

Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П. Г. Демидова
Кафедра компьютерной безопасности и математических
методов обработки информации

Н. Б. Федотов

Практикум на ЭВМ. Ассемблер

Методические указания

*Рекомендовано
Научно-методическим советом университета
для студентов, обучающихся по специальности
Математика*

Ярославль 2011

УДК 002
ББК 3973.2–018.1я73
Ф33

*Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2010/2011 учебного года*

Рецензент

кафедра компьютерной безопасности и математических методов обработки информации

Федотов, Н. Б. Практикум на ЭВМ. Ассемблер : методические указания / Н. Б. Федотов ; Яросл. гос. ун-т им. П. Г. Демидова. – Ярославль: ЯрГУ, 2011. – 68 с.

Данная работа содержит описание лабораторных работ для практического освоения программирования на языке ассемблера. Для каждой лабораторной работы формулируется ее цель, приводится список заданий и для одного из заданий дается его полное решение в виде отлаженной программы и набора тестов, на котором она проверялась. Каждая программа содержит достаточно подробные комментарии, полезные для самостоятельного обучения, а каждый тест включает в себя его описание, исходные данные и ожидаемые результаты.

Предназначено для студентов, обучающихся по специальности 010101.65 «Прикладная математика и информатика» (дисциплина «Практикум на ЭВМ», блок ЕН), очной формы обучения. .

УДК 002
ББК 3973.2–018.1я73

© Ярославский государственный
университет им. П. Г. Демидова, 2011

Прежде чем переходить к описанию лабораторных работ, сделаем замечания общего характера. Каждая работа должна содержать комментарии, описывающие содержание работы, смысл важнейших ее частей, а также отдельных команд, имеющих ключевое значение. Далее, крайне важно приучиться использовать мнемонические имена. И наконец, важнейшей частью работы является разработка достаточно полного набора тестов, который должен следовать определенной логике, быть упорядочен, как правило, по размерности исходных данных и содержать описание теста, исходные данные и ожидаемый результат. Полезно набор тестов разработать прежде, чем писать программу. Кроме того, программы необходимо тестировать каждый раз заново при внесении любых изменений в ее текст.

Лабораторная работа № 1

Цель работы – познакомиться со структурой программы, технологией выполнения (см. образец программы и приложение 1), арифметическими командами (см. приложение 3), внутренним представлением целых чисел, работой в отладчике td.exe. В работе предполагается сравнение результатов выполнения арифметических команд в отладчике с результатами расчетов, выполненных вручную.

Образец программы

```
title    lab1      первая программа на ассемблере
stk      segment para stack 'stack'
          db       64 dup('stack')
stk      ends
data     segment para public 'data'
x        dw       112
y        dw       122
a        dw       100 dup(0)
data     ends
code     segment para public 'code'
          assume   cs:code,ds:data
start:   mov      ax,data
          mov      ds,ax
          mov      ax,x                ;ax=x
```

	add	ax,y	;ax=x+y
	mov	a,ax	;a[0]=x+y
	mov	ax,x	;ax=x
	sub	ax,y	;ax=x-y
	mov	a+2,ax	;a[1]=x-y
	mov	ax,x	
	imul	y	;dx:ax=x*y
	mov	a+4,ax	;a[2]=ax
	mov	a+6,dx	;a[3]=dx
	mov	ax,x	
	cwd		;dx:ax=ax
	idiv	y	;ax=x/y dx=x mod y
	mov	a+8,ax	;a[4]=x/y
	mov	a+10,dx	;a[5]=x mod y
	mov	ax,x	
	inc	ax	;ax=x+1
	mov	a+12,ax	;a[6]=x+1
	mov	ax,y	
	dec	ax	;ax=y-1
	mov	a+14,ax	;a[7]=y-1
	mov	ax,x	
	sar	ax,3	;ax=x/8
	mov	a+16,ax	;a[8]=x/8
	mov	ax,y	
	sal	ax,4	;ax=y*16
	mov	a+18,ax	;a[9]=y*16
	mov	ah,4ch	;ah=4ch
	int	21h	
code	ends		
	end	start	

Варианты заданий

Варианты заданий определяются значениями констант x,y:

- | | | | |
|--------------|--------------|--------------|--------------|
| 1. 135,1222 | 2. 89,4177 | 3. 92,1158 | 4. 101,412 |
| 5. 201,1276 | 6. 105,2242 | 7. 139,3157 | 8. 1322,178 |
| 9. 161,3183 | 10. 111,2136 | 11. 151,3223 | 12. 79,2177 |
| 13. 99,3157 | 14. 121,2173 | 15. 133,3226 | 16. 145,4252 |
| 17. 119,1217 | 18. 111,4148 | 19. 141,1193 | 20. 120,4255 |

Лабораторная работа № 2

Цель работы – научиться программировать арифметические выражения с рационализацией по числу промежуточных вычислений, по выбору оптимальных команд. Использование внешних подпрограмм ввода-вывода и макросов.

Образец программы

```
        title    lab2
stk      segment para stack 'stack'
        db      64 dup('stack')
stk      ends
data     segment para public 'data'
input_x  db      10,13,'x= ','$'
input_y  db      10,13,'y= ','$'
output_z db      10,13,'z= ','$'
data     ends
code     segment para public 'code'
        extrn   rdint:far
        extrn   wrint:far
;--- макрос вывода строки
wrstr    macro str
        mov     dx,offset str
        mov     ah,9
        int     21h
        endm
        assume  cs:code,ds:data,ss:stk
start:   mov     ax,data
        mov     ds,ax
;--- распределение регистров
x        equ     bx
y        equ     bp
z        equ     si
;--- вывод запроса на число x
wrstr    input_x
        call    rdint
        mov     x,ax
;--- вывод запроса на число y
wrstr    input_y
        call    rdint
        mov     y,ax
;--- вычисление  $((y-1) * x - (y-1)/x + (y-1) \bmod x) * 16$ 
;--- проверка на ОДЗ
        cmp     x,0
```

```

        jnz      raschet
        mov     z,9999;y=9999 - индикатор x вне ОДЗ
        jmp     rez
raschet:dec     y          ;y=y-1
        mov     ax,y
        imul    x          ;ax=(y-1)*x
        mov     z,ax       ;z=(y-1)*x
        mov     ax,y
        cwd
        idiv    x          ;ax=(y-1)/x      dx=(y-1)mod x
        sub     z,ax       ;z=(y-1) * x - (y-1)/x
        add     z,dx       ;z=(y-1) * x - (y-1)/x + (y-1)mod x
        mov     cl,4       ;cl=4
        sal     z,cl       ;z=((y-1) * x - (y-1)/x + (y-1)mod x)*16
;--- вывод результата
rez:     wrstr    output_z
        mov     ax,z
        call    writ
        mov     ah,4ch
        int     21h
code     ends
        end      start

```

Варианты заданий

- $(x * (y-1) - x/(y-1) + x \bmod (y-1))/4;$
- $(x * (y+1) - x/(y+1) + x \bmod (y+1))/4;$
- $((x+1) * (y-1) - (x+1)/(y-1) + (x+1) \bmod (y-1))/8;$
- $((x+1) * (y+1) - (x+1)/(y+1) + (x+1) \bmod (y+1))/8;$
- $((x+1) * y - (x+1)/y + (x+1) \bmod y)/16;$
- $((x-1) * (y-1) - (x-1)/(y-1) + (x-1) \bmod (y-1))/8;$
- $((x-1) * (y+1) - (x-1)/(y+1) + (x-1) \bmod (y+1))/8;$
- $((x-1) * y - (x-1)/y + (x-1) \bmod y)/16;$
- $(y * (x-1) - y/(x-1) + y \bmod (x-1))/4;$
- $(y * (x+1) - y/(x+1) + y \bmod (x+1))/8;$
- $((y+1) * (x+1) - (y+1)/(x+1) + (y+1) \bmod (x+1))/16;$
- $((y+1) * (x-1) - (y+1)/(x-1) + (y+1) \bmod (x-1))/16;$
- $((y+1) * x - (y+1)/x + (y+1) \bmod x)/8;$
- $((y-1) * (x+1) - (y-1)/(x+1) + (y-1) \bmod (x+1))/8;$
- $((y-1) * (x-1) - (y-1)/(x-1) + (y-1) \bmod (x-1))/16;$
- $((y-1) * x - (y-1)/x + (y-1) \bmod x)/16.$

Тесты

Описание теста	Исходные данные	Результат
1. x вне ОДЗ	x=0 y=1	z=9999
2. z=0 при y=1	x=5 y=1	z=0
3. z=0 при x=1	x=1 y=5	z=0
4. общий случай	x=2 y=2	z=48

Лабораторная работа № 3

Цель работы – программирование циклов, условных переходов. Использование внутренних подпрограмм.

Образец (без внутренней подпрограммы)

```
;
;   табулирование   y=((x-1) mod |b^3+1|)*4, x<=-1
;                   y=(|b^3+1| / (x-1))*4, x>-1
;   при x=a, a+h, ..., a+(n-1)h
;
;   int x,y,a,b,h,n,c,d;
; ввод исходных данных
;   cout << "a=", cin >>a;
;   cout << "h=", cin >>h;
;   cout << "n=", cin >>n;
;   cout << "b=", cin >>b;
;   cout << "\n\\t\\t\\t\\n\\n";
;   подготовка числа
;   c=abs(b*b*b+1);
;   x=a;
;   цикл табулирования
;   for (int i=0; i<n; i++) {
;       d=x-1;
;       if (x<=-1)
;           if (c!=0)           // знаменатель не нуль
;               y=d % c;
;           else
;               y=9999; // индикатор - вне ОДЗ
;       else
;           if (d!=0)           // знаменатель не нуль
;               y=c / d;
;           else
;               y=9999; // индикатор - вне ОДЗ
;   }
```

```

;               if (y!=9999)
;                   y=y*4;
;               cout << x << "\t\t" << y << "\n" ;
;               x=x+h;
;           }
;       cout << "\n" ;
;
;       title      lab3 Табулирование функции
stk      segment para stack'stack'
;       db        64 dup('stack')
stk      ends
data     segment para public 'data'
a        dw      ?
h        dw      ?
n        dw      ?
b        dw      ?
a_       db      13,10,'a=$'
h_       db      13,10,'h=$'
n_       db      13,10,'n=$'
b_       db      13,10,'b=$'
shapka   db      13,10,'x',09,09,'y',13,10,13,10,'$'
nl       db      13,10,'$'
tab      db      09,'$'
data     ends
code     segment para public 'code'
;       extrn     rdint:far
;       extrn     wrint:far
;--- макрос вывода строки
wrstr    macro    str
;       mov     dx,offset str
;       mov     ah,9
;       int     21h
;       endm
;       assume   cs:code,ds:data,ss:stk
start:   mov     ax,data
;       mov     ds,ax
;--- распределение регистров
x        equ     bx
y        equ     bp
c        equ     si      ;abs(b^3+1)
d        equ     di      ;x-1
;--- ввод a
wrstr    a_
;       call    rdint
;       mov     a,ax

```



```

;--- ввод h
        wrstr    h_
        call     rdint
        mov      h,ax
;--- ввод n
        wrstr    n_
        call     rdint
        mov      n,ax
;--- ввод b
        wrstr    b_
        call     rdint
        mov      b,ax
;--- вывод шапки
        wrstr    shapka
;--- вычисление c=abs(b*b*b+1)
        mov      ax,b
        imul     b
        imul     b
        inc      ax
        mov      c,ax
        jns      nachx
        neg      c
;--- начальное x
nachx:   mov      x,a
;--- подготовка цикла
        mov      cx,n
;--- защита от n<=0
        cmp      cx,0
        jle      fin
;--- цикл табулирования
cirklv:  mov      ax,x
        dec      ax
        mov      d,ax           ;d=x-1
        cmp      x,-1
        jg       div2
        cmp      c,0           ;x<=-1
        jz       zero1
        mov      ax,d           ;c!=0
        cwd
        idiv     c              ;dx=d mod c
        mov      y,dx
        jmp      short mult
zero1:   mov      y,9999         ;y=9999 - индикатор x вне ОДЗ
        jmp      short mult
div2:    cmp      d,0           ;x>-1

