ЛАБОРАТОРНА РОБОТА №4-1

Організація взаємозв'язку програм мовою Асемблера з програмами мовою Паскаль

Мета роботи – ознайомлення з організацією взаємозв'язку програм мовою Ассемблера з програмами мовою Паскаль, вивчення методів опрацювання складних структур даних

4-1.1. Зміст роботи

Робота виконується на двох заняттях. На першому занятті студенти, використовуючи приклади програм у файлах lab4.pas та BigShow.asm, вивчають правила взаємозв'язку окремо скомпільованих програм мовою Асемблера та мовою Паскаль. Шляхом модифікації Паскаль програми ознайомлюються з машинною структурою різних типів даних, також з індивідуальним варіантом завдання щодо реалізаціїї елементарних операцій з надвеликими цілими числами, які застосовуються в сучасній комп'ютерній криптографії.

На другому занятті складають власну підпрограму мовою Асемблера для виконання елементарної операції з надвеликими числами згідно варіанта, складають мовою Паскаль тестову програму з тестовими даними для виклику асемблерної процедури, використовують процедуру BigShow для відображення тестових даних і результатів виконання відповідної операції.

4-1.2. Теоретичні відомості

Взаємозв'язок програм мовою Асемблера і мовою Паскаль

Взаємозв’язок програм мовою високого рівня і програм мовою Асемблера здійснюється двома наступними методами:

- вставками асемблерного тексту в текст програми мовою високого рівня;

- використання окремо скомпільованих асемблерних процедур, які на мові високого рівня об’являються як зовнішні (*external*).

З першим методом студенти ознайомлювались у Лабораторній роботі №2. Використання же окремо скомпільованих асемблерних процедур пов’язане з виконанням додаткових спеціальних вимог до програм як мовою Паскаль, так і мовою Асемблера.

Додаткові вимоги до програм мовою Паскаль в інтегрованому середовищі Турбо Паскаль

1) В програмі повинна бути директива компілятора **L**, яка забезпечить підключення об'єктного файла, сформованого Асемблером (Tasm або Masm). Наприклад, *{$L BigShow.obj}*, при цьому файл BigShow.obj повинен знаходитись у тому ж каталозі, що і компілятор Паскаля. Якщо файл BigShow.obj знаходиться в іншому каталозі, тоді необхідно вказати повний шлях деревом каталогу, наприклад, *{$L / ..шлях.. /BigShow.obj}*.
2) В розділі опису підпрограм і функцій Паскаль програми за правилами мови Паскаль повинні бути описані асемблерні процедури. Додатковою вимогою є доповнення опису кожної процедури ключовим словом *External*, наприклад:

```
Procedure BigShow(Var m; len:word); external;
Function Xyz(x,y:integer):integer; external;
```

3) Для виклику підпрограм використовується внутрішньосегментна або міжсегментна команда процесора *CALL*. Компілятор Паскаля сам вирішує, яку і коли команду використовувати відповідно до контексту програми і ключів компіляції інтегрованового середовища. Тип процедури (*near* чи *far*) мовою Асемблера компілятором при цьому не враховується. Тому, для запобігання неузгодженості і досягнення максимального універсалізму, рекомендується всі процедури мовою Асемблера оформляти за типом *far*, а в Паскаль-програмі перед описом асемблерних процедур помістити директиву far-компіляції *{$F+}*. Після

опису асемблерних процедур рекомендується помістити директиву "ближньої" адресації *{$F-}*, наприклад:

```
{$F+}
Procedure BigShow(Var m; len:word); external;
Function Xyz(x,y:integer):integer; external;
{$F-}
```

Додаткові вимоги до програм мовою Асемблера

1) Поле операндів директиви END

Програма мовою Асемблера не повинна бути основною, а це означає, що поле операндів директиви *END* має бути порожнім.

2) Імена логічних сегментів

Повинні використовуватися певні імена логічних сегментів:

- _Text – для сегменту кодів;
- _Data – для сегменту даних, початкові значення яких визначаються під час завантаження;
- _Bss – для сегменту даних, значення яких під час завантаження не визначені.

