А. В. Карпов

СОЗДАНИЕ И ВЫПОЛНЕНИЕ ПРОСТЫХ ПРОГРАММ НА АССЕМБЛЕРЕ МП i8086 (часть I)

Методические указания к лабораторной работе

Волгоград 2006

УДК 681.325

Создание и выполнение простых программ на Ассемблере МП i8086 (часть I): Методические указания к лабораторной работе/ Сост. А. В. Карпов. -- Волгоград, 2006.-- 17 с.

Приведено описание лабораторной работы “Создание и выполнение простых программ на Ассемблере МП i8086” по курсу “Вычислительные машины, системы и сети”, в которой изучаются директивы языка, элементы и структура простейших программ на Ассемблере МП i8086. Издание предназначено для студентов дневной, вечерней и заочной форм обучения специальности 2102.

Цель работы

Целью настоящей работы является изучение директив языка, элементов и структуры простейших программ на Ассемблере МП i8086.

1 Элементы и структура программы на языке Ассемблера

Формат строки

Строки исходного кода на языке Ассемблера имеют следующий формат:

<метка> <команда/директива> <операнды> <;комментарий>

где <метка> представляет собой необязательное имя идентификатора,
<команда/директива> - это мнемоника команды или директивы,
<операнды> - содержат сочетание 0, 1 или 2 (иногда более) констант, ссылок на память или регистры, или текстовых строк, как это требуется в каждой конкретной команде или директиве,
<;комментарий> - необязательный комментарий.

В любом месте в строки качестве символа продолжения строки можно поместить символ обратной косой черты (\). Однако данный символ нельзя использовать для разбиения идентификаторов. Обратная косая черта означает: "считать следующую строку, и продолжить обработку с данной точки". При этом можно комментировать каждую строку. Например:

```
foo mystructure \     ; Начать заполнение структуры
<0,\                  ; Сначала - нулевое значение
1,\                   ; Затем - единица
2>\                   ; Структура завершается значением 2
```

Метки

Метки - это имена, использующиеся в программе для ссылки на числа и строки символов или ячейки памяти.
Метки позволяют присваивать имена переменным в памяти, значениям и адресам, где находятся конкретные инструкции. Например:

```
dseg SEGMENT 'DATA'
  FactorialValue DW ?
  Factorial DW ?
dseg ENDS
cseg SEGMENT 'CODE'
  ASSUME CS: cseg, DS: dseg
  FiveFactorial PROC
start:
    MOV AX, dseg
    MOV DS, AX
    MOV [FactorialValue], 1
```

```
    MOV [Factorial], 2
    MOV CX, 4
FiveFactorialLoop:
      MOV AX, [FactorialValue]
      MUL [Factorial]
      MOV [FactorialValue], AX
      INC [Factorial]
    LOOP FiveFactorialLoop
  FiveFactorial ENDP
END
```

Метки ***FactorialValue*** и ***Factorial*** эквивалентны адресам двух 16-битовых переменных. Они используются для последующей ссылки в программе на эти две переменные. Метка ***FiveFactorial*** - это имя подпрограммы (процедуры или функции), содержащей код. Она позволяет вызывать этот код в других частях программы. Метки ***start*** и ***FiveFactorialLoop*** есть адреса команд:

```
MOV AX, dseg
```

и

```
MOV AX, [FactorialValue]
```

соответственно.

Метки могут состоять из следующих символов:

A – Z a – z _ @ $? 0 – 9.

Цифры 0 - 9 не могут использоваться в качестве первых символов метки. Символы $ и ? имеют специальное значение, поэтому их не следует использовать в именах идентификаторах.

Каждая метка должна определяться только один раз, то есть метки должны быть уникальными. Как операнды метки могут использоваться любое число раз.

Метка может занимать всю строку, то есть на этой строке кроме метки может отсутствовать команда или директива. В этом случае значением метки является адрес команды или директивы на следующей строке программы. Например:

```
JMP DoAddition
...
DoAddition:
ADD AX, DX
```

следующей инструкцией, выполняемой после инструкции ***JMP***, которая выполняет переход на метку ***DoAddition***, является команда ***ADD AX, DX***. Предыдущий пример эквивалентен следующему:

```
JMP DoAddition
...
DoAddition: ADD AX, DX
```

Метка не может совпадать с любым из зарезервированных слов Ассемблера.