```

```

        jz      zero2
        mov     ax,c                ;d!=0
        cwd
        idiv    d                   ;ax=c / d
        mov     y,ax
        jmp     short mult
zero2:   mov     y,9999              ;y=9999 - индикатор x вне ОДЗ
mult:    cmp     y,9999
        jz      rez
sal      y,2
;--- вывод результата
rez:     wrstr    nl
        mov     ax,x
        call    wrint
        wrstr    tab
        wrstr    tab
        mov     ax,y
        call    wrint
        add     x,h                 ;x=x+h
        loop    cirklv
        wrstr    nl
fin:     mov     ah,4ch
        int     21h
code     ends
        end      start

```

Образец (с внутренней подпрограммой)

```

        title    lab3 Табулирование функции
stk      segment para stack 'stack'
        db       64 dup('stack')
stk      ends
data     segment para public 'data'
a        dw      ?
h        dw      ?
n        dw      ?
b        dw      ?
a_       db      13,10,'a=$'
h_       db      13,10,'h=$'
n_       db      13,10,'n=$'
b_       db      13,10,'b=$'
shapka   db      13,10,'x',09,09,'y',13,10,13,10,'$'
nl       db      13,10,'$'
tab      db      09,'$'
data     ends
code     segment para public 'code'

```

```

        extrn    rdint:far
        extrn    wrint:far
;--- макрос вывода строки
wrstr    macro    str
        mov     dx,offset str
        mov     ah,9
        int     21h
        endm
        assume  cs:code,ds:data,ss:stk
func     proc     near
;--- вычисление c=abs(b*b*b+1)
        mov     ax,b
        imul    b
        imul    b
        inc     ax
        mov     c,ax
        jns     nachx
        neg     c
nachx:
        mov     ax,x
        dec     ax
        mov     d,ax
        cmp     x,-1
        jg      div2
        cmp     c,0
        jz      zero1
        mov     ax,d
        cwd
        idiv    c
        mov     y,dx
        jmp     short mult
zero1:mov     y,9999
        jmp     short mult
div2:  cmp     d,0
        jz      zero2
        mov     ax,c
        cwd
        idiv    d
        mov     y,ax
        jmp     short mult
zero2:mov     y,9999
mult:  cmp     y,9999
        jz      rez
sal     y,2
        ret

```

```

func      endp
start:    mov     ax,data
          mov     ds,ax
;--- распределение регистров
x         equ     bx
y         equ     bp
c         equ     si          ;abs(b^3+1)
d         equ     di          ;x-1
;--- ввод a
          wrstr    a_
          call     rdint
          mov      a,ax
;--- ввод h
          wrstr    h_
          call     rdint
          mov      h,ax
;--- ввод n
          wrstr    n_
          call     rdint
          mov      n,ax
;--- ввод b
          wrstr    b_
          call     rdint
          mov      b,ax
;--- вывод шапки
          wrstr    shapka
;--- начальное x
          mov      x,a
;--- подготовка цикла
          mov      cx,n
;--- защита от n<=0
          cmp      cx,0
          jle      fin
;--- цикл табулирования
cirklv:   call func
;--- вывод результата
rez:      wrstr    nl
          mov      ax,x
          call     wrint
          wrstr    tab
          wrstr    tab
          mov      ax,y
          call     wrint
          add      x,h          ;x=x+h
          loop     cirklv

```

```

fin:      wrstr      nl
          mov        ah,4ch
          int        21h
code      ends
          end        start

```

Варианты заданий

- | | |
|--|---------------------------------------|
| 1. $y = ((x^2-1) \bmod 8b+1)/8, x \leq 1;$ | $y = (8b+1 / (x^2-1))/8, x > 1;$ |
| 2. $y = ((x-1) \bmod 8b^2+1)+100, x \leq 1;$ | $y = (8b^2+1 / (x-1))+100, x > 1;$ |
| 3. $y = ((x^3-1) \bmod 8b+1)+20, x \leq 1;$ | $y = (8b+1 / (x^3-1))+20, x > 1;$ |
| 4. $y = ((x-1) \bmod 8b^3+1)-100, x \leq 1;$ | $y = (8b^3+1 / (x-1))-100, x > 1;$ |
| 5. $y = ((x^2-1) \bmod 8b^2+1)-20, x \leq 1;$ | $y = (8b^2+1 / (x^2-1))-20, x > 1;$ |
| 6. $y = ((x-1) \bmod 8b+1)^2, x \leq 1;$ | $y = (8b^2+1 / (x-1))^2, x > 1;$ |
| 7. $y = ((x^3-1) \bmod 8b+1)^3, x \leq 1;$ | $y = (8b+1 / (x^3-1))^3, x > 1;$ |
| 8. $y = ((x-4) \bmod 8b^3+1)-1, x \leq 4;$ | $y = (8b^3+1 / (x-4))-1, x > 4;$ |
| 9. $y = (x^2-1 \bmod (16b+1))/8, x \leq 1;$ | $y = ((16b+1) / x^2-1)/8, x > 1;$ |
| 10. $y = (x-1 \bmod (16b^2+1))+100, x \leq 1;$ | $y = ((16b^2+1) / x-1)+100, x > 1;$ |
| 11. $y = (x^3-1 \bmod (4b+1))+20, x \leq 1;$ | $y = ((4b+1) / x^3-1)+20, x > 1;$ |
| 12. $y = (x-1 \bmod (2b^3+1))-100, x \leq 1;$ | $y = ((2b^3+1) / x-1)-100, x > 1;$ |
| 13. $y = (x^2-1 \bmod (8b^2+1))-20, x \leq 1;$ | $y = ((8b^2+1) / x^2-1)-20, x > 1;$ |
| 14. $y = (x-1 \bmod (8b+1))^2, x \leq 1;$ | $y = ((8b^2+1) / x-1)^2, x > 1;$ |
| 15. $y = (x^3-1 \bmod (8b+1))^3, x \leq 1;$ | $y = ((8b+1) / x^3-1)^3, x > 1;$ |
| 16. $y = (x-4 \bmod 8b^3+1)-1, x \leq 4;$ | $y = ((8b^3+1) / x-4)-1, x > 4;$ |

Тесты

Описание теста	исходные данные	результат
1. число точек таб. ≤ 0	a=-2 h=1 n=0 b=-1	x y
2. b=-1 - вне ОДЗ	a=-2 h=1 n=5 b=-1	x y
		-2 9999
		-1 9999
		0 0
		1 9999
		2 0
3.общий случай с точкой x=1 вне ОДЗ	a=-2 h=1 n=5 b=1	x y
		-2 -4
		-1 0
		0 -8
		1 9999
		2 8

Лабораторная работа № 4

Цель работы – работа с одномерными массивами. Переадресация. Использование внешних подпрограмм. Передача параметров.

Образец программы

```
;      В сегменте данных лежит одномерный массив а.
;      Сдвинуть элементы на четных местах вправо циклически.
      title      lab4 основная программа
stk    segment stack
      dw        256 dup(0)
stk    ends
data segment
n      dw      ?
a      dw      100 dup(0)
in_a   db      13,10,'input a:','$'
out_a  db      13,10,'output a:','$'
outn   db      13,10,'n=','$'
outa   db      'ai=','$'
tab    db      9,'$'
nl     db      13,10,'$'
data   ends
code   segment
wrstr  macro    str                ;!!!описание макроса
      mov      dx,offset str
      mov      ah,9
      int      21h
      endm
      extrn    rdint:far
      extrn    wrint:far
      extrn    shiftcr:far
      assume   cs:code,ds:data,ss:stk
;--- подпрограмма ввода вектора
invect  proc
;--- ввод n
      wrstr    outn
      call     rdint
      mov      n,ax
;--- подготовка цикла
      mov      cx,n
      cmp      cx,0
      jle      fin_in              ;защита от n<=0
      xor      si,si
      wrstr    nl
```

```

;--- цикл ввода а
in_ai:  wrstr    outa
        call    rdint
        mov     a[si],ax
        wrstr    nl
        inc     si
        inc     si
        loop    in_ai
fin_in:  ret
invect  endp
;--- подпрограмма вывода вектора
outvect proc
;--- подготовка цикла
        mov     cx,n
        cmp     cx,0
        jle     fin_out      ;защита от n<=0
        xor     si,si
        wrstr    nl
;--- цикл вывода а
out_ai: mov     ax,a[si]
        call    wrint
        wrstr    tab
        inc     si
        inc     si
        loop    out_ai
fin_out:ret
outvect endp
start:  mov     ax,data
        mov     ds,ax
        call    invect      ;ввод массива
;--- вывод исходного вектора
        wrstr    in_a      ;!!!вызов макроса
        call    outvect
;--- подготовка параметров и вызов процедуры преобразования
        push    n
        lea     ax,a
        push    ax
        call    shiftcr     ;вызов процедуры преобразования
;--- вывод преобразованного вектора
        wrstr    out_a      ;!!!вызов макроса
        call    outvect
        mov     ah,4ch
        int     21h
code    ends
        end      start

```

Образец внешней процедуры

```

;      В сегменте данных лежит одномерный массив a.
;      Сдвинуть элементы на четных местах вправо циклически.
;      int imax=n-1,imin=1;
;      if (n % 2 ==1) imax--;
;      a1=a[imax];
;      for (int i=imax; i>imin; i--,i--)
;          a[i-2]=a[i];
;      cout << "\n";
;      for (in i=0; i<n; i++)
;          cout << a[i] << "\t";

title    lab4 процедура вычисления
prcode segment
extrn    wrint:far
public   shiftcr
assume   cs:prcode
shiftcr proc    far
;--- сохранить bp и запомнить в нем sp для доступа к параметрам
    push    bp
    mov     bp,sp
;--- сохранение регистров в стек
    push    ax
    push    bx
    push    cx
    push    si
    push    di
    pushf
;--- распределение регистров
n      equ    [bp+8]
aai    equ    di      ;адрес a[i]
alast  equ    si      ;значение посл. элемента на четном месте
;--- организация доступа к параметрам
    mov     aai,[bp+6]
;--- подготовка цикла
;--- вычисление числа итераций
    mov     cx,n
    sar     cx,1
    dec     cx      ;cx=n/2-1      число итераций
    cmp     cx,0
    jle     fin      ;защита от n<=0
;--- вычисление адреса последнего элемента с четным номером
    add     aai,2
    mov     ax,cx

```



```

        sal     ax,2
        add     aai,ax           ;aai=aai+2+4*число итераций
;--- запомнить значение последнего элемента с четным номером
        mov     alast,[aai]
;--- цикл вычислений
cicl:   mov     ax,[aai-4]
        mov     [aai],ax        ;!!!косвенная адресация
        dec     aai             ;!!!переадресация
        dec     aai
        dec     aai
        dec     aai
        loop    cicl
;--- записать значение последнего элемента в первый
        mov     [aai],alast
;--- восстановление регистров
fin:
        popf
        pop     di
        pop     si
        pop     cx
        pop     bx
        pop     ax
        pop     bp
        ret     4               ;возврат к программе
shiftcr endp
prcode  ends
end

```

Варианты заданий

Преобразовать числовой массив:

1. Сдвинуть элементы влево на 1 позицию. Последний обнулить.
2. Сдвинуть элементы вправо на 1 позицию. Первый обнулить.
3. Сдвинуть элементы влево на 1 позицию циклически.
4. Сдвинуть элементы вправо на 1 позицию циклически.
5. Вставить 1 элемент под заданным номером.
6. Удалить 1 элемент с заданным номером.
7. Сдвинуть элементы влево на 2 позиции. 2 последних обнулить.
8. Сдвинуть элементы вправо на 2 позиции. 2 первых обнулить.
9. Сдвинуть элементы влево на 2 позиции циклически.
10. Сдвинуть элементы вправо на 2 позиции циклически.

