000H.

s\ p /\ m /

У пам'яті: 00 00 C9 C2.

При підсумовуванні одержимо:

Z = X + Y = 22.75 + (–100.5) = –77.75

Число $Z = -77.75D = (-1)^1 \cdot 1001101.11B \cdot 2^0$.

Після нормалізації $Z = (-1)^1 \cdot 1.00110111B \cdot 2^6$. істинний порядок п = 6, значить p = п + 7FH = 85H. Число Z у ФПК буде виглядати так:

Z = 1100 0010 1001 1011 1000 0000 0000 0000 B = C29B8000H.

s\ p /\ m/

У результаті виконання програми в пам'яті повинно бути:

Adress Z: +00 +01 +02 +03

Data : 00 80 9B C2

Алгоритм додавання (віднімання) двох чисел, поданих у ФПК записаний нижче.

1. Порівняння порядків для визначення більшого.
2. Знаходження різниці порядків.
3. Перетворення мантис доданків у додатковий код.
4. Значення отриманої різниці використовується як кількість зсувів управо мантиси числа з меншим порядком.
5. Додавання мантис за правилами додавання чисел із фіксованої коми.
6. Перетворення мантиси результату в прямий код.

7. Нормалізація мантиси результату і при необхідності корекція значення порядку суми.

8. Одержання порядку суми й остаточного результату.

Примітка: 1. З більшого порядку віднімається менший, щоб різниця була позитивна.

2. Знак суми збігається зі знаком числа що має більший порядок.

Віднімання може бути замінено додаванням за формулою $Z = X + (-Y)$. Отже, віднімання реалізується зміною знака що віднімається і наступного додавання з усіма особливостями операції додавання.

Програма 5.1

```
TITLE ПРОГРАМА ДЛЯ ДОДАВАННЯ І ВІДНІМАННЯ
TITLE ЧИСЕЛ У ФОРМАТІ З ПЛАВАЮЧОЇ КОМИ
 1 ; Визначення сегмента стека
 2 SSEG SEGMENT PARA STACK 'STACK'
 3 DB 256 DUP(0)
 4 SSEG ENDS
 5 ; Визначення сегмента даних для доданків і результату
 6 DSEG SEGMENT PARA PUBLIC 'DATA'
 7 X DB 4 DUP (0) ; 1-е число, 4 байта
 8 Y DB 4 DUP (0) ; 2-е число, 4 байта
 9 Z DB 4 DUP (0) ; Результат, 4 байта
 10 P DB 1 DUP (0) ; Більший порядок
 11 SIGN DB 1 DUP (0) ; Знак результату
 12 SIGL DB 1 DUP (0) ; Знак меншого числа
 13 DSEG ENDS
 14 ; Визначення сегмента коду програми
 15 CSEG SEGMENT PARA PUBLIC 'CODE'
 16 ASSUME CS:CSEG,DS:DSEG,SS:SSEG
 17 ;
 18 FSUMMA PROC FAR
 19 Start:
 20 mov ax,DSEG ; Визначення адреси сегмента даних і
 21 mov ds,ax ; пересилка в сегментний регістр DS
 22 ;
 23 ; 1. Порівняння порядків для визначення більшого
 24 @S:
 25 mov cl,X+2 ; Занесення в регістр CX
 26 mov ch,X+3 ; двох старших байтів числа X у ФПК.
 27 mov dl,Y+2 ; Занесення в регістр DX
 28 mov dh,Y+3 ; двох старших байтів числа Y у ФПК.
 29 ; ДЛЯ ВІДНІМАННЯ ТРЕБА В РЯДКУ 30 ВИЛУЧИТИ
    ;КРАПКУ З КОМОЮ
 30 ; XOR DH,80H
```

```
31 ;
32 rol cx,1 ; Занесення порядку числа X у CH
33 ror cl,1 ; (і знака в біт 23).
34 rol dx,1 ; Занесення порядку числа Y у DH
35 ror dl,1 ; (і знака в біт 23).
36 ;
37 cmp ch,dh ; Порівняння порядків X і Y
38 js @L1 ; Перехід, якщо (CH) < (DH) (Px < Py).
39 ;
40 ; 2. Знаходження різниці порядків
41 ; Примітка.  З більшого порядку віднімається менший,
42 ; щоб різниця порядків була додатньою
44 ; 2.1. Порядок X не менше порядку Y (Px ≥Py) (CH)≥(DH).
45 mov ah,ch ; Знаходження різниці порядків або, іншими
46 sub ah,dh ; словами, кількості зсувів меншої мантиси
47 mov P,ch ; Занесення в P більшого порядку
48 mov al,dl ;      \
49 mov bl,Y ; У AX,BX заноситься менше число
50 mov bh,Y+1 ;/
51 mov SIGL,al ; Зберігання знака меншого числа
52 mov dl,X ; У CX,DX заноситься більше число
53 mov dh,X+1 ; (за модулем)
54 mov SIGN,cl ; Зберігання знака результату
55 jmp @L2
56 ;
57 ; 2.2. Порядок X менше порядку Y (Px < Py) (CH) < (DH).
58 @L1:
59 mov ah,dh ; Знаходження різниці порядків або
60 sub ah,ch ; кількості зсувів меншої мантиси.
61 mov P,dh ; Занесення в P більшого порядку
62 mov al,cl ;\
63 mov bl,X ; У AX,BX заноситься менше число
64 mov bh,X+1 ;/
65 mov SIGL,al ; Зберігання знака меншого числа
66 mov cl,dl ;      \
67 mov dl,Y ; У CX,DX заноситься більше число
68 mov dh,Y+1 ;/ (за модулем)
69 mov SIGN,cl ; Зберігання знака результату
70 ;
71 ; 3. Перетворення мантис доданків у додатковий код.
72 ;
73 ; 3.1. Переведення трьох байтів мантиси меншого числа (al,bx)
74 ; із прямого коду в додатковий
75 @L2:
76 sal al,1 ; біт (C) ← (регістр al) ←0
```

```
 77 jae @L3 ; Перехід до мітки @L3, якщо (C) = 0, тобто
 78 ; число в регістрах (al,bx) - додатне
 79 rcr al,1 ; Повернення "1" із біта (C) у старший біт (al).
 80 not bx ;   \
 81 not al ; Переведення в ДК
 82 add bx,1 ; від'ємного числа в регистрах (al,bx)
 83 adc al,0 ;/
 84 jmp @L4
 85 @L3:
 86 rcr al,1 ; Повернення "0" із біта (C) у старший біт (al).
 87 add al,80H ; Примусова установка старшого біта
 88 ; регістра (al) у "1" (його вага = 1/2)
 89 ;
 90 ; 3.2. Переведення трьох байтів мантиси більшого числа (cl,dx)
 91 ; із прямого коду в додатковий
 92 @L4:
 93 sal cl,1 ; біт (C) ←(регістр cl) ←0
 94 jae @L5 ; Перехід до мітки @L5, якщо (C) = 0, тобто
 95 ; число в регістрах (cl,dx) - невід'ємне
 96 rcr cl,1 ; Повернення "1" із біта (C) у старший біт (cl)
 97 not dx ;\
 98 not cl ; Переведення в ДК
 99 add dx,1 ; від'ємного числа в регістрах (cl,dx)
100 adc cl,0 ;/
101 jmp @L6
102 @L5:
103 rcr cl,1 ; Повернення "0" із біта (C) у старший біт (cl)
104 add cl,80H ; Примусова установка старшого біта
105 ; регістра (al) у "1" (його вага = 1/2)
106 ;
107; 4. Значення отриманої різниці використовується як кількість
108 ; зсувів управо мантиси числа з меншим порядком
109 @L6:
110 cmp ah,0
111 je @L7
112 mov ch,SIGL ;\
113 sal ch,1 ; Зсув управо
114 rcr al,1 ; меншого числа
115 rcr bx,1 ;/
116 dec ah ; Зменшення на 1 значення лічильника зсувів
117 jmp @L6
118 ;
119 ; 5. Додавання мантис за правилами додавання чисел з
120 ; фіксованою комою
121 @L7:
```

```
122 add bx,dx ; Додавання
123 adc al,cl ; мантис
124 ; 6. Перетворення мантиси результату в прямий код
125 @L8:
126 mov cl,SIGN ; Дублювання
127 sar cl,1 ; знака результату
128 sal cl,1 ; Перевірка знака результату (числа з великим
129 ; порядком)
130 jae @L9 ; Перехід до мітки @L9, якщо (C) = 0, тобто
131 ; якщо мантиса суми додатня
132 sub bx,1 ;\
133 sbb al,0 ; Переведення негативної мантиси суми в
134 not bx ; (al,bx) із додаткового коду в прямий
135 not al ;/
136 ;
137 ; 7. Нормалізація мантиси результату і при необхідності
138 ; корекція значення його порядку
139 @L9:
140 sal bx,1
141 rcl al,1
142 jb @L10
143 sub ah,1
144 jmp @L9
145 ;
146 ; 8. Одержання порядку й остаточного результату
147 @L10:
148 add ah,P
149 ;
150 sal cl,1 ; біт (C) ← 0, (регістр al) ← 0
151 rcr ax,1
152 rcr bx,1
153 ;
154 mov Z,bl ;\
155 mov Z+1,bh ; Пересилка результату
156 mov Z+2,al ; в пам'ять
157 mov Z+3,ah ;/
158 ;
159 jmp @S
160 ;
161 FSUMMA ENDP
162 CSEG ENDS
163 END FSUMMA
```

Множення і ділення

Множення. Множення чисел із плаваючою комою принципових труднощів не викликає, тому що з представлення чисел у формі

$$X = m_x \cdot 2^{п_x},\ Y = m_y \cdot 2^{п_y}$$

випливає, що

$$Z = X \cdot Y = (m_x \cdot m_y) 2^{п_x + п_y}.$$

Іншими словами, мантиса добутку дорівнює добутку мантис співмножників, а порядок добутку дорівнює сумі їхніх порядків. Для множення мантис, як чисел із фіксованою комою, можна застосувати будь-який із розглянутих у лабораторній роботі 2 варіантів множення. Знак добутку визначається окремою дією – шляхом додавання по модулю 2 знаків співмножників.

При множенні нормалізованих мантис може виникнути тільки порушення нормалізації вправо максимум на один розряд, тому що добуток мінімальних мантис, рівних 1/2, дає 1/4 (або 0.01B). Воно усувається звичайною уявою: мантиса зсувається вліво на один біт, а порядок зменшується на 1.

В операції множення можуть виникати як переповнювання, так і антипереповнювання. Переповнювання виявляється, коли сума порядків співмножників більше максимального припустимого порядку, а антипереповнювання – після того, як зроблений декремент порядку при усуненні порушення нормалізації вправо, і порядок виявився менше мінімального припустимого.

Роздивимося два варіанти множення: у програмі 5.2 реалізований стандартний алгоритм із циклом множення на окремі біти множника, а в програмі 5.3 застосовується команда множення MUL.

Програма 5.2

```
TITLE Множення чисел із плаваючою комою
;
SSEG    SEGMENT PARA STACK 'STACK'
        DB 256 DUP(0)
SSEG    ENDS
;
DSEG    SEGMENT PARA PUBLIC 'DATA'
X       DD      64.0  ;1 DUP(0)
Y       DD      126. 25  ;1 DUP(0)
Z       DD      1 DUP(0)
DSEG    ENDS
;
CSEG    SEGMENT PARA PUBLIC 'CODE'
ASSUME CS:CSEG,DS:DSEG,SS:SSEG
;
MULT    PROC    FAR
        mov     ax,DSEG
        mov     ds,ax
; Перший операнд X знаходиться в регістрах BX:SI,
```

```
; другий Y –  в регістрах DX:DI, добуток
; повертається в регістри BX:SI.
; При виникненні особливого випадку CF=1
;
Start:  lea    si,X
        lea    di,Y
        mov    bx,[si+2]
        mov    si,[si]
        mov    dx,[di+2]
        mov    di,[di]
MULF:   ; Перевірити операнди на нуль
        mov    ax,bx   ; Перевірити на 0
        or     ax,si       ; перший операнд
        jz     MULF8  ; Добуток дорівнює 0
        mov    ax,dx    ; Перевірити на 0
        or     ax,di       ; другий операнд
        jnz    MULF1   ; Операнди не рівні 0
        xchg   bx,dx    ; Добуток дорівнює 0
        xchg   si,di
        jmp    MULF8
; Операнди не нульові, можна множити.
; Утворити знак добутку, відновити мантиси.
MULF1:  mov    ch,bh   ; Знак добутку
        xor    ch,dh   ; у регістрі CH
        shl    bx,1    ; Відновити схований біт
        stc            ; мантиси першого операнда
        rcr    bl,1
        shl    dx,1    ; Відновити схований біт
        stc            ; мантиси другого операнда
        rcr    dl,1
        ; Скласти порядки.
        mov    al,bh   ; Скласти в AL
        add    al,dh    ; усунуті порядки
        jc     MULF2   ; Виник перенос
        sub    al,127   ; Відняти зсув
        jnc    MULF3   ; Можна множити
        jmp    MULF8   ; Виникло антипереповнювання
MULF2:  add    al,129  ; Врахувати втрату 256 через перенос
        jnc    MULF3   ; Можна множити
        jmp    MULF8   ; Виникло переповнювання
        ; Можна множити мантиси.
MULF3:  mov    bh,al   ; Порядок добутку в BH
        xor    dh,dh   ; Підготувати місце
        xor    ax,ax   ; для добутку
        mov    cl,24   ; Утворити лічильник бітів
```

```
        ; Тут починається цикл множення
MULF4:  rcr     dl,1    ; Зсунути множник
        rcr     di,1    ; вліво на один біт
        jnc     MULF5   ; Біт множника дорівнює 0
        add     ax,si   ; Додати множене
        adc     dh,bl   ; до добутку
MULF5:  rcr     dh,1    ; Зсунути суму
        rcr     ax,1            ; часткових добутків
        dec     cl
        jnz     MULF4
        mov     bl,dh   ; Повернути добуток
        mov     si,ax   ; у регістри BL:SI
        ; Перевірити порушення нормалізації вліво.
MULF6:  or      bl,bl   ; Перевірити старший біт мантиси
        jns     MULF7       ; Порушення нормалізації немає
        inc     bh              ; Збільшити порядок на 1
        stc                     ; CF=1
        jz      MULF8       ; Виникнуло переповнювання
        jmp     MULFA   ; Переповнювання немає
MULF7:  shl     si,1        ; Зсунути мантису
        rcl     bl,1            ; вліво на один біт
        ; Формування результату
MULFA:  add     ch,ch   ; Знак у прапорі переносу
        rcr     bh,1    ; Знак числа на місці
        rcr     ch,1    ; Молодший біт порядку в CH
        or      ch,7fh  ; Утворити маску
        and     bl,ch   ; Утворити 2-й байт добутку
        lea     di,Z mov
                  [di],si
        mov     [di+2],bx
MULF8:  jmp Start       ; RET
MULT    ENDP
CSEG    ENDS
END     MULT
```

Програма 5.3

```
TITLE Множення чисел із плаваючою комою
TITLE із застосуванням команди MUL МП 1810
;
SSEG    SEGMENT PARA STACK 'STACK'
        DB 256 DUP(0)
SSEG    ENDS
;
DSEG    SEGMENT PARA PUBLIC 'DATA'
X       DD      64.0 ;1 DUP(0)
```

```
Y       DD      126. 25 ;1 DUP(0)
Z       DD      1 DUP(0)
TEMP    DD      1 DUP(0)
DSEG    ENDS
;
CSEG    SEGMENT PARA PUBLIC 'CODE'
ASSUME CS:CSEG,DS:DSEG,SS:SSEG
;
MULT    PROC    FAR
        mov     ax,DSEG
        mov     ds,ax
; Перший операнд X знаходиться в регістрах BX:SI,
; другий Y – у регістрах DX:DI, добуток
; повертається в регістри BX:SI.
; При виникненні особливого випадку CF=1
;
Start:  lea     si,X
        lea     di,Y
        mov     bx,[si+2]
        mov     si,[si]
        mov     dx,[di+2]
        mov     di,[di]
MULF:   ; Перевірити операнди на нуль
        mov     ax,bx   ; Перевірити на 0
        or      ax,si   ; перший операнд
        jz      @@M1    ; Добуток дорівнює 0
        mov     ax,dx   ; Перевірити на 0
        or      ax,di   ; другий операнд
        jnz     MULF1   ; Операнди не рівні 0
        xchg    bx,dx   ; Добуток дорівнює 0
        xchg    si,di
@@M1:   jmp     MULF8
; Операнди не нульові, можна множити.
; Утворити знак добутку, відновити мантиси.
MULF1:  mov     ch,bh   ; Знак добутку
        xor     ch,dh   ; у регістрі CH
        shl     bx,1    ; Відновити схований біт
        stc             ; мантиси першого операнда
        rcr     bl,1
        shl     dx,1    ; Відновити схований біт
        stc             ; мантиси другого операнда
        rcr     dl,1
        ; Скласти порядки.
        mov     al,bh   ; Скласти в AL
        add     al,dh   ; усунуті порядки
```

```
        jc      MULF2   ; Виник перенос
        sub     al,127  ; Відняти зсув
        jnc     MULF3   ; Можна множити
        jmp     MULF8   ; Виникло антипереповнювання
MULF2:  add     al,129  ; Врахувати втрату 256 через перенос
        jnc     MULF3   ; Можна множити
        jmp     MULF8   ; Виникло переповнювання
        ; Можна множити мантиси.
MULF3:  mov     bh,al   ; Порядок добутку в BH
        xor     dh,dh   ; Підготувати місце
        xor     ax,ax   ; для добутку
        mov     cl,dl   ; Звільнити регістр DX
        ; Тут починається цикл множення
MULF4:  mov     ax,si   ; Умножити молодші
        mul     di      ; слова мантис
        mov     WORD PTR TEMP,dx ; Зберегти старшу частину
                                 ;добутку
        mov     al,bl
        mov     ah,0
        mul     di
        add     WORD PTR TEMP,ax ; Врахувати результат
        jnc     NEXT    ; у повному добутку
        inc     dx
NEXT:   mov     WORD PTR TEMP+2,dx
        mov     al,cl   ; Умножити наступні
        mov     ah,0    ; частини мантис
        mul     si
        add     WORD PTR TEMP,ax
        jnc     NEXT1
        inc     dx
NEXT1:  add     dx,WORD PTR TEMP+2
        mov     al,bl   ; Умножити старші
        mul     cl      ; байти мантис
        add     ax,dx   ; Утворити повний добуток
        mov     bl,ah   ; і розмістити його
        mov     ah,al   ; в регістрах BL:SI
        mov     al,BYTE PTR TEMP+1
        mov     si,ax
        ; Перевірити порушення нормалізації вліво.
MULF6:  or      bl,bl   ; Перевірити старший біт мантиси
        jns     MULF7   ; Порушення нормалізації немає
        inc     bh      ; Збільшити порядок на 1
        stc             ; CF=1
        jz      MULF8   ; Виникло переповнювання
        jmp     MULFA   ; Переповнювання немає
```

```
MULF7:  shl     si,1    ; Зсунути мантису
        rcl     bl,1    ; вліво на один біт
        ; Формування результату
MULFA:  add     ch,ch   ; Знак у прапорі переносу
        rcr     bh,1    ; Знак числа на місці
        rcr     ch,1    ; Молодший біт порядку в CH
        or      ch,7fh  ; Утворити маску
        and     bl,ch   ; Утворити 2-й байт добутку
        lea     di,Z mov
                  [di],si
        mov     [di+2],bx
MULF8:  jmp Start       ; RET
MULT    ENDP
CSEG    ENDS
END     MULT
```

Ділення. Із представлення діленого X і дільника Y у формі

$$X = m_x \cdot 2^{n_x}, \quad Y = m_y \cdot 2^{n_y}$$

випливає, що частка

$$Z = X/Y = (m_x/m_y)2^{n_x - n_y}.$$

Таким чином, мантиса частки дорівнює результату ділення мантис операндів як чисел із плаваючою крапкою, а порядок частки дорівнює різниці їхніх порядків. Для ділення мантис застосуємо будь-який варіант, але звичайно застосовується варіант із зсувом залишків уліво. Якщо mx>my, ділення мантис дає цілу частину, рівну 1. Звичайно така ситуація при діленні чисел із фіксованою крапкою вважається переповнюванням, але в даному випадку цього не відбувається, тому що результуючу мантису завжди можна зрушити на один біт вправо і відповідно скорегувати (інкремент) порядок. Порушення нормалізації вправо при діленні нормалізованих чисел не відбувається.

В операції ділення, як і при множенні, можуть виникати переповнювання й антипереповнювання. Переповнювання має місце, коли після інкременту порядку (для усунення порушення нормалізації вліво) одержують число, більше максимального припустимого порядку, а антипереповнювання – коли різниця порядків виявилася менше мінімального припустимого порядку, в зв'язку з цим використання команди DIV МП 1810 для ділення мантис не є можливим.

Програма 5.4

```
TITLE Ділення чисел у форматі з плаваючою комою
SSEG    SEGMENT PARA STACK 'STACK'
        DB 256 DUP(0)
SSEG    ENDS
DSEG    SEGMENT PARA PUBLIC 'DATA'
X       DD      64.0  ;1 DUP(0)
Y       DD      126. 25  ;1 DUP(0)
```

```
Z       DD      1 DUP(0)
DSEG    ENDS
CSEG    SEGMENT PARA PUBLIC 'CODE'
ASSUME CS:CSEG,DS:DSEG,SS:SSEG
DIVFLT  PROC    FAR
        mov     ax,DSEG
        mov     ds,ax
; Перший операнд X знаходиться в регістрах BX:SI,
; другий Y – у регістрах DX:DI, добуток
; повертається в регістри BX:SI.
; При виникненні особливого випадку CF=1
Start:  lea     si,X
        lea     di,Y
        mov     bx,[si+2]
        mov     si,[si]
        mov     dx,[di+2]
        mov     di,[di]
DIVF:   ; Перевірити операнди на нуль
        mov     ax,bx   ; Перевірити на 0
        or      ax,si   ; ділене
        jz      DIVF7   ; Не нульовий результат у BX:SI
        mov     ax,dx   ; Перевірити на 0
        or      ax,di   ; дільник
        stc             ; Якщо ділення на 0
        jz      DIVF7   ; CF = 1
        ; Обидва операнди не нульові
        mov     ch,bh   ; Утворити знак частки
        xor     ch,dh   ; в регістрі CH
        shl     bx,1    ; Відновити схований біт
        stc             ; мантиси діленого
        rcr     bl,1
        shl     dx,1    ; Відновити схований біт
        stc             ; мантиси дільника
        rcr     dl,1
        ; Відняти порядки
        mov     al,bh   ; Утворити різницю порядків
        sub     al,dh   ; в регістрі AL
        jnc     DIVF1   ; Порядок діленого більше
        add     al,127  ; Додати зсув
        cmc             ; Якщо немає переносу,
        jc      DIVF7   ; виникло антипереповнювання
        jmp     DIVF2   ; Перейти до ділення мантис
DIVF1:  add     al,127  ; Додати зсув
        jc      DIVF7   ; Виникло переповнювання
        ; Ділення мантис.
```

```
DIVF2:  mov     bh,0    ; Очистити старші байти
        mov     dh,0    ; перед мантисами
        mov     cl,24   ; Утворити лічильник біт
DIVF3:  sub     si,di   ; відняти мантису дільника
        sbb     bx,dx   ; із мантиси діленого
        cmc             ; Утворити в CF біт частки
        pushf           ; Зберегти його в стеку
        jc      DIVF4   ; Залишок позитивний
        add     si,di   ; Відновити попередній
        adc     bx,dx   ; позитивний залишок
DIVF4:  popf            ; Повернути біт частки в CF
        rcl     bp,1    ; Передати біт частки
        rcl     ah,1    ; у регістри AH:BP
        shl     si,1    ; Зсунути залишок
        rcl     bx,1
        dec     cl      ; Декремент лічильника
        jnz     DIVF3   ; Повторювати до завершення
        ; Перевірити порушення нормалізації вправо
        test    ah,80H  ; Перевірити старший біт мантиси
        jnz     DIVF5   ; Порушення нормалізації немає
        dec     al      ; Декремент порядку
        stc             ; Перевірити можливість
        jz      DIVF7   ; антипереповнювання
        sub     si,di; Визначити ще один біт
        sbb     bx,dx   ; мантиси частки,
        cmc             ; помістити біт частки
        rcl     bp,1    ; в потрібне місце
        rcl     ah,1
DIVF5:  mov     bl,ah   ; Розмістити частку
        mov     si,bp   ; у регістрах BX:SI
        mov     bh,al
        ; Формувати результат
        add     ch,ch   ; Знак у прапорі переносу
        rcr     bh,1    ; Знак числа на місці
        rcr     ch,1    ; Молодший біт порядку в CH7
        or      ch,7fh  ; Утворити маску
        and     bl,ch   ; Утворити другий байт добутку
        lea     di,Z
        mov     [di],si
        mov     [di+2],bx
DIVF7:  jmp Start       ; RET
DIVFLT  ENDP
CSEG    ENDS
END     DIVFLT
```

Домашнє завдання

1. Вивчіть команди арифметичних і логічних зсувів МПК 1810.
2. Вивчіть команди умовного і безумовного переходів МПК 1810.
3. Намалюйте в протоколі схему алгоритмів програм.
4. Для двох пар чисел X і Y докладно запишіть їхнє перетворення у ФПК, обчисліть значення Z у десятковій системі і запишіть перетворення Z у ФПК.

Порядок виконання роботи

1. Скопіюйте файл fadd-sub.asm у файл під ім'ям st???.asm (??? – останні три цифри номера залікової книжки).
2. Відтранслюйте програму, набрав у командному рядку tasm /ZI /L st???.asm (не забудьте, що замість ??? треба набирати цифри).
3. Якщо помилок немає (з'явився файл із розширенням *.obj) , то переходіть до п. 4 виконання роботи. При наявності помилок перегляньте файл із розширенням *.lst (F3), у якому вони відзначені, виправте їх у файлі з розширенням *.asm (F4) і повторіть п.п. 2 і 3 виконання роботи.
4. Скомпонуйте завантажувальний файл, набрав у командному рядку tlink /v st???.obj (тут теж замість ??? треба набирати потрібні цифри).
5. Запустіть налагоджувач, набрав у командному рядку td st???.exe (знову замість ??? треба набирати цифри).

Примітка. Якщо ви не набрали імені програми (тобто запустили налагоджувач командним рядком td), то уважно вивчіть розділ "Завантаження нової програми для налагодження" стислого опису налагоджувача TD.

6. У налагоджувачі, після появи на екрані вихідного тексту програми у вікні Module, викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Registers (натисніть R). Перемістіть вікно Registers (Регістри) у правий верхній кут екрана (натисніть послідовно клавіші Ctrl-F5, End, PgUp, Enter).
7. Для перегляду пам'яті викличте меню VIEW (натисніть Alt-V). У меню VIEW виберіть для перегляду вікно Dump (натисніть D). Перемістіть вікно Dump вниз екрана (натисніть послідовно клавіші Ctrl-F5, PageDown, Enter).
8. У вікні Dump побайтно занесіть значення чисел X і Y.
9. Трасування програми виконуйте клавішею F7. Після кожного натискання клавіші F7 переглядайте вміст регістрів і пам'яті.
10. Після переходу до мітки "@S:", тобто до початку програми, змініть значення чисел X і Y, і повторіть трасування при нових значеннях доданків, які задані викладачем.
11. Вийдіть з програми TD.EXE (натисніть комбінацію клавіш Alt-X).
12. Зведіть в таблицю результати обчислень (значення X, Y і Z), отримані вручну і програмно. Порівняйте їх. Зробить висновки.

Таблиця 5.1

	Заповнюється до виконання програми		Заповнюється після виконання програми	
	DEC	HEX (4 БАЙТ)	HEX (4 БАЙТ)	DEC
X				
Y				
Z				

13. Повторіть п.п. 1–12 для файлів floatmul.asm, fltmul.asm, divflt.asm.

Контрольні питання

1. Які команди зсувів ви знаєте?
2. Які команди переходів ви знаєте?
3. За допомогою яких команд можна перекласти число (довжиною 1; 2; 3; 4 байта) із прямого коду в додатковий і навпаки?

ЛАБОРАТОРНА РОБОТА 6

ЗАСТОСУВАННЯ МІКРОПРОЦЕСОРІВ У ЦИФРОВІЙ ОБРОБЦІ СИГНАЛІВ (ЦИФРОВА ФІЛЬТРАЦІЯ)

Мета роботи: ознайомлення із засобами цифрової обробки сигналів із застосуванням мікропроцесорів, на прикладі цифрового фільтра.

Стислі теоретичні відомості й приклади програм

Основні поняття

При обробці радіотехнічної інформації великий обсяг займає фільтрація.

Фільтр – пристрій із мінімальним ослабленням, що передає коливання частот, що потрапляють в область смуги перепускання.

У задачах фільтрації можуть застосовуватися як аналогові, так і цифрові фільтри.

Аналоговий фільтр являє собою частотно-виборчий ланцюг (рис.6.1), що здійснює деяке лінійне перетворення над безупинним вхідним сигналом $u_{вх}(t)$. Результатом такого перетворення є неперервний вихідний сигнал $u_{вих}(t)$. Особливість цифрового фільтра складається в тому, що вказане вище перетворення виконується не над неперервним сигналом $u_{вх}(t)$, а над вхідною цифровою по-

слідовністю x(n), і одержуваний на виході результат перетворення y(n) являє собою також цифрову послідовність.

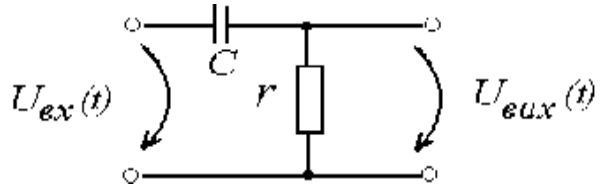

Рис. 6.1

Розглянемо відгук $u_{вих}(t)$ даного rC-ланцюга на вхідний вплив $u_{вх}(t)$. Струм i(t) у ланцюзі визначається виразом

$$i(t) = C\frac{d(u_{вх}(t) - u_{вих}(t))}{dt} = \frac{u_{вих}(t)}{r}. \qquad (6.1)$$

Приймаючи до уваги цифровий засіб рішення задачі, представимо вхідну $u_{вх}(t)$ і вихідну $u_{вих}(t)$ напруги відповідними цифровими послідовностями $x_n=x(nT)$ и $y_n=y(nT)$. Тоді похідна в (6.1) може бути замінена таким наближенням:

$$\frac{d(u_{вих}(t) - u_{вих}(t))}{dt}\Big|_{t=nT} \approx \frac{(x_n - y_n) - (x_{n-1} - y_{n-1})}{T} \qquad (6.2).$$

Підставивши (6.2) у (6.1) і розв'язавши отриманий вираз щодо y_n, одержимо:

$$y_n = a_0 x_n + a_1 x_{n-1} - b_1 y_{n-1}, \qquad (6.3)$$

де $a_0 = 1/(1+T/rC)$, $a_1 = -1/(1+T/rC)$, $b_1 = -1/(1+T/rC)$.

Отримане різницеве рівняння може бути використане для побудови цифрового фільтра 1-го порядку.