Определения меток в коде программы (и содержащиеся на отдельной строке, и после которых на той же строке следуют директивы или инструкции) должны заканчиваться двоеточием (:). Двоеточие просто завершает метку и не является ее частью. Например, в следующем фрагменте программы:

```
LoopTop:
mov al,[si]
inc si
and al,al
jz Done
jmp LoopTop
Done: ret
```

метки ***LoopTop*** и ***Done*** определены с двоеточиями, но в ссылках на эти метки двоеточия не указываются.

Другие метки в общем случае не должны завершаться двоеточием. В примере программы в начале данной лабораторной работы содержатся несколько меток без двоеточий.

Мнемоники команд и директивы

Основным полем в строке программы на Ассемблере является поле <команда/директива>. Это поле может содержать мнемонику команды или директиву.

Команды непосредственно выполняются процессором i8086. Ассемблер компилирует каждую мнемонику команды непосредственно в соответствующую команду на машинном языке.

В отличие от мнемоник команд, директивы не генерируют выполняемого кода. Вместо этого они управляют различными аспектами работы Ассемблера - от типа ассемблируемого кода до используемых сегментов и формата создаваемых файлов листингов и т. д.

Директивы Ассемблера

Директива SEGMENT используется для того, чтобы начать сегмент. Каждой директиве SEGMENT должна соответствовать директива ENDS, завершающая сегмент.

Полная форма директивы SEGMENT имеет следующий вид:

```
имя SEGMENT выравнивание комбинирование использование 'класс'
```

где поля "выравнивание", "комбинирование", "использование" и "класс" необязательны[1].

Поле "имя" задает имя сегмента. Имена сегментов являются метками, поэтому в своих исходных модулях они должны быть уникальны. При завершении сегмента то же имя должно использоваться в директиве ENDS. Пример:

[1] См. Приложение 1.

```
cseg SEGMENT
  ...
cseg ENDS
```

Сегменты могут быть вложенными, что означает, что можно определить сегмент до того, как закончится предыдущий сегмент, например:

```
DataSeg SEGMENT PARA PUBLIC 'DATA'
     ...
     DataSeg2 SEGMENT PARA PRIVATE 'FAR_DATA'
          ...
     DataSeg2 ENDS
     ...
DataSeg ENDS
```

Директива ENDS определяет конец сегмента. Например:

```
cseg ENDS
```

завершает сегмент с именем ***cseg***, который начинался по директиве SEGMENT. При использовании стандартных директив определения сегментов необходимо явным образом завершать каждый сегмент.

Директива ASSUME позволяет сообщить Ассемблеру, на какой сегмент или группу указывает данный сегментный регистр. Это не то же самое, что действительная загрузка сегментного регистра для ссылки на данный сегмент (это необходимо делать отдельно с помощью инструкции MOV). Назначение директивы ASSUME состоит в том, чтобы дать Ассемблеру возможность проверить допустимость ссылок на память и при обращении к памяти автоматически включить префиксы переопределения сегментов (если это требуется).

Директива ASSUME для регистра CS должна следовать перед любым кодом в каждом исходном модуле. Благодаря этому Ассемблер знает, в каком сегменте подразумевается размещение инструкций[1].

Пример директивы ASSUME:

```
DataSeg SEGMENT PARA PUBLIC 'DATA'
  ...
DataSeg ENDS
CodeSeg SEGMENT PARA PUBLIC 'CODE'
     ASSUME CS:CodeSeg,DS:DataSeg
     ...
CodeSeg ENDS
```

Директива END. Каждая программа должна содержать директиву END, отмечающую конец исходного кода программы. Все строки, которые следуют за директивой END, Ассемблером игнорируются. Если

[1] По мере необходимости в любой исходный модуль могут включаться другие директивы ASSUME (с указанием различных сегментных регистров). Подразумеваемый сегмент для любого сегментного регистра может быть изменен, когда это необходимо. В одной директиве ASSUME можно изменить любой или все сегментные регистры.

опустить директиву END, то генерируется ошибка, поэтому необходимо всегда указывать директиву END.

Например:

```
sseg SEGMENT 'STACK'
  DB 200h DUP (?)
sseg ENDS
dseg SEGMENT 'DATA'
  ...
dseg ENDS
cseg SEGMENT 'CODE'
  ASSUME CS: cseg, DS: dseg, SS: sseg
start:
  MOV AX, dseg
  MOV DS, AX
  MOV AH, 4ch
  INT 21h
cseg ENDS
END start
```

Это простейшая программа на Ассемблере. Она ничего не делает, просто немедленно возвращает управление DOS.