11. Сдвинуть элементы на нечетных местах влево. Последний на нечетной позиции обнулить.

12. Сдвинуть элементы на нечетных местах вправо. Первый на нечетной позиции обнулить.

13. Сдвинуть элементы на нечетных местах влево циклически.

14. Сдвинуть элементы на нечетных местах вправо циклически.

15. Сдвинуть элементы на четных местах влево. Последний на четной позиции обнулить.

16. Сдвинуть элементы на четных местах вправо. Первый на четной позиции обнулить.

17. Сдвинуть элементы на четных местах влево циклически.

18. Сдвинуть элементы на четных местах вправо циклически.

Тесты

Описание теста	исходные данные	результат
1. вырожденный массив	n=0	
2. нет четных мест	n=1 a=(1)	a=(1)
3. 1 четное место(n чет)	n=2 a=(1,2)	a=(1,2)
4. 1 четное место(n неч)	n=3 a=(1,2,3)	a=(1,2,3)
5. 2 четных места(n чет)	n=4 a=(1,2,3,4)	a=(1,4,3,2)
6. 2 четных места(n неч)	n=5 a=(1,2,3,4,5)	a=(1,4,3,2,5)
7. 3 четных места	n=6 a=(1,2,3,4,5,6)	a=(1,6,3,2,5,4)

Лабораторная работа № 5

Цель работы – написать внешнюю подпрограмму с двумерным массивом и вызов ее из программы на C++. В программе на C++ предусмотреть меню со следующими пунктами: 1) ввод матрицы из файла; 2) вывод матрицы на экран; 3) проверка условия – вызов подпрограммы на ассемблере; 4) выход из программы.

Образец программы файл main.cpp

```
// main.cpp : main project file.  
// в программе тестируется подпрограмма на ассемблере, которая  
// определяет, существует ли в двумерном массиве столбец,  
// элементы которого – пилообразная последовательность  
#include <iostream>  
#include <fstream>  
#include <cstdlib>
```

```

#include <iomanip>
#include <locale>
#include <conio.h>
using namespace std;
// прототип подпрограммы на ассемблере
extern "C"
{
    bool isPila(int **a,int n);
}
// двумерный массив представлен динамическим массивом адресов
// столбцов (n элементов) и динамическими массивами столбцов
// (n столбцов). Хранение массива по столбцам, а не по строкам
// определяется задачей его обработки в подпрограмме на ассемб-
// лере, а именно тем, что требуется установить некоторое свойство
// столбцов (а не строк), т. е. во внутреннем цикле необходимо
// передвигаться по элементам столбцов.

int menu()
{
    int k;
    system("cls");
    cout<<"1. Ввод матрицы из файла"<<endl;
    cout<<"2. Печать матрицы на экран"<<endl;
    cout<<"3. Проверка пилообразности"<<endl;
    cout<<"4. Выход"<<endl;
    cout<<endl<<" Введите номер пункта меню "<<endl;
    cin>>k;

    return k;
}

// функция ввода матрицы учитывает размещение ее в памяти в
// в виде набора столбцов
int inMatrFile(int **a,int &n)
{
    ifstream f;
    char filename[100];
    // для удобства тестирования имена файлов с тестами имеют
    // вид числа без расширения имени
    cout<<"введите номер теста "<<endl;
    cin>>filename;

    f.open(filename);
    if(!f)
        return 0; // признак неудачи

```

```

        f>>n;
        a=new int *[n];           // динамич. массив адресов столбцов
        for(int i=0;i<n;i++)
            a[i]=new int[n];      // динамич. массивы столбцов
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                f>>a[j][i];

        f.close();
        return 1;                 // признак успешного завершения
    }
    // функция вывода матрицы учитывает размещение ее в памяти в
    // в виде набора столбцов
    void outMatr(int **a,int n)
    {
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
                cout<<setw(5)<<a[j][i];
            cout<<endl;
        }
    }
    int main()
    {
        int n, n_p=0;              //размерн. матрицы и № пункта меню
        int **a= NULL;             //матрица
        setlocale(LC_ALL,"");
        while (n_p!=4)
        {
            n_p=menu();
            cin.ignore(10,'\n');
            switch (n_p)
            {
                case 1:
                    if(a != NULL)   // если уже вводили матрицу и
                        {           // хотим ввести новую, то
                                    // освобождаем память
                                    for(int i=0;i<n;i++)
                                        delete []a[i];
                                    delete a;
                                }
                    if (!inMatrFile(a,n))
                        cout<<"Файл не найден";
                    break;
                case 2:
                    if(a)
                        20

```

```

        outMatr(a,n);
        else cout<<"Матрица не введена";
        break;
    case 3:
        if(a)
        {
            if(isPila(a,n)==1)
            cout<<"Характеристика верна"<<endl;
            else
            cout<<"Характеристика не верна"<<endl;
        }
        else cout<<"Матрица не введена";
        break;
    case 4:
        return 0;
        break;
    default:
        cout<<"Введено недопустим. значен.";
        break;
    }
    cout<<endl<<"Для продолж. нажм. любую клав."<<endl;
    getch();
}
if(a != NULL)    // если мы вводили матрицу - освоб. память.
{
    for(int i=0;i<n;i++)
        delete []a[i];
    delete a;
}
return 0;
}

```

Образец подпрограммы файл pila.asm

```

; определяет, есть ли в двумерном массиве столбец, элементы
; которого образуют пилообразную последовательность
;
;-----
;      заготовка на C++
;-----
;
;      if (n<3) return 0;
;      int i,j,goodStlb,dif1,dif2;
;      // цикл по столбцам
;      for (j=1; j < n; j++)

```

```

;
; {
; // начальные присвоения для цикла по столбцу
; goodStlb=1; // стлб. хороший - презумпция невиновности
; i=0;
; dif1=a[i+1][j]-a[i][j];
;
; // цикл по столбцу
; while (goodStlb && i < n-2)
; {
;     dif2=a[i+2][j]-a[i+1][j];
;     if !((dif1>0)&&(dif2<0))||((dif1<0)&&(dif2>0))
;         goodStlb=0;
;     dif1=dif2; i++;
; }
; if (goodStlb) return 1;
; }
; return 0;
;
;-----
; соображения о представлении двумерного массива в C++ для удобства
; работы в ассемблере
;-----
;--- так как при расположении строк исходного двумерного массива в
;--- виде динамических одномерных строк доступ к соседним элементам
;--- столбца делается в 2 такта
;
;     mov     esi,[edi]
;     mov     eax,[esi][ebx]    ;eax=a[i][j]
;
;
;     mov     esi,[edi+4]
;     mov     eax,[esi][ebx]    ;eax=a[i+1][j]
;--- то целесообразнее расположить в динамических строках столбцы
;--- исходного двумерного массива
;
;     mov     eax,[esi][ebx]    ;eax=a[i][j]
;
;
;     mov     eax,[esi][ebx+4]; eax=a[i+1][j]
;
;
; public isPila
;
; .586
; .model flat,c
; .data
; n          dd      ?          ;размер массива
; n_2        dd      ?          ;n_2=разм.массива – 2 (число итераций по стлб)
; .stack     ;сегмент стека
;           db 256 dup (" ") ;сегмент стека
; .code

```

```

isPila    proc
;--- сохранение регистров и фиксация в ebp значения указателя стека
    push    ebp
    mov     ebp,esp          ;ebp – указывает на вершину стека
    push    esi
    push    ecx
    add     ebp,8            ;пропускаем в стеке адрес возврата
;--- сохранение параметров в n и edi, вычисление n_2
    mov     eax,[ebp+4]
    mov     n,eax
    dec     eax
    dec     eax
    mov     n_2,eax          ;n_2=n-2
    mov     edi,[ebp]        ;адрес массива указателей на стлб
;--- теперь ebp можно задействовать в вычислениях
    mov     ecx,n            ;счетчик столбцов
;--- проверка на n < 3
    cmp     n,3
    jl      badSb
;-----
;--- цикл по столбцам
cStlb:
;--- начальные присвоения для цикла по столбцу
    mov     esi,[edi]        ;esi=адрес первого элемента в стлб
    mov     ebx,0            ;смещение по столбцу
    mov     ebp,ecx          ;сохр. знач. сч-ка внешнего цикла
    mov     ecx,n_2          ;выставить счетчик внутр. цикла в n-2
;--- сравнение a[i+1][j] и a[i][j] – первая пара элементов
    mov     eax,[esi][ebx+4]
    cmp     eax,[esi][ebx]
    jz      no               ;нет приращения
    js      minus1
    mov     dl,1             ;признак возрастания
    jmp     plus1
minus1:   mov                 dl,-1    ; признак убывания
plus1:
;--- цикл по элементам столбца
;-----
cStr:    mov     eax,[esi][ebx+8]
    cmp     eax,[esi][ebx+4] ;сравн. 2-й пара элементов
    jz      no               ;приращ. по столбцу нулевое
    js      minus2
    mov     dh,1             ;приращ. по столбцу положительное
    jmp     plus2
minus2:   mov                 dh,-1    ;приращ. по столбцу отрицат.

```

```

plus2:  cmp     dl,dh
        jz      no      ;плохо – два приращения одного знака
        mov     dl,dh   ;2-я пара на след. итер. станет первой
        add     ebx,4    ;переход к след. элементу столбца
        loop    cStr
;-----
;--- успешно прошли столбец – значит он хороший
        jmp     goodSb
;--- unsuccessfully passed column – try the next
no:
        mov     ecx,ebp ;восст. знач. счетчика внеш. цикла
        add     edi,4
        loop    cStlb
;-----
        jmp     badSb
goodSb:  mov     eax,1    ;успех – передача рез. через регистр eax
        jmp     fin
badSb:   mov     eax,0    ;неуспех – передача рез. через eax
fin:
;--- восстановление регистров
;       push     eax
;       pop      ecx
;       pop      esi
;       pop      ebp
        ret                ;возврат в программу
isPila  endp
end

```

Создание проекта matr_cpp_asm:

1. File -> new -> Project -> Visual C++ -> Win32 Console Application
-> Next -> Empty Project

2. В Solution Explorer

контекстное меню для проекта matr_cpp_asm -> Custom Build Rules ->

поставить галочку для Microsoft Macro Assembler

контекстное меню для папки Source Files -> Add -> New Item ->

C++ File -> main.cpp -> в окно скопировать, например, образец

Text File -> proiz0.asm -> в окно скопировать, например, образец

3. Поместить файл 3. в папку проекта

4. Запустить

Варианты условий

Они получаются при выборе номеров из 3-х групп (I,II,III)

(I)	(II)	(III)
1. каждая	1. строка	1. строго монотонная последовательность
2. существует	2. столбец	2. симметричная относительно центра последовательность
		3. сумма элементов нечетна
		4. количество нечетных элементов четно
		5. несимметричная относительно центра последовательность
		6. не является строго монотонной последовательность
		7. произведение элементов неотрицательное

Пример: вариант 227 означает выбор 2 из группы (I), 2 из группы (II) и 7 из группы (III), т. е. условие звучит так: существует столбец, в котором произведение элементов неотрицательное.

