Матвеева Т.К., Матвеев Ф.В.

Инструктивно-справочные материалы по практикуму на MASM'е. Часть 1.

(исправленный и дополненный вариант 2012 г.)

Настоящая разработка предназначена для студентов 1 курса, изучающих язык программирования MASM.
Все ссылки на страницы в дальнейшем тексте являются ссылками на учебное пособие В.Н.ПИЛЬЩИКОВ "Программирование на языке ассемблера IBM PC", первые главы которого посвящены темам

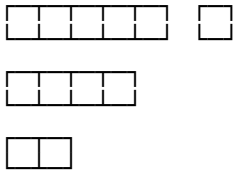
Глава 1. ОСОБЕННОСТИ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА
Глава 2. ЯЗЫК АССЕМБЛЕРА. НАЧАЛЬНЫЕ СВЕДЕНИЯ.
Глава 3. ЯЗЫК АССЕМБЛЕРА. ПЕРЕСЫЛКИ. АРИФМЕТИЧЕСКИЕ КОМАНДЫ.
Глава 4. ЯЗЫК АССЕМБЛЕРА. ПЕРЕХОДЫ. ЦИКЛЫ.

Целью настоящей методической разработки является проиллюстрировать минимальным набором решенных задач главы 1-4.

ПРИМЕРЫ ЗАДАЧ С РЕШЕНИЯМИ.

1. Указать значение константы K после выполнения выписанных директив.

```
N EQU 5
X DB N DUP(0), 5
Y DD ?
Z DW ?
K EQU Z – X
```



Ответ: K = Z – X = это константное выражение; оно равно количеству байтов памяти, начиная с байта с именем X до байта с именем Z, т.е. 5+1 + 4 = 10

2. Указать значение константы Lx после выполнения выписанных директив.

```
K  EQU 20
X  DB (3*K–1)/2 DUP(?)
Y  DW ?
Lx EQU Y–X
```

Ответ: см. на с39 таблицу всех операторов по старшинству (она же приведена в приложении А)
Lx = Y–X = (3*20–1)/2=59/2 { оператор / соответствует div} = 29

3.
```
A DB 5, 3, 8, 4
B DB 6, 6, 6
```
Указать значение регистра AL после выполнения команд:

MOV AL, A ADD AL, 2	MOV AL, A+2	MOV AL, B–3

Ответ: простейшие адресные выражения вида <имя> ± <целое используются для ссылок на безымянные байты.

AL = 7	AL = 8	AL = 3

4. Описать переменную T размером слово, начальным значением которой является:

а) адрес этой же переменной; ответ: T DW $
б) адрес следующего слова памяти; ответ: T DW $+2
в) адрес предыдущего слова памяти. ответ: T DW $–2

К простейшим адресным выражениям относится счетчик размещения, он записывается как $ и обозначает адрес того предложения, в котором он встретился (см сс32,40).

5. Описать байтовый массив X из 101 элемента со следующими начальными значениями
первые 50 элементов имеют значение 10, следующие 20 элементов – значение '+', остальные – без начального значения;

Ответ: X DB 50 DUP(10), 20 DUP('+'), 31 DUP(?) (см. сс26-30 ; в память записывается код символа '+').

6.
```
N EQU 10
```
Описать единичную матрицу E размера NxN из элементов-слов.

Ответ:
```
N EQU 10
K EQU N–1                ;        см. сс34,35 имея N, можно определить K при желании
E DW K DUP ( 1, N DUP(0) ), 1
;        E DW N–1 DUP ( 1, N DUP(0)), 1  ; другой вариант ответа без K
```

Пояснение: для N=4 имеем массив (~матрицу) 1000 0100 0010 0001 или запишем так: 10000 10000 10000 1

7. X db 97h

Какое a) *беззнаковое десятичное* число b) *знаковое десятичное* число представляет байт X ?

Ответы: **a) 151 b) -105**

РЕШЕНИЕ:

a) Возможные значения X ∈ [0, 255]. Вычислим 97h = 9*16+7 = 151d ; получили допустимое значение, 151 является ответом.

b) Возможные ***знаковые*** значения байта X ∈ [−128, +127] , соответствующие коды [128, 255] для отрицательных и [0, 127] для положительных чисел.

У нас $X_{доп.код}$ = 151(=97h), значит X<0 и |X|= 256 − 151 = 105 , **вывод:** X= **−** 105

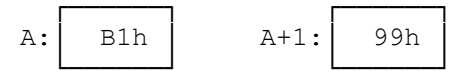
(или |X|= 100h − 97h = 69h = 105d , **вывод:** X= **−** 105)

```
( или          1 0000 0000b
             −  1001 0111b
                0110 1001b = 69h = 105d , вывод: X= − 105
                64+32+8+1= 105d )
```

ПЕРЕСЫЛКИ

8. A DW 99B1h .

A) Какое десятичное знаковое число окажется в регистре BH после выполнения команды MOV BH, byte ptr A ? (числа размером слово хранятся в памяти в перевернутом виде см. с32, оператор ptr с45)



РЕШЕНИЕ: 0B1h = 177 ==> −79

Ответ: BH=**−79**

Б) Какое десятичное знаковое число окажется в регистре BH после выполнения команды MOV BH,byte ptr A+1?