На одной строке с директивой END содержится метка ***start***. Кроме завершения программы, директива END может выполнять еще и вторую функцию, указывая, где должно начинаться выполнение при запуске программы. Иногда требуется начать выполнение программы в файле .EXE не с первой команды. Директива END предусматривает такие случаи. Пример:

```
sseg SEGMENT 'STACK'
  DB 200h DUP (?)
sseg ENDS
dseg SEGMENT 'DATA'
  ...
dseg ENDS
cseg SEGMENT 'CODE'
  ASSUME CS: cseg, DS: dseg, SS: sseg
Delay:
  MOV CX, 40
DelayLoop:
  LOOP DelayLoop
  RET
start:
  MOV AX, dseg
  MOV DS, AX
  CALL Delay
  MOV AH, 4ch
  INT 21h
```

```
cseg ENDS
END start
```

Выполнение здесь начинается не с первой команды исходного кода (***MOV CX, 40***) по метке ***Delay***, а с команды ***CALL Delay*** по метке ***start***, как определено в директиве END.

Если программа состоит только из одного модуля, то в директиве END всегда нужно определять адрес запуска программы. В программе, состоящей из нескольких модулей, определять адрес запуска программы следует только в директиве END модуля, содержащего команду, с которой должно начаться выполнение программы. В директивах END других модулей должно указываться только ключевое слово END.

Инициализированные данные

Директивы определения данных DB, DW, DD, DF, DP, DQ и DT позволяют определить переменные в памяти различного размера:

DB	1 байт
DW	2 байта (1 слово)
DD	4 байта (1 двойное слово)
DF, DP	6 байт
DQ	8 байт (1 четверное слово)
DT	10 байт

Например:

```
dseg SEGMENT 'DATA'
  ByteVar     DB    'Z'  ; 1 байт
  WordVar     DW    101b ; 2 байта (1 слово)
  DwordVar    DD    2BFh ; 4 байта (1 двойное слово)
  ...
dseg ENDS
cseg SEGMENT 'CODE'
  ...
  MOV AH, 2
  MOV DL, ByteVar
  ADD BX, WordVar
  ...
cseg ENDS
```

Здесь определяются и используются пять переменных памяти, и показывается, как некоторые из таких переменных можно использовать.

Инициализация строк символов

Символы представляют собой допустимые операнды директив определения данных, поэтому строку символов можно определить следующим образом:

```
String DB 'A', 'B', 'C', 'D'
```

В Ассемблере в этом случае предусмотрена также удобная сокращенная форма:

```
String DB 'ABCD'
```

Если необходимо завершить строку символами возврата каретки и перевода строки, их также нужно включить в строку. В следующем примере определяется текстовая строка, за которой следует символ возврата каретки, символ перевода строки:

```
HelloString DB 'Привет!',0dh,0ah,'$'
```

Например, программа:

```
dseg SEGMENT 'DATA'
  String1 DB 'Line1','$'
  String2 DB 'Line2','$'
  String3 DB 'Line3','$'
dseg ENDS
cseg SEGMENT 'CODE'
  ASSUME CS: cseg, DS: dseg
  ProgramStart:
  MOV AX, dseg
  MOV DS, AX
  MOV AH,9                    ; функция DOS печати строки
  MOV DX, OFFSET String1
  INT 21h                     ; вызвать DOS для печати строки
                              ; String1
  MOV DX, OFFSET String2
  INT 21h                     ; вызвать DOS для печати строки
                              ; String2
  MOV DX, OFFSET String3
  INT 21h                     ; вызвать DOS для печати строки
                              ; String3
  MOV AH, 4ch                 ; функция DOS завершения программы
  INT 21h
cseg ENDS
END ProgramStart
```

печатает следующее:

Line1Line2Line3

Однако если в конце каждой строки добавить пару символов "возврат каретки/перевод строки":

```
String1 DB 'Line1',0dh,0ah,'$'
String2 DB 'Line2',0dh,0ah,'$'
String3 DB 'Line3',0dh,0ah,'$'
```

то вывод будет выглядеть следующим образом:

Line1
Line2
Line3

2 Порядок выполнения работы

1. Введите в микроЭВМ с помощью редактора EDIT программу №1:

```
; Создаем сегмент данных
dseg SEGMENT 'DATA'
  msg DB "Привет!", 13, 10, "$"; Строка, которую программа
                                ; выведет на экран
dseg ENDS
; Создаем сегмент кода
cseg SEGMENT 'CODE'
; Директива ASSUME сообщает ассемблеру, как будут использоваться
; сегментные регистры. Эта директива не выполняет загрузку
; сегментных регистров, она нужна ассемблеру только для правильного
; вычисления смещений
  ASSUME CS:cseg, DS:dseg
; При запуске программы управление будет передано на оператор с меткой
; start. Первое, что должна сделать программа – правильно загрузить
; сегментные регистры. Регистр CS загружается операционной системой
; при запуске программы, поэтому его загружать не надо. Следующие
; два оператора инициализируют сегментный регистр данных DS.
start:
  MOV AX, dseg
  MOV DS, AX
; Выводим сообщение msg из сегмента данных
  MOV AH, 9h                    ; функция DOS вывода строки на
                                ; экран
  MOV DX, OFFSET msg
  INT 21h                       ; выводим строку на экран
  MOV AH, 4Ch                   ; функция DOS завершения
                                ; программы
  INT 21h                       ; завершаем программу
cseg ENDS
END start
```

Сохраните на диске набранный текст программы с именем
hello.asm

2. Создайте исполняемый файл программы №1 типа .EXE с помощью компилятора tasm:

tasm hello.asm

компоновщика tlink

tlink hello.obj

Примечания. А. Если файл с текстом программы **hello.asm** не содержит ошибок, то в результате создается файл **hello.obj**

В. Если файл с текстом программы **hello.asm** содержит ошибки, на экран выводятся сообщения, указывающие строки, где содержатся ошибки. При получении сообщений об ошибках необходимо проверить исходный текст программы и убедиться, что он выглядит точно так, как в примере, а затем снова ассемблировать программу.

С. **Tlink.exe** создаст файл **hello.exe**, который и нужно запустить на выполнение.

3. Запустите программу №1 на выполнение.

4. Напишите программу №2 выполняющую те же действия что и программа №1, но имеющую тип .COM.

5. Отладьте программу №2.

6. Создайте исполняемый файл программы №2 типа .COM с помощью компилятора tasm, компоновщика tlink, используя параметр /t.

tlink /t <имя Вашего файла>

7. Запустите программу №2 на выполнение.

8. Введите в микроЭВМ программу №3, которая меняет порядок символов во введенной строке на обратный.

Программа №3

```
dseg SEGMENT 'DATA'
  Max_String_Lenght EQU 1000
  StringToReverse DB Max_String_Lenght DUP (?)
  ReverseString DB Max_String_Lenght DUP (?)
dseg ENDS
sseg SEGMENT 'STACK'
  DB 64 DUP (?)
sseg ENDS
cseg SEGMENT 'CODE'
  ASSUME CS:cseg, DS:dseg, SS:sseg
start:
  MOV AX, dseg
  MOV DS, AX
; Инициализируем сегментный регистр стека и указатель стека. Эта
; операция должна выполняться в состоянии процессора с запрещенными
; прерываниями, иначе если регистр SS будет содержать уже новое
; значение, а SP - старое и если в этот момент произойдет прерывание,
; адрес возврата и значение регистра флагов будут записаны в не
; предназначенную для этого область.
CLI
  MOV AX, sseg
  MOV SS, AX
  MOV SP, OFFSET sseg
  STI
  MOV AH, 3fh                  ; функция DOS чтения ввода
  MOV BX, 0                    ; описатель стандартного ввода
                               ; (клавиатура)
  MOV CX, Max_String_Lenght    ; считать до максимального числа
                               ; символов
  MOV DX, OFFSET StringToReverse
  INT 21h                      ; получить строку
  AND AX, AX                   ; были считаны символы?
  JZ Done                      ; нет, конец
```

```
  MOV CX, AX                    ; поместить длину строки в
                                ; регистр CX
  PUSH CX                       ; сохранить в стеке длину строки
  MOV BX, OFFSET StringToReverse
  MOV SI, OFFSET ReverseString
  ADD SI, CX
  DEC SI                        ; указывает на конец буфера строки
ReverseLoop:
  MOV AL, [BX]                  ; получить следующий символ
  MOV [SI], AL                  ; сохранить символы в обратном
                                ; порядке
  INC BX                        ; указатель на следующий символ
  DEC SI                        ; указатель на предыдущую ячейку
  LOOP ReverseLoop              ; переместить следующий символ,
                                ; если он имеется
  POP CX                        ; извлечь длину строки
  MOV AH, 40h                   ; функция записи DOS
  MOV BX, 1                     ; описатель стандартного вывода
                                ; (экран)
  MOV DX, OFFSET ReverseString  ; напечатать строку
  INT 21h
Done:
  MOV AH, 4Ch
  INT 21h
cseg ENDS
END start
```

9. Создайте исполняемый файл программы №3 типа .EXE.
10. Запустите программу №3 на выполнение.
11. Напишите программу №4 выполняющую те же действия что и программа №3, но имеющую тип .COM.
12. Отладьте программу №4.
13. Создайте исполняемый файл программы №4 типа .COM.
14. Запустите программу №4 на выполнение.