Тесты

Описание теста	Исходные данные	Результат
1. мала размерность	$n=2$	0
2. мин. размерность	$n=3$	
2a нет пилю		
2a1. (а по столбцам)	$a=\{\{1\ 1\ 2\}\{1\ 2\ 2\}\{1\ 2\ 3\}\}$	0
2a2. (а по столбцам)	$a=\{\{3\ 2\ 1\}\{1\ 1\ 1\}\{1\ 2\ 3\}\}$	0
2b есть пила		
2b1. (а по столбцам)	$a=\{\{1\ 2\ 1\}\{1\ 1\ 2\}\{2\ 2\ 1\}\}$	1
2b2. (а по столбцам)	$a=\{\{1\ 1\ 2\}\{2\ 1\ 2\}\{2\ 1\ 1\}\}$	1
2b3. (а по столбцам)	$a=\{\{1\ 2\ 3\}\{1\ 1\ 2\}\{1\ 2\ 1\}\}$	1
3. немин. размерность	$n=4$	
3a нет пилю		
3a1. (а по столбцам)	$a=\{\{1\ 1\ 2\ 1\}\{1\ 2\ 2\ 1\}\{1\ 2\ 1\ 1\}\{1\ 2\ 3\ 1\}\}$	0
3a2. (а по столбцам)	$a=\{\{2\ 1\ 2\ 3\}\{1\ 2\ 3\ 4\}\{4\ 3\ 2\ 1\}\{4\ 3\ 2\ 1\}\}$	0

3b есть пила		
3b1. (а по столбцам)	$a=\{\{1\ 2\ 1\ 2\}\{1\ 1\ 2\ 1\}\{1\ 2\ 2\ 1\}\{1\ 2\ 1\ 1\}\}$	1
3b2. (а по столбцам)	$a=\{\{1\ 1\ 2\ 1\}\{1\ 2\ 1\ 2\}\{2\ 1\ 1\ 2\}\{1\ 2\ 1\ 1\}\}$	1
3b3. (а по столбцам)	$a=\{\{1\ 1\ 2\ 1\}\{1\ 2\ 2\ 1\}\{1\ 2\ 1\ 2\}\{1\ 2\ 1\ 1\}\}$	1
3b4. (а по столбцам)	$a=\{\{1\ 1\ 2\ 1\}\{1\ 2\ 2\ 1\}\{1\ 2\ 1\ 1\}\{1\ 2\ 1\ 2\}\}$	1
3b5. (а по столбцам)	$a=\{\{1\ 1\ 2\ 1\}\{1\ 2\ 1\ 2\}\{1\ 2\ 1\ 1\}\{1\ 2\ 1\ 2\}\}$	1

Задание. Сформулировать, в чем отличие тестов, кроме тех, что указаны явно.

При отладке подпрограммы на ассемблере используйте технику отладки в проекте на C++: 1) установите точку прерывания на строке программы на C++, на которой идет обращение к подпрограмме на ассемблере; 2) запустите приложение Debug->Start Debugging; 3) при остановке в точке прерывания нажмите F11(Step Into); 4) открыть окно для наблюдения за регистрами: Debug->Windows->Registers; 5) нажимая на F10, выполнять строку за строкой подпрограмму на ассемблере, наблюдая при этом за содержимым регистров в окне Registers и наслаждаясь отладкой.

Лабораторная работа № 6

Цель работы – в программе на C++ сделать включение фрагмента на ассемблере, в котором выполняется обработка символьной строки с использованием строковых команд. В программе на C++ предусмотреть: 1) ввод строки; 2) вставка фрагмента обработки строки на ассемблере; 3) вывод результата.

Образец программы файл main.cpp

```
#include <iostream>
using namespace std;
// вводится строка из слов, которые разделяются ровно 1 пробелом.
// преобразовать эту строку в результирующую, которая состоит слов
// исходной строки с номерами 1 или 2 по модулю 3, и перевернуто каж-
// дое // слово с нечетным номером в результирующей строке.
void main()
{
    char str1[100];           // исходная строка
    int a[100];              // вспомогательный массив адресов //разделителей
                             // слов с нечетными номерами
```

```

char result[100];           // результирующая строка
cout << "Vvedite stroky" << endl;
gets(str1);
__asm
{
    push    ds                ;не трогать регистр EBP
    pop     es                ;es=ds т.е. es и ds адресует один сег-
МЕНТ
                                ;---обработка строки нулевой длины
    cmp     str1,0            ;str1==" "?
    jne     no_zero           ;результирующая строка
                                ;ненулевой длины
    mov     result,0          ;результир. строка нулевой длины
    jmp     fin
    ;--- подготовка цикла по str1
no_zero:
    lea     esi,str1          ;esi=адрес str1
    lea     edi,result        ;edi=адрес result
    lea     edx,a              ;edx=адрес массива адресов
    mov     a,edi
    dec     a                  ;адрес "разделителя" перед
1-м словом
    xor     ecx,ecx           ;номер слова по модулю 3
    inc     ecx
    cld
                                ;установить перемещ.
                                ;по строке вперед
    ;--- цикл по str1 - формирование результ. строки, но пока без ; перево-
    ;рота и формирование массива адресов разделителей ; слов в результи-
    ;рующем массиве
nextwrд:
    ;--- цикл по слову
nextch:lodsb
    cmp     al,0              ;сравн. с призн. конца строки
    je      finstr            ;конец строки
    cmp     al,' '             ;сравн. с пробелом между
словами
    je      finwrд            ;конец слова

    cmp     ecx,0              ;куда девать символ в al ?
    je      propusk           ;символ не берем в result
    stosb                     ;символ берем в result
propusk:
    jmp     nextch

finwrд:
                                ;постобработка слова

```

```

        cmp     ecx,0
        je      propus2 ;для пропущенного слова не записы-
вам
                                ;разделитель после него
        add     edx,4
        mov     [edx],edi ;сохранить адрес разделителя
                                ;после слова
        mov     al,' '
        stosb   ;записываем разделительной пробел
propus2:
        inc     ecx      ;ведение ecx – № слова по модулю 3
        cmp     ecx,3
        jne     no3
no3:
        xor     ecx,ecx  ;обнулить счетчик слов

        jmp     nextwrд
        ;-----
finstr:
                                ;постобработка посл. слова
        add     edx,4
        mov     [edx],edi ;сохраняем адрес разделителя после
слова
        mov     al,0
        cmp     ecx,0    ;если посл. слово в str1 не попадает
                                ;в result
        jne     m2
dec     edi              ;то пробел после него заменяется на
                                ;признак конца строки
m2:
        stosb   ;ввод признака конца строки
        ;--- подготовка к преобразованию result
        mov     edx,0    ;смещение в a
        xor     ecx,ecx  ;нач. уст. счетчика слов в result
        mov     esi,a[edx]
        inc     esi      ;уст. esi на первый символ слова
        ;--- цикл перебора слов в result
c_words:
        mov     edi,a[edx+4]
        dec     edi      ;установка edi на посл. символ слова
        inc     ecx      ;ведение счетчика слов
        test    cl,01h
        jz      no_preo ;слово с четным номером не преобраз.
        ;--- цикл переворота слова
c_preob:

```

```

mov     al,[esi]
mov     ah,[edi]
mov     [esi],ah
mov     [edi],al
inc     esi
dec     edi
cmp     esi,edi
jl      c_preob
;-----
no_preob:add     edx,4      ;переход к след. слову в рез.строке
mov     esi,a[edx]        ;стоим на разделителе
cmp     [esi],0
jz      fin              ;строка result закончилась
inc     esi               ;стоим на 1-м символе след. слова
jmp     c_words
;-----
fin:
}
cout << endl << "str1=" << str1 << endl;
cout << endl << "result=" << result << endl;
}

```

Варианты заданий

Вводится строка из слов, которые разделяются ровно 1 пробелом. Преобразовать эту строку в результирующую, которая состоит из слов исходной строки, определяемых условием I, причем часть из них перевернута (условие перевертывание II).

Условия на отбор в результирующую строку - это условие на номер слова в исходной строке:

- 1) номер равен 0 по модулю 2;
- 2) номер равен 1 по модулю 2;
- 3) номер равен 0 по модулю 3;
- 4) номер равен 1 по модулю 3;
- 5) номер равен 2 по модулю 3;
- 6) номер равен 0 или 1 по модулю 3;
- 7) номер равен 0 или 2 по модулю 3;
- 8) номер равен 1 или 2 по модулю 3.

Условие на переверот слова в результирующей строке:

- 1) длина слова четная
- 2) длина слова нечетная
- 3) номер слова в результирующей строке четный

4) номер слова в результирующей строке нечетный

В результате каждый вариант задания определяется парой номеров условий на отбор и переверот, например, вариант 84 означает условие на отбор номер 8 и условие на переверот номер 4.

Тесты

Описание теста	Исходные данные	Результат
1.пустая строка	str=""	result=""
1 слово		
2.1 длины 1	str="1"	result="1"
2.2 длины 2	str="12"	result="21"
2.3 длины 3	str="123"	result="321"
2.4 длины 4	str="1234"	result="4321"
2.5 длины 5	str="12345"	result="54321"
2 слова		
3.1 2е-длины 1	str="12 3"	result="21 3"
3.2 2е-длины 2	str="12 34"	result="21 34"
3 слова		
4.1 3е-длины 1	str="12 34 5"	result="21 34"
4.2 2е-длины 2	str="12 34 56"	result="21 34"
4 слова		
5.1 4е-длины 1	str="12 34 56 7"	result="21 34 7"
5.2 4е-длины 2	str="12 34 56 78"	result="21 34 87"
6. 5 слов	str="12 34 56 78 90"	result="21 34 87 90"
7. 6 слов	str="12 34 56 78 90 12"	result="21 34 87 90"
8. 7 слов	str="12 34 56 78 90 12 34"	result="21 34 87 90 43"

Приложения

Приложение 1

Структура программы и технология выполнения

```
title    <Разместите здесь заголовок>
page     ,132
stk      segment para stack 'stack'
db       64 dup('stack')
stk      ends
data     segment para public 'data'
;<Поместить здесь данные>
data     ends
;<Если требуется оператор extrn, поместить его здесь>
;<Если требуется оператор public, поместить его здесь>
code     segment para public 'code'
assume   cs:code,ss:stk,ds:data
start:   mov     ax,data
         mov     ds
;<Пометите здесь команды>
         mov     ah,4ch
         int     21h
code     ends
end      start
```

Создание объектного модуля:

```
tasm имя.asm ,,,;
```

Создание загрузочного модуля:

```
tlink имя.asm+имя1.asm+имя2.asm
```

Запуск отладчика:

```
td имя.exe
```

Для отладки полезно использование bat-файла a.bat с таким содержанием

```
tasm имя.asm ,,,;
```

```
tlink имя.asm+имя1.asm+имя2.asm
```

```
имя.exe
```

Подпрограммы rdint и writn ввода и вывода целых чисел

```

title      Подпрограммы ввода-вывода
stacksg segment stack
dw         130 dup(?)
stacksg ends
iodata segment
w10 dw 10
msg0 db 10,13,'overflow',10,13,'$'
msg1 db 10,13,'error simbol',10,13,'$'
msg2 db 10,13,'abort',10,13,'$'
iodata ends
io segment
public rdint
public writn
assume cs:io,ds:iodata,ss:stacksg

```

; после ввода целого числа с клавиатуры и нажатия на Enter число окажется в ;регистре ax

```

rdint proc far
push bx
push ds
pushf
mov ax,iodata
mov ds,ax
mov bx,0
mov ah,1
int 21h
cmp al,'-'
pushf
jne cif
incif: mov ah,1
int 21h
cif:   cmp al,'0'
jl next
cmp al,'9'
jg next
and al,0fh
cbw
xchg ax,bx
imul w10
jo over

```



```

        add     bx,ax
        jmp     incif
next:    cmp     al,0dh
        je      sign
        mov     dx,offset msgsg
        jmp     short outmsg
over:    mov     dx,offset msggo
outmsg:  mov     ah,9
        int     21h
        mov     dx,offset msgga
        int     21h
        mov     ah,4ch
        int     21h
sign:    popf
        jnz     fin
        neg     bx
fin:     mov     ax,bx
        popf
        pop     ds
        pop     bx
        ret
rdint    endp
; целое число из ax выводится на экран
wrint    proc    far
        push    ax
        push    dx
        push    cx
        push    bx
        push    ds
        pushf
        mov     bx,ax
        mov     ax,iodata
        mov     ds,ax
        cmp     bx,0
        jnl     cfr
        mov     dl,'-'
        mov     ah,2
        int     21h
        neg     bx
cfr:     mov     cx,0
        mov     ax,bx
nextcfr:cwd
        idiv    w10
        push    dx
        inc     cx

```

```

        cmp     ax,0
        jne     nextcfr
        mov     ah,2
outcfr: pop     dx
        or      dl,'0'
        int     21h
        loop    outcfr
        popf
        pop     ds
        pop     bx
        pop     cx
        pop     dx
        pop     ax
        ret
wrint   endp
io      ends
        end

```

Команды ассемблера

ADD

(ADDition)

Сложение

Схема команды: add приемник,источник

Назначение: сложение двух операндов источник и приемник размерностью байт, слово или двойное слово.