РЕШЕНИЕ: 99h =9*16+9=90+54+9=153 ==>256-153=|103|==> -103)

Ответ: BH=**−103**

В) Какое десятичное знаковое число окажется в регистре после выполнения команды MOV AX, A ?

Ответ: AX = **99B1h** (не путать с хранением в памяти)

Решения следующих задач представлены в виде фрагментов сегмента данных и сегмента команд.

АРИФМЕТИКА

9. Вычислить Y:= (X div 38)*(X mod 38) (не более чем 6 команд), если

```
X DB ?
Y DW ?
```

Ответ: числа БЕЗ знака	Ответ: числа СО знаком
...	
MOV AL, X	MOV AL, X
MOV AH, 0	CBW
MOV BL, 38	MOV BL, 38
DIV BL	IDIV BL
MUL AH	IMUL AH
MOV Y, AX	MOV Y, AX
...	

10. Идет n-я секунда суток. Определить, сколько полных часов (h) и минут (m) прошло к этому моменту (Выбираем размеры: n – двойное слово, т.к. секунд в сутках 86400, h и m – байты).

```
  ...
mov ax,word ptr n
mov dx,word ptr n+2     ; (dx,ax):=n (=h*3600+m*60+s)
mov bx,3600             ; число секунд в часе
div bx                  ; dx:=n mod 3600,  ax:=n div 3600 (=h)
mov h,al                ; h:=часы   { (ah,al)=(0,h) }
mov ax,dx               ; ax:=m*60+s
mov bh,60               ; число секунд в минуте
div bh                  ; al:=m, ah:=секунды
mov m,al                ; m:=минуты
  ...
```

СРАВНЕНИЕ. ПЕРЕХОДЫ. ЦИКЛЫ.

11. Выпишите значение байта A (A db ?) и флагов CF, OF, ZF и SF после выполнения следующих команд:

```
MOV A, -1
MOV AL, 96h     ; 96h = 150 => -106
CMP AL,-100d    ; -106 < -100 ==> goto L
JLE  L          ; сравнение знаковых чисел, т.к. код JLE
MOV A, 1        ; эта команда не выполняется, т.к. goto L выполняется
L:
```

Ответы: A=–1 CF=1 OF=0 ZF=0 SF=1

РЕШЕНИЕ: 96h=9*16+6=90+54+6=150==>256-150=|106|==>AL:=-106 (см.7)

CMP – вычитание без сохранения результата см.с67, переходы после сравнения – с68 .

```
-100==>256-100=156=9Ch

 _-106d=96h   _96h    _1001 0110    CF=1, т.к.нужен заем
  -100d=9Ch    9Ch     1001 1100
-------------------------------------------------------
    -6d        FAh     1111 1010    ZF=0, SF=1, OF=0,
                                                т.к.(-)-(-)=( нет переполнения)
 |res_sub| =100h-0FAh=6 ==>  res_sub = -6
```

12. Вычислить u = max (x,y,z) для знаковых чисел размером в слово.

```
        ...
      mov ax,x
      cmp ax,y
      jL  m1          ;if x<y goto m1
      cmp ax,z
      jge ex          ;if x>=z (max=x) goto ex
      jmp m2          ; y <= x < z goto m2
m1:   mov ax,y
      cmp ax,z
      jge ex          ;if y>=z (max=y) goto ex
m2:   mov ax,z        ;(max=z)
ex:   mov u,ax
       ...
```

13. Определить c – наибольший общий делитель (НОД) натуральных чисел a и b (a, b, c – беззнаковые числа размером в слово).

```
; цикл while
        ...
      mov ax,a
      mov bx,b
m1:   cmp ax,bx     ; цикл while
      je  m3        ; ax=bx  -> m3 ( команды перехода флаги не меняют )
      jb  m2        ; ax<bx  -> m2
      sub ax,bx     ; при ax>bx из ax вычесть bx
      jmp m1
m2:   sub bx,ax     ; при ax<bx из bx вычесть ax
      jmp m1
m3:   mov c,ax      ; c:=НОД(a,b)
        ...
```

ВВОД-ВЫВОД.

Далее используется набор команд ввода-вывода, которые описаны в гл.4.4 с.75.
В приложении Б перечислены все наиболее употребительные команды.

14. Ввести 15 различных знаковых целых чисел размером слово, найти и напечатать наибольшее из них и его номер.

```
; цикл for             ...
       n equ 15        ; константа
         outch '>'     ; приглашение к вводу
         inint dx      ; ввести 1-е число ==> dx (max)
         mov  di,1     ; di - номер max
         mov  si,1     ; si - номер очередного числа
         mov  cx,n-1   ; счетчик цикла
    lp:  inint ax      ; ввести очередное число ==> ax
         inc  si       ; вычислить его номер
         cmp  ax,dx
         jLe  lp1      ; ax<=dx -> lp1
         mov  dx,ax    ; появилось большее число - запомнить его
         mov  di,si    ; и его номер
    lp1: loop lp       ; цикл cx раз см.с71
         outint dx     ; вывод ответа
         outword di,4
         newline
           ...
```

ЛИТЕРНЫЕ ДАННЫЕ.