Примітка. Не рекомендується використовувати інші назви логічних сегментів, навіть якщо це дозволяє інтегроване середовище Турбо Паскаль, оскільки вищезазначені назви є стандартними для всіх компіляторів, у тому числі і для інших мов програмування. Крім того, можуть виникнути проблеми з початковими значеннями даних.

3) Операнди директив SEGMENT

Тип об’єднання для всіх сегментів – *Public*.

Класи сегментів:

- для сегментів _Text – *'Code'*
- для сегментів _Data – *'Data'*
- для сегментів _Bss – *'Bss'*

Розрядність: за умови використання директиви .386 або будь-якої іншої директиви 32-розрядного мікропроцесора, обов’язково задається операнд *Use16*. Приклад подання директиви опису логічного сегмента кодів:

```
_Text        Segment     Word  Public  'Code'  Use16
```

4) Доступ до параметрів процедур

Параметри перед викликом процедури записуються в стек в порядку їх слідування (конвенція Паскаля): спочатку перший, потім другий і т.д., а після останнього параметра командою CALL в стек записується адреса повернення (рис.4-1.1).

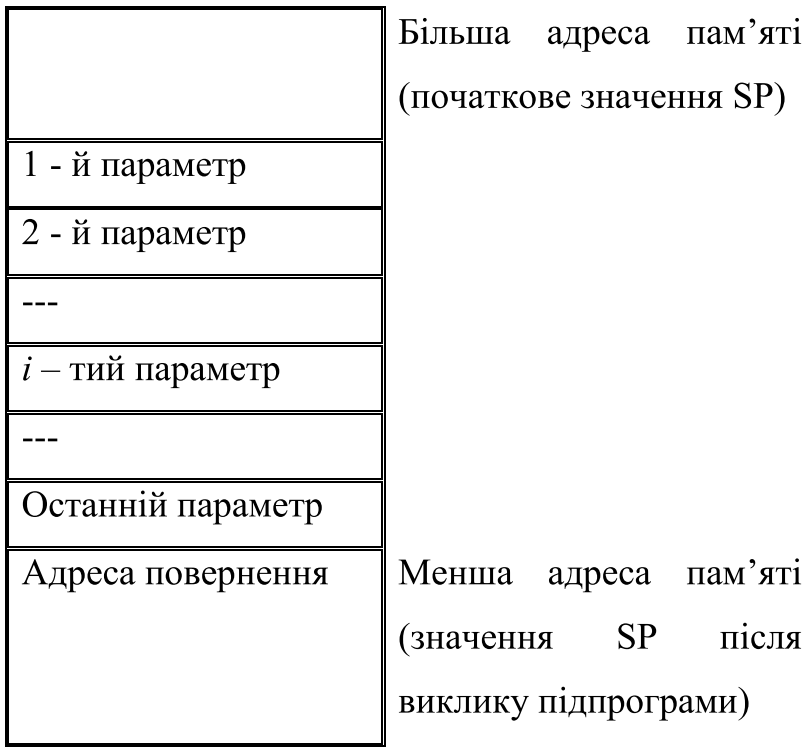

Рис.4-1.1. Послідовність запису параметрів в стек при виклику процедури

Можливі два варіанти запису параметра в стек:

- в стек записується повна логічна адреса параметра (сегментна складова та зміщення в сегменті) – 2 слова;
- в стек записується значення параметра – не більше 2-х слів (4-х байтів).

Перший варіант (повна логічна адреса) використовується у випадках:

- коли параметр передається за посиланням;

- коли параметр передається за значенням, але займає в пам'яті більше 4-х байтів.

Якщо параметр передається за значенням і займає в пам'яті один байт, тоді в стек записуються 2 байти. Таким чином, будь-який параметр займає в стеку одне або два слова.