Відомо, що однією з характеристик фільтра, що цілком визначають виконувані ним перетворення, є перехідна характеристика. Перехідна характеристика аналогового фільтра є відгук на вплив, що має форму одиничного стрибка (одиничної функції). Перехідна характеристика цифрового фільтра є цифрова послідовність, що являє собою визначене лінійне перетворення, виконане над вхідною послідовністю такого вигляду:

$$x(nT) = \begin{cases} 1 \text{ при } n \ge 0, \\ 0 \text{ при } n < 0. \end{cases}$$

Вирішуючи різницеве рівняння (6.3) за даною вхідною послідовністю x(n), можна одержати перехідну характеристику h цифрового фільтра 1-го порядку.

Результати обчислень за різноманітними значеннями кроку інтегрування T приведені в табл. 6.1. У цій же таблиці приведена перехідна характеристика аналогового фільтра.

Таблиця 6.1

Фільтр	h при t/r, рівному				
	0	0.5	1.0	1.5	2.0
Аналоговий фільтр	1	0.7788	0.3679	0.2231	0.136
Цифровий					

фільтр T=0,5rC	0.6667	0.4444	0.2963	0.1975	0.132
T=0,25rC	0.8000	0.5120	0.3277	0.2097	0.134
T=0,125rC	0.8889	0.5549	0.3464	0.2163	0.135

На рис. 6.2 ці характеристики подані у формі графіків. Як випливає з цих даних, обираючи достатньо малим значення кроку інтегрування T, можна за допомогою цифрового фільтра відтворити з будь-якою точністю перехідну характеристику аналогового фільтра.

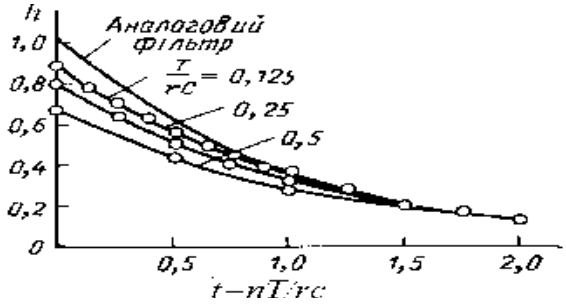

Рис. 6.2

Звернемо увагу на таку особливість цифрового фільтра. Він виконує те ж саме перетворення, що й аналоговий фільтр при визначених значеннях коефіцієнтів у виразі (6.3). При інших значеннях коефіцієнтів цифровий фільтр виконує перетворення, що може виявитися не реалізованим за допомогою аналогового фільтра.

Поряд із перехідною характеристикою виконуване аналоговим фільтром перетворення може бути описане за допомогою передатної функції, обумовленої відношенням H(s)=y(s)/x(s), де x(s) і в(s) – перетворення по Лапласу відповідно вхідного x(t) і вихідного в(t) сигналів. Передатна функція цифрового фільтра визначається за виразом H(z)=y(z)/x(z), де x(z) і в(z) являють собою z-перетворення відповідно до вхідної і вихідної цифрових послідовностей. Z-перетворення цифрової послідовності x(n), n = 0, 1, визначається за формулою

$$X(z) = \sum_{n=0}^{\infty} x(nT) z^{-n}. \qquad (6.4)$$

Обернене z-перетворення

$$x(nT) = \frac{1}{2\pi j} \oint X(z) z^{n-1} dz. \qquad (6.5)$$

Наводимо результати z-перетворення для деяких окремих випадків (табл. 6.2).

Таблиця 6.2

Цифрова послідовність	z-перетворення послідовності
Одиничний імпульс $x(nT)=\begin{cases}1, & n=0,\\ 0, & n\neq 0\end{cases}$	$X(z)=1$
Одиничний стрибок $x(nT)=\begin{cases}1, & n\geq 0\\ 0, & n<0\end{cases}$	$X(z)=\frac{1}{1-z^{-1}}$
Затримка на один тактовий період T $y(nT)=x(nT-T).$	$Y(z)=X(z)z^{-1}.$

Таким чином, застосовуючи z-перетворення до послідовностей, що входять у вираз (6.3), можна це різницеве рівняння записати в такому вигляді:

$$Y(z)=a_0 X(z)+a_1 X(z)z^{-1}-b_1 Y(z)z^{-1} \qquad (6.6)$$

Рішення даного рівняння реалізується поданої на рис. 6.3 схемою цифрового фільтра 1-го порядку .

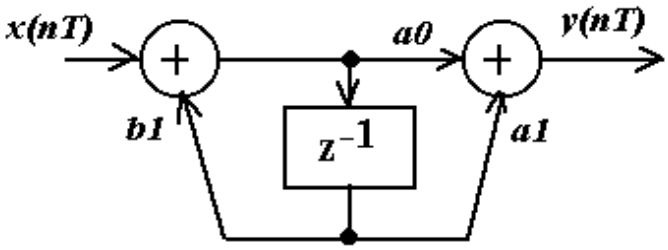

Рис. 6.3

У схемі цифрового фільтра позначений z^{-1} елемент виконує затримку цифрової послідовності на один тактовий період T, показані на лініях числа – коефіцієнти, на які умножаються відповідні послідовності.

Вирішимо вираз (6.6) відносно Y(z):

$$Y(z)=\frac{a_0+a_1 z^{-1}}{1+b_1 z^{-1}}X(z).$$

Звідси передатна функція цифрового фільтра 1-го порядку

$$H(z)=(a_0+a_1 z^{-1})/(1+b_1 z^{-1}). \qquad (6.7)$$

Передатна функція аналогового ланцюга дозволяє одержувати його частотну характеристику, для чого достатньо зробити в H(s) заміну s на $j\omega$.

Таким чином, $H(j\omega)$ являє собою коефіцієнт передачі для гармоніки вхідного сигналу частоти ω. Передатна функція цифрового фільтра H(z) при підстановці замість z цифрової послідовності експоненційного коливання $e^{j\omega T}$ дозволяє визначити $H(e^{j\omega T})$ частотну характеристику цифрового фільтра. Ця характеристика визначає коефіцієнт передачі лінійної цифрової системи для вхідної цифрової послідовності, що відображає комплексне експоненційне коливання $x(nT)=e^{jn\omega T}$. Так, частотна характеристика цифрового фільтра 1-го порядку, передатна функція якого визначається за виразом (6.7), має такий вид:

$$H(e^{j\omega T})=(a_0+a_1 e^{-j\omega T})/(1+b_1 e^{-j\omega T}). \qquad (6.8)$$

Для цифрового фільтра 2-го порядку різницеве рівняння буде таким:

$$y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} - b_1 y_{n-1} - b_2 y_{n-2}, \qquad (6.9)$$

а передатна функція:

$$H(z) = \frac{(a_0 + a_1 z^{-1} + a_2 z^{-2})}{(1 + b_1 z^{-1} + b_2 z^{-2})}. \qquad (6.10)$$

Функціональна схема такого цифрового фільтра показана на рис. 6.4.

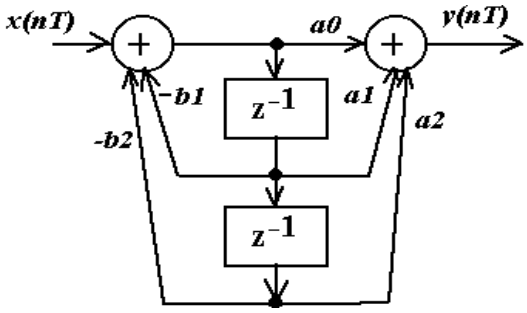

Рис. 6.4

У загальному випадку різницеве рівняння цифрового фільтра буде:

$$y_n = \sum_{i=0}^{M} a_i x_{n-i} - \sum_{i=1}^{N} b_i y_{n-i}. \qquad (6.11)$$

При N=0 утвориться нерекурсивний цифровий фільтр, що характеризується тим, що відгук його представляється сумою деякого числа членів вхідної послідовності x(n).

При N > 0 утворюється рекурсивний цифровий фільтр із відгуком, що виражається сумою, у якій виявляються не тільки члени вхідної цифрової послідовності, але і попередні члени вихідної послідовності. Рекурсивні фільтри за певних умов можуть бути нестійкими і значення у вихідній послідовності можуть необмежено наростати.

Загальній формі різницевого рівняння (6.11) відповідає такий вираз передатної функції:

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_M z^{-M}}{1 + b_1 z^{-1} + b_2 z^{-2} + ... + b_N z^{-N}}. \qquad (6.12)$$

Звернемо увагу на таку особливість цифрових фільтрів. Аналогові фільтри фізично реалізовані, якщо в їхніх передатних функціях степінь полінома чисельника не вище степені полінома знаменника. Цифрові фільтри не пред'являють таких обмежень, і, таким чином, вони можуть мати характеристики, домогтися яких в аналогових фільтрах неможливо.

Звернемося до питань, пов'язаних із реалізацією цифрових фільтрів із використанням мікропроцесорних пристроїв. При побудові цифрового фільтра на МП часто зштовхуються з необхідністю рішення проблеми підвищення швидкодії. На вхід цифрового фільтра надходить вихідна цифрова послідовність x(n), на вихід видається послідовність y(n), що являє собою результат обробки вхідної послідовності. Швидкодія може оцінюватися припустимим для даної реалізації мінімальним значенням тактового періоду T_{min} вхідної цифрової послідовності x(n) або пов'язаним із T_{min} максимальним значенням ширини смуги

частот $F_{max}=1/(2T_{min})$ сигналу, дискретизацією якого отримана вхідна послідовність x(n). Чим менше T_{min} або більше F_{max}, тим вище швидкодія цифрового фільтра. Необхідність у високій швидкодії пов'язана з прагненням обробляти у реальному масштабі часу широкосмужні сигнали.

Однокристальні універсальні МП мало придатні для побудови швидкодіючих цифрових фільтрів. Більша швидкодія досягається при використанні мікропроцесорів із розрядно-модульною організацією. Проте й останні часто не спроможні забезпечити необхідну швидкодію без використання спеціальних прийомів організації обробки інформації у фільтрі, розгляду яких присвячується подальший виклад.

Реалізація цифрового фільтра шляхом послідовного включення елементарних фільтрів

Якщо позначити $p = z^{-1}$, то чисельник виразу H(z) (6.12) представиться поліном $a_0 + a_1 p + ... + a_M p^M$. Прирівнявши поліном до нуля і вирахувавши корені отриманого рівняння p_i, i = 1,..., M, можна уявити поліном добутком:

$$a_0 + a_1 p + ... + a_M p^M = a_M(p - p_1)(p - p_2) ... (p - p_M). \qquad (6.13)$$

Тут p_i можуть бути дійсними або комплексними числами. В останньому випадку для кожного комплексного p_i серед коренів знайдеться корінь p_j , що має сполучене з p_i значення. Пари членів, що відповідають таким комплексно-сполученим членам у (6.13), можна уявити тричленами виду $p^2-(p_i+p_j)p+p_ip_j$. При цьому коефіцієнти тричлена $-(p_i + p_j)$ та p_ip_j будуть дійсними членами. Можна також об'єднати в тричлени пари двочленів, що відповідають дійсним кореням. Тоді поліном може бути поданий виразом

$$a_0 + a_1 z^{-1} + ... + a_m z^{-m} = \prod_{i=1}^{m} (a_{0i} + a_{1i} z^{-1} + a_{2i} z^{-2}). \qquad (6.14)$$

Якщо M непарне, число двочленів буде непарним й один з них не буде мати пари. Цей двочлен у (6.14) відобразиться тричленом, у котрому $a_{2i} = 0$.

Аналогічно в формі добутку може бути поданий і знаменник виразу (6.12)

$$1 + b_1 z^{-1} + ... + b_n z^{-n} = \prod_{i=1}^{n} (1 + b_{1i} z^{-1} + b_{2i} z^{-2}). \qquad (6.15)$$

Таким чином, відношення розглянутих поліномів, тобто передатна функція H(z), може бути подана добутком дробів, у яких чисельник і знаменник є поліном степені не вище другої:

$$H(z) = \prod_{i=1}^{k} \frac{a_{0i} + a_{1i} z^{-1} + a_{2i} z^{-2}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} = H_1(z) H_2(z) ... H_k(z), \qquad (6.16)$$

де $H_i(z) = \dfrac{a_{0i} + a_{1i} z^{-1} + a_{2i} z^{-2}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}}$.

Для одержання добутку передатних функцій $H(z) = \prod_{i=1}^{k} H_i(z)$ необхідно пристрої, які реалізують елементарні функції $H_i(z)$, включити послідовно, як показано на рис. 6.5.

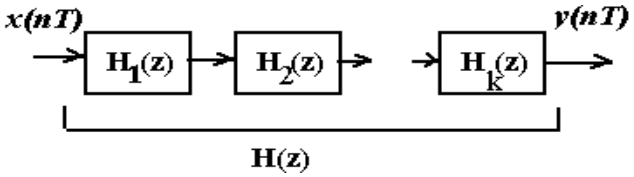

Рис. 6.5

Для кожної елементарної передатної функції $H_i(z)$ може бути побудований елементарний цифровий фільтр (1-го або 2-го порядку), а послідовне їхнє з'єднання (рис. 6.5) забезпечить результуючу передатну функцію H(z). Відповідне виразу (6.16) з'єднання фільтрів 2-го порядку показане на рис. 6.6.

Приклад 1. Нехай потрібно побудувати цифровий фільтр 5-го порядку з передатною функцією

$$H(z)=\frac{2-3z^{-1}+2z^{-2}-0.5z^{-3}}{1+1.9z^{-1}+1.8z^{-2}+0.95z^{-3}+0.3z^{-4}+0.05z^{-5}}.$$

Нулі передатної функції: 1+j; 1-j; 2; полюси: -1+j; -1-j; -1+2j; -1-2j; -2.

Отже, передатна функція може бути подана в такому виді:

$$H(z)=\frac{-0.5(z^{-1}-1-j)(z^{-1}-1+j)(z^{-1}-2)}{0.05(z^{-1}+1-j)(z^{-1}+1+j)(z^{-1}+1-2j)(z^{-1}+1+2j)(z^{-1}+2)}=\frac{-0.5(z^{-2}-2z^{-1}+2)(z^{-1}-2)}{0.05(z^{-2}+2z^{-1}+2)(z^{-2}+2z^{-1}+5)(z^{-1}+2)}=$$

$$=\frac{1}{1+z^{-1}+0.5z^{-2}}\,\frac{-1+z^{-1}-0.5z^{-2}}{1+0.4z^{-1}+0.2z^{-2}}\,\frac{-2+z^{-1}}{1+0.5z^{-1}}.$$

Таким чином, передатна функція виявилася поданою в формі добутку передатних функцій трьох елементарних цифрових фільтрів, і реалізований фільтр може мати структуру, подану на рис. 6.7.

При такому послідовному з'єднанні елементарних цифрових фільтрів виконання операцій, що пов'язані із реалізацією окремих елементарних цифрових фільтрів, може провадитися в окремих МП. Кожний МП у цьому випадку виконує не всю обробку, а лише її частину, видаючи проміжні значення для подальшої обробки в наступний МП. Останній МП буде видавати кінцевий результат обробки y(n). Якщо послідовно включено k МП, то при надходженні на вхід даних вихідної цифрової послідовності x(n) із тимчасовим інтервалом T, час, що витрачається на обробку, пов'язану з формуванням кожного вихідного значення, може складати kT.

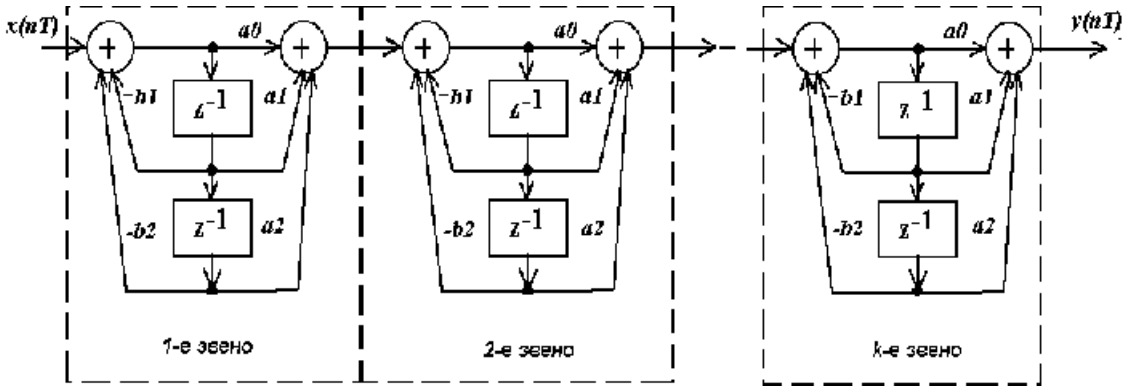

Рис. 6.6

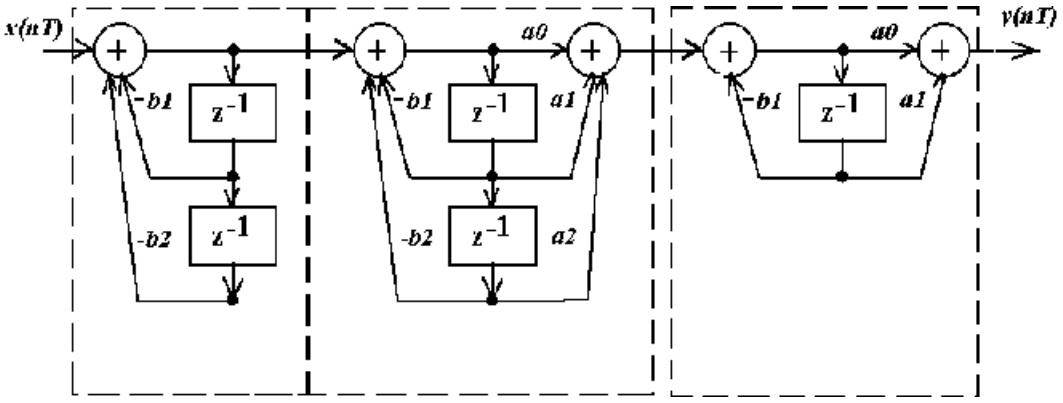

Рис. 6.7

Очевидно, якби вся обробка була б зосереджена в одному МП, він затрачував би на обробку час kT і допускав надходження на вхід даних із тимчасовим інтервалом kT. Отже, в аналізованому випадку, коли обробка розподіляється між k мікропроцесорами швидкодія (а виходить, і широкосмужність) цифрового фільтра виростає в k разів.

Реалізація цифрового фільтра паралельним включенням елементарних фільтрів.

Можливий інший спосіб розпаралелювання процесу обробки сигналу. Вираз H(z) (6.12) може бути подано не добутком, як розглянуто вище, а сумою елементарних дробів, знаменники яких формуються, як і в попередньому випадку, шляхом визначення полюсів за виразом (6.12):

Рис. 6.6

Рис. 6.7

$$H(z) = \sum_{i=1}^{k} \frac{c_{0i} + c_{1i} z^{-1}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}} = \sum_{i=1}^{k} H_i(z). \qquad (6.17)$$

Для одержання результуючої передатної функції H(z) у цьому випадку буде потрібно рівнобіжне включення елементарних цифрових фільтрів, кожний із яких реалізує одну зі складових передатної функції у виразі (6.17). Як і в послідовній реалізації, для побудови елементарних фільтрів можуть бути використані окремі МП. При цьому швидкодія фільтра зростає в стільки разів, скільки мікропроцесорів бере участь в обробці сигналів. Відмінність від попереднього випадку в тому, що результати обробки з'являються на виході із запізнюванням, рівним не kT а T.

Приклад 2. Розглянемо реалізацію тієї ж передатної функції, що й у попередньому прикладі, і представимо її у формі

$$H(z) = \frac{A + Bz^{-1}}{1 + z^{-1} + 0.5z^{-2}} + \frac{C + Dz^{-1}}{1 + 0.4z^{-1} + 0.2z^{-2}} + \frac{E}{1 + 0.5z^{-1}}.$$

Тут A, B, C, D, E – коефіцієнти, значення яких знаходяться з такої умови: після приведення суми дробів до загального знаменника вираз в чисельнику повинен співпасти з чисельником виразу заданої передатної функції H(z). Ці коефіцієнти мають значення: A=16,551726; B=–8,965525; C=–2,55124; D=–4,85517; E=21,10345.

Таким чином, передатна функція:

$$H(z)=\frac{-16.55173-8.96553z^{-1}}{1+z^{-1}+0.5z^{-2}}+\frac{-2.55124-4.85517z^{-1}}{1+0.4z^{-1}+0.2z^{-2}}+\frac{21.10345}{1+0.5z^{-1}}.$$

Цій формі передатної функції відповідає схема цифрового фільтра з паралельним включенням елементарних фільтрів (рис. 6.8).

Алгоритм обчислення і приклад програми цифрового фільтра

Схема цифрового фільтра 2-го порядку показана на рис.6.9, на рис. 6.10 – алгоритм обробки сигналу в ньому. Нумерація крапок у схемі цифрового фільтра відповідає нумерації перемінних y_i, що бережуть формовані в цих крапках значення, тобто y_1, y_2, y_3, y_4 (і, отже, комірки пам'яті, виділені для збереження значень цих перемінних) мають значення, що збігаються із значеннями величин у крапках 1, 2, 3, 4 схеми цифрового фільтра.

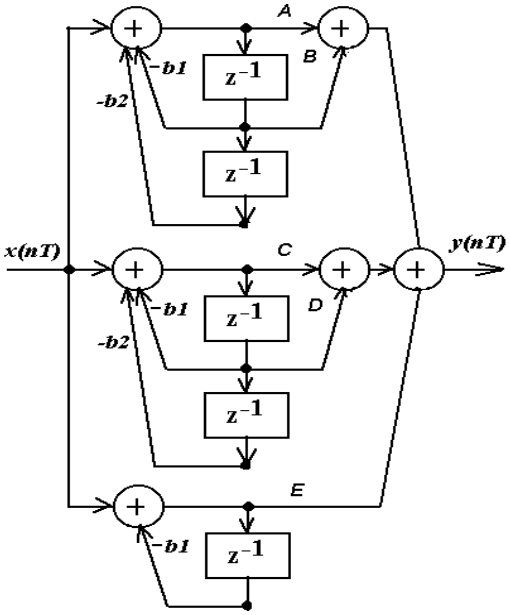

Рис. 6.8

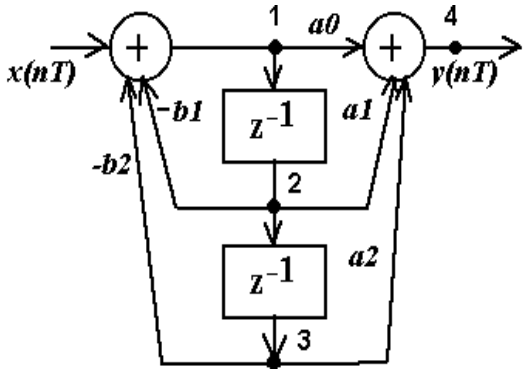

Рис. 6.9

Блоки 1, 2 схеми алгоритму значення y2, y1(рис.6.10), сформовані в попередньому повторенні циклу, передають відповідно в y3, y2 і відтворюють та-

ким чином затримку, передбачену в схемі цифрового фільтра між крапками 3 і 2, 2 і 1. Комірка y5 використовується як допоміжна комірка для формування добутків.

З 14 блоків, що утримуються в схемі алгоритму, п'ять блоків (блоки 4, 6, 8, 10, 12) передбачають виконання операції множення, і основний час, що затрачується на виконання алгоритму, пов'язаний саме з виконанням цих блоків. Можливо прискорення виконання алгоритму, якщо передбачити паралельне виконання операцій множення, тобто п'ять передбачених в алгоритмі операцій множення виконувати одночасно, використовуючи п'ять різноманітних пристроїв множення. Тому що час, що затрачується на виконання інших блоків у схемі алгоритму малий, то при паралельному виконанні множення приблизно в п'ять разів скоротиться час одноразового виконання циклу алгоритму і, отже, у п'ять разів зросте швидкодія цифрового фільтра і гранична широкосмужність оброблюваних фільтром сигналів.

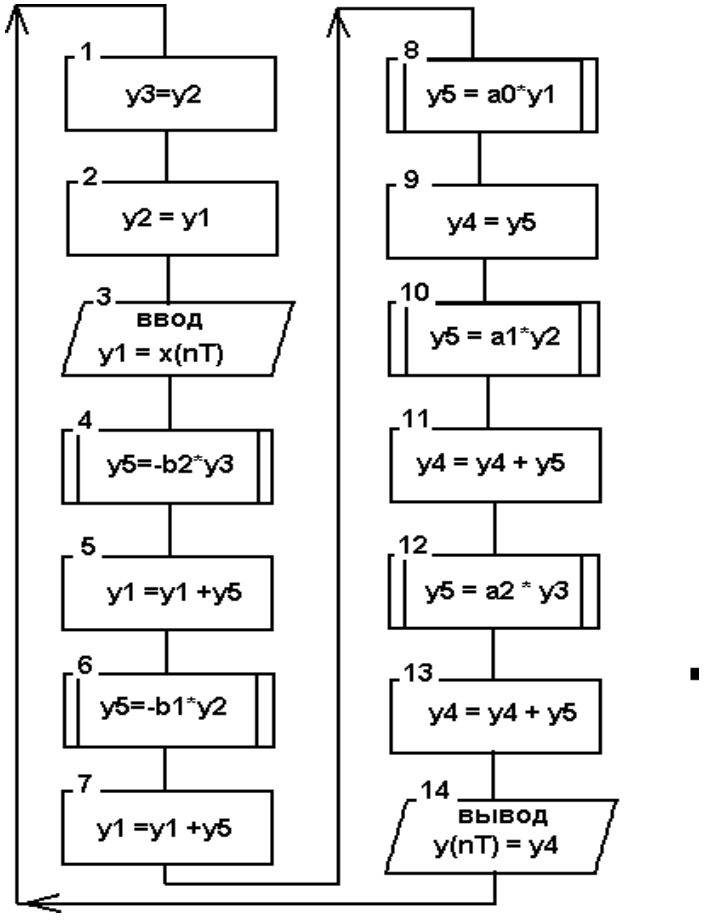

Рис. 6.10

Програма 6.1

```
TITLE Рекурсивний цифровий фільтр(ЦФ) 2-го порядку
;---------------------------------
include asm1. inc
;---------------------------------
```

```
TITLE Макрокоманда зсуву SHIFT@
SHIFT@ MACRO var1, var2
        lea      di,var1
        lea      si,var2
        mov      cx,2
rep     movsw
        ENDM
;----------------------------------
EXTRN  addf4:proc, mulf4:proc  ; Зовнішні п/п
;----------------------------------
SSEG SEGMENT PARA STACK 'STACK'
        db      256 dup (0)
SSEG ENDS
;----------------------------------
DSEG  SEGMENT  PARA  PUBLIC  'DATA'
        xnt      dd 256 dup (0) ; Вхідний сигнал
        ynt     dd 256 dup (0) ; Вихідний сигнал
        a0      dd 0. 067456 ; Fд=1000 Гц  //1 dup (0)
        a1      dd 0. 134913 ; Fв=100 Гц   //1 dup (0)
        a2      dd 0. 067456 ; Fн=0 Гц     //1 dup (0)
        b1      dd 1. 14299  ; ФНЧ         //1 dup (0)
        b2      dd -0. 412825 ;Баттерворда //1 dup (0)
        y1       dd 1 dup (0)
        y2       dd 1 dup (0)
        y3      dd 1 dup (0)
        y4      dd 1 dup (0)
        y5      dd 1 dup (0)
        count   dw 256         ; Лічильник циклів
        index   dw 1 dup (0)   ; Індекс у масиві
        in_file db 'signal. in',0
        outfile db 'signal. out',0
        handle  dw ?
;       filsize dw ?
DSEG ENDS
;----------------------------------
CSEG SEGMENT PARA PUBLIC 'CODE'
        ASSUME CS:CSEG, DS:DSEG, ES:DSEG, SS: SSEG
;----------------------------------
INSIGN PROC    FAR ; Читання сигналу з файла
;--- Відчиняємо файл
        lea     dx,in_file
        mov     al,0
        mov     ah,3d
        int     21H
        mov     handle,ax
```

```
;--- Читаємо файл
        mov     ah,3f
        mov     bx,handle
        mov     cx,1024
        lea     dx,xnt
        int     21H
;---  Закриваємо  файл
        mov      bx,handle
        mov     ah,3e
        int     21H
        ret
INSIGN ENDP
;---------------------------------
OUTSIGN PROC    FAR ; Запис сигналу у файл
;--- Створимо файл
        lea     dx,outfile
        mov     cx,0
        mov     ah,3c
        int     21H
        mov     handle,ax
;--- Записуємо у файл вихідний сигнал
        mov     ah,40H
        mov     bx,handle
        mov     cx,1024
        lea     dx,ynt
        int     21H
;---  Закриваємо  файл
        mov      bx,handle
        mov     ah,3e
        int     21H
        ret
OUTSIGN ENDP
;---------------------------------
;----- Основна програма ----------
LINK2   PROC    FAR      ; Головна підпрограма
        mov     ax,DSEG
        mov     ds,ax
        mov     es,ax
        call    INSIGN ; П/п читання вхідного сигналу з диска
        mov     cx,count
start:  push    cx
        SHIFT@ y3, y2 ; y3 = y2
        SHIFT@ y2, y1 ; y2 = y1

        lea     di,y1    ;          ⎫
```

```
        lea     si,xnt  ;       ⎫
        add     si,index ;      ⎬ y1 = x(n)
        mov     cx,2    ;       ⎭
rep     movsw           ;       ⎭

        SHIFT@ y5, y3 ; y5 = y3
        lea     si,y5
        lea     di,b2
        call    mulf4  ; y5 = y5*b2
        lea     si,y1
        lea     di,y5
        call    addf4  ; y1 = y1+y5
        SHIFT@ y5, y2
        lea     si,y5
        lea     di,b1
        call    mulf4  ; y5 = y5*b1
        lea     si,y1
        lea     di,y5
        call    addf4  ; y1 = y1+y5
;-----------------------------------
        SHIFT@ y5, y1
        lea     si,y5
        lea     di,a0
        call    mulf4  ; y5 = y5*a0
        SHIFT@ y4, y5  ; y4 = y5
        SHIFT@ y5, y2
        lea     si,y5
        lea     di,a1
        call    mulf4  ; y5 = y5*a1
        lea     si,y4
        lea     di,y5
        call    addf4  ; y4 = y4+y5
        SHIFT@ y5, y3
        lea     si,y5
        lea     di,a2
        call    mulf4  ; y5 = y5*a2
        lea     si,y4
        lea     di,y5
        call    addf4  ; y4 = y4+y5


        lea     di,ynt  ;       ⎫
        lea     si,y4   ;       ⎬
        add     di,index ;      ⎭ y(n) = y4
```

```
        mov     cx,2    ;       |
rep     movsw           ;       |
        add     index,4
        pop     cx
        dec     cx
        jz      @end
        jmp     start
@end:   call    OUTSIGN ; П/п запису вихідного сигналу на диск
        mov     ah,4c ; Вихід із програми у відповідності
        int     21H         ; вимогам DOS.  Стандартне переривання DOS.
LINK2   ENDP
;---------------------------------
CSEG    ENDS
        END     LINK2
```

У програмі цифрового фільтра використовуються зовнішні підпрограми додавання і множення чисел у форматі з плаваючою комою (addf4, mulf4), розташовані в зовнішній бібліотеці float.lib.