15. (Перевод строки символов в целое число)

Ввести непустую последовательность десятичных цифр, за которой задан пробел, и перевести ее в целое число k (считать, что число "умещается" в слово).

```
       k dw ?
; цикл while
         outch '>'        ; приглашение к вводу
         mov bx,0         ; bx - вводимое число
         mov ch,0         ; очистка старшего байта регистра cx
    beg: inch cl          ; ввести очередной символ => cl
         cmp cl," "
         je  fin          ; пробел -> fin

         sub cl,"0"       ; перевод цифры из символа в число
         mov ax,10
         mul bx           ; ax:=10*bx   dx=0
         mov bx,ax
         add bx,cx        ; bx:=10*bx+цифра
         jmp beg          ; на начало цикла
    fin: mov k,bx         ; k:=введенное число
           ...
; цикл repeat
         mov bx,0         ; bx – число
         mov ch,0         ; очистка старшего байта регистра cx
         inch cl          ; ввод 1-го символа, т.к. текст непустой по условию
    beg: sub cl,'0'       ; перевод цифры из символа в число
         mov ax,10
         mul bx           ; (dx,ax):=10*bx, dx=0
         mov bx,ax
         add bx,cx        ; bx:=10*bx+цифра
         inch cl          ; ввести очередной символ => cl
         cmp cl,' '       ; пробел ли ?
         jne beg          ; на начало цикла repeat
         mov k,bx         ; k – введенное посимвольно число
           ...
```

ПРАКТИЧЕСКАЯ РАБОТА.

Образец оформления файла с текстом программы (программного файла).

```
;       файл:  aschem.asm
Include ..\masm_exe\io.asm ; путь зависит от среды
;подключены операции вв/вывода

S Segment stack
      db 128 dup (?)
S Ends

D Segment
;==============================
; ПЕРЕМЕННЫЕ И КОНСТАНТЫ


;==============================
D Ends

C Segment
   Assume ss:S, ds:D, cs:C
start:
   mov ax, D
   mov ds, ax
;==============================
; КОМАНДЫ ПРОГРАММЫ


;==============================
   finish
C Ends

   End start
```

Далее приводятся четыре программы, оформленные в соответствии с образцом aschem.asm.

```
;                    файл: ex1.asm
;Вывод чисел знаковых / беззнаковых
Include ..\masm_exe\io.asm
;подключены операции ввода-вывода
S Segment stack
      db 128 dup (?)
S Ends

Cd Segment
   Assume ss:S, cs:Cd
start:
;============================
mov ax,-1000
; нет операндов, обращающихся к ячейкам памяти, нет загрузки DS
;mov aL,-100     ; aL:=-100
;cbw             ; ax:=-100
outint ax        ; -1000 ;   -100
outword ax,8     ; 64536 ;  65436
newline
;============================
   finish
Cd Ends
   End start


;                     файл: ex2.asm
;   X:= ( 2*A - 400 div (A+B)2 ) mod 7
; описание аналогичной программы можно найти в [3] гл.7
Include ..\masm_exe\io.asm
;подключены операции вв/вывода
S Segment stack
      dw 128 dup (?)
S Ends
```

```
D Segment
  A dw ?
  B db -10
  X dw ?
D Ends
C Segment
  Assume ss:S, ds:D, cs:C
start:
  mov ax, D
  mov ds, ax
;--------------------------------
inint A     ; A=20
mov bx, A   ; bx:=A
mov al, B   ; B=-10
cbw         ; ax:= B слово
add ax, bx  ; ax:=B+A=A+B
add bx, bx  ; bx:=2*A
imul ax     ; (dx,ax):=(A+B)2
mov cx, ax  ; cx:=(A+B)2
mov ax, 400 ;
cwd         ; (dx,ax):=400
idiv cx     ; ax:=400 div (A+B)2
sub bx, ax  ; bx:=2*A-400div(A+B)2
mov ax, bx  ; ax:=bx=y
cwd
mov bx,7
idiv bx     ; dx:= y mod 7
mov X, dx
outint X    ; X=1
;--------------------------------
   finish
C Ends
   End start


;                  файл: ex3.asm
; Условные переходы после cmp
Include ..\masm_exe\io.asm
S Segment stack
      db 128 dup (?)
S Ends

Dat Segment

; ПЕРЕМЕННЫЕ И КОНСТАНТЫ
 A DB  20 ; -20 ; 120 ; 120 ; 216
 B DB  20 ; -40 ;  70 ; 216 ; 120
;      <=    >     >     <=    <
Dat Ends

Cd Segment
   Assume ss:S, ds:Dat, cs:Cd
start:
   mov ax,Dat
   mov ds,ax
;============================
; КОМАНДЫ ПРОГРАММЫ
  mov AL, A
  cmp AL, B
  JL   L1
  JBE  L2
  JG   L3
    outch '-'
    JMP L4
L1:
    outch '<'
    JMP L4
L2:
    outch '<'
    outch '='
    JMP L4
```

```
L3:
    outch '>'
L4:    NEWLINE
;============================
   finish
Cd Ends
   End start


;               файл: ex4.asm
; y:=n!=n(n-1)(n-2) ... *2
Include ..\masm_exe\io.asm
stack segment stack
      dw 128 dup (?)
stack ends
data  segment
;============================
; переменные и константы
  n db  ?    ; n>1
  y DW  ?
;============================
data  ends
code segment
     assume ss:stack, ds:data
     assume cs:code
start:
     mov ax, data
     mov ds, ax
;============================
 outch ">"
 inint bx
 mov n, bl
 mov ax, bx
 dec bx
 mov cx, bx
 dec cx
L: mul bx    ; ax=n(n-1)
 cmp dx, 0   ; dx ? 0
 jne M1
 dec bx
 outword ax

 outch ' '
 loop L
 mov y, ax
 newline
 outword y
 jmp M2
M1:
 outch 'E'
 outch 'R'
M2:
;============================
    finish
code ends
    end start

 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
>5
20 60 120
120

>8
56 336 1680 6720 20160 40320
40320

>9
72 564 3024 15120 60480 ER
```