Для доступу до параметрів процедур використовується базовий регістр стеку *BP*. Оскільки в програмах широко використовується виклик одних процедур з інших, то необхідно зберегти попередній вміст цього регістра, а потім записати в нього вміст покажчика стеку (*SP*):

```
push    bp
mov     bp, sp
```

В результаті, для адресації останнього аргументу можна використати адресний вираз [BP+6] (4 байти – адреса повернення і ще два байти – попередній вміст регістра BP). Для адресації решти аргументів (пробігаючи в зворотному напрямку від останнього до першого) необхідно кожний раз збільшувати константу на 2 або 4, в залежності від кількості байтів попереднього аргументу. Нехай, наприклад, маємо наступний опис зовнішньої асемблерної процедури в програмі мовою Паскаль:

```
Procedure pp(var arg1, arg2; arg3:char, arg4:integer); external;
```

Тоді адреса аргументу arg4 в стеку буде [BP+6], аргументу arg3 – [BP+8], аргументу arg2 – [BP+10], аргументу arg1 – [BP+14]. При багаторазових звертаннях до аргументів в асемблерній програмі доцільно задати наступну послідовність директив *EQU*:

```
Arg4    EQU     [BP+6]
Arg3    EQU     [BP+8]
Arg2    EQU     [BP+10]
Arg1    EQU     [BP+14]
```

або

```
Arg4    EQU   word ptr [BP+6]
Arg3    EQU   byte ptr [BP+8]
Arg2    EQU   dword ptr [BP+10]
Arg1    EQU   dword ptr [BP+14]
```

За умови повного розуміння механізму передачі та доступу до параметрів, можна використати спеціальну директиву Асемблера *ARG*, яка заміняє дії вищезазначених директив EQU:

```
ARG   Arg4:Word, Arg3:Byte, Arg2:Dword, Arg1:Dword
```

5) Вміст регістрів процесора

Вміст усіх сегментних регістрів, а також регістра BP(EBP) повинен не змінюватись або бути відновленим при поверненні з підпрограми.

6) Доступ до змінних програми мовою Паскаль

За умови незмінності вмісту регістра DS, ідентифікатори змінних, визначених в розділі опису змінних глобального блоку мовою Паскаль, можна використовувати в асемблерній програмі як символічні адреси (зміщення в сегменті даних). Для цього в асемблерній програмі такі ідентифікатори необхідно визначити в директиві *Extrn*, що має наступний формат:

```
Extrn   ім'я:тип, ...,  ім'я:тип
```

Тут мається на увазі асемблерний тип: byte, word, dword (необхідно знати кількість байтів, що відводиться для змінних Паскаля).

7) Доступ до процедур і функцій програми мовою Паскаль

Для забезпечення доступу ідентифікатори необхідних паскальних процедур і функцій необхідно визначити в директиві Extrn з типом *near* або *far*. Тип *far* задається для процедур і функцій, описаних в секції Interface, або при явному використанні режиму far-компіляції (наприклад, шляхом завдання директиви **{$F+}**).

Для надійного визначення типу рекомендується за допомогою методики, викладеній в Лабораторній роботі №2-1, визначити код команди *RET*, яка міститься в скомпільованій Паскаль процедурі, а потім по коду визначити, чи вона є міжсегментною, чи внутрішньосегментною.

8) Організація повернення з підпрограми

У відповідності до конвенції Паскаля, на підпрограму покладаються обов'язки щодо забезпечення початкового значення SP. Тому підпрограма повинна закінчуватись командами:

```
pop   bp
ret   const
```

Зауважимо, що команда *ret const* після виконання дії повернення додає до вмісту регістра SP значення const. Значення const обчислюється шляхом додавання кількості байтів, які відведені в стеку на кожний параметр. Наприклад, для раніше розглянутої процедури

Procedure pp(var arg1,arg2;arg3:char,arg4:integer); External;

значення const буде дорівнювати 12 (4 байти на аргумент arg1, 4 байти на аргумент arg2, 2 байти на аргумент arg3 і 2 байти на аргумент arg4), а закінчення асемблерної процедури буде мати вигляд:

```
pop   bp
ret   12
```

Директива *ARG* дозволяє доручити обчислення const Асемблеру. Для цього необхідно закінчити директиву виразом *=ідентифікатор,* наприклад:

```
ARG   Arg4:Word,Arg3:Byte,Arg2:Dword,Arg1:Dword=arg_size
```

Тоді закінчення асемблерної процедури буде таким:

```
pop   bp
ret   arg_size
```

9) Повернення значень функцій

Якщо мовою Асемблера необхідно реалізувати процедуру-функцію, яка використовується в Паскаль програмі, тоді перед поверненням із асемблерної процедури значення функції необхідно розмістити:

- в регістрі *AL*, якщо на паскальний тип даних відводиться один байт;
- в регістрі *AX*, якщо на паскальний тип даних відводиться одно слово;
- в парі регістрів *DX* і *AX*, якщо на паскальний тип даних відводиться подвійне слово, при цьому в регістрі *DX* розміщується старша частина даних або сегментна частина логічної адреси.