Алгоритм работы: сложить операнды источник и приемник; записать результат сложения в приемник; установить флаги.

Состояние флагов после выполнения команды:

OF	SF	ZF	AF	PF	CF
г	г	г	г	г	г

Применение: команда add используется для сложения двух целочисленных операндов. Результат сложения помещается по адресу первого операнда. Если результат сложения выходит за границы операнда приемник (возникает переполнение), то учесть эту ситуацию следует путем анализа флага cf и последующего возможного применения команды adc. Например, сложим значения в регистре ax и области памяти ch. При сложении следует учесть возможность переполнения

CALL

(CALL)

Вызов процедуры или задачи

Схема команды: call цель

Назначение: передача управления близкой или дальней процедуре с запоминанием в стеке адреса точки возврата; переключение задач.

Алгоритм работы: определяется типом операнда:

– метка ближняя – в стек заносится содержимое указателя команд eip/ir и в этот же регистр загружается новое значение адреса, соответствующее метке;

– метка дальняя – в стек заносится содержимое указателя команд `ip/ip` и `cs`. Затем в эти же регистры загружаются новые значения адресов, соответствующие дальней метке;

– `ri6, 32` или `mi6, 32` – определяют регистр или ячейку памяти, содержащие смещения в текущем сегменте команд, куда передается управление. При передаче управления в стек заносится содержимое указателя команд `ip/ip`;

– указатель на память – определяет ячейку памяти, содержащую 4- или 6-байтный указатель на вызываемую процедуру. Структура такого указателя 2+2 или 2+4 байта. Интерпретация такого указателя зависит от режима работы микропроцессора:

– в реальном режиме – в зависимости от размера адреса (`use16` или `use32`) первые два байта трактуются как сегментный адрес, вторые два/четыре байта – как смещение целевой метки передачи управления. В стеке запоминается содержимое регистров `cs` и `ip/ip`;

– в защищенном режиме – интерпретация цели передачи управления зависит от значения байта `AR` дескриптора, определяемого селекторной частью указателя. Целью здесь являются дальний вызов процедуры без изменения уровня привилегий, дальний вызов процедуры с изменением уровня привилегий или переключение задачи.

Состояние флагов после выполнения команды (кроме переключения задачи): выполнение команды не влияет на флаги.

При переключении задачи значения флажков изменяются в соответствии с информацией о регистре `eflags` в сегменте состояния `TSS` задачи, на которую производится переключение.

CBW/CWDE

(Convert Byte to Word/Convert Word to Double Word Extended)

Преобразование байта в слово/слова в двойное слово

Схема команды: `cbw`

`cwde`

Назначение: расширение операнда со знаком.

Алгоритм работы:

– `cbw` — при работе команда использует только регистры `ax` и `ax`:

- анализ знакового бита регистра al: если знаковый бит al=0, то ah=00h; если знаковый бит al=1, то ah=0ffh.

– cwde – при работе команда использует только регистры ax и eax:

- анализ знакового бита регистра ax: если знаковый бит ax=0, то установить старшее слово eax=0000h; если знаковый бит ax=1, то установить старшее слово eax=0ffffh.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги.

Применение: данные команды используются для приведения операндов к нужной размерности с учетом знака. Такая необходимость может, в частности, возникнуть при программировании арифметических операций.

CLD

(CLear Direction flag)

Сброс флага направления

Схема команды: cld

Назначение: сброс в ноль флага направления df.

Алгоритм работы: установка флага df в ноль.

Состояние флагов после выполнения команды:

10	DF	0
----	----	---

Применение: данная команда используется для сброса флага df в ноль. Такая необходимость может возникнуть при работе с цепочечными командами. Нулевое значение флага df вынуждает микропроцессор при выполнении цепочечных операций производить инкремент регистров si и di.

CMP

(CoMPare operands)

Сравнение операндов

Схема команды: cmp операнд1,операнд2

Назначение: сравнение двух операндов.

Алгоритм работы:

– выполнить вычитание (операнд1-операнд2);

– в зависимости от результата установить флаги, операнд1 и операнд2 не изменять (то есть результат не запоминать).

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
r	r	r	r	r	r

Применение: данная команда используется для сравнения двух операндов методом вычитания, при этом операнды не изменяются. По результатам выполнения команды устанавливаются флаги. Команда `cmp` применяется с командами условного перехода и командой установки байта по значению `setcc`.

CMPS/CMPSB/CMPSW/CMPSD

(CoMPare String Byte/Word/Double word operands)

Сравнение строк байтов/слов/двойных слов

Схема команды: `cmps` приемник, источник

`cmpsb`

`cmpsw`

`cmpsd`

Назначение: сравнение двух последовательностей (цепочек) элементов в памяти.

Алгоритм работы:

– выполнить вычитание элементов (источник - приемник), адреса элементов предварительно должны быть загружены:

- адрес источника — в пару регистров `ds:esi/si`;
- адрес назначения — в пару регистров `es:edi/di`;

– в зависимости от состояния флага `df` изменить значение регистров `esi/si` и `edi/di`: если `df=0`, то увеличить содержимое этих регистров на длину элемента последовательности; если `df=1`, то уменьшить содержимое этих регистров на длину элемента последовательности;

– в зависимости от результата вычитания установить флаги: если очередные элементы цепочек не равны, то `cf=1`, `zf=0`; если очередные элементы цепочек или цепочки в целом равны, то `cf=0`, `zf=1`;

– при наличии префикса выполнить определяемые им действия (см. команды `repe/repe`).

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
г	г	г	г	г	г

Применение: команды без префиксов осуществляют простое сравнение двух элементов в памяти. Размеры сравниваемых элементов зависят от применяемой команды. Команда `cmps` может работать с элементами размером в байт, слово, двойное слово. В качестве операндов в команде указываются идентификаторы последовательностей этих элементов в памяти. Реально эти идентификаторы используются лишь для получения типов элементов последовательностей, а их адреса должны быть предварительно загружены в указанные выше пары регистров. Транслятор, обработав команду `cmps` и выяснив тип операндов, генерирует одну из машинных команд `cmpsb`, `cmpsw` или `cmpsd`. Машинного аналога для команды `cmps` нет. Для адресации назначения обязательно должен использоваться регистр `es`, а для адресации источника можно делать замену сегмента с использованием соответствующего префикса.

Для того чтобы эти команды можно было использовать для сравнения последовательности элементов, имеющих размерность байт, слово, двойное слово, необходимо использовать один из префиксов `rep` или `repne`. Префикс `rep` заставляет циклически выполняться команды сравнения до тех пор, пока содержимое регистра `ecx/cx` не станет равным нулю или пока не совпадут очередные сравниваемые элементы цепочек (флаг `zf = 1`). Префикс `repne` заставляет циклически производить сравнение до тех пор, пока не будет достигнут конец цепочки (`ecx/cx = 0`) либо не встретятся различающиеся элементы цепочек (флаг `zf = 0`).

CWD

(Convert Word to Double word)

Преобразование слова в двойное слово

Схема команды: `cwd`

Назначение: расширение слова со знаком до размера двойного слова со знаком.

Алгоритм работы: копирование значения старшего бита регистра *ax* во все биты регистра *dx*. Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

Применение: команда *cwd* используется для расширения значения знакового бита в регистре *ax* на биты регистра *dx*. Данную операцию, в частности, можно использовать для подготовки к операции деления, для которой размер делимого должен быть в два раза больше размера делителя, либо для приведения операндов к одной размерности в командах умножения, сложения, вычитания.

DEC

(DECrement operand by 1)

Уменьшение операнда на единицу

Схема команды: *dec* операнд

Назначение: уменьшение значения операнда в памяти или регистре на 1.

Алгоритм работы: команда вычитает 1 из операнда.

Состояние флагов после выполнения команды:

11	07	06	04	02
OF	SF	ZF	AF	PF
r	r	r	r	r

Применение: команда *dec* используется для уменьшения значения байта, слова, двойного слова в памяти или регистре на единицу. При этом заметьте то, что команда не воздействует на флаг *cf*.

IDIV

(Integer DIVide)

Деление целочисленное со знаком

Схема команды: *idiv* делитель

Назначение: операция деления двух двоичных значений со знаком.

Алгоритм работы: для команды необходимо задание двух операндов — делимого и делителя. Делимое задается неявно, и размер его зависит от размера делителя, местонахождение которого указывается в команде: если делитель размером в байт, то делимое должно быть расположено в регистре *ax*. После опера-

ции частное помещается в `al`, а остаток — в `ah`; если делитель размером в слово, то делимое должно быть расположено в паре регистров `dx:ax`, причем младшая часть делимого находится в `ax`. После операции частное помещается в `ax`, а остаток — в `dx`; если делитель размером в двойное слово, то делимое должно быть расположено в паре регистров `edx:eax`, причем младшая часть делимого находится в `eax`. После операции частное помещается в `eax`, а остаток — в `edx`. Остаток всегда имеет знак делимого. Знак частного зависит от состояния знаковых битов (старших разрядов) делимого и делителя.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?	?	?	?	?	?

Применение: команда выполняет целочисленное деление операндов с учетом их знаковых разрядов. Результатом деления являются частное и остаток от деления. При выполнении операции деления возможно возникновение исключительной ситуации: 0 — ошибка деления. Эта ситуация возникает в одном из двух случаев: делитель равен 0 или частное слишком велико для его размещения в регистре `eax/ax/al`.

IMUL

(Integer MULtiply)

Умножение целочисленное со знаком

Схема команды: `imul множитель_1`

`imul множ_1,множ_2`

`imul рез-т,множ_1,множ_2`

Назначение: операция умножения двух целочисленных двоичных значений со знаком.

Алгоритм работы: алгоритм работы команды зависит от используемой формы команды. Форма команды с одним операндом требует явного указания местоположения только одного сомножителя, который может быть расположен в ячейке памяти или регистре. Местоположение второго сомножителя фиксировано и зависит от размера первого сомножителя: если операнд, указан-

ный в команде, – байт, то второй сомножитель располагается в `al`; если операнд, указанный в команде, – слово, то второй сомножитель располагается в `ax`; если операнд, указанный в команде, – двойное слово, то второй сомножитель располагается в `eax`.

Результат умножения для команды с одним операндом также помещается в строго определенное место, определяемое размером сомножителей: при умножении байтов результат помещается в `ax`; при умножении слов результат помещается в пару `dx:ax`; при умножении двойных слов результат помещается в пару `edx:eax`.