Домашнє завдання

1. Вивчіть опис програми GEN.EXE генератора стандартних сигналів (дод. 7).
2. Намалюйте в протоколі схему алгоритму програми 1.
3. Вивчіть систему команд мікропроцесора 1810ВМ86 (дод. 8).
4. Вивчіть стислий опис програми TURBO ASSEMBLER (дод. 1).
5. Вивчіть стислий опис налагоджувача TURBO DEBUGGER (дод. 2).
6. Вивчіть стислий опис програми NORTON COMMANDER (дод. 6).
7. Використовуючи програму 1, напишіть програму не рекурсивного цифрового фільтра 2-го порядку.

Порядок виконання роботи

1. Скопіюйте файл filtr2.asm у файл під ім'ям st???.asm (??? – останні три цифри номера залікової книжки).
2. В отриманому тексті програми, в сегменті даних замініть рядок signal.in на st???.in, рядок signal.out на st???.out. Замість заданих у програмі коефіцієнтів a0, a1, a2, b1, b2 занесіть значення з табл. 6.3 для фільтра НЧ. Апроксимація частотної характеристики задається викладачем.
3. Проведіть асемблювання програми. Наберіть у командному рядку tasm /ZI /L st???.asm (не забудьте, що замість ??? треба набирати цифри).
4. Якщо помилок немає (з'явився файл із розширенням *.obj), то переходіть до п. 3.5 виконання роботи. При наявності помилок перегляньте файл із розширенням *.lst (F3), у якому вони відзначені, виправіть їх у файлі з розширенням *.asm (F4) і повторіть п.п. 2 і 3 виконання роботи.

Таблиця 6.3

Фільтр Баттерворда $\frac{f_д}{f_{ср}} = 10$					
	a0	a1	a2	b1	b2
ФНЧ	0.067456	0.134913	0.067456	-1.14299	0.41283
ФВЧ	0.638956	-1.27791	0.638956	-1.14299	0.41283
Фільтр Чебишева $\frac{f_д}{f_{ср}} = 10$					
	a0	a1	a2	b1	b2
ФНЧ	0.50754	0.101507	0.50754	-1.39019	0.59321
ФВЧ	0.630169	-1.26033	0.630169	-1.0539	0.46677

5. Скомпонуйте завантажувальний файл, набрав у командному рядку tlink /v st???,,,float.lib (тут теж замість ??? треба набирати потрібні цифри). Проконтролюйте створення файла st???.exe.

6. Використовуючи програму GEN.EXE створіть файл, що містить відліки прямокутного відеоімпульса з T=0.01 с, τ=0.003 с, U=1 В, і відліки випадкового процесу з σ^2=0.5, $f_Д$=2500 Гц.

7. Проведіть фільтрацію отриманого сигналу, набравши у командному рядку st???.exe.

Переконайтеся, що отримали вихідний файл st???.out.

8. Перегляньте і замалюйте в протоколі форму вхідного і вихідного сигналів, набрав у командному рядку grapher st???.in st???.out.

9. Повторюйте п.п.3.6-3.8 для наступних параметрів сигналу:

T=0.05 с	τ=0.025 с	σ^2=0	$f_Д$=2500 Гц
T=0.05 с	τ=0.01 с	σ^2=0	$f_Д$=2500 Гц
T=0.004 с	τ=0.002 с	σ^2=0	$f_Д$=2500 Гц

10. Повторюйте п.п. 2 – 8 для ФВЧ.

11. Для заданого викладачем типу фільтра побудуйте амплітудно-частотну характеристику (АЧХ). Створіть гармонійний сигнал і пропускайте його через фільтр. Запишіть максимальне значення вихідного сигналу.

12. Пункт 3.11 повторюйте 12 –15 разів для значень частот у діапазоні від 0 до $3f_{ср}$.

13. Отримані дані зведіть в таблицю, побудуйте графік АЧХ досліджуваного фільтра.

13. Оформіть протокол і зробіть висновки.

Контрольні питання

1. Перерахуйте всі вмонтовані функції програми GEN.EXE.

2. Зобразіть ідеальну АЧХ ФНЧ, ФВЧ, смугового фільтра, фільтра, що загороджує.

3. Визначіть мінімальну частоту дискретизації, якщо відома частота зрізу фільтра.

4. Визначіть частоту зрізу ФНЧ, щоб він пропускав перші три гармоніки прямокутного відеоімпульса з параметрами T=0.025 с, τ=0.0025 с, U= 12.6 В. Перші чотири гармоніки.

5. Визначіть частоту зрізу ФВЧ, щоб він не пропускав перші п'ять гармонік пилообразного відеоімпульсу з параметрами T=0.025 с, U= 0.1586 В. Перші десять гармонік.

6. Визначіть частоту зрізу ФНЧ, щоб він придушував гармоніки прямокутного відеоімпульса з параметрами T=0.33 с, τ=0.025 с, U= 6.3 В вище шостої. Вище першої гармоніки.

ЛАБОРАТОРНА РОБОТА 7

ДОСЛІДЖЕННЯ АЛГОРИТМІВ ГЕНЕРАЦІЇ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ

Мета роботи: ознайомлення із принципами програмування на мові Асемблер ОЕОМ 1816ВЕ51 із використанням налагоджених засобів на прикладі алгоритмів генерації псевдовипадкових чисел.

Стислі теоретичні відомості й приклад програми

Поява обчислювальних машин дозволило багато натурних експериментів (які потребують великих матеріальних і часових витрат) замінити моделюванням на ЕОМ. При проведенні моделювання фізичних процесів вимагаються ефективні (швидкі) методи генерації випадкових чисел. Програми, що виробляють випадкові числа, називають давачами.

Найкращі з відомих сьогодні давачів випадкових чисел являють собою окремі випадки наступної схеми запропонованої в 1948 році Дж. Лемером.

Вибираємо чотири числа:

X_0 – початкове значення ($X_0 >= 0$);

a – множник ($a >= 0$);

c – збільшення ($c >= 0$);

m – модуль ($m > X_0, m > a, m > c$).

Тоді шукана послідовність випадкових чисел <Xn> утворюється зі співвідношення

$$X_{n+1} = (a X_n + c) \bmod m, \quad n > 0.$$

Таку послідовність називають лінійною конгруентною послідовністю. Метод викликає досить очевидне заперечення. Як може бути випадковою вироблена таким способом послідовність, якщо кожний її член цілком визначений своїм попередником? Відповідь полягає в тому, що ця послідовність не випадкова, але виглядає як випадкова. Звичайно немає значення, як пов'язані один з одним два наступні числа послідовності. У науково-технічній літературі послі-

довності, вироблені детермінированим способом, називають псевдовипадковими або квазівипадковими.

Наприклад, при $X_0 = a = c = 7$, $m = 10$ послідовність виглядає так:

7, 6, 9, 0, 7, 6, 9, 0, ...

Як видно з даного приклада, послідовність не завжди виявляється "випадковою", якщо вибрати X_n, a, c, m довільно. Приведений приклад ілюструє той факт, що конгруентні послідовності завжди "зациклюваються", тобто зрештою, числа утворять цикл, що повторюється безкінечне число разів. Повторюваний цикл називають періодом. Довжина періоду в цьому прикладі дорівнює 4. У даному методі довжина періоду залежить від усіх параметрів датчика.

Вибір модуля

Значення модуля m повинна бути достатньо великою, щоб забезпечити можливо більший період послідовності, тому що максимальна довжина періоду не може бути більше за m.

Другий чинник що впливає на вибір m, це швидкість виробітки чисел: m варто вибрати таким, щоб ефективно (із мінімальним числом дій) обчисляти

$$(a X_n + c) \bmod m.$$

Для обчислення y mod m необхідно значення y розділити на m і в якості результату взяти залишок від ділення. Але тому що ділення – порівняно повільна операція, її можна уникнути за рахунок зручного вибору m, прийнявши його рівним розміру оброблюваного слова. Максимальне ціле число, що можна записати в n-розрядне двійкове слово складає 2^n-1. Для правильного вибору множника a потрібно, щоб m було простим числом, тобто ділилося тільки саме на себе і не розкладалося на прості множники. У табл. 7.1 приведені значення розкладання 2^n-1 на прості множники .

Таблиця 7.1

Розрядність n	Максимальне число 2^n-1	Розкладання на прості множники
6	63	3x3x7
7	127	-
8	255	3x5x17
9	511	7x73
10	1023	3x11x31
11	2047	23x89
12	4095	3x3x5x7x13
13	8191	-
14	16383	3x43x127
15	37367	7x31x151
16	65535	3x5x17x257
17	131071	-
18	261143	3x3x7x17x73
19	524287	-
. . .	. . .	. . .
31	2147483647	-

Як видно з табл. 7.1, при n < 32 усього 5 чисел є простими. У табл. 7.2 зазначене необхідне число байтів для збереження цих простих чисел.

Таблиця 7.2

Розрядність n	Максимальне число 2^n-1	Число байтів для збереження числа
7	127	1
13	8191	2
17	131071	3
19	524287	3
31	2147483647	4

При роботі з однокристальною ЕОМ необхідно вибирати невелике число n = 13, тому що при цьому для збереження кожного числа буде потрібно всього 2 байта пам'яті, а це значно спростить обробку чисел й упорядкування програми генерації "псевдовипадкових" чисел. При меншому n=7 утворюється занадто малий період генератора, а при великих n=17, 19 або 31 значно ускладнюється програма.

Число n=13 забезпечує довжину періоду "псевдовипадкових" чисел рівною 8191. Далі буде показано, як збільшити довжину періоду.

Вибір множника

Розглянемо, як слід вибирати множник a, щоб одержати період максимальної довжини. Для будь-якої послідовності, призначеної для використання в якості джерела випадкових чисел, важливий великий період, що містить достатнє число випадкових чисел. Проте, варто пам'ятати, що великий період – це тільки один із необхідних ознак випадку послідовності. Цілком можливі абсолютно невипадкові послідовності з дуже великим періодом. Наприклад, при a=c=1 послідовність чисел, що генеруються, зводиться просто до $X_{n+1}=(X_n+1)\text{mod}m$. Без сумніву, її період дорівнює m, проте її ніяк не можна назвати випадковою. Коли період послідовності має довжину m, кожне число від 0 до (m-1) зустрічається за період рівно один раз. Тому вибір початкового значення X_0 не впливає на довжину періоду. Для забезпечення лінійної конгруентної послідовності максимальної довжини необхідно, щоб:

– a, c і m , були взаємно простими числами;

– якщо m являє собою степінь двійки (тобто процесор працює в двійковій системі числення), то вибираємо множник a таким, щоб a mod 8 = 5;

– множник a повинний перевищувати величину $\sqrt{m}$, бажано, щоб він був більше m/100, але менше $m - \sqrt{m}$.

У табл. 7.3 приведені значення 10 множників, які задовольняють приведеним вище критеріям, що накладаються на вибір множника a для лінійної конгруентної послідовності при m = 8191.

Таблиця 7.3

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
101	107	131	157	163	173	179	197	211	227

Вибір збільшення

Розглянемо тепер, як вибрати збільшення c, щоб одержати максимальний період послідовності й забезпечити невеликий коефіцієнт послідовної кореляції. Вище вже було відзначено, що c і m повинні бути взаємно простими числами. Для забезпечення невеликих значень послідовної кореляції при виробітку псевдовипадкових чисел відношення c/m повинно відповідати кореням рівняння $1-6x+6x^2=0$. При рішенні цього рівняння одержуємо x=c/m = 0.2113248654.

У табл. 7.4 приведені значення збільшень c, які задовольняють приведеним критеріям з похибкою не більш 1%.

Таблиця 7.4

Збільшення	Значення	Погрішність %
c_1	1721	-0.57
c_2	1723	-0.46
c_3	1733	+0.12
c_4	1741	+0.58
c_5	1747	+0.93

Збільшення довжини періоду

Розглянемо методи, що дозволяють збільшити довжину періоду вироблюваних псевдовипадкових чисел. Якщо параметри генератора лінійної конгруентної послідовності вибирати з приведеними вище критеріями, то кожний набір параметрів a, c і m дозволяє одержати максимальний період послідовності, що генерується. Причому при кожному наборі це буде незалежна послідовність. У аналізованих вище прикладах генерували псевдовипадкові числа від 0 до m. Розділивши, при необхідності їх на m, одержимо числа в діапазоні від 0 до 1.

Використання декількох незалежних генераторів дозволяє одержати випадкову послідовність, довжина періоду якої дорівнює добутку періодів використовуваних генераторів. Для цього використовують алгоритм із перемішуванням:

Крок 1. Використовуючи параметри першого генератора a_1, c_1 и m_1, сгенеруємо k випадкових чисел X_j і заповнити ними робочий масив S розмірністю k. Значення k може бути довільним і визначається величиною доступної пам'яті ОЕОМ і зручністю обчислень – звичайно 8-20 елементів. Приймемо k=8.

Крок 2. Використовуючи параметри другого генератора a_2, c_2 и m_2, сгенеруємо випадкове число W_j, необхідне для обчислення поточного індексу L у робочому масиві S, для чого умножити випадкове число W_j на k.

Крок 3. Використовуючи поточний індекс L у робочому масиві S, добуваємо випадкове число $Z=S_L$ і приймаємо його в якості вихідного.

Крок 4. Використовуючи параметри першого генератора a_1, c_1 и m_1 , сгенеруємо нове випадкове число X_j і замінюємо ним обране з робочого масиву S на 3-му крокці число за індексом L.

Крок 5. При необхідності генерації нових псевдовипадкових чисел повертаємось назад на крок 2 алгоритму.

Використання двох незалежних генераторів з періодами $m_1 = m_2 = 8191$ дозволяє отримати випадкову послідовність з періодом рівним $m_1 \times m_2 = 67092481$, що є цілком прийнятним для більшості застосувань ОЕОМ. При необхідності довжину періоду можна додатково збільшити в m раз, використовуючи замість одновимірного масиву S – двомірний, замінивши крок 2 алгоритму.

Приклад програми

Програма 7.1

```
;*********************************************************
;Програма генерації ПВЧ лінійним конгруентним методом
;       із використанням перемішування
;*********************************************************
AJMP   START ;Передати керування на початок програми
;       Встановити початкову адресу програми
       ORG    030H
;
START:      ACALL  S1 ;ВИКОНАТИ настроювання параметрів
;
;       Підготувати бункер на 8 ПВЧ
;
      MOV   PSW,#10000B      ;Встановити банк регістрів - 2
      MOV   R7,#8            ;розмір бункера 8 слів
L1:    ACALL      GEN        ;згенерувати ПВЧ1
      MOV   A,R7             ;підготувати номер до запису
      ACALL      M1          ;записати ПВЧ у бункер
      DJNZ   R7,L1;продовжити до заповнення бункера
;       підготувати масив частот
      MOV       R7,#8        ;розмір буфера 8 байтів
      MOV       R0,#70H ;початок буфера
      MOV       A,R7         ;підготувати номер до запису
L01:  MOV       @R0,#10H     ;занести початкове значення INC
                R0 ;підготувати адресу для наступного запису
      DJNZ      R7,L01       ;продовжити до заповнення буфера
;
;      згенерувати 64 ПВЧ і записати в РПД з адреси 30H
;
YES: MOV      PSW,#11000B ;встановити банк регістрів – 3
      ACALL    GEN64        ;визначити ПВЧ2 (індекс)
      MOV      PSW,#0       ;встановити банк регістрів – 0
      ACALL    STAT1   ;підрахувати частоти влучення в інтервали
      ACALL    STAT2   ;відняти порівн. значення з частот улучення
      AJMP     YES     ;повторити генерацію
```

```
;
; п/п генерації 64 ПВЧ
;
GEN64:      MOV      R7,#64   ;установити лічильник циклів
GG1: ACALL      GEN  ;визначити ПВЧ2 (індекс)
      ACALL      M2    ;вибрати з бункера і записати в РПД
      MOV   PSW,#10000B      ;встановити банк регістрів – 2
      ACALL      GEN  ;згенерувати нове ПВЧ1
      MOV   A,1EH       ;індекс помістити в акумулятор з R6
      ACALL      M01   ;замінити в бункері число
      MOV   PSW,#11000B      ;відновити банк регістрів – 3
      DJNZ   R7,GG1      ;продовжити цикл, якщо не кінець
      RET    ;повернутися з підпрограми
;
; підрахувати частоти влучення в інтервали
;
STAT1:      MOV R7,#64          ;встановити лічильник циклів
   MOV       R1,#30H   ;встановити початкову адресу буфера ПВЧ
MOV R6,#70H   ;встановити початкову адресу буфера частот
ST1:  MOV A,@R1       ;прочитати ПВЧ із буфера
      ANL    A,#1CH      ;виділити розряди 2-4
      RR     A      ;зрушити циклічно вправо
      RR     A      ;зсунути циклічно вправо
      ADD    A,R6  ;обчислити адресу інтервалу
      MOV    R0,A  ;переслати в R0
      INC    @R0  ;збільшити лічильник частоти влучення
      INC    R1     ;перемістити покажчик на наступне ПВЧ
      DJNZ   R7,ST1              ;продовжити цикл, якщо не кінець
      RET    ;повернутися з підпрограми
;
; відняти середні значення з частоти влучення в інтервали
;
STAT2:      MOV R7,#8 ;встановити лічильник циклів
        MOVR0,#70H    ;встановити початкову адресу буфера ;частот
ST2:  MOV A,@R0       ;прочитати число з буфера
      SUBB  A,#8  ;відняти порівн. значення
      MOV   @R0,A       ;переслати в буфер
      INC    R0     ;перейти до наступного числа
      DJNZ   R7,ST2       ;продовжити цикл, якщо не кінець
      RET            ;повернутися з підпрограми
;
; п/п генерації ПВЧ
;       R2,R1         - поточне ПВЧ (R1-молодший байт,R2-старший байт)
;       R3     - множник
;       R5,R4- зсув (R4-молодший байт,R5-старший байт)
```

```
;
GEN: MOV A,R3 ;переслати множник в акумулятор
     MOV  B,R1  ;молодший байт у додатковий акк.
     MUL  AB    ;помножити
     MOV  R1,A  ;молодший байт зберегти в R1
     MOV  R6,B  ;старший байт зберегти в R6
     MOV  A,R3  ;відновити множитель
     MOV  B,R2  ;старший байт у додатковий акк.
     MUL  AB    ;умножити
     ADD  A,R6  ;молодший байт скласти зі старшим від
     MOV  R2,A  ;попереднього множення і запам'ятати в R2
;      додати зсув
     MOV  A,R1  ;;
     ADD  A,R4  ;; скласти молодші байти
     MOV  R1,A  ;;
     MOV  A,R2  ;;
     ADDC A,R5  ;; скласти старші з урахуванням переносу
     ANL  A,#1FH;виділити 5 молодших розрядів (узяти модуль)
     MOV  R2,A  ;зберегти старше слово
     RET        ;повернутися з підпрограми
;
;      п/п запису в бункер за адресами 20H-2FH
;
M1:  DEC  A     ;зменшити на одиницю
       RL  A     ;помножити на 2
;      додаткова точка входу в підпрограму
M01: ADD  A,#21H ;додати адресу початку буфера
     MOV  R0,A  ;переслати в R0 для непрямої адресації
     MOV  A,R2  ;взяти старший байт ПВЧ
     MOV  @R0,A      ;записати в бункер
     DEC  R0    ;зменшити поточну адресу
     MOV  A,R1  ;взяти молодший байт ПВЧ
     MOV  @R0,A      ;записати в бункер
     RET        ;повернутися з підпрограми
;
;      п/п заміни ПВЧ у бункері
;
M2:  MOV A,R2  ;переслати старший байт ПВЧ -2 в акк.
     RR   A    ;розділити на 2
     ANL  A,#0EH ;виділити молодші разряды
     MOV  R6,A  ;зберегти в регістрі R6 3-го банку
     ADD  A,#21H ;додати адресу початку бункера
     MOV  R0,A  ;переслати в R0 для непрямої адресації
     MOV  A,@R0      ;витягти байт із бункера
     MOV  2H,A  ;помістити в R2 0-го банку регістрів
```

```
      DEC   R0            ;зменшити поточний покажчик
      MOV   A,@R0         ;витягти байт із бункера
      MOV   1H,A  ;помістити в R1 0-го банку регістрів
      MOV   A,#70H ;занести в акумулятор адреса кінця масиву
      CLR   C       ;скинути прапор
      SUBB  A,R7  ;обчислити поточну адресу для запису ПВЧ
      MOV   R0,A  ;помістити в R0 для непрямої адресації
      MOV   A,2H ;взяти старший байт ПВЧ
      MOV   @R0,A         ;записати його в ділянку виводу в РПД
      RET             ;повернутися з підпрограми
;
;       п/п початкового настроювання генераторів
;
S1:   MOV DPTR,#G1 ;завантажити покажчик на 1-й генератор
      CLR   A           ;очистити акумулятор
      MOVC A,@A+DPTR       ;переслати байт із ПП в акумулятор
      MOV   13H,A         ;зберегти в R3 2-го банку регістрів
      INC   DPTR;перемістити покажчик
      CLR   A       ;очистити акумулятор
      MOVC A,@A+DPTR       ;переслати байт із ПП в акумулятор
      MOV   15H,A         ;зберегти в R5 2-го банку регістрів
      MOV   A,#1  ;занести зсув
      MOVC A,@A+DPTR ;переслати байт із ПП в акумулятор
      MOV   14H,A         ;зберегти в R4 2-го банку регістрів
      MOV   A,#2  ; занести зсув
      MOVC A,@A+DPTR ;переслати байт із ПП в акумулятор
      MOV   1BH,A         ;зберегти в R3 3-го банку регістрів
      MOV   A,#3  ;занести зсув
      MOVC A,@A+DPTR ;переслати байт із ПП в акумулятор
      MOV   1DH,A         ;зберегти в R5 3-го банку регістрів
      MOV   A,#4  ;занести зсув
      MOVC A,@A+DPTR ;переслати байт із ПП в акумулятор
      MOV   1CH,A         ;зберегти в R4 3-го банку регістрів
      RET             ;повернутися з підпрограми настроювання
; параметри 1 - го генератора (основного)
G1:   DB    101   ; множник
      DW    1733  ; зсув
; параметри 2 - го генератора (допоміжного)
G2:   DB    163   ; множник
      DW    1721  ; зсув
      END
```

Програма складається з основної частини від мітки START до мітки GEN64 і ряду допоміжних підпрограм.

На початку основної частини виконується настроювання параметрів генераторів – викликається п/п S1 (параметри першого генератора позначені міткою G1, він використовується для формування ПВЧ послідовності, а другого – міткою G2, він використовується для обчислення адреси в "бункері" перемішування ПВЧ).

Потім виконується підготовка «бункера» для перемішування ПВЧ, при цьому використовуються п/п GEN – для генерації ПВЧ і M1 – для запису ПВЧ у «бункер».

На наступному етапі провадиться підготовка масиву частот влучень в інтервали – адреса початку масиву 70H, довжина масиву 8 байт, у кожний елемент масиву записується початкове значення 10H.

Міткою YES позначений фрагмент програми, що виконує генерацію 64 псевдовипадкових чисел – п/п GEN64 генерує 64 ПВЧ і записує їх у буфер по адресах 30H – 6FH, а потім провадиться їхня обробка – п/п STAT1 робить підрахунок частот влучень ПВЧ у 8 інтервалів, а п/п STAT2 віднімає середнє значення з отриманих частот, потім провадиться повернення на мітку YES для повторної генерації наступних 64 ПВЧ.

Домашнє завдання

1. Вивчіть систему команд однокристальної ЕОМ 1816ВЕ51 (дод. 9).
2. Вивчіть стислий опис Кросс-Ассемблера X8051 (дод. 3).
3. Вивчіть стислий опис компановщика LINK (дод. 4).
4. Вивчіть стислий опис емулятора FD51 (дод. 5).
5. Вивчіть стислий опис програми NORTON COMMANDER (дод. 6).
6. Намалюйте в протоколі схему алгоритму програми генерації ПВЧ конгруентним методом.
7. Обчисліть параметри генератора для m = 127.
8. Намалюйте в протоколі схеми алгоритму програми 7.1.

Порядок виконання роботи

1. Використовуючи NORTON COMMANDER, ввійдіть у підкаталог 1816 каталога LAB_RAB для однієї з панелей керування.
2. На сусідній панелі керування ввійдіть у підкаталог SOURCE каталога 1816.
3. Скопіюйте файл LR5.a51 із підкаталога SOURCE у каталог 1816, привласнивши йому ім'я st???.asm (де ??? три останні цифри номера залікової книжки студента).
4. Використовуючи редактор тексту NORTON COMANDER відредагуйте файл тексту програми st???.asm, замінивши в ньому параметри опису генераторів відповідно до таблиць 7.5 і 7.6 для множників і збільшень за трьома останніми цифрами залікової книжкиі I-J-K (де I – розряд сотень, J – розряд десятків, K – розряд одиниць).
5. Виконайте трансляцію вихідного тексту програми, ввівши команду X8051 у командний рядок NORTON COMANDER.

Приклад діалогу при трансляції програми з ім'ям ST000.ASM приведений нижче.

```
>x8051.exe
   8051 Macro Assembler   -   Version 4.02a
   Copyright (C) 1985 by 2500 A.D. Software, Inc.
Listing Destination  (N, T, D, E, L, P, <CR> = N) : d
Generate Cross Reference ? (Y/N  <CR> = No) :
Input  Filename : st000
Output Filename : <CR>
```

У процесі трансляції програми створюється файл із розширенням *.obj, що містить об'єктний код програми, і файл із розширенням *.lst, який містить лістинг програми.

Якщо при трансляції програми будуть видані повідомлення про помилки, то усуньте помилки у вихідному тексті своєї програми. Перелік помилок й їхні можливі причини наведені у дод. 3.

Таблиця 7.5

Основний генератор		Допоміжний генератор	
K	Множник	(I+K+5) mod 10	Множник
0	a_0	0	a_0
1	a_1	1	a_1
2	a_2	2	a_2
3	a_3	3	a_3
4	a_4	4	a_4
5	a_5	5	a_5
6	a_6	6	a_6
7	a_7	7	a_7
8	a_8	8	a_8
9	a_9	9	a_9

Таблиця 7.6

Основний генератор		Допоміжний генератор	
J	Збільшення	(I+J+3) mod 10	Збільшення
0	c_1	0	c_1
1	c_2	1	c_2
2	c_3	2	c_3
3	c_4	3	c_4
4	c_5	4	c_5
5	c_1	5	c_1
6	c_2	6	c_2
7	c_3	7	c_3
8	c_4	8	c_4
9	c_5	9	c_5

6. Виконайте побудову виконавчого коду програми, ввівши команду link у командний рядок NORTON COMANDER.

Приклад діалогу при побудові виконавчого коду програми з ім'ям ST000.OBJ приведений нижче:

```
>link

2500 A.D. Linker  Copyright (C) 1985  -  Version 4.02d
Input   Filename : st000
Enter Offset For 'CODE'              : 0

Input   Filename :
Output  Filename :
Library Filename :

Options (D, S, A, M, Z, X, H, E, T, 1, 2, 3,  <CR>= Default) : H
```

У результаті створюється файл із розширенням *.hex, що містить виконавчий код програми.

7. Перегляньте файл st???.lst і заповніть табл. 7.7.

Таблиця 7.7

Ім'я мітки	Адреса мітки (HEX код)

8. Запустіть програму моделювання роботи однокристальной ЕОМ увівши команду fd51 у командний рядок NORTON COMANDER.

Опис роботи програми-емулятора приведено в дод. 5.

9. Завантажте програми в пам'ять програм ЕОМ набравши таку команду: L st???.hex

10. Встановіть точку зупину на адресу мітки "YES", для цього натисніть клавішу F5 і викличте вікно "Breakpiont". У колонку PC встановіть адресу мітки "YES", у колонку "Counter" – 2 (число проходів програми). Поверніться в основне вікно, натиснувши клавішу F5.

Запустіть програму на виконання, ввівши команду G.

11. Після виконання програми перегляньте область пам'яті даних, використовуючи клавіші F7 або F9, і запишіть отримані числа, розташовані в пам'яті даних за адресою з 30H по 6FH. Для них визначіть математичне чекання і дисперсію.

Запишіть отримані числа, розташовані в пам'яті даних за адресою з 70H по 77H, за отриманими даними побудуйте гістограми.

Повторіть генерацію ПВЧ і підрахунок частот для 128, 512 і 1024 чисел і запишіть отримані частоти влучення в інтервали, за отриманими даними побудуйте гістограми.

Контрольні питання

1. Як виконують генерацію ПВЧ конгруентним методом.
2. Як вибирають період генератора для конгруентного методу.
3. Як вибирають множник при використанні конгруентного методу.
4. Як вибирають збільшення при використанні конгруентного методу.
5. Як збільшити період генерації ПВЧ.
6. В яких регістрах розташовані поточні ПВЧ, множник і збільшення для основного генератора.
7. В яких регістрах розташовані поточні ПВЧ, множник і збільшення для допоміжного генератора.
8. За якими адресами розташований бункер для перемішування ПВЧ.
9. За якими адресами розташований буфер для виводу ПВЧ що генеруються програмою.
10. За якими адресами розташований буфер для підрахунку частот влучень в інтервали.
11. Як підрахувати матсподівання, дисперсію і середнє квадратичне відхилення випадкових розмірів.
12. Які параметри (математичне сподівання, дисперсію і середнє квадратичне відхилення) має випадковий розмір з рівномірним законом розподілу на інтервалі від 0 до 1 і від 0 до 31.
13. Де в програмі розташовані числа, що задають значення множника і збільшення для основного й допоміжного генераторів.
14. Які дії необхідно виконати перед запуском програми на виконання.
15. Як виконати програму за кроками (за процедурами).
16. Як переглянути область даних ОЕОМ.

ДОДАТОК 1

СТИСЛИЙ ОПИС ПРОГРАМИ TASM (TURBO ASSEMBLER)

Загальні відомості

Турбо Асемблер фірми Borland являє собою багатопрохідний Асемблер з дозволом посилань, що випереджають, швидкістю асемблювання до 48000 рядків у хвилину, сумісний з макроасемблером фірми Microsoft MASM, і додатковою можливістю використання режиму розширеного синтаксису.

У Турбо Асемблері здійснюється підтримка таких функцій:

– повна підтримка процесора 80386;
– поліпшена синтаксична перевірка типів;
– спрощені директиви визначення сегментів;
– поліпшене управління лістингом;
– розширення інструкцій POP і PUSH;
– розширений оператор CALL з аргументами і необов'язковим параметром мови;
– локальні мітки;
– локальні ідентифікатори в стеку й аргументи виклику в процедурах;
– структури й об'єднання;
– вкладені директиви;
– режим QUIK, що емулює MASM
– повне налагодження на рівні вихідного тексту за допомогою Турбо налагоджувача;
– умонтована утиліта генерації перехресних посилань (TCREF);
– файли конфігурації й командні файли.

Турбо Асемблер є потужним Асемблером, що працює з командним рядком, який сприймає ваші вихідні файли (файли з розширенням *.ASM) і створює з них об'єктні модулі (файли з розширенням *.OBJ). Після цього можна використовувати програму – укладач фірми Borland TLINK. EXE, що відрізняється високою швидкістю копонування, для компонування отриманих об'єктних модулів і створення виконуваних файлів (файлів з розширенням *.EXE).