4-1.3. Приклад організації взаємодії програми мовою Паскаль і програми на Асемблері

Як приклад організації взаємодії програм мовою Паскаль і мовою Асемблера пропонується програма lab4.pas і відповідно програма BigShow.asm. Програма lab4.pas містить визначення мовою Паскаль двох цілих беззнакових чисел великої розрядності – байтових масивів x, y – та їх початкове заповнення, а також виклики процедури BigShow для відображення на екрані значень цих чисел у 16-ковому форматі.

```
Program lab4(input,output);
var
    i    : word;
    x    :array [1..2000] of byte;
    y    :array [1..1000] of word;
{$L bigshow.obj}
{$F+}
 Procedure BigShow(var p1;p2:word);external;
{$f-}
 begin  {Main program}
 for i:=1 to 300 do
 begin
  x[i]:=i;
  y[i]:=i;
 end;
 for i:=1 to 30 do
 begin
   writeln('x= ');
   BigShow(x,301-i);
   writeln('y= ');
   BigShow(y,301-i);
   readln;
 end;
  end.
```

В оперативному запам’ятовуючому пристрої дані цілого беззнакового типу великої розрядності займають *k* комірок, де *k* - довільне значення. Нехай A – адреса даних такого типу. Тоді адреси комірок пам’яті та нумерацію двійкових розрядів надвеликого числа можна подати наступним чином:

A+k-1	...	A+i-1	...	A+1	A
b_{k*8-1} $b_{(k-1)*8}$	...	b_{i*8-1} $b_{(i-1)*8}$	...	b_{15} b_8	b_7 b_0

Значення B такого числа визначається стандартним чином:

$$B = \sum_{J=0}^{k*8-1} b_i * 2^j$$

Мова Паскаль не підтримує такий тип даних. Для подання мовою Паскаль даних надвеликого цілого беззнакового типу доцільно використовувати байтові масиви. Тобто, один байтовий масив використовується для вмісту ОДНОГО надвеликого цілого беззнакового числа.

Процедура BigShow реалізована як асемблерна процедура в окремому програмному модулі BigShow.asm. Вона працює з надвеликими цілими додатними числами, які розміщуються у байтових масивах. Процедура призначена для перевірки правильності результатів виконання завдання.

Процедура BigShow має два параметри: перший із них – повна логічна адреса байтового масиву, другий параметр передається за значенням і задає кількість байтів у масиві. Програма виводить байти масиву на екран у шістнадцятковому форматі. Байти групуються при відображенні у подвійні слова (8 шістнадцяткових символів для подвійного слова). Байт з найменшою адресою (заданою першим параметром) завжди виводиться в крайній правій позиції останнього рядка, що зручно для зорового порівняння двох масивів.