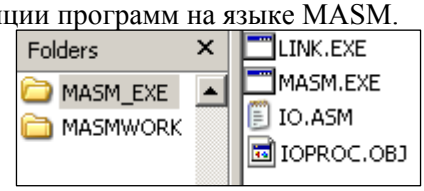
КРАТКАЯ ИНСТРУКЦИЯ ПО ВЫПОЛНЕНИЮ ЗАДАНИЙ ПРАКТИКУМА НА ЯЗЫКЕ MASM

В конкретных условиях (например, в машинном классе факультета) инструкция может отличаться.

1. СОСТАВ КАТАЛОГОВ MASM_EXE И MASMWORK

Каталоги MASM_EXE и MASMWORK находятся на одном уровне в дереве каталогов.

В каталоге MASM_EXE собраны все файлы, необходимые для компиляции программ на языке MASM.



masm.exe - ассемблер (компилятор языка ассемблера и макрокоманд)
link.exe - редактор связей (компановщик)
io.asm - операции ввода-вывода, а точнее, описание макросов, вызывающих процедуры ввода/вывода из файла ioproc.obj
ioproc.obj - файл с оттранслированными процедурами ввода-вывода; эти процедуры подключаются к программе автоматически при выполнении командного файла maa.bat

В каталоге MASMWORK находятся следующие файлы:
aschem.asm - образец структуры программы на языке MASM.
maa.bat - командный файл, выполняющий ассемблерирование, загрузку и счет (одномодульной) программы на языке MASM, имя которой (без расширения!) задается как параметр для maa.bat (например: maa firstprg или MAA FIRSTPRG или MAA.BAT FIRSTPRG)

Кроме того, в каталоге MASMWORK должны находиться компилируемые программы.

2. ПОДГОТОВКА ПРОГРАММЫ

Файл с текстом программы на языке MASM должен иметь расширение asm (например, firstprg.asm) и соответствовать образцу aschem.asm .

Одной из возможных сред для работы является файловый менеджер FAR с помощью которого можно скопировать образец под новым именем и отредактировать. Аналоги VC, NC, TC, ...

Директива `Include ..\masm_exe\io.asm` в тексте программы предполагает, что файл io.asm лежит в каталоге MASM_EXE ; компилируемая программа находится в каталоге MASMWORK и каталоги MASMWORK и MASM_EXE находятся на одном уровне в дереве каталогов.

3. КОМПИЛЯЦИЯ И ВЫПОЛНЕНИЕ СКОМПИЛИРОВАННОЙ ПРОГРАММЫ

Чтобы оттранслировать, загрузить и выполнить программу необходимо сформировать команду (например: **maa.bat firstprg**) в командной строке и выполнить её нажатием клавиши Enter, находясь в каталоге MASMWORK. Параметром командного файла **maa.bat** должно быть имя файла с текстом программы **без расширения .asm** .

В файле maa.bat команды ..\masm_exe\masm %1,%1,%1 и ..\masm_exe\link %1+..\masm_exe\ioproc.obj,%1 предполагают, что файлы masm.exe, link.exe и ioproc.obj находятся в каталоге MASM_EXE , находящемся на том же уровне в дереве каталогов, что и MASMWORK (текущий каталог), из которого запускается maa.bat .

4. АНАЛИЗ ОШИБОК КОМПИЛЯЦИИ ПРОГРАММЫ.

При успешной компиляции будут созданы два новых файла - файл с расширением .lst, в который записывается листинг программы с дополнительной информацией и файл с расширением .exe, в который помещается оттранслированная программа. В случае, если при трансляции были обнаружены ошибки или предупреждения, они указываются в листинге программы.

Внимание! Исправлять ошибки надо в файле с расширением .asm , а не .lst .

5. ПОВТОРНОЕ ВЫПОЛНЕНИЕ ПРОГРАММЫ.

Оттранслированную программу можно выполнить снова без перекомпиляции, если не требуется никаких изменений в тексте программы. Для этого можно установить курсор на полученный файл с расширением .exe и нажать клавишу Enter .

Во время трансляции, загрузки и счета программы с экрана исчезнут панели FAR, но после выполнения программы они снова появятся. Поэтому на экране не будет видно результатов работы программы. Чтобы их увидеть, надо нажать клавиши Ctrl+O (латинская буква O), а для возврата в обычный режим нажать эти же клавиши повторно.