Турбо Асемблер створений для роботи з процесорами серії 80x86 і 80x87 (більш докладно набір інструкцій процесорів серії 80x86/80x87 описаний у відповідних посібниках фірми Intel).

Турбо Асемблер генерує інструкції процесорів 8086, 80186, 80286 і 80386, а також інструкції з плаваючою капкою для арифметичних сопроцесорів 8087, 80287 і 80387.

Запуск Турбо Асемблера з DOS

У Турбо Асемблері є дуже потужний і гнучкий синтаксис командного рядка. Якщо запустите Турбо Асемблер, не задавши ніяких аргументів, наприклад:

TASM,

то на екран виводиться довідкова інформація (англійською мовою), що описує множину параметрів командного рядка й синтаксис для специфікації асембльованих файлів. Нижче показано, як вона виглядає.

```
---------------------------------------------------------------
Turbo Assembler Version 2.0 Copyright (C) 1990
by Borland International, Inc
Usage:
TASM [параметри] іcх_файл [,об'єкт_файл] [,лістинг] [,ссилки]
/a,/s            Порядкування сегментів за алфавітним порядком або порядком вихідного коду
/c               Генерація в лістинзі перехресних посилань
/dSYM[=VAL]      Визначення SYM = 0 або SYM = VAL
/e,/r            Інструкції, які емулюються, або дійсні інструкції з плаваючою крапкою
/h,/?            Виведення довідкової інформації
/lPATH           Включення файлів, що шукаються за маршрутм, обумовленим PATH
/jCMD            Визначення початкової директиви Асемблера (наприклад, jIDEAL)
/kh#,/ks#        Потужність хеш-таблиці #, потужність обсягу рядка #
```

/l,/la	Генерація лістингу: l – звичайний лістинг, la – розширений
/ml,/mx,/mu	Розрізненість у регістрі букв ідентифікаторів: ml – усе, mx – глобальні, mu – не різняться
/mv#	Завдання максимальної довжини ідентифікаторів
/m#	Дозволяє виконання декількох проходів для задоволення посилань, що випереджають
/n	Придушення в лістингах таблиці символів (ідентифікаторів)
/p	Перевірка перекриття сегмента коду в захищеному режимі
/q	Придушення записів *.OBJ, що не будуть потрібні при компонуванні
/t	Придушення повідомлень при успішному асемблюванні
/w0,/w1,/w2	Завдання рівня попередження: w0 – немає попереджень, w1=w2 – є попередження
/w-xxx,/w+xxx	Заборона або дозвіл попередження типу xxx
/x	Вмикання в лістинги блоків умовного асемблюванні
/zi,/zd	Інформація про ідентифікатори для налагодження: zi – повна, zd – тільки про номери рядків

За допомогою параметрів командного рядка можна задавати ім'я одного або декількох файлів, а також параметри, що керують їх асемблюванням.

Загальний вид командного рядка такий:

TASM файли [; файли]...

Крапка з комою після лівої квадратної дужки дозволяє в одному командному рядку асемблювати декілька груп файлів. За бажанням можна задати для кожної групи файлів різноманітні параметри, наприклад:

TASM /E FILE1; /A FILE2.

У загальному випадку група файлів у командному рядку може мати вид:

[параметр]... іcх_файл [[+] вихідний_файл]...
[,[об'єктний_файл] [, [файл_лістингу],
[, [файл_перехресних_посилань]]

Цей синтаксис показує, що група файлів може починатися з будь-якого параметра, що можна за бажанням застосувати до цих файлів, а потім можуть випливати файли, які за бажанням асемблюються. Ім'ям файла можна бути одне ім'я файла, або ви можете використовувати звичайні трафаретні символи DOS * і ? для завдання групи файлів. Якщо розширення імені файла не зазначено, Турбо Асемблер використовує за умовчанням розширення *.ASM.

TASM MYFILE,,,MYXREF

По цій команді файл MYFILE.ASM асемблює у файл MYFILE.OBJ, лістинг виводиться у файл з ім'ям MYFILE.LST, а перехресні посилання – у файл MYXREF.XRF.

Якщо при специфікації вихідних файлів використовувати трафаретні символи, їх можна використовувати також для завдання імен файла лістингу й об'єктного файла. Наприклад, якщо в поточному каталозі утримуються файли XX1.ASM і XX2.ASM, то командний рядок:

TASM XX*,YY*

асемблює всі файли, що починаються з букв XX, генерує об'єктні файли, імена яких будуть починатися з YY, а іншу частину імені формує відповідно до імені вихідного файла. Результуючі об'єктні файли одержать, таким чином, імена YY1.OBJ і YY2.OBJ.

Якщо не хочете створювати об'єктний файл, але при цьому одержати файл лістингу, або якщо одержати файл перехресних посилань, але не створюючи файл лістингу або об'єктний файл, можна в якості імені файла задати нульовий (фіктивний) пристрій.

Наприклад:

TASM FILE1,,NUL,

Ця команда асемблює файл FILE1.ASM в об'єктний файл FILE1.OBJ. При цьому файл лістингу не створюється, а створюється файл перехресних посилань FILE1.XRF.

Модель програмного модуля на Асемблері

Стандартний програмний модуль на мові Асемблера має таку структуру:

```
TITLE (Розмістите тут заголовок)
(Якщо потрібно оператор EXTERN помістите його тут).

STACK  SEGMENT PARA STACK 'STACK'
       DB      64 DUP (0)
STACK  ENDS
```

```
DSEG  SEGMENT PARA  PUBLIC 'DATA'

(Помістіть  тут дані)

DSEG   ENDS

SUBTITL   Основна програма

CSEG   SEGMENT PARA  PUBLIC 'CODE'
       ASSUME  CS:CSEG, DS:DSEG, SS:STACK

ENTRY          PROC FAR     ; Точка  входу

(Помістіть  тут код програми на Асемблері)

ENTRY          ENDP
CSEG   ENDS
       END    ENTRY
```

Базові директиви для оформлення модулів
на мові Асемблера 1810

Текст програми модуля на мові Асемблера оформляють як послідовність операторів псевдокоманд (директив і операцій) і операторів команд мікропроцесора. Оператори псевдокоманд (директиви й операції) використовуються для опису модуля і даних. Оператори команд мікропроцесора визначають виконувані в модулі дії. Оператори записуються знаками алфавіту мови: звичайно це цифри (0...9), букви латинського алфавіту (A ... Z, a ... z), спеціальні символи (? , :, ;, +, -, *, /, ( ), ').

Операція машиною команди має формат

Ім'я мітки:	операція	операнд(и)	;коментар

у якому обов'язковим є лише поле команди. Поле мітки заповнюється при необхідності посилань на операцію. Вміст поля операндів визначають типом машинної команди. Поле коментарів заповнюють, якщо потрібні пояснення вмісту окремих полів оператора. Практично всі мови Асемблера припускають формат запису оператора, коли число пропусків між полями не обмежене, але не менше одного. Для позначення імен і міток оператора використовують букви та цифри, але першою повинна бути буква. Не можна використовувати постійні символічні імена Асемблера (імена регістрів, мнемокоди машинних команд, мнемокоди псевдокоманд і операції Асемблера). Машинні команди записують як мнемонічні позначення команд мікропроцесора. Операндами машинних команд можуть бути числові або символьні константи; імена, числові значення яких визначають за псевдокомандами; мітки, числові значення яких визначають за адресами; імена регістрів, числові значення яких визначають за їхнім кодуванням в машинних кодах: арифметико-логічні вираження над усіма розглянутими типами. Використані в операндах константи записують у двійковому, восьмеричному, десятковому, шістнадцятковому і символьному кодах. У виразах використовують операції: арифметичні, логічні, зсуву, відношень.

Адресний простір МП 1810 сегментований. Тому програму модуля будують як послідовність описів окремих сегментів. Сегменти у функціональному відношенні не однорідні. Сегменти даних і стека використовують для збереження даних, а сегмент команд береже програму і константи.

Сегмент обмежується сегментними директивами Асем-блера. У Асемблері МП 1810 використовуються два види сегментних директив: спрощені сегментні директиви і стандартні сегментні директиви.

Спрощені сегментні директиви. Вони роблять управління сегментами відносно легким й ідеально підходять для компонування модулів Асемблера з мовами високого рівня, але підтримують тільки деякі із сегментних засобів, наявних у Асемблері МП 1810.

Основними спрощеними директивами визначення сегментів є директиви .STACK, .CODE, .DATA, .MODEL і DOSSEG. Розглянемо ці директиви, розбивши їх на дві групи. Перша буде група директив .STACK, .CODE і .DATA, друга – .MODEL і DOSSEG.

Директиви .STACK, .CODE і .DATA. Директиви визначення сегментів .STACK, .CODE і .DATA визначають сегмент стека, сегмент коду і сегмент даних відповідно. Наприклад, директива:

```
.STACK  200h
```

визначає стек розміром у 200h (512) байт. Що стосується стека, то це все, що можна зробити. Необхідно просто переконатися, що в даній програмі є директива .STACK, і Асемблер виділить стек. Для звичайних програм цілком підходить стек розміром 200h, хоча в програмах, що інтенсивно використовують стек (наприклад, у програмах, що містять рекурсивні виклики), може знадобитися стек більшого розміру.

Директива .CODE відзначає початок сегмента коду. Можна визначити, що для Асемблера всі ваші інструкції відносяться до сегмента коду. Насправді Асемблер дозволяє (за допомогою стандартних директив визначення сегментів) використовувати декілька сегментів коду, а директива .CODE вказує Асемблеру, в який саме сегмент треба помістити ваші інструкції. Визначення сегмента коду ще простіше, чим визначення сегмента стека, тому що аргументи для директиви .CODE вказувати не потрібно. Наприклад:

```
.
.CODE
sub   ax,ax   ; встановити акумулятор у значення 0
mov   cx,100  ; число виконуваних циклів
.
```

Директива .DATA більш складна. Як можна зрозуміти, директива .DATA відзначає початок сегмента даних. У цьому сегменті варто розміщати змінні пам'яті. Наприклад:

```
.
.DATA
TopBoundary  DW     100
Counter      DW     ?
ErrorMessage DB     0dh,0dh,'***Помилка***',0dh,0ah,'$'
.
```

Це досить просто. Вся "складність" директиви .DATA полягає в тому, що до того, як будуть звертатися до комірок пам'яті в сегменті, визначеному за допомогою директиви .DATA, потрібно завантажувати сегментний регістр DS ідентифікатором @data. Тому що сегментний регістр можна завантажити з регістра загального призначення або з комірки пам'яті, регістр DS завантажується за допомогою послідовності з двох інструкцій:

```
.
mov   ax,@data
mov   ds,ax
.
```

(Замість регістра AX можна використовувати будь-який загальний регістр.) Дана послідовність інструкцій встановлює DS таким чином, щоб він указував на сегмент даних, що починається по директиві .DATA.

Директива DOSSEG. Директива DOSSEG призводить до того, що сегменти в програмі Асемблера будуть згруповані відповідно до угод про упорядкування сегментів фірми Microsoft і є обов'язковою.

Директива .MODEL. Директива .MODEL визначає модель пам'яті в модулі Асемблера, де використовуються спрощені директиви визначення сегментів.

У Асемблері використовується шість моделей пам'яті.

Tiny — модель використовується в тих випадках, коли абсолютним критерієм гідності програми є розмір коду. У цій моделі всі чотири сегментних регістри встановлюються на той самий адрес. Загальний обсяг пам'яті – 64 кілобайта.

Small — сегменти коду і даних розташован окремо один від одного і не перекривають один одного, що дозволяє мати 64К пам'яті програм і 64К даних і стека.

Medium — код програми може займати обсяг до 1Мб. Дані і стек обмежені розміром 64К.

Compact — код програми може бути обмежений обсягом 64К. Дані і стек можуть мати розмір до 1 Мб.

Large, Huge — дані і код можуть займати граничний розмір — 1Мб.

Директиву .MODEL необхідно вказувати при використанні спрощених директив визначення сегментів, тому що в противному випадку Асемблер не буде знати, як установлювати сег-

менти, визначені за допомогою директив .CODE і .DATA. Директива .MODEL повинна бути першою перед директивами .CODE, .DATA і .STACK.

Приведемо приклад начерку програми, що використовує спрощені директиви визначення сегментів:

```
      DOSSEG
      .MODEL   small
      .STACK   200h
      .DATA
MemVar  DW     0
      .
      .
      .CODE
ProgramStart:
      mov    ax,@data
      mov    ds,ax
      mov    ax,[MemVar]
      .
      .
      mov    ah,4ch
      int    21h
      END    ProgramStart
```

Стандартні директиви визначення сегментів. Далі приведемо такий же приклад програми, як і в попередньому розділі, але цього разу використовуємо стандартні директиви визначення сегментів SEGMENT, ENDS, ORG і ASSUME.

```
DGROUP   GROUP   _DATA, STACK
       ASSUME   CS:_TEXT, DS:_DATA, SS:STACK
STACK    SEGMENT PARA STACK 'STACK'
       DB     200h DUP (?)
STACK    ENDS
_DATA    SEGMENT WORD PUBLIC 'DATA'
MemVar   DW     0
       .
       .
_DATA    ENDS
_TEXT    SEGMENT WORD PUBLIC 'CODE'
ProgramStart:
       mov    ax,_DATA
       mov    ds,ax
       mov    ax,[MemVar]
       .
       .
       mov    ah,4ch
       int    21h
_TEXT    ENDS
       END    ProgramStart
```

Директива SEGMENT.

Ім'я SEGMENT [вирівнювання] [об'єднання] [тип] ['клас']

Директива SEGMENT визначає початок сегмента. Мітка, що вказується в даній директиві, визначає початок сегмента та його ім'я. Якщо раніше був уже визначений сегмент за тим же ім'ям, то даний сегмент буде інтерпретуватися як продовження визначеного раніше сегмента, що має теж ім'я. Одне й теж ім'я сегмента можна вказувати в різноманітних модулях. На етапі компонування сегменти з однаковими іменами будуть об'єднані в один сегмент. Наприклад, директива:

Cseg SEGMENT PARA PUBLIC 'CODE' визначає початок сегмента з ім'ям Cseg. Директива SEGMENT може також (необов'язково) визначати атрибути сегмента, включаючи вирівнювання в пам'яті на межу байта (BYTE), слова (WORD), подвійного слова (DWORD), параграфа (16 байт) (PARA) або сторінки (256 байт) (PAGE). Інші атрибути містять у собі засіб, за допомогою якого сегмент буде комбінуватися з другими сегментами з тим же ім'ям і класом сегмента.

Директива ENDS визначає кінець сегмента. Наприклад:

Cseg ENDS

завершує сегмент з ім'ям Cseg, що починався по директиві SEGMENT. При використанні стандартних директив визначення сегментів треба завершувати кожний сегмент.

Директива ORG встановлює значення лічильника адреси або поточного сегмента. Наприклад:

ORG 100h

Директива ASSUME вказує Асемблеру, в значення якого сегмента встановлений даний сегментний регістр. Директиву ASSUME CS: потрібно вказувати в кожній програмі, в якій використовують стандартні сегментні директиви, тому що Асемблеру необхідно знати про сегмент коду для того, щоб установити виконувану програму. Крім того, звичайно використовують директиви ASSUME DS:, і ASSUME ES: завдяки котрим Асемблер знає, до яких комірок можна адресуватися в даний момент.

Директива ASSUME дозволяє Асемблеру перевірити допустимість кожного звернення до іменованої комірки пам'яті з урахуванням значення поточного сегментного регістра. Директива ASSUME дає засіб у будь-який момент повідомити Асемблеру про значення сегментного регістра, після чого Асемблер буде повідомляти про намагання щось зробити неможливе. Якщо сегмент не організується, те в якості імені задається значення NOTHING або даний сегментний регістр не згадується.

Загальні директиви. Разом із сегментними директивами в програмному модулі використовуються загальні директиви Асемблера, що визначають розміщення даних і управляють роботою програми.

Директива DB. Опис байта розміщеного в пам'яті.

[name] DB initialvalue,,, .

Розміщає й ініціалізує в пам'яті змінну довжиною в один байт (8 бітів). Name – ім'я змінної в пам'яті МП. Аргумент initialvalue може бути заданий одним із таких засобів:

- ціле число (наприклад, 12);
- рядок (наприклад, VAR1 DB 'message');
- константний вираз (наприклад, 2·5);
- операція DUP (наприклад, 10 DUP (?));
- знак питання (?) (наприклад, 0,1,? ,2).

Знак питання (?) дає вказівка Асемблеру лишити початкове значення невизначеним. Для того, щоб задати більш одного початкового значення, треба відокремити їх комами.

Директива DD. Опис подвійного слова, розміщеного в пам'яті.

[name] DD initialvalue,,, .

Розміщає й ініціалізує в пам'яті змінну довжиною в одне подвійне слово (4 байта) пам'яті.

Аргумент initialvalue може бути заданий одним із таких засобів:

– ціле число, наприклад, 1234;

– дійсне число, наприклад, 2.3;

– закодоване дійсне число;

– константний вираз ,наприклад, 2·12;

– адресний вираз, наприклад, farArrayPtr;

– операція DUP, наприклад, 10 DUP (?) .

Директива DF. Опис довгого слова розміщеного в пам'яті.

[name] DF initialvalue,,, .

Розміщає й ініціалізує в пам'яті змінну довжиною в 6 байтів пам'яті.

Аргумент initialvalue може бути заданий одним із таких засобів:

– ціле число, наприклад, 1234;

– дійсне число, наприклад, 2.3;

– закодоване дійсне число;

– константний вираз ,наприклад, 2·12;

– адресний вираз, наприклад, farArrayPtr;

– операція DUP, наприклад, 10 DUP (?) .

Директива DQ. Опис розміщеного в пам'яті слова, збільшеного учетверо.

[name] DQ initialvalue,,, .

Розміщає й ініціалізує в пам'яті змінну довжиною в одне слово, збільшене учетверо, (8 байтів) пам'яті.

Аргумент initialvalue може бути заданий одним із таких засобів:
– ціле число, наприклад, 1234;
– дійсне число, наприклад, 2.3;
– рядкова константа;
– закодоване дійсне число;
– константний вираз ,наприклад, 2·12;
– адресний вираз, наприклад, farArrayPtr;
– операція DUP, наприклад, 10 DUP (?);
– знак питання (?), наприклад, 0,1,?,2.
Директива DT. Опис десятибайтної перемінної розміщеної в пам'яті.
[name] DT initialvalue,,,
Розміщає й ініціалізує в пам'яті змінну довжиною в одну десятибайтну одиницю пам'яті.
Аргумент initialvalue може бути заданий одним із таких засобів:
– цілочисловий вираз (наприклад, 12334d);
– упаковане десяткове число;
– рядкова константа;
– закодоване дійсне число, наприклад, 2F0000000000000r);
– операція DUP, наприклад, 10 DUP (?);
– знак питання (?), наприклад, 0,1,?,2 .
Директива DW. Опис розміщеного в пам'яті слова.
[name] DW initialvalue,,, .
Розміщає й ініціалізує в пам'яті змінну довжиною в одне слово (2 байта) пам'яті.
Аргумент initialvalue може бути заданий одним із таких засобів:
– ціле число, наприклад, 1234;
– рядкова константа;
– константний вираз ,наприклад, 2·12;
– адресний вираз, наприклад, arrayAddres;
– операція DUP, наприклад, 10 DUP (?);
– знак питання (?), наприклад, 0,1,?,2.
Директива END. Кінець модуля.
END [expression] .
Помічає кінець модуля. Всі команди, що слідують за цією директивою, ігноруються.
Аргумент expression, якщо присутніq, задає точку входження програми, тобто ім'я (адресу), із якого починається виконання програми. Програма може містити більш одного модуля, але тільки в одному модулі дозволяється задавати точку входження. Модуль, в якому задана точка входження, називають головним.

Директива EQU. Створення символу.
name EQU expression .
Створює абсолютні символи (імена, що представляють 16-бітні значення), "прізвиська" (імена, що представляють другі символи) або текстові символи (імена, що представляють рядки) шляхом присвоєння імені name значення вираження expression.
DataSize EQU 23.
Параметром expression може бути:
– ціле число (234);
– дійсне число (3.23);
– рядкова константа ('abc');
– закодоване дійсне число (2F000000000000r);
– мнемокод команди (mov ax, 1);
– константний вираз (23·4);
– адресний вираз (arrayPtr).
Параметр name не повинний бути раніше визначений і не може бути перевизначений.
Якщо значення вираз є число між 0 і 65535, то Асемблер заміняє ім'я name на це число. В всіх інших випадках Асемблер заміняє ім'я name на текст.
Директива EXTRN. Опис зовнішнього імені.
EXTRN name:type,,, .

Описує змінну, мітку або символ як зовнішні. Зовнішньої рахується одиниця, що описана в другому програмному модулі, але використовується в програмному модулі, що містить директиву EXTRN.

EXTRN dmult, fltdiv.

У модулі, де одиниця вводиться, вона повинна бути описана директивою PUBLIC.

Директива INCLUDE. Вмикання кодів із зовнішнього файла.

INCLUDE ім'я_файла.

Змушує Асемблер завантажити вихідний код із зовнішнього файла й опрацювати його.

Директива LABEL. Створення змінної або мітки.

ім'я LABEL тип.

Створює нову змінну або мітку, надаючи їй поточне значення лічильника розміщення. Ім'я не повинно бути раніше визначено.

Типом може бути BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, FAR або ім'я описаного структурного типу.

Директива MACRO. Початок опису макрокоманди.

ім'я MACRO [dummyparameter,,,]

.

команди

.

ENDM .

Починає опис макрокоманди, що складається з імені і внутрішніх команд.

```
SWAP  MACRO a, b
        mov     ax,b
        mov     a,b
        mov     b,ax
ENDM
```

Макрокоманди аналогічні файлам, що включаються. Текст макрокоманди асемблює кожного разу, коли зустрічається її ім'я. Макрокоманди можуть мати прийняті формальні параметри.

Директива NAME. Завдання імені модуля.

NAME modulename .

Присвоює ім'я даному модулю.

Директива PROC. Початок опису процедури (підпрограми).

ім'я PROC [відстань]

команди

ім'я ENDP .

Відзначає початок процедури. Ім'я повинно бути унікальним. Відстань може бути або типу NEAR (всередині того ж сегмента), або FAR (за межами даного сегмента). Якщо відстань не вказана, передбачається тип NEAR. Припускається укладення процедур.

```
MultiplyBy4     PROC FAR
        sub     dx,dx
        shl     ax,1
        rcl     dx,1
        shl     ax,1
        rcl     dx,1
        ret
MultiplyBy4     ENDP
```

Процедура повинна містити хоча б одну команду RET, оскільки наприкінці процедури команда RET не виробляється автоматично, як у других мовах програмування.

Ім'я має ті ж можливості, що і мітка, вирішується виклик, перехід або цикл, що використовують ім'я в якості призначення.

Директива PUBLIC. Оголошення символу доступним для всіх модулів.

PUBLIC ім'я,,, .

Робить мітки, змінні або абсолютні символи доступними у всіх модулях програми.

PUBLIC MultiplyBy4, DivdBy4

Директива .RADIX. Установка системи числення для запровадження.

.RADIX вираження .

Встановлює систему числення для введення чисел, що входять у вираз.

Аргумент виразу розглядається як десяткове число, незалежно від поточної системи числення. Воно може бути будь-яким цілим числом від 2 до 16.

Стандартна система числення – десяткова.

Числа, що вводять у виді аргументів команд DD, DQ і DT, завжди вважаються десятковими, незалежно від поточної системи числення, якщо вони не супроводжуються покажчиком системи числення.

Операції. За допомогою операцій можна складати складні вирази, що можуть бути використані в якості операндів у командах і директивах. Основні операції над змінними, які дозволені до застосування у виразах операндів і директив, приведені в табл. Д1.1.

Таблиця Д1.1

Позначення	Формат	Найменування	Приклад	
			Операція	Результат (HEX)
Арифметичні операції				
+	+<число>	унарний плюс	+5	0005
+	<число>+<число>	бінарний плюс (додавання)	7+14	0015
-	-<число>	унарний мінус	-1	FFFF
-	<число>-<число>	бінарний мінус (віднімання)	2-1	0001
*	<число>*<число>	множення	5*5	0019
/	<число>/<число>	ціла частина ділення	2/5	0000
MOD	<число> MOD<число>	залишок від ділення	17 MOD 3	0005
SHORT	SHORT<число>	усікання величини до 1 байта	SORT 25F5	0F5
HIGH	HIGH<число>	виділення старшого байта	HIGH 25F5	25
LOW	LOW<число>	виділення молодшого байта	LOW 25F5	0F5
Логічні операції				
NOT	NOT <число>	інверсія	NOT 7	FFF8
AND	<число> AND<число>	логічне І	65535AND7	0007
OR	<число> OR<число>	логічне АБО	0AH OR 5H	000F
XOR	<число>XOR<число>	АБО, що виключає	0AH XOR 8	0002
Операції зсувів				
SHL	<число> SHL <число>	зсув уліво (другий операнд □ розрядність зсуву)	8888 SHL 1	1110
SHR	<число> SHR <число>	зсув управо (----\\-----)	8888 SHR 1	4444
Операції відношень				
GT	<число> GT <число>	більше	3 GT 5	0 неправда
GE	<число> GE <число>	більше або дорівнює	3 GE 5	0 неправда
EQ	<число> EQ <число>	дорівнює	3 EQ 5	0 неправ-

Позначення	Формат	Найменування	Приклад	
			Операція	Результат (HEX)
NE	<число> NE <число>	не дорівнює	3 NE 5	правда FFFF
LE	<число> LE <число>	менше або дорівнює	3 LE 5	істина FFFF
LT	<число> LT <число>	менше	3 LT 5	істина FFFF

Поряд з операціями арифметичними, логічними, зсуву і відношень, Асемблер містить операції зміни типу змінної, визначення характеристики змінної.

SEG. Повертає сегмент адреси виразу.

SEG вираз

Вираження може бути будь-яким виразом або операндом, що посилається на місце в пам'яті. SEG повертає константу, що подає сегментну частину адреси місця в пам'яті.

```
.DATA
REZULT      DW     5 DUP (0)
.CODE
mov     ax, SEG REZULT.
mov     ds,ax
```

Використовуючи директиви OFFSET і SEG, можна оперативно обчисляти значення зсуву або адреси сегмента будь-якої змінної або мітки.

OFFSET. Повертає зміщення в границях сегмента.

OFFSET вираз

OFFSET автоматично повертає значення зміщення щодо початку області, зазначеної базовим сегментом. Якщо при використовуванні директиви сегментації необхідно зміщення щодо початку групи, а не сегмента, треба точно зазначити групу як частину вираження.

```
mov     ax, OFFSET REZULT.
```

[ ]. Визначає додавання або операнд індексування в пам'яті.

[вираз 1] [вираз 2].

Регістри при непрямої адресації повинні бути замкнуті в [ ]. Зміщення при непрямої адресації може знаходитися як усередині, так і поза дужками ([ ]).

```
mov al,BYTE PTR ES:[BX]
mov al,CS:10h
mov [SI][BX],0FFFFH
```

BYTE. Задає розмір для виразу в один байт.

BYTE вираження

Вираз повинен бути адресою. Результатом є вираз, що вказує на ту саму адресу в пам'яті, але завжди має розмір BYTE, не рахуючи дійсного розміру виразу.

Для виконання цієї функції необхідно використовувати операцію PTR із попереднім йому BYTE.

```
mov [BYTE PTR BX],1
```

DUP. Повторює розміщення даних.

count DUP (вираз 1,[вираз]...)

count визначає скільки разів будуть повторені дані, визначені виразом. Операція DUP ставиться після однієї з директив розміщення даних (DB, DW і т.п.).

Кожний вираз є ініціалізованим значенням визначеного типу даних і слідує за DUP.

```
WRDBUF  DW 40 DUP(1)      ;ініціалізує
                          ;40 слів із значенням 1
SQUARE DB 4 DUP(4  DUP(0)) ;масив 4x4 із
                           ;значенням 0
```

DWORD. Задає розмір для виразу в подвійне слово.

DWORD вираз

Вираз повинен бути адресою. Результатом є вираз, що вказує на ту саму адресу в пам'яті і завжди має розмір DWORD, незважаючи на його дійсний формат.

Для виконання цієї функції необхідно використовувати директиву PTR з попереднім йому DWORD.

```
CALL DWORD PTR ADSUM
```

FAR. Задає вираз для дальнього посилання.

FAR вираження

Вираз повинен бути адресою. Результатом є вираз, що вказує на адресу в пам'яті і є дальнім посиланням, що містить базовий сегмент і зміщення незалежно від реального типу виразу.

```
CALL FAR ABC ; дальнє посилання
```

LENGTH. Повертає кількість розміщених елементів даних.

LENGTH ім'я

Ім'я є ідентифікатором, що вказує на розміщені дані за допомогою однієї з директив розміщення даних (DB, DD і т.д.). LENGTH повертає кількість повторюваних в імені елементів.

Якщо ім'я не було оголошено за допомогою операції DUP, то завжди повертається 1. LENGTH повертає 1 також у випадку, коли ім'я посилається на дані, що розміщені за допомогою виділення їх комами.

```
MSG    DB "Hello"
array  DW 10 DUP(0)
numbrs DD 1,2,3,4
var    DQ ?
lmsg =LEHGTH MSG ;=1 не використовувалася DUP
larray =LENGTH ARRAY ;=10
lnumbrs =LENGTH NUMBRS;=1,не використовувалася
                       ;DUP
lvar =LENGTH VAR ;=1,не використовувалася DUP
```

NEAR. Задає вираз для ближнього посилання.

NEAR вираз

Вираження повинен бути адресою. Результатом є вираз, що вказує на адресу в пам'яті і є ближнім посиланням, що містить тільки зміщення без базового сегмента, незалежно від реального типу виразу.

PTR. Задає вираз, що має особливий розмір.

type PTR expression

Expression (вираз) повинен бути адресою. Результатом є вираз, що вказує на адресу в пам'яті і що має завжди формат type, незважаючи на дійсний розмір виразу.

Type повинний приймати одне з таких значень: UNKNOWN, BYTE, WORD, DWORD, FWORD, PWORD, QWORD, TBYTE, DATAPTR, CODEPTR або ім'я структури — для даних. SHORT, PROC, NEAR, FAR — для коду.

SIZE. Повертає розмір розміщених даних.

SIZE ім'я

Ім'я є ідентифікатором, що посилається на дані, розміщені однією з директив розміщення даних (DB, DD і т.д.). SIZE повертає значення LENGTH, помножене на TYPE. Тому це не варто брати до уваги багатьох даних, і не розраховувати групу операцій DUP.

```
msg    DB "Hello"
array  DW 10 DUP(4 DUP(1),0)
numbrs DD 1,2,3,4
var    DQ ?
smsg=SIZE msg ;1, рядок має довжину 1
sarray=SIZE array ;=20,10 DUP подвійне слово
snumbrs=SIZE numbrs;4, довжина=1,DD=4 байт
svar=SIZE var ;=8,1 елемент=8 байт
```

TYPE. Повертає число, що визначає розмір або тип ідентифікатора.