Для виведення на екран використовується функція MS-DOS 02h. Для виклику функції використовується команда програмного переривання Int 21h. Параметром виклику є символ ASCII, який необхідно записати в регістр DL. Номер функції (02h) розміщують в регістрі AH.

```
; програмний модуль BigShow.asm
.386
_text    segment word public 'text' use16
        assume  cs:_text
;
;****************************************
; п/п виведення на екран в hex-форматі даних із регістра ebx:
; якщо di=28 – виводяться усі 4 байти
; якщо di=20 – виводяться 3 молодших байти
; якщо di=12 – виводяться 2 молодших байти
; якщо di=4 – виводиться один молодший байт
 show_bt        proc
        pushad
        mov       cx,di
```

```
        mov     ah,2
bt0:
        mov     edx,ebx
        shr     edx,cl
        and     dl,00001111b
        cmp     dl,10
        jl      bt1
        add     dl,7
bt1:
        add     dl,30h
        int     21h
        sub     cl,4
        jnc     bt0
        popad
        ret
show_bt         endp

BigShow         proc    far                     ; procedure BigShow(var mas, len:word)
      public    BigShow
; mas - адреса байтового масиву
@mas            equ     [bp+8]                  ; адреса адреси
; len - кількість байт масива, які необхідно вивести на екран
@len            equ     [bp+6]                  ; адреса кількості

        push    bp
        mov      bp,sp                  ; базова адреса фактичних параметрів
;  перехід на новий рядок екрану
        mov     ah,2
        mov      dl,13
        int     21h
        mov     dl,10
        int     21h
; обчислення кількості пробілів у першому рядку
        mov     ax,@len
        test    ax,00000011b
        pushf
        shr     ax,2
        popf
        jz      @1
        inc     ax
@1:
        xor     cx,cx
        mov     di,28
        and     ax,00000111b
        jz      @2
; формування пробілів на відсутніх подвійних словах
        mov     ah,8
        sub     ah,al
        mov     al,ah
        xor     ah,ah
        imul    ax,8+1
        mov     cx,ax
```

```
@2:
        mov     dx,@len
        and     dx,00000011b
        jz      l000
; формування початкового значення кількості зсувів
        mov     di,dx           ;di -  1  2   3
        dec     di              ;di -  0  1   2
        shl     di,3            ;di -  0  8  16
        add     di,4            ;di -  4  12 20
;  формування пробілів на відсутніх байтах у подвійному слові
        mov     dh,4
        xchg    dh,dl           ;dh -  1  2  3
        sub     dl,dh           ;dl -  3  2  1
        shl     dl,1            ;dl -  6  4  2
        xor     dh,dh           ;dx -  6  4  2
        add     cx,dx
l000:
        jcxz    l002
; виведення початкових пробілів у першому рядку
l001:
        mov     ah,2
        mov     dl," "
        int     21h
        loop    l001
l002:
        mov     cx,@len
        shr     cx,2
        cmp     di,28
        jz      @3
        inc     cx
@3:
        xor     esi,esi
        lds     si,@mas
        lea     esi,[esi+ecx*4]-4
        std
; виведення  масиву
l004:
        lodsd
        mov     ebx , eax
        call    show_bt
        mov     di,28
        mov     ah , 2
        mov     dl,20h
        int     21h
        dec             ecx
        test    ecx,7
        jne     l005
; перехід на новий рядок
        mov      ah,2
        mov      dl,13
        int     21h
        mov     dl,10
```

```
        int     21h
l005:
        jcxz    L006
        jmp     l004
l006:
        mov     ah,2
        mov     dl,13
        int     21h
        mov     dl,10
        int     21h
 ; mov   ah,1
 ; int   21h
        pop     bp
        ret     6
BigShow endp
_text   ends
        end
```

Програма lab4.pas містить всі необхідні елементи для забезпечення зв'язку з асемблерною процедурою BigShow. Вона демонструє незалежність процедури BigShow від паскального типу даних. Це означає, що процедура BigShow (або подібні їй процедури) можна використовувати також для аналізу машинного формату типів даних мови Паскаль. Наприклад, за допомогою процедури BigShow легко визначається формат логічних значень *True* та *False*.

4-1.4. Завдання на виконання роботи

Перше заняття

1) Скопіювати програми lab4.pas та BigShow.asm в окремий робочий каталог. Ознайомитись з їх призначенням і вмістом.
2) Протранслювати за допомогою tasm (або masm) програму мовою Асемблера BigShow.asm. В директиві L програми мовою Паскаль lab4.pas відкоригувати (при необхідності) шлях до файлу BigShow.obj. Відкомпілювати Паскаль-програму (разом з підключеним файлом BigShow.obj) та перевірити її працездатність.
3) Вивчити правила взаємозв'язку окремо скомпільованих програм мовою Асемблера та мовою Паскаль (див. Теоретичні відомості);

4) Розробити алгоритм реалізації операції з надвеликими цілими додатними числами згідно варіанта завдання (табл. 4-1.1).