ФАЙЛ: maa.bat

```
@echo off
rem Если в качестве параметра передано имя файла с программой без расширения
if not exist %1.asm goto f3 ; при передаче masm.exe в качестве параметра полного имени файла
rem                         ; (с расширением) текст программы может быть потерян
rem             Компилятор создает из него объектный модуль %1.obj
..\masm_exe\masm %1,%1,%1;
if errorlevel 1 goto f1
echo                    Этап компиляции преодолен ...
rem
rem Редактор связей создает из объектного модуля исполняемый файл %1.exe
..\masm_exe\link %1+..\masm_exe\ioproc.obj,%1;
rem
rem Объектный модуль удаляется
del %1.obj
if errorlevel 1 goto f2
rem
echo Выполнение программы началось:
%1.exe
echo.
echo Выполнение программы закончено.
goto fin
rem _____________________________________________________________
:f1
echo Ошибка компиляции !!!
del %1.obj
goto fin
:f2
echo Ошибка редактирования связей !
goto fin
:f3
echo Файл с программой не обнаружен.
echo Возможно, в качестве параметра ошибочно указано имя файла с расширением.
:fin
echo on
```

ПРИЛОЖЕНИЯ.

Приложение А. Старшинство всех операторов ЯА (Языка Ассемблера)

В порядке убывания; в каждой строке указаны операторы одного старшинства:

1. `(), [], LENGTH, SIZE, WIDTH, MASK`
2. `.`
3. `:`
4. `PTR, OFFSET, SEG, TYPE, THIS`
5. `HIGH, LOW`
6. одноместные **+** и **-**
7. `*, /, MOD, SHL, SHR`
8. двухместные **+** и **-**
9. `EQ, NE, LT, LE, GT, GE`
10. `NOT`
11. `AND`
12. `OR, XOR`
11. `SHORT, .TYPE`

Операторы одного старшинства выполняются слева направо.
Например, A+B–C выполняются так (A+B)-C.

Приложение Б.

Команды языка Ассемблера для микропроцессора 8086 (1978г. 16-ти разрядный с памятью 1Мб)
(Используемые обозначения см. в конце файла перед литературой).

1 КОМАНДЫ **ПЕРЕСЫЛОК**

Синтаксис**:** КОП op1, op2 (или КОП op или КОП) **Действие:** op1 **:=** op2 (основное действие)

MOV	r**,** i|r|m	B|W	Пересылка значения операнда op2 в op1	c43
MOV	m**,** i|r	B|W	Пересылка значения операнда op2 в op1	c43
MOV	r|m**,** SR	W	Пересылка значения сегментного регистра SR в op1	c43
MOV	SR**,** r|m	W	Пересылка значения op2 в сегментный регистр SR, но не в CS	c43
XCHG	r**,** r|m	B|W	Обмен содержимым между операндами op1 и op2	c47
XCHG	m**,** r	B|W	Обмен содержимым между операндами op1 и op2	c47
PUSH	r16|m16	W	Загрузка содержимого операнда op в стек	c143
PUSH	SR	W	Загрузка значения сегментного регистра SR в стек	c143
PUSHF		W	Загрузка содержимого регистра FLAGS в стек	c145
POP	r16|m16	W	Пересылка слова из стека в регистр или по исполнительному адресу	c144
POP	SR	W	Пересылка слова из стека в сегментный регистр SR (кроме CS)	c144
POPF		W	Пересылка слова из стека в регистр FLAGS флагов	c145
LEA	r16**,** m16	W	Загрузка в регистр op1 вычисленного исполнительного адреса op2	c91
LDS	r16, m32	D	Загрузка содержимого двойного слова в r16:= [m32] и DS := [m32+2]	c178
LES	r16, m32	D	Загрузка содержимого двойного слова в r16:= [m32] и ES := [m32+2]	c178

В командах MOV, XCHG операнды должны быть одинакового! размера и *не* быть вида память-память (*не* m, m).
Команды пересылок не меняют флаги, кроме команды POPF.
Запись слова в стек происходит так**:** сначала значение регистра SP уменьшается на 2 (сдвигается вверх), затем в свободную ячейку стека идет запись. Считывание слова из стека**:** верхнее слово, на которое указывает SS**:**SP, пересылается в операнд и значение регистра SP увеличивается на 2 (сдвигается вниз). Отметим, что вычисление SP ± 2 происходит по модулю 2^{16} .

2 **АРИФМЕТИЧЕСКИЕ** команды.