Команды с двумя и тремя операндами однозначно определяют расположение результата и сомножителей следующим образом: в команде с двумя операндами первый операнд определяет местоположение первого сомножителя. На его место впоследствии будет записан результат. Второй операнд определяет местоположение второго сомножителя; в команде с тремя операндами первый операнд определяет местоположение результата, второй операнд — местоположение первого сомножителя, третий операнд может быть непосредственно заданным значением размером в байт, слово или двойное слово.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
г	?	?	?	?	г

Команда `imul` устанавливает в ноль флаги `of` и `cf`, если размер результата соответствует регистру назначения. Если эти флаги отличны от нуля, то это означает, что результат слишком велик для отведенных ему регистром назначения рамок и необходимо указать больший по размеру регистр для успешного завершения данной операции умножения. Конкретными условиями сброса флагов `of` и `cf` в ноль являются следующие условия: для однооперандной формы команды `imul` регистры `ax/dx/edx` являются знаковыми расширениями регистров `al/ah/eax`; для двухоперандной формы команды `imul` для размещения результата умножения достаточно размерности указанных регистров назначения `r16/r32`; то же для трехоперандной команды умножения.

INC

(INCRement operand by 1)

Увеличить операнд на 1

Схема команды: inc операнд

Назначение: увеличение значения операнда в памяти или регистре на 1.

Алгоритм работы: команда увеличивает операнд на единицу.

Состояние флагов после выполнения команды:

11	07	06	04	02
OF	SF	ZF	AF	PF
r	r	r	r	r

Применение:

Команда используется для увеличения значения байта, слова, двойного слова в памяти или регистре на единицу. При этом команда не воздействует на флаг cf.

INT

(INTerrupt)

Вызов подпрограммы обслуживания прерывания

Схема команды: int номер_прерывания

Назначение: вызов подпрограммы обслуживания прерывания с номером прерывания, заданным операндом команды.

Алгоритм работы: записать в стек регистр флагов eflags/flags и адрес возврата. При записи адреса возврата вначале записывается содержимое сегментного регистра cs, затем содержимое указателя команд eip/ip; сбросить в ноль флаги if и tf; передать управление на программу обработки прерывания с указанным номером. Механизм передачи управления зависит от режима работы микропроцессора (см. уроки 15 и 17).

Состояние флагов после выполнения команды:

09	08
IF	TF
0	0

Применение: как видно из синтаксиса, существуют две формы этой команды: `int 3` – имеет свой индивидуальный код операции `0ссh` и занимает один байт. Это обстоятельство делает ее очень удобной для использования в различных программных отладчиках для установки точек прерывания путем подмены первого байта любой команды. Микропроцессор, встречая в последовательности команд команду с кодом операции `0ссh`, вызывает программу обработки прерывания с номером вектора 3, которая служит для связи с программным отладчиком.

Вторая форма команды занимает два байта, имеет код операции `0сdх` и позволяет инициировать вызов подпрограммы обработки прерывания с номером вектора в диапазоне 0–255. Особенности передачи управления, как было отмечено, зависят от режима работы микропроцессора.

JCC

JCXZ/JECXZ

(Jump if condition)

(Jump if CX=Zero/ Jump if ECX=Zero)

Переход, если выполнено условие

Переход, если CX/ECX равен нулю

Схема команды: jcc метка

jcxz метка

jecxz метка

Назначение: переход внутри текущего сегмента команд в зависимости от некоторого условия.

Алгоритм работы команд (кроме `jcxz/jecxz`):

– проверка состояния флагов в зависимости от кода операции (оно отражает проверяемое условие): если проверяемое условие истинно, то перейти к ячейке, обозначенной операндом; если проверяемое условие ложно, то передать управление следующей команде.

Алгоритм работы команды `jcxz/jecxz`:

– проверка условия равенства нулю содержимого регистра `есх/сх`: если проверяемое условие истинно, то есть содержимое `есх/сх` равно 0, то перейти к ячейке, обозначенной операндом метка; если проверяемое условие ложно, то есть содержимое `есх/сх` не равно 0, то передать управление следующей за `jcxz/jecxz` команде программы.

Состояние флагов после выполнения команды:

11	07	06	05	04	03	02	01	00
OF	SF	ZF	0	AF	0	PF	1	CF
?	?	?		r		?		r

Применение (кроме `jcxz/jesxz`): команды условного перехода удобно применять для проверки различных условий, возникающих в ходе выполнения программы. Как известно, многие команды формируют признаки результатов своей работы в регистре `eflags/flags`. Это обстоятельство и используется командами условного перехода для работы. Ниже приведены перечень команд условного перехода, анализируемые ими флаги и соответствующие им логические условия перехода.

Команда	Состояние проверяемых флагов	Условие перехода
JA	CF = 0 и ZF = 0	если выше
JAЕ	CF = 0	если выше или равно
JB	CF = 1	если ниже
JBE	CF = 1 или ZF = 1	если ниже или равно
JC	CF = 1	если перенос
JE	ZF = 1	если равно
JZ	ZF = 1	если 0
JG	ZF = 0 и SF = OF	если больше
JGE	SF = OF	если больше или равно
JL	SF <> OF	если меньше
JLE	ZF=1 или SF <> OF	если меньше или равно
JNA	CF = 1 и ZF = 1	если не выше
JNAЕ	CF = 1	если не выше или равно
JNB	CF = 0	если не ниже
JNBE	CF=0 и ZF=0	если не ниже или равно
JNC	CF = 0	если нет переноса
JNE	ZF = 0	если не равно
JNG	ZF = 1 или SF <> OF	если не больше

JNGE	SF \diamond OF	если не больше или равно
JNL	SF = OF	если не меньше
JNLE	ZF=0 и SF=OF	если не меньше или равно
JNO	OF=0	если нет переполнения
JNP	PF = 0	если количество единичных битов результата нечетно (нечетный паритет)
JNS	SF = 0	если знак плюс (знаковый (старший) бит результата равен 0)
JNZ	ZF = 0	если нет нуля
JO	OF = 1	если переполнение
JP	PF = 1	если количество единичных битов результата четно (четный паритет)
JPE	PF = 1	то же, что и JP, то есть четный паритет
JPO	PF = 0	то же, что и JNP
JS	SF = 1	если знак минус (знаковый (старший) бит результата равен 1)
JZ	ZF = 1	если ноль

Логические условия “больше” и “меньше” относятся к сравнениям целочисленных значений со знаком, а “выше и “ниже” – к сравнениям целочисленных значений без знака. Если внимательно посмотреть, то у многих команд можно заметить одинаковые значения флагов для перехода. Это объясняется наличием нескольких ситуаций, которые могут вызвать одинаковое состояние флагов. В этом случае с целью удобства ассемблер допускает несколько различных мнемонических обозначений одной и той же машинной команды условного перехода. Эти команды ассемблера по действию абсолютно равнозначны, так как это одна и та же машинная команда. Изначально в микропроцессоре i8086 команды условного перехода могли осуществлять только короткие переходы в пределах -128...+127 байт, считая от следующей команды. Начиная с микропроцессора i386, эти команды уже могли выполнять любые переходы в пределах текущего сегмента

команд. Это стало возможным за счет введения в систему команд микропроцессора дополнительных машинных команд. Для реализации межсегментных переходов необходимо комбинировать команды условного перехода и команду безусловного перехода `jmp`. При этом можно воспользоваться тем, что практически все команды условного перехода парные, то есть имеют команды, проверяющие обратные условия.

Применение `jcxz/jecxz`:

Команда	Состояние флагов в <code>eflags/flags</code>	Условие перехода
<code>JCXZ</code>	не влияет	если регистр <code>CX</code> =0
<code>JECXZ</code>	не влияет	если регистр <code>ECX</code> =0

Команду `jcxz/jecxz` удобно использовать со всеми командами, использующими регистр `ecx/cx` для своей работы. Это команды организации цикла и цепочечные команды. Очень важно отметить то, что команда `jcxz/jecxz`, в отличие от других команд перехода, может выполнять только близкие переходы в пределах -128...+127 байт, считая от следующей команды. Поэтому для нее особенно актуальна проблема передачи управления далее чем в указанном диапазоне. Для этого можно привлечь команду безусловного перехода `jmp`. Например, команду `jcxz/jecxz` можно использовать для предварительной проверки счетчика цикла в регистре `ecx` для обхода цикла, если его счетчик нулевой.

JMP

(**JuMP**)

Переход безусловный

Схема команды: `jmp метка`

Назначение: используется в программе для организации безусловного перехода как внутри текущего сегмента команд, так и за его пределы. При определенных условиях в защищенном режиме работы команда `jmp` может использоваться для переключения задач.

Алгоритм работы: команда `jmp` в зависимости от типа своего операнда изменяет содержимое либо только одного регистра `eax`, либо обоих регистров `cs` и `eax`: если операнд в команде `jmp` – метка в текущем сегменте команд (а8, 16, 32), то ассемблер формиру-

ет машинную команду, операнд которой является значением со знаком, являющимся смещением перехода относительно следующей за `jmp` команды. При этом виде перехода изменяется только регистр `еір/ір`; если операнд в команде `jmp` — символический идентификатор ячейки памяти (`m16`, `32`, `48`), то ассемблер предполагает, что в ней находится адрес, по которому необходимо передать управление. Этот адрес может быть трех видов:

1) значением абсолютного смещения метки перехода относительно начала сегмента кода. Размер этого смещения может быть 16 или 32 бит в зависимости от режима адресации;

2) дальним указателем на метку перехода в реальном и защищенном режимах, содержащим два компонента адреса — сегментный и смещение. Размеры этих компонентов также зависят от установленного режима адресации (`use16` или `use32`). Если текущим режимом является `use16`, то адрес сегмента и смещение занимают по 16 бит, причем смещение располагается в младшем слове двойного слова, отводимого под этот полный адрес метки перехода. Если текущим режимом является `use32`, то адрес сегмента и смещение занимают соответственно 16 и 32 бит — в младшем двойном слове находится смещение, в старшем — адрес сегмента;

3) адресом в одном из 16 или 32-разрядных регистров — этот адрес представляет собой абсолютное смещение метки, на которую необходимо передать управление, относительно начала сегмента команд.

Для понимания различий механизмов перехода в реальном и защищенном режимах нужно помнить следующее. В реальном режиме микропроцессор просто изменяет `сs` и `еір/ір` в соответствии с содержимым указателя в памяти. В защищенном режиме микропроцессор предварительно анализирует байт прав доступа `AR` в дескрипторе, номер которого определяется по содержимому сегментной части указателя. В зависимости от состояния байта `AR` микропроцессор выполняет либо переход, либо переключение задач.

Состояние флагов после выполнения команды (за исключением случая переключения задач): выполнение команды не влияет на флаги

LDS/LES/LFS/LGS/LSS

(Load pointer into ds/es/fs/gs/ss segment register)

Загрузка сегментного регистра ds/es/fs/gs/ss указателем из памяти

Схема команды: lds приемник,источник

les приемник,источник

lfs приемник,источник

lgs приемник,источник

lss приемник,источник

Назначение: получение полного указателя в виде сегментной составляющей и смещения.

Алгоритм работы: алгоритм работы команды зависит от действующего режима адресации (use16 или use32): если use16, то загрузить первые два байта из ячейки памяти источник в 16-разрядный регистр, указанный операндом приемник. Следующие два байта в области источник должны содержать сегментную составляющую некоторого адреса; они загружаются в регистр ds/es/fs/gs/ss; если use32, то загрузить первые четыре байта из ячейки памяти источник в 32-разрядный регистр, указанный операндом приемник. Следующие два байта в области источник должны содержать сегментную составляющую, или селектор, некоторого адреса; они загружаются в регистр ds/es/fs/gs/ss.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги.