3 Содержание отчета

Отчет о лабораторной работе должен содержать следующие сведения: 1) цель работы; 2) текст программ.

4 Контрольные вопросы

1. Перечислите известные Вам директивы Ассемблера МП i8086.
2. Перечислите известные Вам элементы и формат строки программы на языке Ассемблер МП i8086.
3. Перечислите известные Вам функции прерывания INT 21h.

Приложения

Приложение 1. ОПИСАНИЕ ПОЛЕЙ ДИРЕКТИВЫ SEGMENT

Поле "выравнивание" задает границу памяти, с которой должен начинаться сегмент:

BYTE - для большинства компактных программ подходит выравнивание на границу байта;

WORD - в 16-разрядных микроЭВМ более предпочтительно выравнивание на границу слова, так как 16-разрядные МП более эффективно работают с данными, выровненными на границу слова;

DWORD - По тем же причинам в 32-разрядных микроЭВМ предпочтительнее выравнивание на границу двойного слова;

PAGE - используется адрес следующей страницы (выравнивается на границу, кратную 256 байтам);

PARA - выравнивание на границу параграфа необходимо для сегментов, которые будут полностью занимать объем 64К.

Поле "комбинирование" управляет тем способом, с помощью которого сегменты с теми же именами в других модулях будут сочетаться с данным модулем при их компоновке. Это поле может принимать следующие значения:

AT - начало сегмента будет размещаться по указанному адресу в памяти. Реальный код не генерируется, вместо этого сегменты с типом комбинирования (сочетания) AT используются, как трафарет для доступа к областям памяти, таким, как сегмент данных ПЗУ BIOS (базовой системы ввода-вывода) и дисплейная память.

COMMON - задает, что начало данного сегмента и начало других сегментов с тем же именем должно выравниваться таким образом, что сегменты будут перекрывать друг друга. Общий размер сегмента представляет собой размер наибольшего сегмента с данным именем. Один из способов использования типа комбинирования COMMON состоит во включении файла, в котором определяется сегмент COMMON, в каждом модуле, где имеется ссылка на данный сегмент, благодаря чему все модули могут совместно использовать один и тот же сегмент.

PUBLIC - указывает компоновщику, что нужно выполнить конкатенацию данного сегмента с другими сегментами с тем же именем, благодаря чему сегменты составляют один большой сегмент. Общий размер сегмента представляет собой сумму размеров всех сегментов с этим именем. Общий размер сегментов PUBLIC не должен превышать 64К. Тип PUBLIC используется, когда несколько модулей совместно используют один и тот же сегмент, но каждый модуль определяет свои собственные переменные. Переменные в сегментах PUBLIC с помощью директивы GLOBAL часто используются модулями совместно.

MEMORY - это тоже самое, что тип PUBLIC.

STACK - указывает компоновщику, что нужно выполнить конкатенацию всех сегментов с данным именем в один сегмент, и построить файл типа .EXE, чтобы при выполнении программы регистры SS:SP указывали на конец этого сегмента. Это специальный тип комбинирования, который должен использоваться только для стека.

VIRTUAL - определяет сегмент специального вида, который будет интерпретироваться, как общая область, и присоединяться во время компоновки к другому сегменту. Предполагается, что сегмент VIRTUAL присоединяется к сегменту, в который он вложен. От этого сегмента сегмент VIRTUAL наследует атрибуты. Директива ASSUME рассматривает сегмент VIRTUAL, как часть его родительского сегмента. Компоновщик рассматривает сегменты VIRTUAL, как общую область, которая будет использоваться разными модулями. Это позволяет совместно использовать в разных модулях статические данные, содержащиеся во включаемых файлах.