ADD	r, i|r|m	B|W	Сложение op1:= op1 + op2	c50
ADD	m, i|r	B|W	Сложение op1:= op1 + op2	c50
SUB	r, i|r|m	B|W	Вычитание op1:= op1 – op2	c50
SUB	m, i|r	B|W	Вычитание op1:= op1 – op2	c50
ADC	r, i|r|m	B|W	Сложение *с учетом предыдущего CF* op1:= op1 + op2 + CF	c51
ADC	m, i|r	B|W	Сложение *с учетом предыдущего CF* op1:= op1 + op2 + CF	c51
SBB	r, i|r|m	B|W	Вычитание *с учетом предыдущего CF* op1:= op1 – op2 – CF	c51
SBB	m, i|r	B|W	Вычитание *с учетом предыдущего CF* op1:= op1 – op2 – CF	c51
CMP	r, i|r|m	B|W	Сравнение операндов, аналог SUB без сохранения результата в op1	c67
CMP	m, i|r	B|W	Сравнение операндов, аналог SUB без сохранения результата в op1	c67
INC	r|m	B|W	Увеличение значения операнда op на 1	c51
DEC	r|m	B|W	Уменьшение значения операнда op на 1	c51
NEG	r|m	B|W	Изменение значения операнда op на противоположное	c51
MUL	r|m	B|W	Умножение *беззнаковое* значения регистра AL (или регистра AX) на значение op в команде; результат в регистре AX (или в регистрах <DX, AX>)	c53
IMUL	r|m	B|W	Умножение *знаковое* значения регистра AL (или регистра AX) на значение op, указанного в команде результат в регистре AX (или в регистрах <DX, AX>)	c53
DIV	r|m	B|W	Деление *беззнаковое* значения в регистре AX (или в регистрах <DX, AX>) на значение op, указанного в команде; результаты в регистрах AH:= *mod* (или в DX:= *mod*) и AL:= *div* (или AX:= *div*)	c55
IDIV	r|m	B|W	Деление *знаковое* значения в регистре AX (или в регистрах <DX, AX>) на значение op, указанного в команде; результаты в регистрах AH:= *mod* (или в DX:= *mod*) и AL:= *div* (или AX:= *div*)	c55
CBW			Преобразование значения *знакового байта*, находящегося в AL, в *слово*-результат в регистре AX	c57
CWD			Преобразование значения *знакового слова*, находящегося в AX, в *двойное слово* – <DX, AX>	c58

В командах **ADD, SUB, ADC, SBB, CMP** операнды должны быть одинакового! размера и *не* быть вида память-память (m, m). По этим командам устанавлива-ются все флаги (c49). Команды **INC, DEC** меняют все флаги, кроме CF. По команде **NEG** операнд не меняется в особых случаях при op= –128 (80h) и op= –32768 (8000h) при этом OF=1, в остальных случаях OF=0; при нулевом операнде CF=0, иначе =1; флаги SF и ZF меняются как обычно. По командам ***умножения*** CF и OF устанавливаются синхронно так: если произведение не превосходит размера сомножителей, то в 0, иначе в 1 (c54).

3 **ЛОГИЧЕСКИЕ** команды и команды **СДВИГА** (выполняются **поразрядно**).

NOT	r|m	B|W	Поразрядное инвертирование каждого бита операнда op	c104
AND	r**,** i|r|m	B|W	Логическое умножение (конъюнкция) поразрядное op1**:=** op1 and op2	c104
AND	m**,** i|r	B|W	Логическое умножение (конъюнкция) поразрядное op1**:=** op1 and op2	c104
TEST	r**,** i|r|m	B|W	Логическое умножение *and без сохранения результата* в op1	c105
TEST	m**,** i|r	B|W	Логическое умножение *and без сохранения результата* в op1	c105
OR	r**,** i|r|m	B|W	Логическое сложение (дизъюнкция) поразрядное op1**:=** op1 or op2	c105
OR	m**,** i|r	B|W	Логическое сложение (дизъюнкция) поразрядное op1**:=** op1 or op2	c105
XOR	r**,** i|r|m	B|W	Исключающее *или* op1**:=** op1 xor op2 : i-й бит результата равен нулю, если i-е биты операндов	
XOR	m**,** i|r	B|W	*совпдают* и i-й бит результата равен 1, если i-е биты операндов *различны* (сложение по mod 2).	

Логические команды меняют все флаги; но после них обычно используется флаг нуля ZF; ZF= 1, если результат нулевой во всех разрядах и ZF= 0, если в результате есть хоть одна двоичная единица. c106

Каждая команда сдвига имеет две разновидности <мнемокод> op, 1 или <мнемокод> op, CL; второй операнд рассматривается как целое без знака и определяет на сколько разрядов сдвигать; слова сдвигаются так, как если бы они в памяти хранились в непере-вернутом виде. Результат сдвига записывается на место первого операнда. Команды сдвига меняют все флаги; но используется CF.

Далее в CF заносится значение "уходящего" бита, а с другого конца добавляется 0.

SHR/**SHL**	r|m, 1	B|W	*Логический* сдвиг op1 вправо/влево на 1 разряд	c108
SHR/**SHL**	r|m, CL	B|W	*Логический* сдвиг op1 вправо/влево на содержимое байтового регистра CL	c108

(**SAL** = другое название команды SHL)

SAR	r|m, 1	B|W	*Арифметический* сдвиг op1 вправо на 1 разряд и знаковый разряд восстанавливается	c110
SAR	r|m, CL	B|W	*Арифметический* сдвиг op1 вправо на CL разрядов и знаковый разряд восстанавливается	c110

Далее попрежнему в CF заносится значение "уходящего" бита и в освобождающийся бит добавляется *он же*!

ROR/**ROL**	r|m, 1	B|W	*Циклический* сдвиг op1 вправо/влево на 1 разряд	c110
ROR/**ROL**	r|m, CL	B|W	*Циклический* сдвиг op1 вправо/влево на CL разрядов	c111

Далее в CF заносится значение "уходящего" бита, но *прежде* старое значение CF заносится в освобождающийся бит. c111

RCR/**RCL**	r|m, 1	B|W	*Циклический* сдвиг через перенос op1 вправо/влево на 1 разряд	c111
RCR/**RCL**	r|m, CL	B|W	*Циклический* сдвиг через перенос op1 вправо/влево на CL разрядов	c111

Быстрое умножение, деление и остаток от деления (cc109,115). Особенности деления отрицательных чисел c110.