Друге заняття

Розробити програму мовою Паскаль і програму мовою Асемблера згідно варіанта завдання та наступних вимог:

1) Програма мовою Паскаль повинна:

- відповідати вимогам зв'язку з асемблерними процедурами;
- містити визначення байтових масивів та їх початкове заповнення;
- містити виклики процедури BigShow для відображення початкових даних,
- містити виклик розробленої асемблерної процедури з відповідними параметрами;
- містити виклики процедури BigShow для відображення результатів;
- перед викликом процедури BigShow у Паскаль програмі забезпечити виведення на екран текстових повідомлень (коментарів).

2) Програма мовою Асемблера повинна:

- розміщуватися в початковому асемблерному модулі;
- містити процедуру відображення BigShow.asm і власну асемблерну процедуру, що виконує ту чи іншу елементарну операцію (згідно варіанта) з надвеликими цілими додатними числами, які розміщуються у байтових масивах. Тобто, один байтовий масив Паскаль програми використовується для вмісту ОДНОГО надвеликого цілого беззнакового числа.
- щоб уможливити виклик власної асемблерної процедури з програми мовою Паскаль, вона повинна відповідати спеціальним вимогам (див. Теоретичні відомості);
- на початку модуля мовою Асемблера розмістіти директиву Title із зазначенням групи та прізвища студента;

3) *Використати процедуру BigShow.asm* для відображення і перевірки коректності роботи розробленої програми на різних тестових наборах значень байтових масивів.

4) Додаткові експерименти:

- визначити, чи може програмний модуль мовою Асемблера, який об'єднується з Паскаль програмою, мати додаткові логічні сегменти з довільними іменами;
- визначити порядок передачі до функції значення типу **String**.

Таблиця 4-1.1

Варіанти завдання

1.	Розробити процедуру **Big2sAdd(var M1,M2;len:word)**, де M1,M2 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M1+M2. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для додавання останніх байт використать команди для 8 - розрядних даних.. Вважати, що M1 і M2 знаходяться в різних сегментах.
2.	Розробити процедуру **Big2Add(var M1,M2,Carry;len:word)**, де M1,M2 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M1+M2. Змінній байтового типу Carry присвоюється значення 1 в разі переповненя і 0 при його відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для додавання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1,M2 знаходяться в одному сегменті.
3.	Розробити функцію **FBig2Add(var M1,M2;len:word):Boolean**, де M1,M2 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M1+M2. Функції **FBig2Add** присвоюється значення False в разі переповненя і True при його відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для додавання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1,M2 і Carry знаходяться в одному сегменті
4.	Розробити процедуру **Big3sAdd(var M1,M2,M3;len:word)**, де M1,M2,M3 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M2+M3. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для додавання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1,M2 і M3 знаходяться в різних сегментах.

5. Розробити процедуру **Big3Add(var M1,M2,M3,Carry;len:word)**, де M1,M2,M3 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M2+M3. Змінній байтового типу Carry присвоюється значення 1 в разі переповненя і 0 при його відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для додавання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1,M2,M3 і Carry знаходяться в одному сегменті.
6. Розробити функцію **FBig3Add(var M1,M2,M3;len:word):Boolean**, де M1,M2,M3 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M2+M3. Функції FBig3Add присвоюється значення False в разі переповненя і True при його відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для додавання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1,M2,M3 знаходяться в одному сегменті.
7. Розробити процедуру **Big2sSub(var M1,M2;len:word)**, де M1,M2 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M1-M2. . Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для додавання останніх байт використать команди для 8 - розрядних даних.. Вважати, що M1 і M2 знаходяться в різних сегментах.
8. Розробити процедуру **Big2Sub(var M1,M2,Carry;len:word)**, де M1,M2 надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M1-M2. Змінній байтового типу Carry присвоюється значення 1 при наявності позики і 0 при її відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для віднімання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1, M2 і Carry знаходяться в одному сегменті.
9. Розробити функцію **FBig2Sub(var M1,M2;len:word):Boolean**, де M1,M2 надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M1-M2. Функції Fbig2Sub присвоюється значення False в разі наявності позики і True при її відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для віднімання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1, M2 знаходяться в одному сегменті.
10. Розробити процедуру **Big3sSub(var M1,M2,M3;len:word)**, де M1,M2,M3 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M2-M3. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для віднімання останніх байт використать команди для 8 - розрядних даних.