TYPE expression

TYPE повертає одне з таких значень, що залежить від типу виразу:

```
BYTE    1
WORD    2
DWORD   4
FWORD   6
PWORD   6
QWORD   8
TBYTE   10
```

```
NEAR          0FFFFh
FAR           0FFFEh
constant      0
structure     кількість байтів у структурі
bvar    DB 1
darray  DD 10DUP(1)
x STRUC
DW ?
DT ?
x ENDS
fp EQU THIS FAR
tbvar=TYPE bvar   ;=1
tdarray=TYPE darray;=4
tx  =TYPE x       ;=12
tfp =TYPE fp      ;0FFFEh
```

WIDTH. Повертає ширину поля запису в бітах.

WIDTH recordfieldname

WIDTH record

Recordfieldname є ім'ям будь-якого поля раніше визначеного запису. WIDTH повертає кількість бітів, що позичаються recordfieldname у записі.

Record – ім'я раніше визначеного запису. WIDTH повертає загальну кількість бітів, що займаються всіма полями даного запису.

WORD. Задає вираз довжиною в слово.

WORD вираз

Вираз повинен бути адресою. Результатом є вирз, що вказує на адресу в пам'яті і завжди має розмір WORD, незалежно від дійсного розміру вираження.

Для виконання цієї функції необхідно використовувати операцію PTR із попереднім йому WORD.

```
X       DW 0
mov [WORD PTR BX],1
mov [WORD PTR X],1
```

Використовуючи директиви й операції мови Асемблера 1810, можна визначати і використовувати сегменти, розподіляти й ініціалізувати змінні в пам'яті і нарешті – проектувати програми будь-якої складності.

Повідомлення про помилки

TASM генерує такі типи повідомлень:

– інформаційні;

– попереджувальні;

– про помилки;

– про фатальні помилки.

Інформаційні повідомлення

Турбо Асемблер TASM генерує два інформаційних повідомлення, одне – на початку асемблювання файла, а друге при завершенні асемблювання. Ці повідомлення мають такий вид:

TurboAssembler Version2. 00 Copyright(C)1990 Borland International Assembling file: TEST. ASM

При завершенні асемблювання видається повідомлення про його результати:

```
Error messages: None      (число помилок)
Warning messages: None    (число попереджень)
Remaining memory: 279k    (обсяг вільної пам'яті)
```

Попередження і повідомлення про помилки

Попереджувальні повідомлення з'являються тоді, коли допущена незначна помилка в операторах вихідного тексту. Проте навіть такі помилки можуть істотно впливати на роботу вашої програми, тому необхідно щораз перевіряти причину, за якою було видано попереджуюче пові-

домлення. Попередження не зупиняють процес генерації об'єктного файла. Вони мають такий формат:

** Warning ** ім'я_файла (рядок) повідомлення

На відміну від попереджень, повідомлення про помилки зупиняють генерацію об'єктного файла, але асемблювання продовжується до досягнення кінця файла. Типовий формат повідомлення про помилку такий:

** Error ** ім'я_файла (рядок) повідомлення

Нижче приведені попереджувальні повідомлення і повідомлення про помилки:

Arguments need types override
(в аргументах потрібно перепризначення типів)
Argument to operation or instruction has illegal size
(аргументи операції або інструкції мають неприпустимий розмір)
Arithmetic overflow (арифметичне
переповнювання) ASSUME must
be segment register
(у ASSUME повинно вказуватися сегментний регістр)
Bad keyword in SEGMENT statement
(невірне ключове слово в операторі SEGMENT)
Can't emulate 8087 instruction
(неможливо емулювати інструкцію процесора 8087)
Can't make variable PUBLIC
(змінну неможливо зробити загальнодоступної)
Code or data emission to undeclared segment
(код або дані в неописаному сегменті)
Constant too large
(константа занадто велика)
CS not correctly assumed
(некоректно припускається CS)
Declaration needs name
(в описах потрібно ім'я)
Duplicate dummy argument:_
(дублює аргумент, що підставляється)
Expecting offset quantity
(очікується зміщення)
Expecting segment or group quantity
(потрібно сегмент або група)
Extra characters on line
(зайві символи в рядку)
Illegal immediate
(неприпустимий безпосередній операнд)
Illegal indexing mode
(неприпустимий режим індексування)
Illegal instruction
(неприпустима інструкція)
Illegal memory reference
(неприпустиме посилання на пам'ять)
Illegal number
(неприпустиме число)
Illegal origin address
(неприпустима початкова адреса)
Illegal override register
(неприпустиме перепризначення регістра)
Illegal radix
(неприпустима основа)
Illegal use of constant
(неприпустиме використання константи)

Illegal use of register
(неприпустиме використання регістра)
Illegal use of segment register
(неприпустиме використання сегментного регістра)
Invalid mode type
(неприпустимий тип режиму)
Invalid operand(s) to instruction
(неприпустимі операнди в інструкції)
Labels can't start with numeric character
(мітка не може починатися з числового символу)
Missing end quote
(пропущена дужка, що закриває,)
Missing module name
(пропущене ім'я модуля)
Model must be specified first
(модель пам'яті повинна задаватися першою)
Need address or register
(потрібний адреса або регістр)
Need file name after INCLUDE
(після INCLUDE потрібно вказувати ім'я файла)
Need left parenthesis
(потрібно ліва дужка) Need
right angle brackets (потрібно
права кутова дужка) Need right
square bracket
(потрібно права квадратна дужка)
Need stack argument
(потрібно аргумент стека)
Open procedure
(відкрита процедура)
Open segment
(відкритий сегмент)
Quantity illegal
(неприпустима величина)
Recursive definition not allowed for EQU
(для EQU не припускається рекурсивне визначення)
Register must be AL or AX
(необхідно використовувати регістр AL або AX)
Register must be DX
(необхідно використовувати регістр DX)
Reserved word used as symbol
(у якості ідентифікатора використане зарезервоване слово)
Rotate count must be constant or CL
(лічильник циклічного зсуву повинний задаватися константою або регістром CL)
Rotate count out of range
(значення лічильника зсуву перевищує припустимий діапазон)
Too few operands to instruction
(в інструкції занадто мало операндів)
Too many registers in expression
(у виразі занадто багато регістрів)
Undefined symbol (невизначений
ідентифікатор) Unexpected end of file (no
END directive)
(непередбачений кінець файла; немає директиви END)
Unknown character
(невідомий символ)
Unmatched ENDP:_
(немає відповідності до ENDP)
Unmatched ENDS:_

(немає відповідності до ENDS)
Value out of range
(значення поза діапазоном)

ДОДАТОК 2
СТИСЛИЙ ОПИС ПРОГРАМИ TD (TURBO DEBUGGER).

У цьому описі приведені необхідні для ознайомлення з роботою програми TD.EXE фірми Borland відомості. Більш докладну інформацію можна одержати в технічному описі цієї програми, що займає 150-200 сторінок (у залежності від версії).

Програма TD (Turbo Debugger) призначена для налагодження написаних на мовах Assembler, Turbo Pascal, C програм.

Турбо налагоджувач дозволяє асемблювати інструкції процесорів 8086, 80186, 80286, 80386, 80486, а також арифметичних сопроцесорів 8087, 80287 і 80387.

Запуск Турбо налагоджувача

Щоб запустити Турбо налагоджувач у відповідь на підказування DOS, уведіть TD, далі – необов'язковий набір аргументів командного рядка і натисніть клавішу Enter. Аргументи командного рядка можуть містити в собі ім'я програми, що налагоджується, і параметри налагоджувача.

Таким чином, формат командного рядка має такий вид:

TD[параметри][ім'я_програми] аргументи_програми]]

Елементи, що замкнуті в квадратні дужки, є необов'язковими. Якщо їх використовують, то набирати їх потрібно без квадратних дужок. "Ім'я_програми" – це ім'я тієї програми, що потрібно налагодити. За цим ім'ям можна зазначити аргументи.

Якщо просто натиснути клавішу Enter, Турбо налагоджувач завантажується і використовує параметри, призначені за умовчанням.

При запуску Турбо налагоджувача повинен бути доступним як виконуваний файл .EXE з включеною в нього налагодженою інформацією, так і вихідні файли програми. Турбо налагоджувач шукає вихідні файли спочатку в тому каталозі, де їх знаходить компілятор при виконанні компіляції, потім – у каталозі, заданому в параметрі Options:Path (Параметри:Маршрут) команди Source (Вихідний модуль), потім у поточному каталозі, і, нарешті, у каталозі, де знаходяться файли .EXE. Перед використанням Турбо налагоджувача треба також мати скомпільований виконуваний файл з повною налагодженою інформацією.

Завантаження нової програми для налагодження

Нову програму для налагодження можна завантажити за допомогою команди File:Open (Файл:Відчинення) рис. Д.2.1.

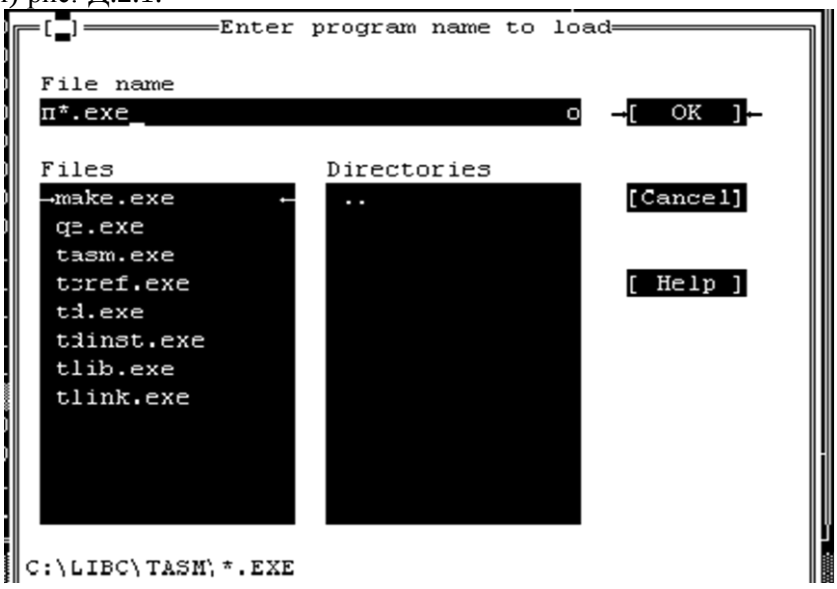

Рис. Д.2.1.

Можна використовувати трафаретні символи DOS для виводу списку файлів для вибору, або ввести ім'я конкретного файла. Якщо у відповідь на підказування натиснути клавішу Enter, то виводиться список усіх файлів *.EXE у поточному каталозі. Треба перемістити підсвічування на ім'я того файла, що завантажється, і натиснути Enter.

Діалогове вікно Open (мал. П.2.1) має такі поля і кнопки: Enter program name to load - введіть ім'я програми для завантаження; File name - ім'я файла; Files - файли; Directories - каталоги; OK - вибір; Cancel - скасування; Help - довідка.

Налагодження програми

Процес налагодження специфічний для кожної завантаженої в TD програми. Але частіше всього він здійснюється за допомогою команди "Трасування" (F7) у вікнах Module або CPU.

Завершення роботи даної програми

Коли ваша програма завершує роботу, і з неї виходять, налагоджувач знову одержує керування. Він виводить повідомлення, яке показує код завершення програми, що буде повертатися в DOS (звичайно 0).

Клавіші Ctrl-Break

Це сполучення клавіш майже завжди перериває програму і повертає керування налагоджувачу. Клавіші Ctrl-Break не можуть скасувати наступні дві умови (і якщо виникне одна з них, доведеться перезавантажити систему):

– попадання в цикл, коли заборонені переривання;

– відбувся збій системи через виконання помилкового коду.

Меню та діалогові вікна

Як і багато продуктів фірми Borland, Турбо налагоджувач має зручну систему меню, які доступні з горизонтального меню у верхньому рядку екрана (основне або головне меню). Головне горизонтальне меню доступно завжди, незалежно від того, яке вікно активно (тобто в якому вікні знаходиться курсор).

Існує три засоби вибору пунктів із головного меню:

– Натиснути F10, підвести курсор до необхідного елемента меню і натиснути клавішу Enter.

– Натиснути клавішу F10 і клавішу з першою буквою елемента меню (F, V, R, B, D, O, W, H).

– Натиснути клавішу Alt одночасно з першою клавішею пункту (елемента) меню (F, V, R, B, D, O, W, H) для активізації обраного меню команд. Наприклад, у будь-якому місці системи натискання клавіш Alt-F перемістить у меню File (Файл).

Меню E (System) буде відкрито при натисканні клавіш Alt-пробіл.

Для переміщення по всім меню, крім головного, використовуйте стрілки вправо/вліво для переміщення від одного меню, що спускається до іншого (наприклад, коли ви знаходитеся в меню File, натискання стрілки вліво перемістить у меню View).

Для переміщення по командах конкретного меню використовуйте стрілки нагору й вниз.

Використовуйте клавіші Home і End для переміщення до першої й останньої альтернативам (командам) меню, відповідно.

Для переміщення в меню або в діалогове вікно більш низького рівня (меню, яке спливає) використовують клавішу Enter.

Вийти з меню або системи меню можна в такий спосіб:

– Натисніть клавішу Esc для виходу з меню і повернення в попереднє меню.

– Натискайте клавішу Esc, якщо знаходитесь в меню другого рівня, це дозволяє вийти із системного меню повернутися в попереднє активне вікно.

- F10 для повернення з меню будь-якого рівня в попереднє активне вікно.

Деякі команди головного (основного) меню відповідають оперативним клавішам (скорочення команд). Там, де можливо, позначення оперативних клавіш зображуються справа від альтернативи меню.

Багато команд Турбо налагоджувача доступні за допомогою діалогових вікон.

Активне вікно

Хоча в Турбо налагоджувачі припускається відчиняти одночасно багато вікон, активним може бути тільки одне. Активне вікно відповідає наступним критеріям:

– має подвійну рамку, а не одинарну;

– містить курсор або рядок підсвічування;

– якщо вікна перекриваються, активне вікно знаходиться перед ними (перше вікно).

Робота з вікнами

При наявності такої розвиненої системи вікон можна, за бажанням, відчинити на екрані одночасно декілька вікон. Турбо налагоджувач дозволяє легко переходити з одного вікна в інше, переміщати їх, відчиняти їх одне за другим, стискувати їх або розширювати, а також закривати.

Більшість команд керування вікнами міститься в меню Windows (Вікна). Ще декілька команд можна знайти в системному меню E (System). Це меню відзначено символом E у лівій частині його верхнього рядка.

Примітка: Для того, щоб відчинити системне меню, натисніть клавіші Alt-пробіл або Alt-W.

Кожне вікно, що відчиняється, містить номер у верхньому правому куті. Звичайно, вікно Module (Модуль) – це вікно номер 1, а вікно Watches (Перегляд) – це вікно номер 2. Наступному вікну, що буде відчинено, буде привласнено номер 3, і т.д.

Можна також циклічно переходити від вікна до вікна, використовуючи команду меню Window: Next (Вікно: Наступне), або оперативну клавішу F6. Це зручно використовувати, якщо номер вікна схований, і невідомо, яку цифру використовувати для переходу до нього.

Для виводу списку усіх відкритих вікон виберають у рядку меню команду Window (Вікно). У нижній частині меню Window виводиться до дев'яти відкритих вікон, із яких можна зробити вибір.

Якщо вікно розбите на області (кожна область резервується для даних конкретного типу), то від однієї області до іншої можна переходити за допомогою команди Window: Next Pane (Вікно: Наступна область) або клавіш Tab (або Shift-Tab).

Коли в Турбо налагоджувачі відчиняють нове вікно, воно з'являється поблизу поточної позиції курсору і має розмір, що використовується за умовчанням для даного типу вікон. Якщо вважати положення або розмір вікна незручним, для їхнього настроювання можна використовувати команду Window: Size/ Move (Вікно: Розмір/ Переміщення).

При переміщенні вікна або зміні його розміру рамка поточного вікна змінюється і стає одинарною. При цьому можна для переміщення вікна використовувати клавіші стрілок, а для зміни його розміру – клавішу Shift разом із клавішами стрілок. Коли задовольнить отриманий результат, натисніть клавішу Enter. Команді Window: Size/Move відповідають оперативні клавіші Ctrl-F5.

Щоб швидко збільшити ("розкрити") або зменшити розмір вікна, виберіть команду Window: Zoom (Вікно: Переключення розміру). Команді Window: Zoom відповідаєть оперативна клавіша F5.

При роботі в вікні його можна закрити за допомогою команди Window :Close (Вікно: Закриття) або натисканням оперативних клавіш Alt-F3.

Можна також відновити для екрана Турбо налагоджувача початкову схему вікон. Це можна зробити за допомогою команди системного меню E: Restore Standard (Система: Відновлення стандартної схеми).

У Турбо налагоджувачі, як і в інших продуктах фірми Borland, вмонтовано контекстно-залежний оперативний довідник. Він доступний як при роботі в системі меню, так і при виводі повідомлення про помилку або підказування.

Для виводу довідкового екрана з інформацією, що відноситься до поточного контексту (вікну або меню) натисніть клавішу F1.

Вікно Module

Робота починається з виводу двох вікон: вікна Module (Модуль) і вікна Watches (Перегляд). Поки не відчиняють іншого вікна або не змінюють (настроюють) ці, вікна будуть повними. Це означає, що вони заповнюють увесь екран без перекриття. Нові вікна автоматично перекривають існуючі, поки по ним переміщаються.

У вікні Module (Модуль) на екран виводиться код програми, що налагоджується. Можна переміщатися по цьому модулі і переглядати дані і код, позиціювати курсор на іменах змінних програми і даючи відповідні команди локальним меню.

Вікно CPU

Вікно CPU (ЦП) показує поточний стан центрального процесора (ЦП) (рис. Д.2.2). За допомогою рис. Д.2.2 можна перевіряти і змінювати біти і байти, що складають код і дані програми. Це вікно містить п'ять областей, в одній з який показані шістнадцяткові байти даних, в іншій виводиться в безпосередньому вигляді (шістнадцяткові байти) зміст стека, у третій — зміст регістрів ЦП, у четвертій — машинні інструкції, а в п'ятій — зміст прапорів ЦП.

У вікні Code (Код) для тимчасової корекції своєї програми використовують вбудований Асемблер. При цьому інструкції вводять точно так, як при наборі вихідних операторів Асемблера.

Можна також одержати доступ до відповідних даних будь-якої структури даних, виводячи і змінюючи їх у різноманітних форматах.

Іноді Турбо налагоджувач відчиняє вікно CPU автоматично, якщо програма зупиняється на інструкції в середині рядка вихідного коду.

Вікно CPU (ЦП) можна створити, обравши команду основного меню View: CPU (Огляд: Центральний процесор).

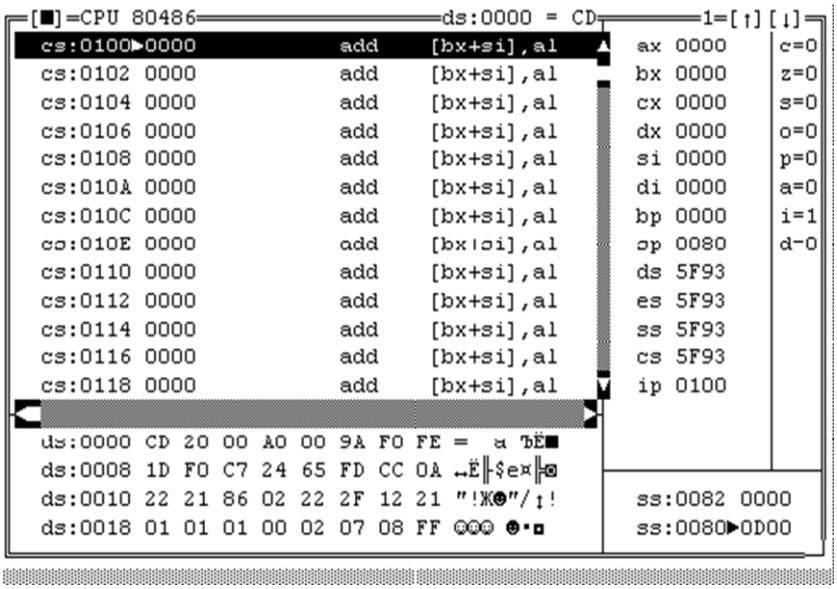

Рис. Д.2.2

Вікно Dump

У вікні Dump (Дамп) (рис. Д.2.3) виводиться в безпосередньому вигляді вміст області пам'яті (це вікно еквівалентно області даних вікна CPU).

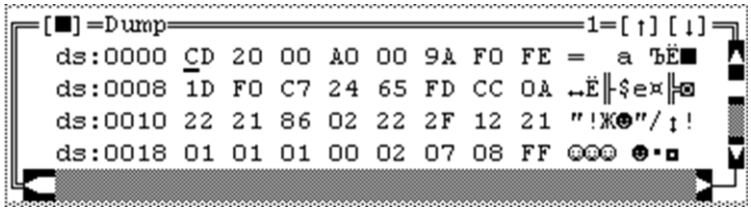

Рис. Д.2.3

Можна виводити дані у вигляді символів, байт, слів, подвійних слів, або в будь-якому форматі з плаваючою крапкою. Дане вікно можна використовувати для перегляду в безпосередньому вигляді деяких даних, коли не потрібні інші частини вікна CPU.

Звичайно це вікно доводиться використовувати при налагодженні програми на Асемблері на рівні вихідного коду, коли хочуть переглянути (на нижньому рівні), як виглядають деякі області даних. Для створення вікна Dump (Дамп) можна використовувати команду View: Dump (Огляд: Дамп).

Вікно Registers

У вікні Registers (Регістри) виводиться вміст регістрів і прапорів центрального процесора. воно працює, як сполучення областей регістрів і прапорів у вікні CPU (ЦП) (рис. Д2.4).

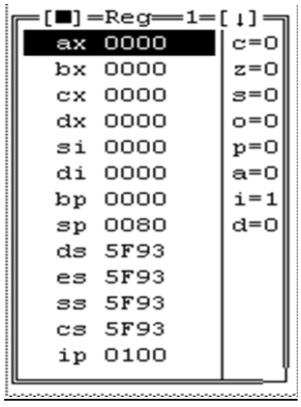

Рис. Д.2.4.

Використовують дане вікно, коли при налагодженні на рівні вихідного коду програми на Асемблері переглядають вміст регістрів. Ви можете скоротити розмір вікна Module (Модуль) і помістити поруч з ним вікно Registers.

Використання Турбо налагоджувача

У верхній частині екрана знаходиться рядок основного меню (рис. Д.2.5). Для виводу меню за допомогою даного рядка, треба натиснути клавішу F10, використати клавіші стрілок для вибору пункту меню, і натиснути клавішу Enter, або клавішу Alt у сполученні з першою літерою одного з пунктів меню.

```
--------------------------------------------------------------------------------
Ë  File  Edit  View  Run  Breakpoint  Data  Option  Window  Help  READY
--------------------------------------------------------------------------------
```

Рис. Д.2.5.

Основне меню (верхній рядок) має наступні команди: Е – системне меню; File – файл; View – огляд; Run – виконання; Breakpoints – точки зупинку; Data – дані; Options – параметри; Window – вікно; Help – довідка.

Приведемо стислу характеристику команд меню Турбо налагоджувача:

Меню =(System) ----------	
Restore Standard	Відновлює стандартний макет вікна
Repaint Desktop	Перемальовує екран
About	Виводить інформацію про Turbo Debugger
Меню File ----------	
Open	Завантажує в налагоджувач нову програму
Change Dir	Виконує зміну диска і/або директорії
Get Info	Видає інформацію про програму
DOS Shell	Запускає командний процесор ДОС
Resident	Робить Turbo Debugger резидентним
Symbol Load Table	Завантажує таблицю символічних імен,
Relocate	незалежну від файла .EXE.
Quit	Виконує повернення в ДОС
Меню View ----------	
Breakpoints	Перегляд точок зупину
Stack	Перегляд стека виклику
Log	Перегляд журналу подій і даних
Watches	Перегляд значень змінних, що відслідковуються
Variables	Перегляд списку локальних і глобальних змінних
Module	Перегляд вихідного тексту модуля програми
File	Перегляд вмісту дискового файла в текстовому або шістнадцятковому форматі
CPU	Перегляд машинних команд, даних і вмісту стека
Dump	Перегляд порядкового дампа області пам'яті

Registers	Перегляд стана регістрів і прапорів процесора
Numeric Processor	Перегляд стана сопроцесора або його емулятора
Execution History	Виводить асембльований код, що був запом'ятован для оберненого трасування або оберненого відтворення натискань клавіш
Hierarchy	Виводить список об'єктів або типів класу й ієрархічне дерево
Another Module	Відчиняє ще одне вікно модуля
Dump	Відчиняє ще одне вікно дампа
File	Відчиняє ще одне вікно файла
Меню Run ----------	
Run	Запускає програму без припинень
Go To Cursor	Виконує програму до поточної позиції курсору
Trace Into	Виконує один рядок вихідного тексту або команду
Step Over	Виконує один рядок вихідного тексту без входження в підпрограму
Execute To	Виконує програму до зазначеної адреси Until
Return	Виконує програму до повернення з функції
Animate	Запускає програму в безупинному покроковому режимі
Back Trace	Реверсує виконання програми на один вихідний рядок або команду
Instruction Trace	Виконує одну команду
Arguments	Встановлює аргументи командного рядка
Program Reset	Заново завантажує поточну програму
Меню Breakpoint----------	
Toggle	Переключає точку зупину в поточній позиції курсору
At	Встановлює точку зупину на рядку із зазначеною адресою
Changed Memory Globa	Встановлює глобальну точку зупину, який включається при зміні вмісту області пам'яті
Expression True Global	Встановлює глобальну точку зупину, який включається, коли вираження стає істинним
Delete All	Видаляє всі точки зупину
Меню Data----------	
Inspect	Дозволяє перевірити елемент даних
Evaluate/Modify	Обчисляє значення виразу
Add Watch	Заносить змінну у вікно спостереження
Function Return	Відображає значення, що повертається поточною підпрограмою
Меню Options----------	
Language	Визначає мову для обчислення виразів за вихідним модулем
Macros Create	Визначає клавішний макрос
Stop Recording	Закінчує запис макроса
Remove	Видаляє клавішний макрос
Delete All	Видаляє всі клавішні макроси
Display Options	Дозволяє встановити опції екрана (свопінг, розмір, табуляція)
Path for Source	Список директорій, у яких знаходяться вихідні файли
Save Options	Зберігає на диску параметри налагоджувача, макроси і параметри вікон
Restore Options	Відновлює параметри з диска
Меню Window----------	
Zoom	Виконує трансформацію до повного розміру вікна і тому
Next	Послідовно активізує вікна, відкриті на екрані
Next Pane	Виконує перехід до наступному підвікна по-

	точного вікна
Size/Move	Дозволяє змінити розміри і положення поточного вікна
Iconize/Restore	Зменшує розмір вікна до невеличкої піктограми і відновлює його
Close	Закриває поточне вікно
Undo Close	Відміняє останню команду Close
Dump Pane to Log	Записує у вікно реєстрації вміст поточного підвікна
User Screen	Виводить на дисплей вихід даної програми
Open window list	Виводить список відкритих вікон для активації
Window Pick	Виводить меню відкритих меню, якщо на екрані їх більш 9
Меню Help----------	
Index	Виводить зміст інтерактивного підказування
Previous Topic	Виводить попередній екран підказування
Help on Help	Виводить підказ по самій інтерактивній системі підказування

У нижній частині екрана знаходиться перелік функціональних клавіш, призначення яких наводиться в табл. Д2.1.

Таблиця Д2.1.

Клавіша	Команда меню	Призначення
F1		Одержання контексто-залежної інтерактивного підказування
F2	Breakpoints| Toggle	Встановлює точку зупину в позиції курсору
F3	View|Module	Список узятих модулів
F4	Run|Go to Cursor	Виконання до позиції курсору
F5	Window|Zoom	Наїзд/від'їзд для поточного вікна
F6	Window|Next Window	Перехід до наступного вікна
F7	Run|Trace Into	Виконання одного вихідного рядка або команди
F8	Run|Step Over	Виконання одного вихідного рядка або команди з пропусканням викликів
F9	Run|Run	Запуск програми
F10		Виклик лінійки меню, вихід із меню

Активні клавіші

Активна клавіша — це клавіша, натискання якої змушує налагоджувач виконувати визначену дію незалежно від поточного стана середовища Turbo Debugger.
У табл. Д2.2 перераховані всі активні клавіші й їхнє призначення.

Таблиця Д2.2

Клавіша	Команда меню	Функція
Alt-F1	Help|Previous Topic	Повернення до минулого екрана підказування
Alt-F2	Breakpoints|At	Встановлює точку зупину за адресою
Alt-F3	Window|Close	Закриває поточне вікно
Alt-F4	Run|Back Trace	Реверсує виконання програми
Alt-F5	Window|User Screen	Показує екран виводу програми
Alt-F6	Window|Undo Close	Відчиняє останнє закрите вікно
Alt-F7	Run|Instruction Trace	Виконує одну команду програми
Alt-F8	Run|Until Return	Виконує програму до повернення з функції

Клавіша	Команда меню	Функція
Alt-F9	Run/Execute To	Виконує програму до заданої адреси
Alt-F10		Викликає локальне меню вікна
Alt-1-9		Робить активним вікно із заданим номером
Alt-пробіл		Викликає меню System
Alt-B		Викликає меню Breakpoints
Alt-D		Викликає меню Data
Alt-F		Викликає меню File
Alt-H		Викликає меню Help
Alt-O		Викликає меню Options
Alt-R		Викликає меню Run
Alt-V		Викликає меню View
Alt-W		Викликає меню Window
Alt-X	File|Quit	Здійснює вихід із налагоджувача і повернення в DOS
Alt- =	Options|Macros| Create	Визначає клавішний макрос
Alt-мінус	Options|Macros| Stop Recording	Завершує запис макроса
Ctrl-F2	Run|Program Reset	Завершує сеанс налагодження і встановлює програму у вихідний стан
Ctrl-F4	Data|Evaluate	Обчислює значення виразу
Ctrl-F5	Window|Size/ Move	Ініціює переміщення або зміна розміру вікна
Ctrl-F7	Data|Add Watch	Заносить змінну у вікно спостереження
Ctrl-F8	Breakpoints| Toggle	Переключає точку зупину в поточній позиції курсору
Ctrl-F9	Run|Run	Запускає програму
Ctrl-F10		Викликає локальне меню вікна
Ctrl- □		Зміщає початкову адресу в підвікні коду, даних або стека вікна процесора на 1 байт нагору
Ctrl- □		Зміщає початкову адресу в підвікні коду, даних або стека вікна процесора на 1 байт униз
Ctrl-A		Переміщення до попереднього слова
Ctrl-C		Скролінг на один екран униз
Ctrl-D		Переміщення вправо на один стовпчик
Ctrl-E		Переміщення нагору на один рядок
Ctrl-F		Переміщення до наступного слова
Ctrl-R		Скролінг на один екран нагору
Ctrl-S		Переміщення вліво на один стовпчик
Ctrl-X		Переміщення на один рядок униз
Shift-F1	Help|Index	Перехід до змісту інтерактивного підказування
Shift-Tab		Переміщення курсору до попереднього підвікна або елементу блока діалогу
Shift- □		Переміщення курсору між підвікнами вікна (підвікно в напрямку стрілки стає активним)
Shift- □		
Shift- □		
Shift- □		
Esc		Закриває вікно перевірки,

Клавіша	Команда меню	Функція
		виконує повернення з меню
Ins		Починає вибір (виділення) блока тексту; для виділення використовують клавіші "Стрілка вліво" і "Стрілка вправо"
Tab	Window|Next Pane	Переміщає курсор у наступне підвікно поточного вікна

ДОДАТОК 3

СТИСЛИЙ ОПИС КРОСС-АСЕМБЛЕРА 1816ВЕ51

Загальні відомості

Цей додаток описує програму транслятора з мови Асемблера процесорів сімейства і8051. Асемблер має два режими роботи: діалогу і командного рядка.