PRIVATE - указывает компоновщику, что данный сегмент не следует сочетать ни с какими другими сегментами. Это позволяет определять сегменты, которые являются локальными по отношению к данному модулю (при этом не нужно беспокоиться о возможных конфликтах с сегментами с теми же именами, использующимися в других модулях. Если тип комбинирования не указывается, то для сегментов по умолчанию задается тип комбинирования PRIVATE.

Поле "использование" предназначено только для работы с МП i80386.

Поле "класс" используется для управления порядком, в котором компоновщик размещает сегменты. Все сегменты данного класса размещаются в непрерывном блоке памяти, независимо от их порядка в исходном коде.

Тип класса (если он присутствует) должен заключаться в кавычки. Тип класса должен быть уникален в данном исходном модуле, то есть никакая используемая в данном модуле метка не может тоже имя, что и имя класса, указанное в этом модуле.

Приложение 2. ФУНКЦИИ ПРЕРЫВАНИЯ INT 21H (DOS)

Вызов функции

1. В регистр AH записать номер функции.

2. Другие регистры - в соответствии с конкретной функцией.

3. Вызвать прерывание INT 21h. Прерывание INT 21h заставляет систему выполнять функцию DOS, номер которой записан в регистре AH.

Пример: Вызов функции «Получить время»

```
MOV AH, 2Ch       ; 2Ch - номер функции «Получить время»
INT 21h           ; Вызов функции
```

Функции

ОТОБРАЗИТЬ СТРОКУ (ФУНКЦИЯ 09H)

Вызов: AH=09H

DS:DX - указатель на отображаемую строку

Возвращает: —

Функция посылает на стандартный вывод строку. $ - признак конца строки. $ не отображается.

DX должен содержать смещение строки (сегментный адрес в DS).

ЧИТАТЬ ВВОД (ФУНКЦИЯ 3FH)

Вызов: AH=3Fh

BX – описатель стандартного ввода

CX – количество байт для чтения

DS:DX – указатель на буфер

Возвращает: CF – установлен:

AX=5 – нет доступа (Стандартный ввод не открыт для чтения)

AX=6 – несуществующий ввод (Стандартный ввод не открыт или не существует)

CF – сброшен:

AX – прочитано байт

Функция 3Fh читает из файла или устройства, ассоциированного с указанным описателем стандартного ввода. BX должен содержать описатель стандартного ввода. CX – количество байт, которое нужно прочитать. DX содержит смещение буфера (сегментный адрес в DS).

Количество байт, указанное в CX, необязательно переносится в буфер. Если эта функция используется для чтения с клавиатуры, то данные читаются до первого Enter.

В случае ошибки устанавливается CF, а AX возвращает код ошибки (см. выше).

ЗАПИСАТЬ В ВЫВОД (ФУНКЦИЯ 40H)

Вызов: AH=40h

BX – описатель стандартного вывода

CX – количество байт для записи

DS:DX – указатель на буфер

Возвращает: CF – установлен:

AX=5 – нет доступа (Стандартный ввод не открыт для чтения)

AX=6 – несуществующий ввод (Стандартный ввод не открыт или не существует)

CF – сброшен:

AX – записано байт

Функция 40h записывает в файл или устройство, ассоциированное с указанным описателем стандартного вывода. BX должен содержать описатель стандартного вывода. В CX нужно загрузить количество записываемых байт. DX содержит смещение этих данных (сегментный адрес в DS).

Если сбросить CX в ноль, то файл будет уменьшен до текущей позиции указателя файла. MS-DOS не выполнит операцию записи, если стандартный вывод открыт только для чтения.

Если нет ошибки, то AX возвращает количество записанных байт. Необходимо проверять содержимое AX после операции записи. Если оно меньше величины, загруженной в CX, то произошла ошибка, даже если сброшен флаг CF. Если AX содержит 0, а адресатом был файл на диске, то это признак переполнения диска.

В случае ошибки устанавливается CF, а AX возвращает код ошибки (см. выше).

ЗАВЕРШИТЬ ПРОГРАММУ (ФУНКЦИЯ 4CH)

Вызов: AH=4Ch

AL - код возврата

Возвращает: --

Функция 4CH завершает программу и передает управление MS-DOS. Регистр AL содержит код возврата, который может быть получен родительской программой с помощью функции 4Dh.

При вызове функции закрывается все открытые стандартные выводы.

Составитель Александр Викторович Карпов

СОЗДАНИЕ И ВЫПОЛНЕНИЕ ПРОСТЫХ ПРОГРАММ НА АССЕМБЛЕРЕ МП i8086 (часть I)

Методические указания к лабораторной работе