Вважати, що M1,M2 і M3 знаходяться в різних сегментах.
11. Розробити процедуру **Big3Sub(var M1,M2,M3,Carry;len:word)**, де M1,M2,M3 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M2-M3. Змінній байтового типу Carry присвоюється значення 1 при наявності позики і 0 при її відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для віднімання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1,M2,M3 і Carry знаходяться в одному сегменті.
12. Розробити процедуру **FBig3Sub(var M1,M2,M3;len:word)**:Boolean, де M1,M2,M3 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - M1=M2-M3 Функції **Fbig3Sub** присвоюється значення False в разі наявності позики і True при її відсутності. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то для віднімання останніх байт використать команди для 8 - розрядних даних. Вважати, що M1,M2,M3 знаходяться в одному сегменті.
13. Розробити функцію **Biggr(var M1,M2;len:word):Boolean**, де M1,M2 - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - якщо M1 > M2 то значення Biggr - True, інакше -False . Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для порівняння останніх байт використать команди для 8 - розрядних даних. Вважати, що M1 і M2 знаходяться в одному сегменті.
14. Розробити функцію **Biggreq (var M1,M2;len:word):Boolean**, де M1,M2 - - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - якщо M1 ≥ M2 то значення Biggreq - True, інакше - False. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для порівняння останніх байт використать команди для 8 - розрядних даних. Вважати, що M1 і M2 знаходяться в одному сегменті.
15. Розробити функцію **Bigeq (var M1,M2;len:word):Boolean**, де M1,M2 - - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - якщо M1 = M2 то значення Bigeq - True, інакше - False. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для порівняння останніх байт використать команди для 8 - розрядних даних. Вважати, що M1 і M2 знаходяться в одному сегменті.
16. Розробити функцію **Bigne (var M1,M2;len:word):Boolean**, де M1,M2 - - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - якщо M1 ≠ M2 то значення Bigne - True, інакше - False. Повинні використовуватись команди для 32-розрядних

даних. Якщо значення len не кратно 4, то при необхідності для порівняння останніх байт використать команди для 8 - розрядних даних. Вважати, що M1 і M2 знаходяться в одному сегменті.
17. Розробити функцію **Bigles (var M1,M2;len:word):Boolean**, де M1,M2 - - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - якщо M1 < M2 то значення Bigles - True, інакше - False. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для порівняння останніх байт використать команди для 8 - розрядних даних. Вважати, що M1 і M2 знаходяться в одному сегменті.
18. Розробити функцію **Bigleseq (var M1,M2;len:word):Boolean**, де M1,M2 - - надвеликі цілі додатні числа (байтові масиви довжиною len). Операція - якщо M1 ≤ M2 то значення Bigleseq - True, інакше - False. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для порівняння останніх байт використать команди для 8 - розрядних даних. Вважати, що M1 і M2 знаходяться в одному сегменті.
19. Розробити процедуру **BigShlCount(var M1;len,count: word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), count - кількість розрядів зсуву. Операція - лінійний зсув вліво (в сторону старших розрядів) на кількість двійкових розрядів, яка задана параметром count. При цьому count старших розрядів втрачаються, а в count молодших розрядів заноситься 0. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для останніх байт використать команди для 8 - розрядних даних.
20. Розробити процедуру **BigShrCount(var M1;len,count: word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), count - кількість розрядів зсуву. Операція - лінійний зсув вправо (в сторону молодших розрядів) на кількість двійкових розрядів, яка задана параметром count. При цьому count молодших розрядів втрачаються, а в count старших розрядів заноситься 0. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для останніх байт використать команди для 8 - розрядних даних.
21. Розробити процедуру **BigRolCount(var M1;len,count: word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), count - кількість розрядів зсуву. Операція - циклічний зсув вліво (в сторону старших розрядів) на кількість двійкових розрядів, яка задана параметром count. При цьому count старших розрядів поступають на місце молодших розрядів. Повинні використовуватись команди для 32-розрядних