Режим діалогу

Для виклику Асемблера пишуть: x8051
Асемблер у відповідь запросить:
Listing Destination ? (N,T,D,E,L,<CR>=N).
Де абревіатури означають наступне:

N = друку немає
T = термінал
P = принтер
D = диск
E = тільки помилки
L = друк вкл/вимк

Потім Асемблер запросить ім'я файла, що містить вихідні коди:
Input filename:
При запровадженні імені файла можна опустити розширення, якщо воно asm. Далі Асемблер запросить ім'я вихідного файла:
Output filename:
Якщо користувач відповість поверненням каретки, ім'я вихідного файла буде утворено з імені вхідного з розширенням obj. Якщо ім'я файла у відповіді не буде містити розширення, то воно передбачається рівним obj.
Якщо видача лістингу йде під керуванням директиви Асемблера LIST ON/OFF, то виникає додатковий запит:
LIST ON/OFF Listing Destination (T,P,D,<CR>=T):
Скорочення відповідають попереднім.
Директива LIST ON/OFF дозволяє користувачу виводити лістинг вихідного тексту частково.
Якщо замовлена видача одних помилок, то Асемблер запитає:
Error Only Listing Destination (T,P,D,<CR>=T):
Якщо лістинг виводиться на принтер (для однокористувальних систем) або на диск, то Асемблер запросить про видачу таблиці посилань.

Режим командного рядка

Асемблер може сприймати командний рядок. У цьому випадку ім'я вхідного файла визначають першим, додатково може йти ім'я вихідного файла і список опцій. Загальна форма команди (необов'язкові поля показані в квадратних дужках) буде наступною:

asm8051[-q]input_filename[output_filename][-t,-p,-d,-px,-dx]

Якщо введена опція -q, на екран виводяться тільки повідомлення про помилки і відповідні рядки. Ця опція повинна передувати імені файла.
Опції асемблювання:

-t = виведення на термінал;
-p = виведення на принтер;
-x = виведення лістингу з таблицею перехресних посилань;
-d = виведення на диск;
-e = виведення тільки повідомлень про помилки;
-l = виведення блоків, позначених у тексті LIST ON/OFF.

Синтаксис мови Асемблера

Визначники підстави системи числення.
Binary - B
Octal - O | Q
Decimal - D
Hex - H
Ascii - "X" | 'X'

Визначено значення двох послідовних символів, замкнутих в одиночні або подвійні символи. Проте для того, щоб користуватися ними необхідна директива TWOCHAR ON.

'CR' – повернення каретки;
'LF' – перехід на другий рядок;
'SP' – пробіл;
'HT' – горизонтальна табуляція;
'NL' – порожній символ.

Коментарії

Рядки коментарів повинні починатися із крапки з комою або із зірочки в першій колонці, за винятком використання директиви COMMENT. Коментар після інструкції не супроводжується крапкою з комою, якщо асемблювання виконується в Spaces Off режимі. Якщо асемблювання виконується в Spaces On режимі, усі коментарі повинні починатися із крапки з комою. Докладніше дивитеся директиву SPACES.

Програмний лічильник

Спеціальні символи долар ($) і зірочка (*) варто використовувати у виразах, щоб визначити програмний лічильник. Розмір, привласнений знаку долар, відповідає значенню лічильника команд на початку цієї інструкції.

Мітки

Нелокальні мітки можуть складатися з будь-якого числа символів, але тільки 32 символи будуть значущими. Мітки ставлять в будь-якій колонці, якщо ім'я закінчується двокрапкою. Якщо двокрапку не використовують, мітка повинна починатися з першої колонки. Великі і маленькі літери вважаються різноманітними.

Локальні мітки можуть використовуватися подібно нелокальним. Різниця в тому, що локальна мітка визначена тільки між нелокальними міткам. Коли програма переходить від однієї локальної зони до іншої, ім'я локальної мітки використовують повторно. Цю особливість використовують для міток, що викликаються тільки в локальній області, як описано вище, і нові імена міток не потрібні.

Асемблер визначає локальні мітки за символом долар на початку або в кінці імені. Цей ідентифікатор може бути змінений за допомогою директиви LLCHAR. Наступні приклади показують використання локальних міток.

```
LABEL1:     OR     LABEL1:
$1:     NOP        1$:  NOP
$2:     JMP  $1    2$:  JMP  1$
        JMP  $2         JMP  2$

LABEL2:            LABEL2:
```

```
$1: NOP          1$:  NOP
$2: JMP  $1      2$:  JMP  1$
    JMP  $2           JMP  2$

 LABEL3:          LABEL3:
 $1: NOP          1$:  NOP
 $2: JMP  $1      2$:  JMP  1$
     JMP  $2           JMP  2$
```

У цьому прикладі використовуються три нелокальні мітки LABEL1, LABEL2 і LABEL3. Локальні мітки $1 і $2 або 1$ і 2$ мають різноманітні значення, коли на них посилаються в різноманітних локальних областях. Зауважимо, що $1 не позначає 1$. В імені локальної мітки може бути використаний будь-який символ. Локальна мітка може мати довжину до 32 символів. У локальних мітках не можуть використовуватися такі оператори, як "+". При використанні директив VAR, DEFL, SECTIONS, ENDS дія локальних міток не буде перериватися.

Режими адресації

Нижче перераховані засоби адресації процесора 8051 і наведені приклади на кожний тип.

Безпосередня. Дані містяться в інструкції.
Наприклад:
```
MOV A,#12H ; Завантажити в А число 12H
MOV A,#DATA ;Завантажити в А число, пов'язане з міткою DATA
```

Реєстрова. Дані містяться в регістрі процесора
Наприклад:
```
MOV R0,A    ;Завантажити в регістр 0 вміст акумулятора
INC R0  ;Збільшити вміст акумулятора на 1
```

Реєстрова непряма. Адреса операнда вказується вмістом регістра.
Наприклад:
```
MOV A,@R0 ; Завантажити акумулятор вмістом байта пам'яті, на який посилається регістр R0
MOV  @R0,A ;Запам'ятати вміст регістра в пам'яті
```

Пряма. Адреса операнда міститься в інструкції.
Наприклад:
```
JMP ADDRESS ; Передати керування на мітку ADDRESS
```

Відносна. Адреса операнда задається стосовно поточної інструкції. Якщо адреса задається з використанням константи, обчислення провадяться з адресою наступної інструкції.
Наприклад:
```
DJNZ 48H,LOOP ;Асемблер обчисляє адресу, віднімаючи значення мітки LOOP з адреси наступної інструкції
JZ 4   ;Перехід до мітки, розташованої на віддаленні 4 байтів від початку наступної команди
```

Директиви Асемблера

Ця частина описує директиви Асемблера. Директиви можуть виділятися крапкою для відокремлення їх від інструкцій програми.

Керування розподілом пам'яті

ORG VALUE, ORIGIN
Встановлює адресу програми. Якщо директива не виконана адреса за умовчанням встановлюється в 0000.

END VALUE

Ця директива відзначає кінець програми або залученого файла. Вираз, що слідує за END, додатковий і, якщо існує, то вказує стартову адресу програми. Цю адресу заносять у вихідний файл, якщо у вихідному форматі існує тип запису стартової адреси.

ASCII STRING

Запам'ятовує STRING в пам'яті, не включаючи обидва символи закінчення рядка, а також вертикальну риску ("|", шістнадцяткове 7C).

```
ASCII   Hello   ;Запам'ятовує символьне уявлення
                ; слова Hello. Цей коментарій
                ; запам'ятовується теж
ASCII   Hello|  ;Тепер коментарій не повинний бути
                ; запам'ятован. Наступний приклад аналогічний
                ;цьому.
ASCII   Hello
```

DB VALUE

FCB
DEFB
BYTE
STRING

Асемблер поміщає величини в послідовні комірки пам'яті. Вираз BYTE дозволяє змішувати типи операндів, розділені комами. Рядки символів варто укладати в апострофи. Якщо рядок містить апострофи, то треба поставити два апострофи підряд. Якщо після директиви не слідує вираз, один байт резервується й обнуляється. Мітка є необов'язковою. Наступний приклад показує застосування директиви BYTE.

```
.BYTE           ;Відводить 1 нульовий байт
.BYTE 10        ;Відводить байт = 10
.BYTE 1,2,3 ;Відводить 3 байти=рівні 1,2 і 3
.BYTE SYMBOL-10 ;Шукає в таблиці символів
                ;мітку SYMBOL віднімає з неї 10,
                ;і запам'ятовує результат
.BYTE 'HELLO'   ;запам'ятовує еквівалент
                ; символів ASCII слова HELLO
.BYTE 'OK',0DH ;Те ж саме, що і попереднє,
                ;але добавляє наприкінці байт 0DH.
.BYTE 'IT"S' ;Вбудовує апостроф
```

DW VALUE

FDB
DEFW
WORD

Ці директиви поміщають 16-бітні величини в пам'ять. Декілька слів можуть бути задані через кому. Якщо вираз не дано, резервується й обнуляється 1 слово. Мітка не є обов'язковою.

LONG VALUE

LONGW
LWORD

Ці директиви поміщають 32-бітні величини в пам'ять. Декілька слів можуть бути задані через кому. Якщо вираз не дано, резервується й обнуляється одне слово. Мітка не є обов'язковою.

DS SIZE,VALUE

RMB
DEFS

Ця директива резервує фіксоване число байтів, обумовлене величиною SIZE. Ніякі величини не запам'ятовуються в області, що резервується.

FLOAT VALUE

Перетворює величину VALUE у формат із плаваючою крапкою одинарної точності. Величина не округлюється, а відкидається, якщо мантиса більше 24 битів.

```
FLOAT  178. 125
FLOAT  100,. 125,-178. 125
```

DOUBLE VALUE

Перетворює величину VALUE у формат із плаваючою крапкою подвійної точності. Величина не округлюється, а відкидається, якщо мантиса більше 24 битів.

```
DOUBLE  178. 125
DOUBLE  100,. 125,-178. 125
```

BLKB SIZE,VALUE

Резервується деяке число байтів, обумовлене SIZE. Якщо поле VALUE є присутнім, величина VALUE запам'ятовується в кожному байті. У противному випадку байти, що резервуються, обнуляються. Мітка є необов'язковою.

```
BLKB 20      ;Запам'ятовує 20 нульових байтів
BLKB 20,0    ;Запам'ятовує 20 нульових байтів
BLKB 20,FFH  ;Запам'ятовує 20 байтів значення 0FFH
```

BLKW SIZE,VALUE

Резервується деяке число 16-бітних слів, обумовлене SIZE. Якщо поле VALUE є присутнім, величина VALUE запам'ятовується в кожному слові. У противному випадку, слова, що резервуються, обнуляються. Мітка є необов'язковою.

```
BLKW 20      ;Запам'ятовує 20 нульових слів
BLKW 20,0    ;Запам'ятовує 20 нульових слів
BLKW 20,FFFFH ;Запам'ятовує 20 слів значення
              ;0FFFFH
```

BLKL SIZE,VALUE

Резервується деяке число 32-бітних слів, обумовлене SIZE. Якщо поле VALUE є присутнім, величина VALUE запам'ятовується в кожному слові. У противному випадку, слова, що резервуються, обнуляються. Мітка є необов'язковою.

```
BLKW 20      ;Запам'ятовує 20 нульових подвійних
            ;слів
BLKW 20,0    ;Запам'ятовує 20 нульових подвійних
              ;слів
BLKW 20,FFFFH ;Запам'ятовує 20 подвійних слів
             ;значення 0FFFFH
```

EQU VALUE

EQUAL

Привласнює імені LABEL значення VALUE. VALUE може бути іншим символом або дозволеним математичним виразом.

```
LABEL:  VAR    VALUE
        DEFL
```

Привласнює імені LABEL значення VALUE, але значення може змінюватися за текстом програми. Ім'я, визначене як VAR, не варто перевизначати директивою EQUEL.

MACRO ARGS

Задає початок макровизначення.

ENDM

MACEND

Задає кінець макровизначення.

MACEXIT

Ця директива викликає безпосередній вихід із макроса. Різниця MACEXIT і MACEND під час виконання макроса полягає в тому, що MACEXIT не завершує макрос, якщо MACEXIT знаходиться умовно в помилковому блоці, що асемблюється , що не виконується. Усі умовно асемблювані величини запам'ятовуються таким же чином, як і при завершенні макроса.

GLOBAL
PUBLIC

Визначивши мітку як глобальну, можна посилатися на неї з інших програм. Декілька міток можна визначити як віддалені, записавши їх через кому. Нижче подано приклад використання GLOBAL.

```
GLOBAL SYM1 ;Оголошує символ SYM1 доступним для інших програм. Лінкер буде відслідковувати зовнішні посилання:
GLOBAL SYM1,SYM2 ;Зовнішніми оголошені два
                 ;символи SYM1 і SYM2
```

GLOBALS ON

Ця директива вказує Асемблеру трактувати всі мітки після GLOBALS ON як глобальні, на які можуть посилатися інші програми. Ця директива не діє на локальні мітки.

GLOBAL OFF

Ця директива повертає Асемблер у режим, що потребує директиви GLOBAL для завдання глобальних символів.

EXTERN
EXTERNAL

Визначає мітку як задану в іншій програмі. Регістри можуть бути задані як зовнішні двома шляхами. Перший – поставить слово REG перед міткою, що задається як зовнішня. Якщо декілька міток міститься в одному визначенні, кожна повинна мати модифікатор REG перед собою. Другий шлях складається в переключенні в секцію регістрів, а потім в завданні зовнішніх міток. У цьому випадку кожній мітці автоматично присвоюється атрибут регістр.

REG ARG
REGISTER

Ця директива дозволяє користувачу визначити ім'я регістра й ім'я біта, що прямо адресується . Ім'я може мати до десятьох символів довжини. Ім'я регістра запам'ятовується в окремому буфері так, що таблиця символів не містить його і не може бути перевірено. Таблиця регістрів заповнюється в наступному порядку:

перші – визначені наперед імена регістрів;

другі – визначені користувачем імена регістрів;

треті – визначені користувачем імена бітів.

Біти обумовлені, як номери регістра або біта, відокремлюються крапкою, як показано нижче. Числа можуть бути або абсолютними, символьними величинами, або їхньою сумішшю. Може бути визначено максимум 512 регістрів. Деякі імена регістрів можуть використовуватися лише якось, проте, регістри можуть рівнятися до інших регістрів.

```
 REG0:REG R4 ;Визначити REG0 як регістр R4
REG1:REG P3 ;Визначити REG1 портом P3
REG2:REG REG1 ;Визначити REG2 рівним REG1
              ;і значить P3
BIT0:REG 20H. 0 ;Визначити BIT0 для
              ;адресації bit#0
 BIT1:REG REG1. 1 ;Визначити BIT1
              ;бітом 1 порта 3
```

SECTION

Дана директива дозволяє генерувати імена секцій, що задаються користувачем. У Асемблері є 4 секції, що задаються стандартно, CODE, DATA, BSECT і RSECT. У файлі допускається існування не більш 256 секцій. Довжина кожного імені не повинна перевищувати 32 символів. Букви, введені по нижньому і верхньому регістрам, вважаються різноманітними. Секції, що описується користувачем, може бути привласнений реєстровий атрибут за допомогою введення модифікатора REG справа від директиви опису секції. Єдиною командою, що може виконуватися в реєстровій секції, є директива DS. Після того, як секція була описана, у програмі можна переходити до цієї секції й навпаки, використовувати ім'я секції в якості мнемонічного позначення. Стандартної (що задається за умовчанням) секцією є секція CODE. Секції можуть бути вкладе-

ними. Аналогічно всім іншим директивам, імені секції може передувати десяткова точка. Далі приведені деякі приклади опису імен секцій і переходів між різноманітними секціями.

```
NOP          ; Дана команда за умовчанням
             ;знаходиться в секції CODE
.DATA        ;Переключення в попередньо
             ;задану секцію DATA
.BYTE        ;Цей байт знаходиться в секції DATA
SECTION1: . SECTION  ;Описана нова секція.
;Даний опис автоматично
;робить активну дану секцію.
NOP      ;Дана команда знаходиться
         ;у секції SECTION1
.CODE    ;Обернене переключення в секцію CODE
NOP      ;Дана команда знаходиться в CODE
.SECTION1        ;Переключення в задану
                 ;користувачем секцію SECTION1
NOP              ;Дана команда знаходиться в
                 ;секцію SECTION1
.BYTE    ;Будь-яка секція може містити код або
         ;дані або те й інше
```

Регістри можуть описуватися за допомогою або директиви REG, або реєстрової секції. Секція RSECT визначена в якості реєстрової секції, але будь-якій секції, що описується користувачем, реєстровий атрибут може бути привласнений за допомогою модифікатора REG. Основне розходження між двома засобами опису регістрів полягає в тому, що описані в реєстровій секції регістри є такими, які переміщаються. Єдиною командою, що може бути виконана в реєстровій секції, є директива DS.

```
RSECT    ;Перехід до реєстрової секції
REG1:  . DS 1  ;Описується кожний використаний
               ;регістр
REG2:  . DS 1
REG3:  . DS 1
REG_FlLE:  SECTlON REG  ;Описується нова
;реєстрова секція
REG4:          . DS 1  ;Описуються регістри
                   ; у нової
REG5:          . DS 1  ;секції
```

Окремі біти, що адресуються, можуть бути описані за допомогою директиви REG або за допомогою директиви DS у секції BSECT. Секція BSECT визначена як бітова секція. Основне розходження між цими двома засобами опису бітів полягає в тому, що описані в бітовій секції біти є такими, які переміщаються. Будь-якій описаній користувачем секції може бути заданий бітовий атрибут за допомогою бітового модифікатора. Єдиною командою, що може бути виконана в бітовій секції, є директива DS.

```
BSECT    ;Перехід до бітової секції
BIT:  . DS 1  ;Описується кожний використовуваний біт
BIT2:         . DS 1
BIT3:         . DS 1
BIT_FILE  SECTION BIT ;Описується нова бітова ;секція
 BIT4:        . DS 1  ;Описується біт і нової секції
BIT5:         . DS 1
```

ENDS

Дана директива використовується в сукупності з директивою SECTION. Директива ENDS забезпечує завершення вкладених секцій у файлі.

ABSOLUTE (Абсолютний режим)

Дана директива дозволяє Асемблеру при можливості використовувати адреси сторінки 0. Виконувані команди завжди повинні асемблюватися у відносному режимі.

RELATIVE (Відносний режим)

Дана директива дозволяє Асемблеру повертатися з абсолютного режиму у відносний. Виконувані команди завжди повинні асемблюватися у відносному режимі. Цей режим є стандартним (використовуваним за умовчанням).

RADIX <значення>

Підстава	Значення
2 або B	Двійкове
8 або O або Q	Восьмеричне
10 або D	Десяткове
16 або H	Шістнадцяткове

Відсутність виразу означає повернення до стандартного (використовуваному по умовчанню) режиму (тобто з основою системи числення 10); при цьому припускається, що всі інші системи числення будуть позначатися за допомогою літер B, Q, D або H після константи. Слід зазначити, що при завданні основи системи числення 16 не існує засобу опису десяткового або двійкового числа, оскільки як літера D, так і літера B є припустимими шістнадцятковими цифрами.

INCLUDE <імені-файла>

Вказує Асемблеру включити даний файл у процес асемблювання. Імена файлів можуть містити в собі маршрутні імена. Розширення імен файлів повинні задаватися цілком. Вкладене вмикання файлів не допускає.

MODULE.

Дана директива призначена для використання в сукупності з директивою ENDMOD і програмою-бібліотекарем. Звичайно бібліотеки складаються з великого числа невеличких за розміром програм. Коли укладач не може знайти глобальний символ (ідентифікатор) у якомусь з файлів, що компонуються, він може здійснити пошук невиявлених символів у бібліотеках. Це означає, що кожна підпрограма повинна знаходитися в окремому файлі, і кожний такий файл повинний асемблювати окремо.

Замість створення окремих файлів кожна підпрограма може бути оточена парою директив MODULE і ENDMOD, що допускають присутність усіх цих підпрограм у тому самому файлі. Це означає, що Асемблер буде інтерпретувати кожний модуль як цілком окрему асемблерну одиницю, тому звертання до зовнішніх символів (ідентифікаторів) повинні бути оголошені відповідно (External), а символи (ідентифікатори), використовувані іншими модулями, повинні бути оголошені як Global (Глобальні). Всі модулі повинні завершуватися директивою ENDMOD. Модулі не можуть бути вкладеними. Усередині модулів можуть знаходитися файли, що включаються, але вони не можуть знаходитися всередині файла, що включається. Число модулів, що можуть знаходитися у файлі, не обмежено. Асемблер буде створювати вихідний файл із розширенням імені pak. Даний файл може бути оброблений тільки програмою-бібліотекарем, проте його обробка за допомогою команд бібліотекаря ADD ALL (Додати всі модулі) і REPLACE ALL (Замінити всі модулі) не представляє утруднень. Далі подано приклад використання директив MODULE і ENDMOD.

```
.MODULE   JUMP_TABLE ;Задати ім'я бібліотеки
.GLOBAL   JUMP_TABLE ;Зробити таблицю
                     ;доступної іншим модулям
                     ;у файлі
.EXTERN   ROUTlNE1   ;Опис зовнішніх
                     ;символів
.EXTERN   ROUTlNE2
JUMP_TABLE:  . WORD ROUTlNE1  ;Завантажити
;адреси підпрограм
.WORD ROUTlNE2
.ENDMOD          ;Задати кінець модуля
.MODULE   ROUTINE1       ;Задати ім'я
                 ;бібліотеки
 .GLOBAL   ROUTlNE1      ;Зробити
                     ;підпрограму
                     ;доступною
ROUTINE1:        NOP
.ENDMOD          ;Задати кінець модуля
.MODULE   ROUTlNE2   ;Задати ім'я бібліотеки
```

```
.GLOBAL  ROUTlNE2  ;Зробити підпрограму
                   ;доступною
ROUTlNE2:      NOP
.ENDMOD        ;Задати кінець модуля
.END           ;Задати кінець файла
```

Якщо поданий вище файл назвати test.asm, то він буде асемблюватися як будь-який інший файл, але вихідний файл одержить ім'я test.pak. Зауважте, як у період асемблювання Асемблер здійснює рестарт на початку кожного модуля.

ENDMOD

Дана директива використовується разом із директивою MODULE і служить ознакою кінця кожного модуля у файлі. За прикладами використання директиви ENDMOD звертайтеся до опису директиви MODULE.

Повідомлення про помилки

CAN' CREATE OUTPUT FILE - DISK MAY BE FULL (Неможливо створення вихідного (результуючого) файла — можливо, диск повний.)

CAN' OPEN INPUT FILE
(Неможливо відкриття вхідного файла.)

CAN' FIND FILENAME. OBJ
(Файл з ім'ям <імені-файла>.OBJ не виявлений.)

SYNTAX ERROR
(Синтаксична помилка.)
Звичайно ця помилка виникає через пропущену кому або круглу дужку.

CAN' RESOLVE OPERAND
(Неможлива ідентифікація звертання до операнда.)

ILLEGAL ADDRESSING MODE
(Невірний режим адресації.)
Адресація операнда з використанням даної форми адресації неприпустима.

ILLEGAL ARGUMENT
(Невірний аргумент.)

MULTIPLY DEFINED SYMBOL
(Символ вже описаний.)

ILLEGAL MNEMONIC
(Невірне мнемонічне позначення.)

TOO LARGE
(Число занадто велике.)

ILLEGAL ASCII DESIGNATOR
(Невірний код таблиці ASCII.)

HEX # AND SYMBOL ARE IDENTICAL
(Шістнадцяткове число і символ є ідентичними.)

UNDEFINED SYMBOL
(Символ не визначений.)

RELATIVE JUMP TOO LARGE
(Занадто далекий відносний перехід.)

EXTRA CHARACTERS AT END OF OPERAND
(Наприкінці операнда присутні зайві символи.)

LABEL VALUE CHANGED BETWEEN PASSES
(Значення мітки змінилося між проходами Асемблера.)

ATTEMPTED DIVISION BY ZERO
(Почата спроба ділення на нуль.)

ILLEGAL EXTERNAL REFERENCE
(Невірне зовнішнє посилання.)

NESTED CONDITIONAL ASSEMBLY UNBALANCE DETECTED
(Виявлений дисбаланс при аналізі вкладених структур Асемблера.)

ILLEGAL REGISTER
(Неприпустиме використання регістра.)

CANT RECOGNIZE NUMBER BASE

(Неможливо визначення основи системи числення числа.)
NOT ENOUGH PARAMETERS
(Не вистачає параметрів.)
ILLEGAL LABEL 1ST CHARACTER
(Невірний перший символ мітки.)
MAXIMUM EXTERNAL SYMBOL COUNT EXCEEDED
(Перевищене максимальне число зовнішніх символів.)
MUST BE IN SAME SECTION
(Повинний знаходитися в тій же самій секції.)
NON-EXISTENT INCLUDE FILE
(Файл, що включається, не існує.)
ILLEGAL NESTED INCLUDE
(Невірне вкладене вмикання файлів.)
NESTED SECTION UNBALANCE
(Дисбаланс вкладених секцій.)
MISSING DELIMITERS ON MACRO CALL LINE (У рядку виклику макроса пропущені обмежувачі.)
MULTIPLE EXTERNALS IN THE SAME OPERAND
(У тому самому операнді виявлене множинне число зовнішніх символів (ідентифікаторів).)
A LABEL IS ILLEGAL ON THIS INSTRUCTION
(Мітка незастосовна до даної команди.)
MACRO STACK OVERFLOW
(Переповнювання стека макросов.)
MISSING LABEL
(Пропущена мітка.)
OPERAND MUST BE DEFINED AS AN 8 BIT RELOCATABLE VALUE
(Операнд може бути описаний як 8-и бітове значення, що переміщається.)
MISSING RIGHT ANGLE BRACKET
(Пропущена права кутова дужка.)
MACRO NAME MUST APPEAR ON SAME LINE AS MACRO DEFINITION
(Ім'я макроса повинно з'являтися на тому ж самому рядку, що й опис макроса.)
ILLEGAL LOCAL LABELS
(Неприпустимо використання локальних міток.)
MISSING MODULE DIRECTIVE
(Пропущена директива опису модуля MODULE.)
MISSING ENDMOD DIRECTIVE
(Пропущена директива ENDMOD (завдання кінця модуля).)
'Module' CAN' BE IN 'Include' FILE
(Директива 'Module' не може знаходитися у файлі, що включається.)
'Endmod' CAN' BE IN 'Include' FILE
(Директива 'Endmod' не може знаходитися у файлі, що включається.)

ДОДАТОК 4

ОПИС КОМПОНОВЩИКА

Далі під лінкером передбачається програма редактора зв'язків.

Лінкер 2500 дозволяє користувачу записати програму мови Асемблера, що містить декілька програмних модулів. Лінкер враховує зовнішні посилання і виконує розміщення в адресному просторі. Він спроможний створювати вихідні файли для всіх найбільше застосовуваних форматів.

Кожний об'єктний файл може мати до 256 різноманітних секцій, визначених користувачем. Лінкер може працювати з 50 різноманітними бібліотечними файлами для визначення зовнішніх символів. Лінкер може працювати в комбінації з 256 вхідними файлами і бібліотечними модулями, і з 256 різноманітними іменами секцій. На розміри секцій обмежень не накладається.

Файли можуть лінковатися відповідно до адреси, що вказується в самому файлі або визначатися в момент лінкування. Спеціальні секції можуть бути використані тільки для посилань. Інформація цих секцій необхідна для процесу лінкування і не включається у вихідний файл. Сек-

ції також можуть бути злінковані в різний час і за різними адресами. Природа лінкера дозволяє створювати коди для виконання з постійним запам'ятовуючим пристроєм (ПЗП) і записувати їх в оперативний запам'ятовуючий пристрій (ОЗП) або ПЗП до моменту запуску.

Лістинг процесу лінкування може бути створений при використанні LINKLIST. Ця директива разом з дозволом запису на диск буде створювати дійсні адреси на момент виконання програми.

Лінкер може бути викликаний у діалогових, командному режимах або під керуванням із файла. Вихідний формат вибирається директивою у вихідному файлі або в параметрах команди LINK. Карта пам'яті, список глобальних символів і помилки лінкування можуть бути записані у файл на диску.

Діалоговий режим

Щоб запустити Лінкер у діалоговому режимі наберіть LINK. Лінкер запросить ім'я вхідного файла. За умовчанням розширення файла OBJ. Після того, як файл буде відкритий на зчитування, Лінкер зажадає зазначити початкову адресу кожної секції ненульового розміру. Цей розмір буде додаватися до адреси, визначеної оператором ORG, що зустрічається у файлі. Натискання тільки клавіші повернення каретки <Enter> змушує Лінкер приєднувати секцію в продовження до попередньої. Знак мінус '–' змушує Лінкер обчисляти положення секції, але не включати її у вихідний файл. Точка з комою ';' після будь-якої введеної адреси змушує розташовувати всі секції в продовження попередньої.

Послідовність вхідних файлів може бути закінчена простим натисканням клавіші повернення каретки замість імені файла. У випадку натискання тільки клавіші повернення каретки ім'я вихідного файла комбінується з іменем першого вхідного файла і розширення, яке залежить від типу вихідного файла.

Після одержання імені вихідного файла Лінкер зажадає ім'я бібліотеки. Лінкер може сприйняти до 50-ти бібліотек із зовнішніми процедурами. Повернення каретки в порожньому рядку закінчить потік імен бібліотечних файлів.

Після того як бібліотеки будуть зазначені, Лінкер зажадає список параметрів.

Режим керування із файла

Файл керування застосовується для довгих послідовностей лінкування або для комбінації цього процесу з іншими. Цей режим аналогічний діалоговому режиму, за винятком того, що діалог – поміщений у файл, і цей файл підставляється Лінкеру. Команда в цьому випадку така:

Link data_file

У цьому випадку файл "data_file.lnk" буде зчитуватися порядково і використовуватися у відповідях на запити. Лінкер припускає розширення lnk для імені файла. Так як порожні рядки у відповідях можуть утруднити контроль умісту файла, то такі рядки можуть бути замінені на символ підкреслення "_". Якщо необхідні додаткові параметри лінкування, то вони розміщаються наприкінці файла аналогічно діалоговому режиму.

У наступному прикладі файл визначає лінкування двох файлів, що містять обидві секції CODE і DATA, які починаються відповідно з адрес 2000H і 4000H. Ім'я вихідного файла взято за умовчанням, а параметри d і 3 дозволяють генерацію файла карти пам'яті і визначають формат вихідного файла як S37 фірми Motorola.