Применение: таким образом, с помощью данных команд в паре регистров ds/es/fs/gs/ss и приемник оказывается полный адрес некоторой ячейки памяти. Это обстоятельство можно использовать, к примеру, при работе с цепочечными командами, где существуют жесткие соглашения на размещение адресов обрабатываемых строк. Помните, что любая загрузка сегментного регистра приводит к обновлению соответствующего теневого регистра (см. урок 16). Смотрите также описание команды `strps` с примером использования.

LEA

(Load Effective Address)

Загрузка эффективного адреса

Схема команды: lea приемник,источник

Назначение: получение эффективного адреса (смещения) источника.

Алгоритм работы: алгоритм работы команды зависит от действующего режима адресации (use16 или use32): если use16, то в регистр приемник загружается 16-битное значение смещения операнда источник; если use32, то в регистр приемник загружается 32-битное значение смещения операнда источник.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

LODS/LODSB/LODSW/LODS

(LOad String Byte/Word/Double word operands)

Загрузка строки байтов/слов/двойных слов

Схема команды: lods источник

lods

lods

lods

Назначение: загрузка элемента из последовательности (цепочки) в регистр-аккумулятор al/ax/eax.

Алгоритм работы: загрузить элемент из ячейки памяти, адресуемой парой ds:esi/si, в регистр al/ax/eax. Размер элемента определяется неявно (для команды lods) или явно в соответствии с применяемой командой (для команд lodsb, lodsw, lodsd); изменить значение регистра si на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df: df=0 – значение положительное, то есть просмотр от начала цепочки к ее концу; df=1 — значение отрицательное, то есть просмотр от конца цепочки к ее началу.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги.

LOOP

(LOOP control by register cx)

Управление циклом по cx

Схема команды: loop метка

Назначение: организация цикла со счетчиком в регистре cx.

Алгоритм работы: выполнить декремент содержимого регистра ecx/cx; анализ регистра ecx/cx: если ecx/cx=0, передать управление следующей за loop команде; если ecx/cx=1, передать

управление команде, метка которой указана в качестве операнда loop.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

LOOPE/LOOPZ

LOOPNE/LOOPNZ

(LOOP control by register cx not equal 0 and ZF=1)

(LOOP control by register cx not equal 0 and ZF=0)

Управление циклом по cx с учетом значения флага ZF

Схема команды: loopе/loopz метка

loopne/loopnz метка

Назначение: организация цикла со счетчиком в регистре cx с учетом флага zf.

Алгоритм работы: выполнить декремент содержимого регистра есх/сх; проанализировать регистр есх/сх: если есх/сх=0, передать управление следующей за loopxx командой; если есх/сх=1, передать управление команде, метка которой указана в качестве операнда loopxx; анализ флага zf: если zf=0, для команд loopе/loopz это означает выход из цикла, для команд loopne/loopnz — переход к началу цикла; если zf=1, для команд loopе/loopz это означает переход к началу цикла, для команд loopne/loopnz — выход из цикла.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

MOV

(MOVe operand)

Пересылка операнда

Схема команды: mov приемник,источник

Назначение: пересылка данных между регистрами или регистрами и памятью.

Алгоритм работы: копирование второго операнда в первый операнд.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

MOVS/MOVSБ/MOVSВ/MOVSД

(MOVe String Byte/Word/Double word)

Пересылка строк байтов/слов/двойных слов

Схема команды: `movs` приемник,источник

`movsb`

`movsw`

`movsd`

Назначение: пересылка элементов двух последовательностей (цепочек) в памяти.

Алгоритм работы: выполнить копирование байта, слова или двойного слова из операнда источника в операнд приемник, при этом адреса элементов предварительно должны быть загружены: адрес источника — в пару регистров `ds:esi/si` (`ds` по умолчанию, допускается замена сегмента); адрес приемника — в пару регистров `es:edi/di` (замена сегмента не допускается); в зависимости от состояния флага `df` изменить значение регистров `esi/si` и `edi/di`: если `df=0`, то увеличить содержимое этих регистров на длину структурного элемента последовательности; если `df=1`, то уменьшить содержимое этих регистров на длину структурного элемента последовательности; если есть префикс повторения, то выполнить определяемые им действия (см. команду `rep`).

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

Применение: команды пересылают элемент из одной ячейки памяти в другую. Размеры пересылаемых элементов зависят от применяемой команды. Команда `movs` может работать с элементами размером в байт, слово, двойное слово. В качестве операндов в команде указываются идентификаторы последовательностей этих элементов в памяти. Реально эти идентификаторы используются лишь для получения типов элементов последовательностей, а их адреса должны быть предварительно загружены в указанные выше пары регистров. Транслятор, обработав команду `movs` и выяснив тип операндов, генерирует одну из машинных команд `movsb`, `movsw` или `movsd`. Машинного аналога для команды `movs` нет. Для адресации операнда приемник обязательно должен использоваться регистр `es`.

Для того чтобы эти команды можно было использовать для пересылки последовательности элементов, имеющих размерность байт, слово, двойное слово, необходимо использовать пре-

фикс гер. Префикс гер заставляет циклически выполняться команды пересылки до тех пор, пока содержимое регистра esх/сх не станет равным нулю.

NEG

(NEGate operand)

Изменить знак операнда

Схема команды: рег источник

Назначение: изменение знака (получение двоичного дополнения) источника.

Алгоритм работы: выполнить вычитание (0 – источник) и поместить результат на место источника; если источник=0, то его значение не меняется.

Состояние флагов после выполнения команды (если результат нулевой):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
г	г	г	г	г	0

Состояние флагов после выполнения команды (если результат ненулевой):

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
г	г	г	г	г	1

NOP

(No OPeration)

Нет операции

Схема команды: пор

Назначение: пустая команда.

Алгоритм работы: не производит никаких действий.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

Применение: команда пор, занимая один байт, может использоваться для резервирования места в сегменте кода или организации программной задержки. В качестве иллюстрации можно обратиться к примеру, приведенному в описании команды hlt. В

этом примере команду `por` можно использовать вместо `jmp $+2`. Назначение `jmp $+2` в этом фрагменте — задержка для синхронизации работы микропроцессора и аппаратуры компьютера.

OR

(logical OR)

Логическое включающее ИЛИ

Схема команды: `or` приемник, маска

Назначение: операция логического ИЛИ над битами операнда назначения.

Алгоритм работы: выполнить операцию логического ИЛИ над битами операнда назначения, используя в качестве маски второй операнд — маска. При этом бит результата равен 0, если соответствующие биты операндов маска и назначения равны 0, в противном случае бит равен 1; записать результат операции в источник (операнд маска остается неизменным); установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
0	г	г	?	г	0

POP

(POP operand from the stack)

Извлечение операнда из стека

Схема команды: `pop` приемник

Назначение: извлечение слова или двойного слова из стека.

Алгоритм работы: алгоритм работы команды зависит от установленного атрибута размера адреса — `use16` или `use32`: загрузить в приемник содержимое вершины стека (адресуется парой `ss:esp/sp`); увеличить содержимое `esp/sp` на 4 (2 байта) для `use32` (соответственно для `use16`).

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

POPF

(POP Flags register from the stack)

Извлечение регистра флагов из стека

Схема команды: `popf`

Назначение: извлечение из стека слова и восстановление его в регистр флагов `flags`.

Алгоритм работы: извлечь из вершины стека слово и поместить его в регистр `flags`; увеличить значение указателя стека `esp` на 2.

Состояние флагов после выполнения команды:

14	1312	11	10	09	08	07	06	04	02	00
NT	IOPL	OF	DF	IF	TF	SF	ZF	AF	PF	CF
г	г	г	г	г	г	г	г	г	г	г

Применение: команда `rorf` по принципу работы является обратной команде `pushf` и используется для восстановления из стека содержимого регистра флагов `eflags`. Возможным вариантом использования этой команды являются программы обработки прерываний или другие случаи, в которых необходимо сохранять некоторый локальный контекст процесса вычисления. Из-за того, что регистр `eflags/flags` непосредственно недоступен, команда `rorf` является одной из немногих возможностей влияния на его содержимое.

PUSH

(PUSH operand onto stack)

Размещение операнда в стеке

Схема команды: `push` источник

Назначение: размещение содержимого операнда источник в стеке.

Алгоритм работы: уменьшить значение указателя стека `esp/sp` на 4/2 (в зависимости от значения атрибута размера адреса — `use16` или `use32`); записать источник в вершину стека (адресуемую парой `ss:esp/sp`).

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

PUSHF

(PUSH Flags register onto stack)

Размещение регистра флагов в стеке

Схема команды: `pushf`

Назначение: размещение в вершине стека (`ss:sp`) содержимого регистра флагов `flags`.

Алгоритм работы: уменьшить значение указателя стека `sp` на 2; поместить в вершину стека содержимое регистра `flags`.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

REP/REPE/REPZ/REPNE/REPNZ

(REPeat string operation)

Повторить цепочечную операцию

Схема команды: гер

gere

gerz

gerne

gernz

Назначение: указание условного и безусловного повторения следующей за данной командой цепочечной операции.

Алгоритм работы: алгоритм работы зависит от конкретного префикса. Префиксы `гер`, `gere` и `gerz` на самом деле имеют одинаковый код операции, их действия зависят от той цепочечной команды, которую они предваряют:

- `гер` используется перед следующими цепочечными командами и их краткими эквивалентами: `movs`, `stos`, `ins`, `outs`. Действия `гер`: анализ содержимого `sx`:

- если $sx > 0$, то выполнить цепочечную команду, следующую за данным префиксом и перейти к шагу 2;

- если $sx = 0$, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по `гер`);

- уменьшить значение $sx = sx - 1$ и вернуться к шагу 1;

- `gere` и `gerz` используются перед следующими цепочечными командами и их краткими эквивалентами: `cmps`, `scas`. Действия `gere` и `gerz`: анализ содержимого `sx` и флага `zf`:

- если $sx > 0$ или $zf < 0$, то выполнить цепочечную команду, следующую за данным префиксом, и перейти к шагу 2;

- если $sx = 0$ или $zf = 0$, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по `гер`); уменьшить значение $sx = sx - 1$ и вернуться к шагу 1;

- `gerne` и `gernz` также имеют один код операции и имеют смысл при использовании перед следующими цепочечными

командами и их краткими эквивалентами: `cmpr`, `scas`. Действия `герне` и `герпз`: анализ содержимого `сх` и флага `zf`:

- если $сх < 0$ или $zf = 0$, то выполнить цепочечную команду, следующую за данным префиксом и перейти к шагу 2;

- если $сх = 0$ или $zf < 0$, то передать управление команде, следующей за данной цепочечной командой (выйти из цикла по `гер`);

- уменьшить значение $сх = сх - 1$ и вернуться к шагу 1.

Состояние флагов после выполнения команды:

06

ZF

г

Применение: команды `гер`, `gere`, `герз`, `герне` и `герпз` в силу специфики своей работы называются префиксами. Они имеют смысл только при использовании цепочечных операций, заставляя их циклически выполняться и тем самым без организации внешнего цикла обрабатывать последовательности элементов фиксированной длины. Большинство применяемых префиксов являются условными, то есть они прекращают работу цепочечной команды при выполнении определенных условий.

RET/RETF

(RETurn/RETurn Far from procedure)

Возврат ближний (дальний) из процедуры

Схема команды: `ret`

`ret` число

Назначение: возврат управления из процедуры вызывающей программе.