даних. Якщо значення len не кратно 4, то при необхідності для останніх байт використать команди для 8 - розрядних даних.
22. Розробити процедуру **BigRorCount(var M1;len,count: word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), count - кількість розрядів зсуву. Операція - циклічний зсув вправо (в сторону молодших розрядів) на кількість двійкових розрядів, яка задана параметром count. При цьому count молодших розрядів поступають на місце старших розрядів. Повинні використовуватись команди для 32-розрядних даних. Якщо значення len не кратно 4, то при необхідності для останніх байт використать команди для 8 - розрядних даних.
23. Розробити процедуру **BigZeroShl(var M1,cnt;len: word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), cnt - кількість розрядів зсуву - змінна типу word. Операція - лінійний зсув вліво (в сторону старших розрядів) до тих пір поки в len*8-1 розряді не з'явиться одиничка. Кількість зсувів записується в параметр cnt. Якщо в початковому значенню числа M1 розряд len*8-1 дорівнює 1, то зсуви не виконуються, а в параметр cnt записується нуль.
24. Розробити процедуру **BigZeroShr(var M1,cnt;len: word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), cnt - кількість розрядів зсуву. Операція - лінійний зсув вправо (в сторону молодших розрядів) до тих пір поки в молодшому розряді числа не з'явиться одиничка. Кількість зсувів записується в параметр cnt. Якщо в початковому значенню числа M1 молодший розряд дорівнює 1, то зсуви не виконуються, а в параметр cnt записується нуль.
25. Розробити процедуру **BigShl(var M1,Carry;len:word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), Carry - змінна типу byte. Операція - лінійний зсув вліво (в сторону старших розрядів) на один розряд. При цьому в змінну Carry заноситься значення len*8-1 розряду числа M1, а в молодший розряд числа M1 заноситься 0.
26. Розробити процедуру **BigShr(var M1,Carry;len:word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len), Carry - змінна типу byte. Операція - лінійний зсув вправо (в сторону молодших розрядів) на один розряд. При цьому в змінну Carry заноситься значення молодшого розряду числа M1, а в старший розряд числа M1 заноситься 0.
27. Розробити функцію **FcBigShl(var M1;len:word):Byte**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len). Операція - лінійний зсув вліво (в сторону

Продовження табл. 4-1.1

старших розрядів) на один розряд. При цьому функція **FcBigShl** приймає значення len*8-1 розряду числа M1, а в молодший розряд числа M1 заноситься 0.
28. Розробити функцію **FBigShl(var M1;len:word):Boolean**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len). Операція - лінійний зсув вліво (в сторону старших розрядів) на один розряд. При цьому функція FBigShl приймає значення False, якщо len*8-1 розряд числа M1 до зсуву дорівнює 1 і True в протилежному випадку. В молодший розряд числа M1 при зсуві заноситься 0.
29. Розробити процедури **BigSetBit(var M1;len,number: word)** та **BigClrBit(var M1;len,number: word)**, де M1 - надвелике ціле додатнє число (байтовий масив довжиною len)., number - номер двійкового розряду числа M1, починаючи з 0. Операція - записать одиничку в розряд number для процедури BigSetBit і 0 для процедури BigClrBit.
30. Розробити процедуру **Extract(var M1,M2;len,ibeg,iend:word)**, надвеликі цілі додатні числа (байтові масиви довжиною len), ibeg,iend - номера розрядів, такі, що len*8-1 ≥ iend ≥ ibeg. Операція - виділить із числа M1 розряди з ibeg по iend включно та одержане таким чином число присвоїть M2. В старші розряди числа M2 занести 0.

4-1.5. Контрольні запитання

1. Чому поле операндів директиви END в модулі мовою Асемблера повинно бути порожнім?
2. Які імена логічних сегментів повинна мати програма мовою Асемблера для забезпечення зв’язку з програмами мовами високого рівня?
3. За якою адресою оперативної пам’яті (більшою чи меншою) буде розташований перший фактичний параметр для процедури, яка визивається Паскаль програмою?
4. Чи можливий в програмі мовою Асемблера виклик процедур мовою Паскаль?
5. До яких змінних програми мовою Паскаль можливий доступ в програмі мовою Асемблера і як він забезпечується?
6. Яка програма (та що викликає чи та яку викликають) відповідає в Паскалі за відновлення вмісту покажчика стеку – вмісту регістра SP?