Приклад:

```
file1     ; Перше ім'я файла
2000      ; Почати секцію CODE з адреси 2000H
4000      ; Почати секцію DATA з адреси 4000H
file2     ; Друге ім'я файла
_         ; Продовжити секцію CODE
_         ; Продовжити секцію DATA
_         ; Немає більше файлів
_         ; Використовувати ім'я вихідного файла за умовчанням
_         ; Немає бібліотечних файлів
d3        ;Створити файл на диску у форматі Motorola S37.
```

Режим командного рядка

Лінкер може бути викликаний у командному рядку. Формат такої команди показаний нижче з наступним розшифровуванням елементів:

Link [-q]-cfile1[-Innnn]file2[-Innnn]... [-ofile][-Llibfile][-opt]

-q — Лінкер у режимі Quit. У цьому випадку видається тільки повідомлення про помилки на консоль;

-c — потрібно для вказівки, що буде використаний режим командного рядка, а не керуючого файла.

Слідом за -c йде список вхідних файлів, що складається (для приведеного рядка) із файлів file1 і file2. За кожним файлом може випливати адреса зміщення секцій, що починається з -I. Якщо ця адреса відсутня, то поточна секція є продовженням попередньої з тим же ім'ям.

-o — використовується для вказівки вихідного файла. Цей елемент не обов'язковий. Якщо він відсутній, Лінкер створить вихідний файл з тим же ім'ям, що і перший вхідний, і з розширенням, обумовленим типом форрмату, що генерується;

-l — використовується для завдання бібліотек. Максимум може бути зазначено 50 бібліотек;

-options — визначає додаткові параметри. Знак мінус потрібен на початку списку, і в нього може входити стільки параметрів, скільки необхідно.

Параметри Лінкера

Параметри вказуються в діалоговому режимі після імені бібліотечних файлів. Ці параметри можуть бути задані й у режимі керування з файла, й у командному режимі. Коли зазначено декілька конкуруючих параметрів, останній параметр відміняє дію попереднього.

Параметри (D,S,A,M,Z,X,H,E,T,1,2,3,<CR>=за умовчанням)

D – створити дисковий файл, що містить усі помилки лінкування, символьну таблицю глобальних змінних і карту пам'яті (MAP) завантаження. Файл має те ж ім'я, що і вихідний файл, але з розширенням map.

S – створити символьний файл для процесу налагодження. Цей файл містить усі глобальні символи й їхні розміри. Кожний символ має в довжину 32 букви.

A – створити символьний файл, але із символами в 10 букв. Він використовується для сумісності з Лінкером 2500 версії 3.0.

M – створити символьний файл з метою налагодження у форматі Microtek. Він містить усі символи як глобальні, так і локальні. Для того, щоб цей формат міг бути створений, у вихідному файлі повинна бути присутня директива SYMBOLS ON.

Z – створити символьний файл із метою налагодження у форматі ZAX. Цей файл містить і локальні, і глобальні змінні. Для того, щоб цей формат міг бути створений, у вихідному файлі повинна бути присутня директива SYMBOLS ON.

X – створити виконуваний вихідний файл.

H – створити шістнадцятковий файл формату Intel.

E – створити шістнадцятковий файл розширеного формату Intel.

T – створити шістнадцятковий файл формату Tektronix.

1 – створити вихідний файл формату S19 фірми Motorola.

2 – створити вихідний файл формату S28 фірми Motorola.

3 – створити вихідний файл формату S37 фірми Motorola.

ДОДАТОК 5

ПОВНОЕКРАННИЙ НАЛАГОДЖУВАЧ АСЕМБЛЕРНИХ ПРОГРАМ ДЛЯ ОДНОКРИСТАЛЬНОЇ МІКРОЕОМ КР1816ВЕ51

Налагоджувач-симулятор для програм, написаних на мові Асемблера однокристальних мікроЕОМ КР1816ВЕ51, КМ1816ВЕ31, КМ1816ВЕ51, призначений для логічного налагодження програм, використовуваних зазначеними МікроЕОМ.

Налагоджувач працює на персональних ЕОМ типу IBM PC XT/AT і сумісних із ними ЕОМ, і потребує для роботи не менше 256 Кбайт оперативної пам'яті. Дана конфігурація дозволяє встановити налагоджувач на жорсткий диск.

Налагоджувач дозволяє:

– завантажити для налагодження HEX-файли, вироблювані наявними кросами-засобами (транслятором із мови Асемблера), а також файли чистого двійкового коду, зчитані з ПЗП;

– переглянути на екрані дизасембльований текст завантаженої програми з адресами і кодами команд, область, що імітується ОЗП даних, область зовнішньої пам'яті, пам'яті програм, уміст усіх регістрів ОМЕОМ;

– виконати завантажену програму по кроках із переглядом результатів після кожного кроку й у безупинному режимі із зупином на точках переривання по досягненні що задаються користувачем адрес;

– внести зміни в завантажену програму в мнемонічних позначеннях мови Асемблера, а також у машинних кодах;

– внести зміни у вміст регістрів, прапорів та пам'яті в командному режимі й у режимі повноекранного редагування;

– вивести на пресу або дискові носії дизасембльований текст, дампы пам'яті;

– зберегти вміст будь-якої області пам'яті у файлі на дисковому носії;

– завантажити пам'ять із дискового файла;

– одержати трасування програми;

– визначити час виконання завантаженої програми й її частин за вмонтованим лічильником.

Запуск налагоджувача виконується викликом файла FD51.

Структура виведених повідомлень на екран налагоджувачем подана на рис. Д.5.1.

Відразу після запуску налагоджувач готовий до прийому команд користувача – курсор знаходиться в командному рядку. У нижньому рядку екрана є меню функціональних клавіш F1-F10, які виконують найбільше використовувані команди. Інші команди вводяться користувачем із клавіатури з використанням алфавітно-цифрових клавіш. При введенні цих команд можна користуватися для редагування клавішами [Ins], [Del], [Backspace], [Home], [End], [Esc]. Пам'ятайте, що після початку введення команди і до натискання клавіші [Enter] функціональної клавіші недоступні. Якщо команда невірна, видається повідомлення про помилку і звуковий сигнал.

Опис команд

Призначення функціональних клавіш FD51

Таблиця Д5.1

Клавіша	Призначення
F1	Виконати поточну інструкцію завантаженої програми. Поточна інструкція це інструкція, виділена в вікні дизасембльванного тексту світлим прямокутником. Після виконання на екрані можна відразу спостерігати результати її виконання
F2	Виконувати програму до наступної за адресою за поточною інструкцією. Ця клавіша дозволяє виконати підпрограму або цикл як одну інструкцію, що зручно, тому що не потрібно переглядати вже налагоджені підпрограми
F3	Дозволяє подати числову інформацію на екрані (уміст регістрів і пам'яті) у десятковій, а при повторному натисканні у двійковій формі. (Після запуску інформація подана в шістнадцятковому виді)
F4	Переключає велике вікно пам'яті з внутрішньої (INT RAM) на зовнішню (EXT RAM) і навпаки
F5	Установка точок переривання
F6	Переключає форму представлення пам'яті в вікні в двійкову й навпаки
F7	Перегортає вікно пам'яті даних нагору на один рядок
F8	Перегортає вікно пам'яті даних униз на один рядок
F9	Перегортає вікно пам'яті програм нагору на один рядок
F10	Перегортає вікно пам'яті програм униз на один рядок

Для швидкого перегортання можна користуватися наступними клавішами:

[Home] – перегортає вікно пам'яті даних нагору на одну сторінку.

[End] – перегортає вікно пам'яті даних униз на одну сторінку.

[PgUp] – перегортає вікно пам'яті програм нагору на одну сторінку.

[PgDn] – перегортає вікно пам'яті програм униз на одну сторінку.

Команди налагоджувача

Для швидкого одержання довідки за командами можна ввести команду "H" або натиснути комбінацію клавіш "Ctrl-H".

В описі використовуються наступні позначення:

замкнуті в кутові дужки параметри, наприклад, <адреса>.

замкнуті необов'язкові параметри в квадратні дужки, наприклад, [<адреса>].

Всі числові значення повинні мати шістнадцятковий формат, при цьому не потрібно вказувати букву "h".

L [<тип пам'яті><нач. адреса>,] <файл. спец.>[/A]

Завантажити файл у пам'ять. <Тип пам'яті> може бути I, E або P. Відповідно до цього параметра файл завантажується у внутрішню (Int), зовнішню (Ext) або програмну (Pgm) пам'ять. <Нач. адреса> і <тип пам'яті> указується тільки при завантаженні чистого двійкового коду. При завантаженні файла, виробленого ISIS-II MACRO-ASSEMBLER'ом, потрібно зазначити тільки специфікацію файла і ключ /A.

Приклад:

L I 01F,A:\PGM\T1 - завантажити двійковий файл у внутрішню пам'ять з адреси 01.

S <тип пам'яті><нач. адреса>-<кін. адреса>, <файл. спец.>

Зберегти область пам'яті в дисковому файлі (в усіх командах у якості <файл. спец.> допускається будь-яка коректна в DOS специфікація файла, наприклад COM1). <Нач.адреса> і <кін.адреса> вказують відповідно початок і кінець області, що зберігається. Збережений командою S файл можна потім знову завантажити командою L.

Приклад:

S P 20-642,C\PGMLIB\MYFILE

PRT <тип пам'яті><нач. адреса>-<кін. адреса>[,<файл. спец.>]

Роздрукувати дамп області пам'яті в шістнадцятковому форматі. Якщо не зазначена <файл. спец.>, то дамп виводиться на принтер.

PRTD <нач. адреса>,<кількість команд>[,<файл. спец.>]

Роздрукувати дизасембльований текст, починаючи з <нач. адреси>. Вивід за умовчанням на принтер.

R <номер регістра>=<число>

Занести число в регістр поточного банку. Число повинно бути байтом.

Приклад:

R4=FF

<Ім'я регістра>=<число>

Занести число в регістр спеціального призначення. Можна використовувати наступні імена: A, У, THO, TH1, TLO, TL1, DPH, DPL, DPTR, SP, IP, IE, TMOD, TCON, SCON, SBUF, PC. Число для PC і DPTR може бути і двобайтової величиною.

Приклад:

SP=20 DPTR=FF00

<Ім'я прапора>=<число>

Встановити або скинути прапор у PSW. Імена прапорів: C, AC, FO, SI, SO, OV, P.

Якщо число дорівнює нулю, то прапор скидається, інакше – установлюється.

Приклад:

S1=0

PO <номер порту>=<число>

Занести число в порт. Номер порту може бути 0-3.

Приклад:

PO2=12

D <адреса>

Встановити адресу дизасембльованого тексту в Вікні.

Приклад:

D 0240

<Тип пам'яті><адреса>[-<кін. адреса>]=<число>

Занести число в пам'ять. Якщо зазначений <кін. адреса>, те цим числом заповнюється область пам'яті.

Приклад:

I 22=55 P 0-40=FF

Можливо виникнення неоднозначності при заповненні деяких комірок пам'яті. Наприклад, команда "P C=23" буде сприйнята не як занесення числа 23 у пам'ять програм за адресою 0C, а як команда установки лічильника команд (PC) у значення 23. У цьому випадку потрібно явно зазначити, що це адреса:

P 0C=23.

<Тип пам'яті><адреса>. <номер біта>=<число>

Встановити або скинути біт у пам'яті. <Номер біта> може бути 7–0 (старший біт – 7).

Приклад:

I 20.6=1

<Ім'я регістра>. <номер біта>=<число>

Встановити або скинути біт у регістрі спеціального призначення (A,B,POO-P03,IP,IE,TMOD,TCON,SCON).

Приклад:

TMOD. 3=0

M <тип пам'яті><нач. адреса>

Встановити початкову адресу пам'яті в Вікні.

Приклад:

M I 20 M E 0FF M P 0

G [<нач. адреса>[,<кін. адреса>]]

Виконати програму з <нач. адреси> до <кін. адреси>. Якщо <нач. адреса> не зазначена, виконання починається з поточної команди (поточна команда виділена білим прямокутником). <Кін. адресу> можна не вказувати, якщо використовуються точки переривання. Програму, що виконується, можна зупинити натисканням будь-якої клавіші. G без параметрів можна ввести натисканням <Alt-F10>. Можна зазначити тільки кінцеву адресу, але кома повинна бути присутня.

Приклад:

G 100-FF0 G,2200

T ON [,<файл. спец.>]

Включити трасування програми. За умовчанням трасировочні записи виводяться на принтер.

T OFF

Виключити трасування.

INT <0/1>=<число>

Імітувати високий або низький рівень на входах INTO або INT1.

Приклад:

INT1=0

BA=<адреса>

Встановити нову "точку зупину" для дизасемблірования. Ця команда корисна при перегляді таблиць, що зашили в пам'яті програм, коли при дизасемблірованні "назад" невідомо, відкіля вести дизасемблірования.

RSTC

Скинути лічильник часу виконання програми.

QUIT Вихід у DOS.

RST

Імітується скид процесора.

N

І начебто тільки що запустили FD51.

Повноекранне редагування

Перехід у режим повноекранного редагування здійснюється натисканням клавіші [Enter] без введення команди. Тепер можна переміщати курсор по екрані за допомогою клавіш керування курсором і змінювати вміст регістрів, пам'яті та прапори набором чисел на клавіатурі. Можна змінити також початкову адресу дизасембльованого тексту (поточної інструкції) і початкові адреси Вікон пам'яті (у перших рядках Вікон). Повноекранне редагування можна робити і при десятковому, і при двійковому представленні інформації на екрані. Під час редагування залишаються доступними всі команди, які вводятьсяі функціональними клавішами. Щоб повернутися в командний рядок, натисніть [Enter] знову. Для швидкого переміщення курсору по екрані можна користуватися клавішами [Tab] і [Shift]-[Tab].

Режим Асемблера

Для переходу в режим Асемблера (уведення команд програми, що налагоджується в мнемонічних позначеннях) потрібно в режимі повноекранного редагування помістити курсор у поле поточної інструкції завантаженої програми. Тепер треба набирати мнемоніку (наприклад, "MOV A,#45") і натиснути [Enter]. Якщо мнемоніка вірна, то відповідні їй коди заносяться в пам'ять програм, а Вікно встановлюється на наступну адресу.

При асемблюванні підтримуються імена регістрів спеціального призначення. При виникненні неоднозначності треба числові значення випереджати нулем. Для виходу з режиму Асемблера натиснути клавішу "Q" або відвести курсор із поля поточної інструкції.

Робота з точками переривання

Меню точок переривання визивається клавішею F5 (рис. Д.5.2). Можна встановити одночасно 8 точок переривання. Переривання (зупин) програми, що виконується, відбувається при досягненні зазначеного в колонці "PC" адреси при виконанні умови "Counter" = "Occur". "Counter" – це лічильник, значення якого визначає, скільки разів програма повинна пройти через зазначену адресу, щоб відбувся зупин. "Occur" показує, скільки разів програма проходила через зазначену адресу.

По закінченні редагування, поточні значення точок переривання можна зберегти на диску (клавіша F2). У користувача запитується номер набору точок переривання (0-9). Інформація записується у файл з ім'ям FD51.BRK. Відновити картинку можна клавішею F1, також указавши її номер.

Для повернення в основне меню натисніть F5. Визначивши точки переривання, можна загустити програму командою "G" без параметрів. При зупині програми за перериванням видається повідомлення із вказівкою номера точки переривання.

ДОДАТОК 6

СТИСЛИЙ ОПИС ПРОГРАМИ NC (NORTON COMMADDER).

Запуск Norton Commander здійснюється набором у командному рядку NC.

Можна запустити версію Norton Commander, що потребує менше оперативної пам'яті, але працюючу повільніше. Для цього скористайтеся командою NCSMALL.

Вихід із Norton Commander. Для виходу з Norton Commander треба натиснути клавішу F10. У центрі екрана з'явиться запит на підтвердження, що хочете вийти з Norton Commander. Щоб вийти, натисніть Enter або "Y". Для скасування виходу, натисніть Esc або "N".

Загальний вигляд екрана. Після запуску Norton Commander у верхній частині екрана з'являються два прямокутних Вікна, обмежені подвійною рамкою (далі ці Вікна будемо називати панелями). Нижче цих панелей розташовується запрошення DOS. Туди вводяться звичайні команди DOS. Ще нижче розташовується рядок, що нагадує про призначення функціональних клавіш Norton Commander. У кожній панелі Norton Commander може утримуватися або зміст каталогу на диску, або дерево каталогів на диску, або зведена інформація про диск і каталог, зображених на іншій панелі.

Якщо в панелі міститься зміст каталогу, то поверх панелі виводиться його ім'я, якщо дерево каталогів на диску – то "Tree", а якщо зведена інформація про диск і каталог, то поверх панелі виводиться "Info".

Імена файлів у змісті каталогу виводяться малими літерами, а підкаталоги – заголовними. Справа від імені підкаталогу зображується <SUB-DIR>.

Самий верхній рядок у змісті займає посилання на батьківський каталог (зрозуміло, для кореневого каталогу диска цей рядок відсутній). У полі імені для батьківського каталогу зображується ... , справа — <UP-DIR>.

Один із файлів або каталогів на екрані виділений сірим кольором (на монохромному дисплеї – інверсним зображенням). Будемо називати такий файл або каталог виділеним.

За допомогою клавіш переміщення курсору, PgUp, PgDn можна пересувати виділену ділянку на екрані, вказуючи інший файл або каталог.

Скориставшись клавішею Tab, можна перекласти виділену ділянку в іншу панель Norton Commander.

Якщо виділити який небудь підкаталог і натиснути Enter, то Norton Commander "увійде" у цей підкаталог і виведе його зміст. Для переходу в батьківський каталог треба його виділити (в його полі імені зображається "UP-DIR") і натиснути Enter, дія Norton Commander залежить від розширення імені цього файла, наприклад:

.COM, EXE, BAT – починається виконання файла;

.DOC – викликається Microsoft Word для редагування файла;

.PAS – викликається Turbo-Pascal;

.C – викликається Turbo-C;

.ZIP – видається зміст архіву.

Дія, виконувана для файла з даним розширенням при натисканні клавіші Enter, задається файлом NC.EXT. Цей файл можна редагувати за допомогою Norton Commander. При відсутності файла NC.EXT і для розширень, не згаданих у файлі NC.EXT, ніяких дій виконано не буде.

У нижньому рядку екрана Norton Commander виводить нагадування про значення функціональних клавіш. Нижче коротко описується їхнє призначення.

[F1] – Help – стисла інформація про призначення клавіш при роботі з Norton Commander;

[F2] – User – запуск команд, зазначених у списку, що задається користувачем (у користувальних меню);

[F3] – View – перегляд файла;

[F4] – Edit – редагування файла;

[F5] – Copy – копіювання файла. У середині екрана з'являється запит, куди копіювати файл. За умовчанням файл копіюється в каталог, зображений на іншій панелі. Можна набрати й інше ім'я каталога. Потім для копіювання треба натиснути Enter, для скасування команди – Esc;

[F6] – Renmov – перейменування файла (каталога) або пересилка файла в інший каталог. Можна задати нове ім'я файла, або каталога, що пересилається або ім'я адресного каталога. Для початку перейменування або пересилки треба натиснути Enter, для скасування команди – Esc;

[F7] – WkDir – створення підкаталога

[F8] – Delete – знищення файла або підкаталога.

[F9] – Menu – вивід меню, що містить режими роботи Norton Commander;

[F10] – Quit – вихід з Norton Commander;

Norton Commander дозволяє вибрати групу файлів, що можна скопіювати, перемістити в інший каталог, видалити і т.д.

Вибір окремого файла (тобто включення цього файла в групу) здійснюється натисканням клавіші Ins. При повторному натисканні клавіші Ins вибір файла скасовується.

Для того, щоб вибрати групу файлів за маскою, треба натиснути [+] (плюс на функціональній клавіатурі) і задати маску для вибору. У масці можна використовувати символи * і ?, їхній зміст — той самий, що й у командах MS-DOS.

Для скасування вибору групи файлів за маскою треба натиснути [–] (мінус на функціональній клавіатурі) і задати маску файлів, вибір яких хочете скасувати.

Обрані файли виділяються жовтим кольором (на монохромному дисплеї – підвищеною яскравістю). Насподі панелі з'являються зведення про загальне число виділених файлів й про їхній загальний розмір.

Обрана група файлів за допомогою функціональних клавіш може бути:

[F5] – Copy – зкопійована в інший каталог;

[F6] – Renmov – переміщена в інший каталог або перейменована;

[F8] – Delet – знищена.

Для того, щоб у панелі Norton Commander вивести зміст іншого диска, варто натиснути:

[Alt-F1] – для лівої панелі;

[Alt-F2] – для правої панелі.

Потім, застосувавши клавіші керування курсором, треба вибрати ім'я потрібного диска і натиснути Enter.

Якщо хочете виконати ту або іншу програму або команду DOS, наберіть, як звичайно, цю команду і натисніть Enter. Після закінчення виконання команди екран буде мати той же вигляд, що і до початку роботи.

Переглянути виведені на екран результати виконання команди (вони можуть бути закриті панелями Norton Commander), зможете, натиснувши Ctrl-O. Повторне натискання Ctrl-O відновить панелі на екрані.

NORTON COMMANDER стисла довідка

Перехід в інший каталог – виділити цей каталог і натиснути [Enter].

Перехід на іншу панель – [Tab].

Включити файл у групу – [Ins].

Виключити файл із групи – [Ins].

Включити в групу файли за маскою – натиснути [+] на функціональній клавіатурі і ввести маску.

Виключити з групи файли за маскою – натиснути [–] на функціональній клавіатурі і ввести маску.

Обрану групу файлів можна:

[F5] – скопіювати;

[F6] – перейменувати або перемістити в інший каталог;

[F8] – знищіти;

Керування панелями Norton Commander:

[Ctrl-O] – прибрати панелі з екрана/вивести панелі на екран;

[Ctrl-P] – прибрати одну з панелей (не поточну) з екрана/вивести панель на екран;

[Ctrl-U] – поміняти панелі місцями;

[Ctrl-F1] – прибрати ліву панель з екрана/вивести ліву панель на екран;

[Ctrl-F2] — прибрати праву панель з екрана/вивести праву панель на екран;

[Alt-F1] – вивести в лівій панелі зміст іншого диска;

[Alt-F2] – вивести в правій панелі зміст іншого диска.

Призначення функціональних клавіш:

[F1] – Help – стисла інформація про призначення клавіш;

[F2] – User – вивід меню команд користувача;

[F3] – View – перегляд файла;

[F4] – Edit – редагування файла;

[F5] – Copi – копіювання файла або групи файлів;

[F6] – Renmov – перейменування файла (файлів) або каталога, пересилка файла (файлів) в інший каталог;

[F7] – MkDir – створення підкаталога;

[F8] – Delet – знищення файла, групи файлів або каталога;

[F9] – Menu – меню Norton Commander;

[F10] – Quit – вихід із Norton Commander;

[Shift-F3] – View – перегляд файла. Ім'я файла запитується;

[Shift-F4] – Edit – редагування файла. Ім'я файла запитується;

[Shift-F5] – Copy – копіювання файла або групи файлів. Запитується, які файли і куди треба скопіювати;

[Shift-F6] – Renmov – перейменування файла (файлів) або каталога, пересилка файла (файлів) в інший каталог. Запитується, які файли і як (куди) перейменовувати або пересилати;

[Shift-F9] – зберігання поточних режимів Norton Commander;

[Alt-F7] – пошук файла на диску;

[Alt-F8] – перегляд і повторне виконання раніше введених команд;

[Alt-F9] – переключення з 25 на 43 рядки на екрані;

[Alt-F10] – швидкий перехід в інший каталог.

ДОДАТОК 7

ОПИС ПРОГРАМИ ГЕНЕРАТОРА СИГНАЛІВ (GEN. EXE)

Програма GEN.EXE призначена для формування дискретних відліків сигналу за математичною залежністю, що описує утворюваний сигнал, і зберіганні цих відліків у файлі на диску в двійковому коді.

Програма має дворівневу структуру. Вищий рівень підтримує головне меню генератора. Другий рівень складають обчислювані й інформаційні функції.

Генератор починає роботу після введення командного рядку: GEN.EXE. Вид головного меню генератора зображений на рис. П.7.1.

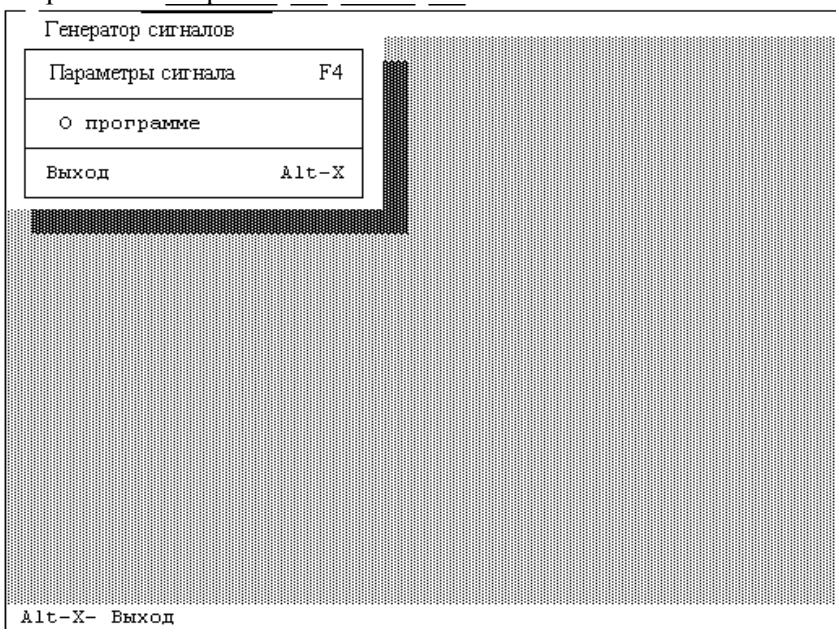

Рис. Д.7.1

Активізувати головне меню можна натисканням клавіші F10. Знаходячись у головному меню, натисканням клавіші "Enter" можна викликати підміню, з якого задаються параметри сигналу, що генерується, і здійснюється вихід із програми.

Вигляд Вікна параметрів сигналу зображений на рис. Д.7.2.

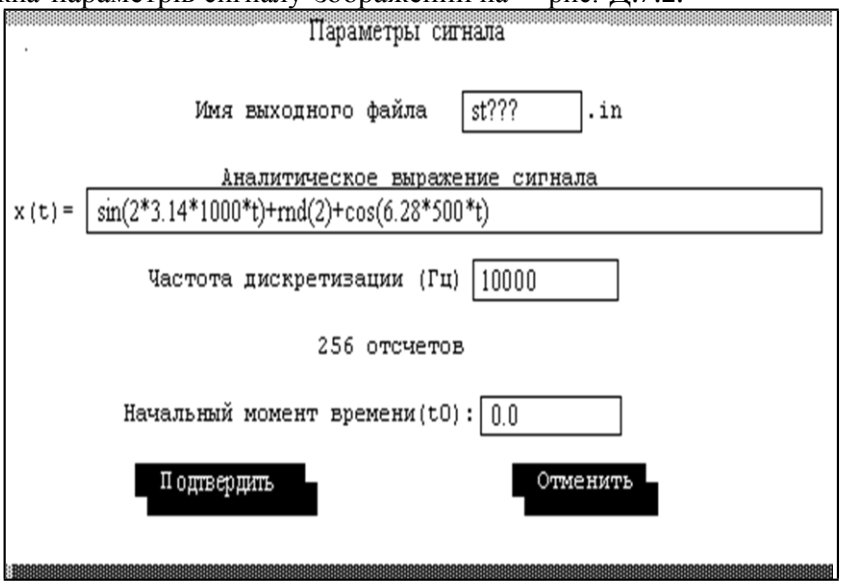

Рис. Д.7.2

Вікно параметрів сигналу можна викликати натисканням клавіші F4, у процесі роботи програми.

Вікно містить чотири поля, у котрі необхідно занести інформацію про сигнал. Перехід з одного поля до іншого здійснюється клавішею "Tab ↔". Насподі Вікна знаходяться дві кнопки: "Підтвердити", натискання якої дозволяє генерувати сигнал, і "Скасувати", що відміняє усі введені параметри.

При успішній генерації сигналу його форма автоматично буде виведена на екран (рис. Д.7.3), а відліки записані у файл із заданим ім'ям.

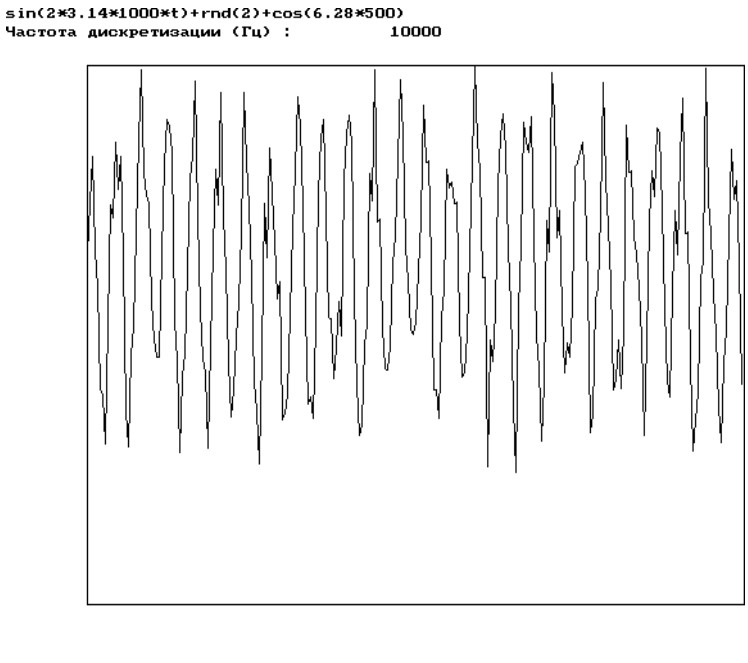

Рис. Д.7.3.

Формулу, що аналітично описує сигнал, необхідно вводити в природному вигляді, використовуючи синтаксис системи, описаний нижче. При цьому припускається використання тільки однієї змінної часу t. Інші параметри повинні задаватися безпосередньо числовими величинами.