4 КОМАНДЫ **УПРАВЛЕНИЯ** (основные, наиболее употребительные варианты)

Синтаксис: КОП op или КОП (не меняют флаги!).

Безусловные близкие (внутрисегментные– меняется только регистр счетчика адреса IP).

JMP	<метка>		Переход прямой (длинный или при наличии short короткий)	cc63–65
JMP	r16 | m16	W	Переход косвенный (длинный) по адресу, содержащемуся в операнде	cc34, 66, 67
JMP word ptr	r16 | m16		или**…** в случае ссылки вперед	cc66, 67,152

Безусловные дальние (межсегментные– устанавливаются оба регистра и IP, и CS).

JMP far ptr	<метка>		Переход прямой (всегда с **far ptr**, т.к. дальний)	c152
JMP	m32	DD	Переход косвенный (X dd L ; X+0 – ofs L , X+2 – seg L)	c151
JMP dword ptr	m32		или**…** в случае ссылки вперед	c152

CALL	<имя процедуры>	Вызов процедуры (переход с возратом**:** запись в стек адреса следующей команды и передача управления на первую команду процедуры)	c155
CALL far ptr	<имя процедуры>	или**…** в случае ссылки вперед	c156
RET		Возврат из процедуры (считывание из вершины стека адреса и переход по нему	c155
RET	i16	Возврат из подпрограммы (после извлечения из стека адреса возврата, увеличивается значение регистра SP на число байтов, равное беззнаковому значению i16, затем передается управление по адресу возврата)	c163

Условные близкие (внутрисегментные) все прямые короткие.

Jxx	<метка>	Условный переход см. далее (на следующем листе уточнения)	c68
LOOP	<метка>	CX := CX–1; переход по адресу adr8, если CX <> 0	c71
LOOPE/Z	<метка>	CX := CX–1; переход по адресу adr8, если CX<>0 и ZF=1	c73
LOOPNE/NZ	<метка>	CX := CX–1; переход по адресу adr8, если CX<>0 и ZF=0	c74
JCXZ	<метка>	Переход по адресу adr8, если CX=0	c69

IP и CS – регистры центрального процессора, задают адрес команды, которая должна выполнится следующей. Короткий переход "adr8" – адрес перехода в пределах области текущего сегмента, ограниченный 128-байтовым "расстоянием" (∈ [–128, +127]) от команды перехода (с 64). NEAR и FAR – стандартные знаковые константы со значениями –1 и –2 (c152).
Type <метка> или type <имя процедуры> имеют эти значения: NEAR или FAR.

4а **УСЛОВИЯ** перехода в командах вида **Jxx** <метка>

Мнемокод		Переход, если условие	Значения флагов	
Переходы после команды сравнения для знаковых чисел.				с68
JL/JNGE	<	меньше/не больше и не равно	SF<>OF	
JGE/JNL	≥	больше или равно/не меньше	SF = OF	
JLE/JNG	≤	меньше или равно/не больше	(SF<>OF) or (ZF=1)	
JG/JNLE	>	больше/не меньше и не равно	(SF=OF) and (ZF=0)	
Переходы после команды сравнения для беззнаковых чисел.				с68
JB/JNAE	<	меньше/не больше и не равно	CF=1	
JAE/JNB	≥	больше или равно/не меньше	CF=0	
JBE/JNA	≤	меньше или равно/не больше	(CF=1) or (ZF=1)	
JA/JNBE	>	больше/не меньше и не равно	(CF=0) and (ZF=0)	
Переходы после команд, устанавливающих тот или иной флаг			Значение флага	с69
JE/JZ	=	результат равен нулю	ZF=1	
JNE/JNZ	≠	результат не равен нулю	ZF=0	
JC/JNC		есть/нет перенос	CF=1 / 0	
JO/JNO		есть/нет переполнение	OF=1 / 0	
JS/JNS		отрицатательный/положительный результат	SF=1 / 0	

Для всех этих команд реализован один формат – близкий прямой короткий относительный переход.
Переход короткий описан выше (следует учитывать при программировании с64).
Переход прямой , т.к. в качестве операнда указывается метка команды, которой надо передать управление.
Переход относительный потому, что в машинной команде указывается не сам адрес, а разность между командой перехода и адресом перехода (адрес, точнее разность вместо метки подставит сам ассемблер).

JB/JNAE/ JC – одна машинная команда, аналогично **JAE/JNB/ JNC** – одна машинная команда.