Алгоритм работы: работа команды зависит от типа процедуры: для процедур ближнего типа — восстановить из стека содержимое `еір/ір`; для процедур дальнего типа — последовательно восстановить из стека содержимое `еір/ір` и сегментного регистра `сs`. если команда `ret` имеет операнд, то увеличить содержимое `esp/sp` на величину операнда число; при этом учитывается атрибут режима адресации — `use16` или `use32`:

- если `use16`, то $sp = (sp + \text{число})$, то есть указатель стека сдвигается на число байт, равное значению число;

– если use32, то $sp=(sp+2*\text{число})$, то есть указатель стека сдвигается на число слов, равное значению число.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги.

SAL

(Shift Arithmetic operand Left)

Сдвиг арифметический операнда влево

Схема команды: `sal операнд, количество_сдвигов`

Назначение: арифметический сдвиг операнда влево.

Алгоритм работы: сдвиг всех битов операнда влево на один разряд, при этом выдвигаемый слева бит становится значением флага переноса cf; одновременно справа в операнд вдвигается нулевой бит; указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение: команда sal используется для сдвига разрядов операнда влево. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов количество_разрядов. Аналогично другим командам сдвига сохраняется эффект, связанный с поведением флага of, значение которого имеет смысл только в операциях сдвига на один разряд: если of=1, то текущее значение флага cf и выдвигаемого слева бита операнда различны; если of=0, то текущее значение флага cf и выдвигаемого слева бита операнда совпадают.

Этот эффект, как вы помните, обусловлен тем, что флаг cf устанавливается в единицу всякий раз при изменении знакового разряда операнда.

Команду sal удобно использовать для умножения целочисленных операндов без знака на степени 2. Кстати сказать, это самый быстрый способ такого умножения; умножить содержимое ax на 16 (2 в степени 4).

SAR

(Shift Arithmetic operand Right)

Сдвиг арифметический операнда вправо

Схема команды: `sar операнд, количество_сдвигов`

Назначение: арифметический сдвиг операнда вправо.

Алгоритм работы: сдвиг всех битов операнда вправо на один разряд, при этом выдвигаемый справа бит становится значением флага переноса cf; обратите внимание: одновременно слева в операнд вдвигается не нулевой бит, а значение старшего бита операнда, то есть по мере сдвига вправо освобождающиеся места заполняются значением знакового разряда. По этой причине этот тип сдвига и называется арифметическим; указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение: команда `sar` используется для арифметического сдвига разрядов операнда вправо. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда `количество_разрядов`. В отличие от других команд сдвига флаг `of` всегда сбрасывается в ноль в операциях сдвига на один разряд.

Команду `sar` можно использовать для деления целочисленных операндов со знаком на степени 2.

SCAS/SCASB/SCASW/SCASD

Сканирование строки байтов/слов/двойных слов

ASCII-коррекция после сложения

Схема команды: `scas приемник`

`scasb`

`scasw`

`scasd`

Назначение: поиск значения в последовательности (цепочке) элементов в памяти.

Алгоритм работы: выполнить вычитание (элемент цепочки- $(eax/ax/al)$). Элемент цепочки локализуется парой $es:edi/di$. Замена сегмента es не допускается; по результату вычитания установить флаги; изменить значение регистра edi/di на величину, равную длине элемента цепочки. Знак этой величины зависит от состояния флага df : $df=0$ — величина положительная, то есть просмотр от начала цепочки к ее концу; $df=1$ — величина отрицательная, то есть просмотр от конца цепочки к ее началу.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
г	г	г	г	г	г

Применение: команды сканирования сравнивают значение в регистре $eax/ax/al$ с ячейкой памяти, локализуемой парой регистров $es:edi/di$. Размер сравниваемого элемента зависит от применяемой команды. Команда $scas$ может работать с элементами размером в байт, слово или двойное слово. В качестве операнда в команде указывается идентификатор последовательности элементов в памяти. Реально этот идентификатор используется лишь для получения типа элементов последовательности, а ее адрес должен быть предварительно загружен в указанную выше пару регистров. Транслятор, обработав команду $scas$ и выяснив тип операндов, генерирует одну из машинных команд: $scasb$, $scasw$ или $scasd$. Машинного аналога для команды $scas$ нет. Для адресации операнда источник обязательно должен использовать регистр es .

Для того чтобы эту команду можно было использовать для поиска значения в последовательности элементов, имеющих размерность байт, слово или двойное слово, необходимо использовать один из префиксов $hrep$ или $repne$. Эти префиксы не только заставляют циклически выполняться команду поиска, пока $ecx \neq 0$, но и отслеживают состояние флага zf (см. команды $hrep/repne$).

SHL

(SHift logical Left)

Сдвиг логический операнда влево

Схема команды: shl операнд, количество_сдвигов

Назначение: логический сдвиг операнда влево.

Алгоритм работы: сдвиг всех битов операнда влево на один разряд, при этом выдвигаемый слева бит становится значением флага переноса cf; одновременно слева в операнд вдвигается нулевой бит; указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	00
OF	CF
?r	r

Применение: команда shl используется для сдвига разрядов операнда влево. Ее машинный код идентичен коду sal, поэтому вся информация, приведенная для sal, относится и к команде shl. Команда shl используется для сдвига разрядов операнда влево. Так же, как и для других сдвигов, значение второго операнда (счетчик сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество_разрядов. Аналогично другим командам сдвига сохраняется эффект, связанный с поведением флага of, значение которого имеет смысл только в операциях сдвига на один разряд: если of = 1, то текущее значение флага cf и выдвигаемого слева бита операнда различны; если of = 0, то текущее значение флага cf и выдвигаемого слева бита операнда совпадают.

Этот эффект, как вы помните, обусловлен тем, что флаг of устанавливается в единицу всякий раз при изменении знакового разряда операнда.

Команду shl удобно использовать для умножения целочисленных операндов без знака на степени 2. Кстати сказать, это самый быстрый способ умножения; умножить содержимое ax на 16 (2 в степени 4).

SHR

Сдвиг логический операнда вправо

Схема команды: shr операнд, кол-во_сдвигов

Назначение: логический сдвиг операнда вправо.

Алгоритм работы: сдвиг всех битов операнда вправо на один разряд, при этом выдвигаемый справа бит становится значением флага переноса cf; одновременно слева в операнд вдвигается нулевой бит; указанные выше два действия повторяются количество раз, равное значению второго операнда.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
?r	r	r	?	r	r

Применение: команда shr используется для логического сдвига разрядов операнда вправо. Так же, как и для других сдвигов, значение второго операнда (счетчика сдвига) ограничено диапазоном 0...31. Это объясняется тем, что микропроцессор использует только пять младших разрядов операнда количество_разрядов. В отличие от других команд сдвига, флаг of всегда сбрасывается в ноль в операциях сдвига на один разряд.

Команду shr можно использовать для деления целочисленных операндов без знака на степени 2.

STD

(SeT Direction Flag)

Установка флага направления

Схема команды: std

Назначение: установка флага направления df в 1.

Алгоритм работы: установить флаг df в единицу.

Состояние флагов после выполнения команды:

10

DF

1

Применение: данная команда используется для установки флага df в единицу. Такая необходимость может возникнуть при работе с цепочечными командами. Единичное состояние флага df вынуждает микропроцессор производить декремент регистров si и di при выполнении цепочечных операций.

STOS/STOSB/STOSW/STOSD

(Store String Byte/Word/Double word operands)

Сохранение строки байтов/слов/двойных слов

Схема команды: stos приемник

stosb

stosw

stosd

Назначение: сохранение элемента из регистра-аккумулятора al/ax/eax в последовательности (цепочке).

Алгоритм работы: записать элемент из регистра al/ax/eax в ячейку памяти, адресуемую парой es:di/edi. Размер элемента определяется неявно (для команды stos) или конкретной применяемой командой (для команд stosb, stosw, stosd); изменить значение регистра di на величину, равную длине элемента цепочки. Знак этого изменения зависит от состояния флага df: df = 0 – увеличить, что означает просмотр от начала цепочки к ее концу; df = 1 – уменьшить, что означает просмотр от конца цепочки к ее началу.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

Применение: команды сохраняют элемент из регистров al/ax/eax в ячейку памяти. Перед командой stos можно указать префикс повторения гер, в этом случае появляется возможность работы с блоками памяти, заполняя их значениями в соответствии с содержимым регистра esх/сх.

SUB

(SUBtract)

Вычитание

Схема команды: sub операнд_1,операнд_2

Назначение: целочисленное вычитание.

Алгоритм работы: выполнить вычитание
операнд_1=операнд_2-операнд_1; установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
г	г	г	г	г	г

Применение: команда sub используется для выполнения вычитания целочисленных операндов или для вычитания младших частей значений многобайтных операндов.

TEST

(TEST operand)

Логическое И

Схема команды: test приемник,источник

Назначение: операция логического сравнения операндов приемник и источник размерностью байт, слово или двойное слово.

Алгоритм работы: выполнить операцию логического умножения над операндами приемник и источник: бит результата равен 1, если соответствующие биты операндов равны 1, в остальных случаях бит результата равен 0; установить флаги.

Состояние флагов после выполнения команды:

11	07	06	02	00
OF	SF	ZF	PF	CF
0	г	г	г	0

Применение: команда test используется для логического умножения двух операндов. Результат операции, в отличие от команды and, никуда не записывается, устанавливаются только флаги. Эту команду удобно использовать для получения информации о состоянии заданных битов операнда приемник. Для анализа результата используется флаг zf, который равен 1, если результат логического умножения равен нулю.

XCHG

(eXCHanGe)

Обмен

Схема команды: xchg операнд_1,операнд_2

Назначение: обмен двух значений между регистрами или между регистрами и памятью.

Алгоритм работы: обмен содержимого операнд_1 и операнд_2.

Состояние флагов после выполнения команды: выполнение команды не влияет на флаги

Применение: команду xchg можно использовать для выполнения операции обмена двух операндов с целью изменения порядка следования байт, слов, двойных слов или их временного сохранения в регистре или памяти. Альтернативой является использование для этой цели стека.

XOR

Логическое исключающее ИЛИ

ASCII-коррекция после сложения

Схема команды: хог приемник,источник

Назначение: операция логического исключающего ИЛИ над двумя операндами размерностью байт, слово или двойное слово.

Алгоритм работы: выполнить операцию логического исключающего ИЛИ над операндами: бит результата равен 1, если значения соответствующих битов операндов различны, в остальных случаях бит результата равен 0; записать результат сложения в приемник; установить флаги.

Состояние флагов после выполнения команды:

11	07	06	04	02	00
OF	SF	ZF	AF	PF	CF
0	г	г	?	г	0

Применение: команда хог используется для выполнения операции логического исключающего ИЛИ двух операндов. Результат операции помещается в первый операнд. Эту операцию удобно использовать для инвертирования или сравнения определенных битов операндов.

Оглавление

Лабораторная работа № 1	3
Лабораторная работа № 2.....	5
Лабораторная работа № 3.....	7
Лабораторная работа № 4.....	14
Лабораторная работа № 5.....	19
Лабораторная работа № 6.....	27
Приложения	33
Приложение 1. Структура программы и технология выполнения.....	33
Приложение 2. Подпрограммы rdint и wrint ввода и вывода целых чисел.....	34
Приложение 3. Команды ассемблера	37

Учебное издание

Н. Б. Федотов

Практикум на ЭВМ. Ассемблер

Учебное пособие

Редактор, корректор М. В. Никулина
Верстка И. Н. Иванова

Подписано в печать 19.07.11. Формат 60×84 1/16.

Бум. офсетная. Гарнитура «Times New Roman».

Усл. печ. л. 3,95. Уч.-изд. л. 2,01.

Тираж 30 экз. Заказ

Оригинал-макет подготовлен в редакционно-издательском отделе
Ярославского государственного университета им. П. Г. Демидова.

Отпечатано на ризографе.

Ярославский государственный университет им. П. Г. Демидова.

150000, Ярославль, ул. Советская, 14.