Наприклад:

R(10,1,0.5,2,t)+sin(6.28*1000*t)/(cos(6.28*500*t))^2

Генератор "розуміє" наступні символи як:

+ – додавання;

– – віднімання;

* – множення;

/ – ділення;

% – залишок від ділення;

^ – зведення в ступінь;

. – роздільник цілої і дробової частин числа;

, – роздільник аргументів;

( ) – аргументи функції або пріоритет виконання операції;

При формуванні сигналу припустимо використання наступних функцій:

sin – синус x, sin(x)| sin(x*t);

cos – косинус x, cos(x)| cos(x*t);

tan – тангенс x, tan(x)| tan(x*t);

ctg – котангенс x, ctg(x)| ctg(x*t);

asin – арксинус x, asin(x)| asin(x*t);

acos – арккосинус x, acos(x)| acos(x*t);

atan – арктангенс x, atan(x)| atan(x*t);

sinh – синус гіперболічний x, sinh(x)| sinh(x*t);

cosh – косинус гіперболічний x, cosh(x)| cosh(x*t);

tgh – тангенс гіперболічний x, tgh(x)| tgh(x*t);

lg – логарифм десятковий x, lg(x)| lg(x*t);

ln – логарифм натуральний x, ln(x)| ln(x*t);

exp – число e в ступеня x, exp(x)| exp(x*t);

sqrt – корінь квадратний від x, sqrt(x)| sqrt(x*t);

fabs – абсолютне значення x, fabs(x)| fabs(x*t);

floor – округлення числа x у більший бік, не більше x, floor(x)| floor(x*t);

ceil – округлення числа x у менший бік, не менше x, ceil(x)| ceil(x*t);

sign – повертає 1, якщо $x \geq 0$, і –1, якщо $x < 0$, sign(x)| sign(x*t);

delt – повертає 1, якщо x≥0, 0 в інших випадках, delt(x*t);

delti – повертає 0, якщо x≥0, 1 в інших випадках, delti(x*t);

rnd – формує випадкове число, розподілене за рівномірним законом від 0 до 1, в якості аргументу потрібно константа більше 1, rnd(2)| rnd(t);

TR(T,A,t1,t2,τ_1,τ_2,t) – формує трапецевидні відеоімпульси періодом T, амплітудою A, зміщенням t1, тривалістю t2, переднім фронтом τ_1, заднім фронтом τ_2;

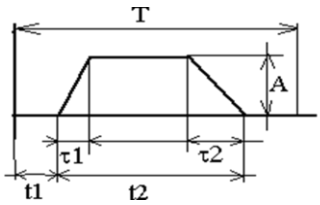

Рис. Д.7.3

T(T,A,t1,τ_1,τ_2,t) – формує трикутні відеоімпульси періодом T, амплітудою A, зміщенням t1, переднім фронтом τ_1, заднім фронтом τ_2;

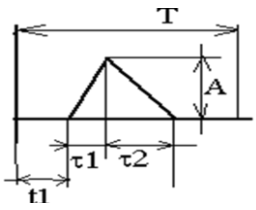

Рис. Д.7.4

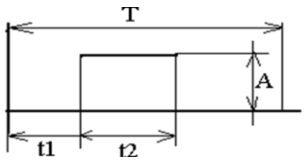

Рис. Д.7.5

R(T,A,t1,t2,t) – формує прямокутні відеоімпульси періодом T, амплітудою A, зміщенням t1, тривалістю t2;

US(T,A,Θ,τ,t) – формує синусоїдальні відеоімпульси періодом T, амплітудою A, кутом отсечки Θ градусів, зміщенням τ;

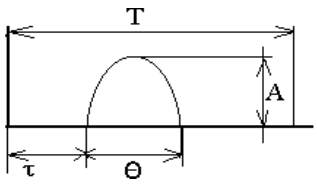

Рис. Д.7.6

Аргументами будь-якої функції можуть бути числа, змінна часу t, вирази, що містять або t, що не містять змінну часу, а також будь-які інші функції.

Для усіх тригонометричних функцій аргумент повинний бути перетворений у радіани (тобто домножений на число 6.28).

Якщо формула, яку вводять за маскою не поміщається в Вікні, система сама здійснить "зсув" формули.

Обмеження: формула не може бути більш 127 символів.

Після введення формули при натисканні клавіші "Enter" автоматично відбувається трансляція формули, синтаксичний і функціональний контроль. При наявності помилок видаються відповідні повідомлення.

Відмова від введення формули здійснюється натисканням клавіші "Esc".

ДОДАТОК 8

СИСТЕМА КОМАНД МІКРОПРОЦЕСОРА КР1810ВМ86

N п/п	Команда	Байти коду команди			Символічний опис і/або зміст операції
		B1	B2	B3 - B6	
1	AAA	00110111			ASCII корекція для додавання
2	AAD	11010101	00001010		ASCII корекція для ділення
3	AAM	11010100	00001010		ASCII корекція для множення
4	AAS	0011111			ASCII корекція для віднімання
5	ADC r,r/m	0001001W	md reg r/m	(disp8/16)	r←r+r/m+CF; додавання з переносом
	r/m,r	0001000W	md reg r/m	(disp8/16)	r/m←r+r/m+CF∧
	r/m,d	1000000W	md 010 r/m	(disp8/16)d8/16	r/m←r/m+d+CF
	r16/m16,d8	10000011	md 010 r/m	data L	r/m←r/m+d+CF
	ac,d	0001010W	data L	(data H)	ac←ac+d
6	ADD r,r/m	0000001W	md reg r/m	(disp8/16)	r←r+r/m; додавання
	r/m,r	0000000W	md reg r/m	(disp8/16)	r/m←r+r/m
	r/m,d	1000000W	md 000 r/m	(disp8/16)d8/16	

N п/п	Команда		Байти коду команди			Символічний опис і/або зміст операції
			B1	B2	B3 - B6	
		r16/m16,d8	10000011	md 000 r/m	data L	r/m←r/m+d
		ac,d	0000010W	data L	(data H)	r/m←r/m+d
						ac←ac+d
7	AND	r,r/m	0010001W	md reg r/m	(disp8/16)	r←r∧r/m; кон'юнкція, I
		r/m,r	0010000W	md reg r/m	(disp8/16)	r/m← r∧r/m
		r/m,d	1000000W	md reg r/m	data L data H	r/m← r/m∧d
		ac,d	0010010W	data L	(data H)	as← as∧d
8	CALL					
	NEAR	sbr	11101000	diff L	diff H	IP← IP+diff; виклик прямий
	NEAR	r16/m16	11111111	md 010 r/m	(disp8/16)	IP← IP+r/m; виклик непрямий
	FAR	sbr	10011010	IP-L	IP-H CS-L CS-H	IP← IP-L, IP-H; CS←CS-L,CS-H; виклик прямий
	FAR	m32	11111111	md 011 r/m	(disp8/16)	IP←m16; CS←m16; виклик непрямий
9	CBW		10011000			Перетворення байта в слово
10	CLC		11111000			CF ← 0; скидання переносу
11	CLD		11111100			DF ← 0; скидання тригера напрямку
12	CLI		11111010			IF ← 0; заборона переривання
13	CMC		11110101			CF ←CF; інверсія переносу
14	CMP	r,r/m	0011101W	md reg r/m	(disp8/16)	r-r/m; порівняння
		r/m,r	0011100W	md reg r/m	(disp8/16)	r/m-r
		r/m,d	1000000W	md reg r/m	(disp8/16) d8/16	r/m-d
		r16/m16,d8	10000011	md reg r/m	data L	r/m-d
		ac,d	0011110W	data L	(data H)	ac-d
15	CMPS		1010011W			Порівняння елементів ланцюжків
16	CWD		10011001			Перетворення слова в подвійне слово
17	DAA		00100111			Десяткова корекція для додавання
18	DAS		00101111			Десяткова корекція для віднімання
19	DEC	r/m	1111111W	md 001 r/m	(disp8/16)	r/m ←r/m-1; декремент
		r	01001reg			
20	DIV	r/m	1111011W	md 110 r/m	(disp8/16)	Ділення чисел без знака
21	ESC	OPCODE,r/m	11011XXX	mdYYY r/m	(disp8/16)	Переключення на сопроцесор
22	HLT		11110100			Зупин
23	IDIV	r/m	1111011W	md 111 r/m	(disp8/16)	Ділення чисел із знаком Множення
24	IMUL	r/m	1111011W	md 101 r/m	(disp8/16)	чисел із знаком Запровадження з
25	IN	ac,port	1110010W	port8		фіксованого порту
		ac,DX	1110110W			Запровадження з порту з непрямою адресацією
26	INC	r/m	1111111W	md 000 r/m	(disp8/16)	r/m ←r/m+1; інкремент
		r	01000reg			r ←r+1
27	INT	type	11001101			Переривання заданого типу
	INTO		001110110			Переривання переповнювання
	INT3		11001100			Переривання типу 3
28	IRET		11001111			Повернення з переривання
29	JA	label	01110111	diff L		IP ←IP+diff L; перейти, якщо вище
30	JAE	label	01110011	diff L		IP←IP+diffL;якщо вище або дорівнює
31	JB	label	01110010	diff L		IP ←IP+diff L; якщо нижче
32	JBE	label	01110110	diff L		IP←IP+diffL;якщо нижче або дорівнює
33	JC	label	01110010	diff L		IP ←IP+diff L; якщо є перенос
34	JCXZ	label	11100011	diff L		IP←IP+diffL;якщо CX дорівнює нулю
35	JE	label	01110100	diff L		IP ←IP+diff L; якщо дорівнює
36	JG	label	01111111	diff L		IP ←IP+diff L; якщо більше
37	JGE	label	01111101	diff L		IP←IP+diffL;якщо більше або дорівнює
38	JL	label	01111100	diff L		IP ←IP+diff L; якщо менше
39	JLE	label	01111110	diff L		IP ←IP+diff L; якщо менше або дорівнює
40	JMP					
	SHORT	label	11101011	diff L		IP ←IP+diff L; прямий короткий
	NEAR	label	11101001	diff L	diff H	перехід
	NEAR	r16/m16	11111111	md 100 r/m	(disp8/16)	IP ←IP+diff L; прямий перехід
	FAR	label	11101010	IP-L	IP-H CS-L CS-H	IP ←IP+r/m; непрямий перехід
	FAR	m32	11111111	md 101 r/m	(disp8/16)	IP ←IP-L, IP-H; CS←CS-L,CS-H; прямий перехід IP←m16; CS←m16; непрямий перехід

N п/п	Команда		Байти коду команди			Символічний опис і/або зміст операції
			B1	B2	B3 - B6	
41	JNA	label	01110110	diff L		IP ←IP+diff L; якщо не вище
42	JNAE	label	01110011	diff L		IP ←IP+diff L; якщо не вище або дорівнює
43	JNB	label	01110111	diff L		IP ←IP+diff L; якщо не нижче
44	JNC	label	01110011	diff L		IP ←IP+diff L; якщо немає переносу
45	JNE	label	01110101	diff L		IP ←IP+diff L; якщо не дорівнює
46	JNG	label	01111110	diff L		IP ←IP+diff L; якщо не більше
47	JNGE	label	0111110	diff L		IP ←IP+diff L; якщо не більше або дорівнює
48	JNL	label	01111101	diff L		IP ←IP+diff L; якщо не менше
49	JNLE	label	01111111	diff L		IP ←IP+diff L; якщо не менше або дорівнює
50	JNO	label	01110001	diff L		IP ←IP+diff L; якщо немає переповнювання
51	JNP	label	01111011	diff L		IP ←IP+diff L; якщо немає паритету
52	JNS	label	01111001	diff L		IP ←IP+diff L; якщо результат позитивний
53	JNZ	label	01110101	diff L		IP ←IP+diff L; якщо не нуль
54	JO	label	01110000	diff L		IP ←IP+diff L; якщо є переповнювання
55	JP	label	01111010	diff L		IP ←IP+diff L; якщо є паритет
56	JPE	label	01111010	diff L		IP ←IP+diff L; якщо паритет парний
57	JPO	label	01111011	diff L		IP ←IP+diff L; якщо паритет непарний
58	JS	label	01111000	diff L		IP ←IP+diff L; якщо результат від'ємний
59	JZ	label	01110100	diff L		IP ←IP+diff L; якщо нуль
60	LAHF		10011111			AH←FL; пересилка молодшого байта F у AH
61	LDS	r16,m32	11000101	md reg r/m	(disp8/16)	r,DS←m32; завантаження покажчика адреси
62	LEA	r16,m	10001101	md reg r/m	(disp8/16)	r← EA; завантаження ефективної адреси
63	LES	r16,m32	11000100	md reg r/m	(disp8/16)	r,ES←m32; завантаження покажчика адреси
64	LODS		1010110W			Завантаження елемента ланцюжка
65	LOOP	label	11100010	diff L		IP ←IP+diff L; зациклить
66	LOOP/ LOOPZ	label	11100001	diff L		IP ←IP+diff L; Зациклить якщо дорівнює
67	LOOPN ELOOP NZ	label	11100000	diff L		IP ←IP+diff L; Зациклить, якщо не дорівнює
68	MOV	r,r/m	1000101W	md reg r/m	(disp8/16)	r←r/m; пересилка
		r/m,r	1000100W	md reg r/m	(disp8/16)	r/m←r;
		r/m,d	1100011W	md 000 r/m	(disp8/16) d8/16	r/m←d;
		r,d	1011Wreg1	data L	(data H)	r←d;
		ac,m	010000W	disp L	disp H	as←m; при прямій адресації m←ac;
		m,ac	1010001W	disp L	disp H	при прямій адресації sr←r/m; запис у сегментний регістр
		sr,r/m	10001110	md 0sr r/m	(disp8/16)	r/m←sr; читання із сегментного регістра
		r/m,sr	10001100	md 0sr r/m	(disp8/16)	
69	MOVS		1010010W			Пересилка елементів ланцюжка
70	MUL	r/m	1111011W	md 100 r/m	(disp8/16)	Множення чисел без знака
71	NEG	r/m	1111011W	md 011 r/m	(disp8/16)	Зміна знака числа
72	NOP		10010000			Порожня операція
73	OR	r,r/m	0000101W	md reg r/m	(disp8/16)	r←rVr/m; диз'юнкція, АБО
		r/m,r	0000100W	md reg r/m	(disp8/16)	r/m←r/mVr
		r/m,d	1000000W	md 001 r/m	(disp8/16) d8/16	r/m←r/mVd
		ac,d	0000110W	data L	(data H)	ac←acVd
74	OUT	ac,port	1110011W	port8		Висновок у фіксований порт
		ac,DX	1110111W			Висновок у порт при непрямій адресації
75	POP	r/m	10001111	md 000 r/m	(disp8/16)	Читання зі стека
		r	01011reg			
		sr	000sr111			

N п/п	Команда		Байти коду команди			Символічний опис і/або зміст операції
			B1	B2	B3 - B6	
76	POPF		10011101			Завантаження регістра прапорів зі стека
77	PUSH	r/m	11111111	md 110 r/m	(disp8/16)	Запис у стек
		r	01010reg			
		sr	000sr110			
78	PUSHF		10011100			Запис регістра прапорів у стек
79	RCL	r/m,1	1101000W	md 010 r/m	(disp8/16)	Циклічний зсув
		r/m,CL	1101001W	md 010 r/m	(disp8/16)	уліво через CF
80	RCR	r/m,1	1101000W	md 011 r/m	(disp8/16)	Циклічний зсув
		r/m,CL	1101001W	md 011 r/m	(disp8/16)	управо через CF
81	RET	(NEAR)	11000011			Повернення з підпрограми
		d(NEAR)	11000010	data L	data H	Повернення з додаванням константи до SP
		(FAR)	11001011			
		d(FAR)	11001010	data L	data H	Повернення з підпрограми Повернення з додаванням константи до SP
82	ROL	r/m,1	1101000W	md 000 r/m	(disp8/16)	Циклічний зсув
		r/m,CL	1101001W	md 000r/m	(disp8/16)	уліво
83	ROR	r/m,1	1101000W	md 001 r/m	(disp8/16)	Циклічний зсув
		r/m,CL	1101001W	md 001 r/m	(disp8/16)	управо
84	SAHF		10011110			Пересилка AH у регістр F
85	SAL	r/m,1	1101000W	md 100 r/m	(disp8/16)	Арифметичний зсув
		r/m,CL	1101001W	md 100 r/m	(disp8/16)	уліво
86	SAR	r/m,1	1101000W	md 111 r/m	(disp8/16)	Арифметичний зсув
		r/m,CL	1101001W	md 111 r/m	(disp8/16)	управо
87	SBB	r,r/m	0001101W	md reg r/m	(disp8/16)	r←r-r/m-CF; віднімання з позикою
		r/m,r	0001100W	md reg r/m	(disp8/16)	r/m←r-r/m-CF
		r/m,d	1000000W	md 011 r/m	(disp8/16) d8/16	r/m←r/m-d-CF
		r16/m16,d8	10000011	md 011 r/m	data L(data H)	r/m←r/m-d-CF
		ac,d	0001110W	data L		ac←ac-d-CF
88	SCAS		1010111W			Сканувати елемент ланцюжка
89	SHL	r/m,1	1101000W	md 100 r/m	(disp8/16)	Логічний зсув
		r/m,CL		md 100 r/m	(disp8/16)	уліво
90	SHR	r/m,1	1101001W	md 101 r/m	(disp8/16)	Логічний зсув
		r/m,CL	1101001W	md 101 r/m	(disp8/16)	управо
91	STC		11111001			CF←1; установка прапора переносу
92	STD		11111101			DF←1; установка тригера напрямку
93	STI		11111011			IF←1; дозвіл переривання
94	STOS		1010101W			[DI] ←ac; запам'ятовування ac у ланцюжку
95	SUB	r,r/m	0010101W	md reg r/m	(disp8/16)	r←r-r/m; віднімання
		r/m,r	0010100W	md reg r/m	(disp8/16)	r/m←r-r/m
		r/m,d	1000000W	md 101 r/m	(disp8/16) d8/16	r/m←r/m-m
		r16/m16,d8	10000011	md 101 r/m	data L(data H)	r/m←r/m-m
		ac,d	0010110W	data L		ac←ac-d;
96	TEST	r,r/m	1000010W	md reg r/m	(disp8/16)	r∧r/m; перевірка, що не руйнує I
		m,r	1000010W	md reg r/m	(disp8/16)	r∧r/m r/m
		r/m,d	1111011W	md 000 r/m	(disp8/16) d8/16	d ac∧d
		ac,d	1010100W	data L	(data H)	Чекання
97	WAIT		00111011			AX↔r; обмін
98	XCHG	AX,r	10010reg			r↔AX
		r,AX	10010reg			r↔r/m
		r,r/m	1000011W	md reg r/m	(disp8/16)	m↔r
		m,r	1000011W	md reg r/m	(disp8/16)	Перетворення байта з AL
99	XLAT		11010111			r←r⊕r/m; підсумовування за модулем 2
100	XOR	r,r/m	0011001W	md reg r/m	(disp8/16)	r/m←r⊕r/m;
		r/m,r	0011000W	md reg r/m	(disp8/16)	r/m←r/m⊕d
		r/m,d	1000000W	md 110 r/m	(disp8/16) d8/16	as←ac⊕d
		ac,d	0011010W	data L	(data H)	
		Префікси:				Префікс блокування шини
		LOCK	11110000			Повторювати ланцюгову операцію
		REP/REPE/REPZ	11110011			
		REPNE/REPNZ	11110010			Те ж саме
		SEG	001sr110			Префікс заміни сегмента

ДОДАТОК 9

СИСТЕМА КОМАНД МП 1816ВЕ51

Таблиця Д9.1

Група команд передачі даних		
Мнемокод	Операція	Назва команди
MOV A, Rn	(A)←(Rn)	Пересилка в акумулятор із регістра (n=0□7)
MOV A, ad	(A)←(ad)	Пересилка в акумулятор байта, що адресується прямо
MOV A, @Ri	(A)←((Ri))	Пересилка в акумулятор байта з РПД (i=0,1)
MOV A, #d	(A)←#d	Завантаження в акумулятор константи
MOV Rn, A	(Rn)←(A)	Пересилка в регістр з акумулятора
MOV Rn, ad	(Rn)←(ad)	Пересилка в регістр байта, що адресується прямо
MOV Rn, #d	(Rn)←#d	Завантаження в регістр константи
MOV ad,A	(ad)←(A)	Пересилка по прямій адресі акумулятора
MOV ad, Rn	(ad)←(Rn)	Пересилка по прямій адресі регістра
MOV add, ads	(add)←(ads)	Пересилка байта, що адресується прямо, по прямій адресі
MOV ad, @Ri	(ad)←((Ri))	Пересилка байта з РПД по прямій адресі
MOV ad, #d	(ad)←#d	Пересилка по прямій адресі константи
MOV @Ri, A	((Ri))←(A)	Пересилка в РПД з акумулятора
MOV @Ri, ad	((Ri))←(ad)	Пересилка в РПД байта, що адресується прямо
MOV @Ri, #d	((Ri))←#d	Пересилка в РПД константи
MOV DPTR, #d16	(DPTR)←#d16	Завантаження покажчика даних
MOVC A, @A+DPTR	(A)←((A)+ +(DPTR))	Пересилка в акумулятор байта з ПП
MOVC A, @A+PC	(PC)←(PC)+1 (A)←((A)+PC))	Пересилка в акумулятор байта з ПП
MOVX A, @Ri	(A)←((Ri))	Пересилка в акумулятор байта з ВПД
MOVX A, @DPTR	(A)←((DPTR))	Пересилка в акумулятор байта з розширеної ВПД
MOVX @Ri,A	((Ri))←(A)	Пересилка у ВПД з акумулятора
MOVX @DPTR, A	((DPTR))←(A)	Пересилка в розширену ВПД з акумулятора
PUSH ad	(SP)←(SP)+1 ((SP))←(ad)	Завантаження в стек
POP ad	(ad)←(SP) (SP)←(SP)-1	Витяг зі стека
XCH A, Rm	(A)↔(Rn)	Обмін акумулятора з регістром
XCH A, ad	(A)↔(ad)	Обмін акумулятора з байтом, що адресується прямо
XCH A, @Ri	(A)↔((Ri))	Обмін акумулятором із байтом із РПД
XCHD A, @Ri	(A0-3)↔$((Ri)_{0-3})$	Обмін молодшої тетради акумулятора з молодшою тетрадою байта РПД
ADD A,Rn	(A)←(A)+(Rn)	Додавання акумулятора з регістром (n=0□7)
ADD A,ad	(A)←(A)+(ad)	Додавання акумулятора до байта, що адресується прямо
ADD A,@Ri	(A)←(A)+((Ri))	Додавання акумулятора до байта із РПД (i=0,1)
ADD A,#d	(A)←(A)+#d	Додавання акумулятора з константою
ADC A,Rn	(A)←(A)+(Rn)+(C)	Додавання акумулятора з регістром і переносом
ADC A,ad	(A)←(A)+(ad)+(C)	Додавання акумулятора з байтом, що адресується прямо, і переносом
ADC A,@Ri	(A)←(A)+((Ri))+(C)	Додавання акумулятора з байтом із РПД і переносом
ADC A,#d	(A)←(A)+#d+(C)	Додавання акумулятора з константою і переносом
DA A	якщо (A0-3)>9V((AC)=1), то (A0-3)←(A0-3)+6, потім якщо (A4-7)>9V((C)=1), то (A4-7)←(A4-7)+6	Десяткова корекція з акумулятором
SUB A,Rn	(A)←(A)-(C)-(Rn)	Віднімання з акумулятора регістра і позики
SUB A,adf	(A)←(A)-(C)-((ad))	Віднімання з акумулятора байта і позики
SUB A,@Ri	(A)←(A)-(C)-((Ri))	Віднімання з акумулятора байта РПД і позики
SUB A,d	(A)←(A)-(C)-#d	Віднімання з акумулятора константою і позики
INC A	(A)←(A)+1	Інкремент акумулятора
INC Rn	(Rn)←(Rn+1)	Інкремент регістра
INC ad	(ad)←(ad+1)	Інкремент байта, що адресується прямо
INC @Ri	((Ri))←((Ri))+1	Інкремент байта в РПД
INC DPTR	(DPTR)←(DPTR)+1	Інкремент покажчика даних
DEC A	(A)←(A)-1	Декремент акумулятора
DEC Rn	(Rn)←(Rn)-1	Декремент регістра
DEC ad	(ad)←(ad)-1	Декремент байта, що адресується прямо
DEC @Ri	((Ri))←((Ri))-1	Декремент байта в РПД
MUL AB	(B)(A)←(A)x(B)	Множення акумулятора на регістр У
DIF AB	(A). (B)←(A)/(B)	Ділення акумулятора на регістр У

Таблиця Д.9.3

Група команд логічних операцій		
Мнемокод	Операція	Назва команд
ANL A,Rn	(A) ←(A)∧(Rn)	Логічне І акумулятора в регістра
ANL A,ad	(A) ←(A)∧(ad)	Логічне І акумулятора в байта, що адресується прямо
ANL A,@Ri	(A) ←(A)∧((Ri))	Логічне І акумулятора в байта з РПД
ANL A,#d	(A) ←(A)∧#d	Логічне І акумулятора в константи
ANL ad,A	(ad) ←(ad)∧(A)	Логічне І в акумулятора
ANL ad,#d	(ad) ←(ad)∧#d	Логічне І байта, що адресується прямо в константи
ORL A,Rn	(A) ←(A)∨(Rn)	Логічне АБО акумулятора і регістра
ORL A,ad	(A) ←(A)∨(ad)	Логічне АБО акумулятора байта, що адресується прямо
ORL A,@Ri	(A) ←(A)∨((Ri))	Логічне АБО акумулятора і байта з РПД
ORL A,#d	(A) ←(A)∨#d	Логічне АБО акумулятора і константи
ORL ad,A	(ad) ←(ad)∨(A)	Логічне АБО байта, що адресується прямо й акумулятора
ORL ad,#d	(ad) ←(ad)∨#d	Логічне АБО байта, що адресується прямо і константи
XRL A,Rn	(A) ←(A)∀(Rn)	Що виключає АБО акумулятора і регістра
XRL A,ad	(A) ←(A)∀(ad)	Що виключає АБО акумулятора і байта, що адресується прямо
XRL A,@Ri	(A) ←(A)∀((Ri))	Що виключає АБО акумулятора і байта з РПД
XRL A,#d	(A) ←(A)∀#d	Що виключає АБО акумулятора і константи
XRL ad,A	(ad) ←(ad)∀(A)	Що виключає АБО байта, що адресується прямо й акумулятора
XRL ad,#d	(ad) ←(ad)∀#d	Що виключає АБО байта, що адресується прямо і константи
CLR A	(A) ←0	Скидання акумулятора
CPL A	(A) ←(A)	Інверсія акумулятора
RL A	(An_{+1})←(An), n=0□6, (A0) ←(A7)	Зсув акумулятора вліво циклічний
RL A	(An_{+1})←(An), n=0□6, (A0) ←(C), (C) ←(A7)	Зсув акумулятора вліво через перенос
RR A	(An)←(An_{+1}), n=0□6, (A7) ←(A0)	Зсув акумулятора вправо циклічний
RRC A	(An)←(An_{+1}), n=0□6, (A7) ←(C), (C) ←(A0)	Зсув акумулятора вправо через перенос
SWAP A	(A0-3) ↔(A4-7)	Обмін місцями тетрад в акумуляторі

Таблиця Д.9.4

Група команд операцій із бітами.		
Мнемокод	Операція	Назва команд
CLR C	(C) ←0	Сброс переносу
CLR bit	(b) ←0	Скидання біта
SETB C	(C) ←1	Установка переносу
SETB bit	(b) ←1_	Установка біта
CPL C	(C)← (C)	Інверсія переносу
CPL bit	(b) ←(b)	Інверсія біта
ANL C,bit	(C) ←(C)∧(b)	Логічне І біта і преноса
ANL C,/bit	(C) ←(C)∧(b)	Логічне І інверсія біта і переносу
ORL C,bit	(C) ←(C)∨(b)	Логічне АБО біта і переносу
ORL C/bit	(C) ←(C)∨(b)	Логічне АБО інверсії біта і переносу
MOV C,bit	(C) ←(b)	Пересилка біта в перенос
MOV bit,C	(b) ←(C)	Пересилка переносу в біт

Таблиця Д.9.5

Група команд передачі керування		
Мнемокод	Операція	Назва команд
LJMP ad16	(PC) ← ad16	Довгий перехід у повному обсязі пам'яті програм
AJMP ad11	(PC) ← (PC)+2; (PC0-10) ← ad11	Абсолютний перехід усередині сторінки в 2 Кбайта
SJMP rel	(PC) ← (PC)+2; (PC) ← (PC)+rel	Короткий відносний перехід усередині сторінки в 256 байт
JMP @A+DPTR	(PC) ← (A)+(DPTR)	Непрямий відносний перехід
JZ rel	(PC)←(PC)+2, якщо (A)=0, те (PC) ← (PC)+rel	Перехід, якщо акумулятор дорівнює нулю
JNZ rel	(PC)←(PC)+2, якщо (A)≠0, те (PC) ← (PC)+rel	Перехід, якщо акумулятор не дорівнює нулю
JC rel	(PC)←(PC)+2, якщо (C)=1, те (PC) ← (PC)+rel	Перехід, якщо перенос дорівнює одиниці
JNC rel	(PC)←(PC)+2, якщо (C)=0, те (PC) ← (PC)+rel	Перехід, якщо перенос дорівнює нулю
JB bit, rel	(PC)←(PC)+3, якщо (b)=1, те (PC) ← (PC)+rel	Перехід, якщо біт дорівнює одиниці
JNB bit, rel	(PC)←(PC)+3, якщо (b)=0, те (PC) ← (PC)+rel	Перехід, якщо біт дорівнює нулю
JBC bit, rel	(PC)←(PC)+3, якщо (b)=1, те (b)←0 і (PC)←(PC)+rel	Перехід, якщо біт установлений, із наступним скиданням біта
DJNZ Rn, rel	(PC)←(PC)+2,(Rn)←(Rn)-1, якщо (Rn)≠0, то (PC)←(PC)+rel	Декремент регістра і перехід, якщо не нуль

DJNZ ad, rel	(PC)←(PC)+2,(ad)←(ad)-1, якщо (ad)≠0, то (PC)←(PC)+rel	Декремент байта, що адресується прямо, і перехід, якщо не нуль
CJNE A, ad, rel	(PC)←(PC)+3, якщо (A)≠(ad), то (PC) ← (PC)+rel, якщо (A)<(ad), то (C)←1, інакше (C)←0	Порівняння акумулятора з байтом, що адресується прямо, і перехід, якщо не дорівнює
CJNE A, #d, rel	(PC)←(PC)+3, якщо (A)≠#d, то (PC) ← (PC)+rel, якщо (A)<#d, то (C)←1, інакше (C)←0	Порівняння акумулятора з константою і перехід, якщо не дорівнює
CJNE Rn, #d, rel	(PC)←(PC)+3, якщо (Rn)≠#d, то (PC) ← (PC)+rel, якщо (Rn)<#d, то (C)←1, інакше (C)←0	Порівняння регістра з константою і перехід, якщо не дорівнює
CJNE @Ri, #d,rel	(PC)←(PC)+3, якщо ((Ri))≠#d, то (PC) ← (PC)+rel, якщо ((Ri))<#d, то (C)←1, інакше (C)←0	Порівняння байта в РПД з константою і перехід, якщо не дорівнює
LCALL ad16	(PC)←(PC)+3, (SP)←(SP)+1, ((SP))←(PC0-7), (SP)←(SP)+1 ((SP))←(PC8-15),(PC)←ad16	Довгий виклик підпрограми
ACALL ad11	(PC)←(PC)+2, (SP)←(SP)+1, ((SP))←(PC0-7), (SP)←(SP)+1 ((SP))←(PC8-15),(PC0-10)←ad11	Абсолютний виклик підпрограми в межах сторінки в 2 Кбайта
RET	(PC8-15)←((SP)), (SP)←(SP)-1, (PC0-7)←((SP)), (SP)←(SP)-1,	Повернення з підпрограми
RETI	(PC8-15)←((SP)), (SP)←(SP)-1, (PC0-7)←((SP)), (SP)←(SP)-1,	Повернення з підпрограми, опрацювання переривання
NOP	(PC)←(PC)+1	Холоста команда