5 МАКРОКОМАНДЫ **ВВОДА/ВЫВОДА.**

INCH	r8|m8	B	ввод символа (его кода) в байтовый операнд: регистр или память	c75
OUTCH	i8|r8|m8		B вывод символа (без кавычек)	c76
ININT	r16|m16		W ввод целого десятичного числа (если число внутри допустимого диапазона, иначе ошибка)	c75
OUTINT	i16|r16|m16	W	вывод десятичного числа со знаком размером в слово	c77
OUTINT	i16|r16|m16**,** leng		W вывод десятичного числа со знаком размером в слово в формате: второй операнд leng типа byte i8|r8|m8 (со значением ≥ 0) задает ширину поля вывода	c77
OUTWORD	i16|r16|m16	W	вывод десятичного числа без знака размером в слово	c77
OUTWORD	i16|r16|m16**,** leng		W вывод десятичного числа без знака размером в слово в формате: второй операнд leng типа byte i8|r8|m8 (со значением ≥ 0) задает ширину поля вывода	c77
OUTSTR			вывод строки символов: DS:DX – адрес начала строки; в конце строки знак $ (код символа 36=24h)	cc76,91
NEWLINE			перевод строки	c76
FINISH			останов	c75

Для inch допускается набор сразу нескольких символов (до Enter, Backspace-отмена последнего символа, Esc-отмена всего текста)
Операнд i8 в outch – это код символа, или символ в кавычках; получаем символ без кавычек в результате.
Особенности ввода числа по inint: число начинается знаком или цифрой, все пробелы и концы строк до пропускаются, ввод идет до первой нецифры, в том числе Enter (он глотается); можно набрать несколько чисел для ввода в цикле, допускается редактирование при наборе.
Особенности вывода числа по outint (аналогично для outword): если ширина поля вывода больше, чем надо, то слева добавляются пробелы; если меньше – выводится только число.

6 КОМАНДЫ **ОБРАБОТКИ СТРОК И БЛОКОВ ДАННЫХ**

Синтаксис строковой команды**:** КОП ; операнды выбираются по умолчению
src (source) источник **DS:SI** "откуда" dst (destination) приёмник **ES:DI** "куда".

CLD (clear) установить флаг DF:=0 или **STD** (set) установить флаг DF:=1 с170

MOVSB | W	Пересылка значения элемента строки из src по адресу строки dst	с176
CMPSB | W	Сравнение значения элемента строки src со значением элемента строки dst Установка флагов условий по результату операции	с169
SCASB | W	Сравнение содержимого srt=AL|AX со значением элемента строки dst Установка флагов условий по результату операции	с175
LODSB | W	Пересылка значения элемента строки src в регистр AL|AX	с177
STOSB | W	Пересылка содержимого регистра AL|AX по адресу dst строки	с176
REP <строковая команда>	Выполнение строковой команды CX раз	с171
REPE <строковая команда>	Повторяй, пока равны, но не более CX раз. if CX=0 then goto L1; ZF:=1; **L:** CX:=CX–1; <строковая команда> if (ZF = 1)and(CX<>0) then goto L; L1**:**	с171
REPNE <строковая команда>	Повторяй, пока не равны, но не более CX раз. if CX=0 then goto L1; ZF:=0; **L:** CX:=CX–1; <строковая команда> if (ZF = 0)and(CX<>0) then goto **L**; L1**:**	с174

Все строковые команды имеют две модификации, которые отличаются только размером операнда байт или слово **(B | W).**
MOVSB | W и **CMPSB | W** являются командами формата память-память (m8 | m16).
Строковые команды **CMPSB|W** и **SCASB | W** устанавливают регистр флагов.
Команда префикс **REPE** имеет другое название синоним **REPZ**, аналогично **REPNE/REPNZ**

Вышеприведенный список наиболее употребительных команд может выполнить роль справочной таблицы.

Использованы следующие обозначения.

На примере команды MOV напомним введенные в гл.3 обозначения для операндов команд. Пробел отделяет мнемокод команды от его первого операнда; *запятая разделяет* первый и второй операнды. Добавим символ | (вертикальная черта), придав ему смысл или одно, или другое; запись вида **B** (byte) | **W** (word) указывает на возможные типы операндов: либо оба размером байт, либо оба размером слово.

r – любой регистр, кроме сегментного (а именно, AH, AL, BH, BL, CH, CL, DH, DL – **r8**;
AX, BX, CX, DX, SI, DI, BP, SP – **r16**).

SR – сегментный регистр (CS, SS, DS, ES).

Если для операнда указан тип только **W**, то **r16** вместо **r** служит для усиления этого факта.

i – непосредственный операнд (immediate), задаваемый в самой команде в виде константного выражения.

m (memory) – адресное выражение (например, X, X[bx], [BX], X+2[bx][si],…).

Для команды MOV выписаны четыре строки. Две первых и две следующих для выделения особенностей работы с сегментными регистрами. Первая и вторая строки для подчеркивания факта недопустимости одновременого указания двух операндов из памяти.
Использование **m16** вместо **m** аналогично r16 вместо r.
Если используется обозначение **m32**, то имеется ввиду адрес двойного слова памяти.

Список литературы.

1. Пильщиков В.Н. Программирование на языке Ассемблера IBM PC. - Диалог= МИФИ, 1994
2. Пильщиков В.Н. Упражнения по языку MASM (Учебное пособие - 4-ое издание - М, : МГУ, 2009. -40с.)
3. Баула В.Г. Введение в архитектуру ЭВМ и системы программирования. - М, : МГУ, 2007, 220с

Кроме того, материалы на интернет-сайте для поддержки обязательных курсов по архитектуре ЭВМ и системам программирования для студентов и слушателей факультета ВМК и его филиалов,. [PDF] http://arch.cs.msu.su

Оглавление.