Федеральное агентство по образованию Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности
электронно-вычислительных систем (КИБЭВС)

В.Н. Кирнос

Основы программирования на языке Ассемблера

Лабораторный практикум

Томск – 2007

В.Н. Кирнос

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ АССЕМБЛЕРА. Лабораторный практикум: Учебно-методическое пособие. – Томск: ТУСУР, 2007, – 106 с.

В пособии кратко рассматривается основы программирования на Ассемблере для процессора 8086. Во введении даются краткие теоретические сведения по организации компьютера. Далее последовательно рассматривается реализация основных типов алгоритмических структур (линейная, разветвленная, циклическая), работа с массивами и функциями на данном языке. Затем особое внимание уделяется организации ввода и вывода на языке ассемблера.

Дополнительно рассматривается работа с файлами и с вещественными числами на Ассемблере.

В конце каждого параграфа даются практические задания для самостоятельного выполнения (их следует использовать при выполнении лабораторных работ).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
Оперативная память. 4
Регистры 5
Регистры общего назначения 6
Сегментные регистры 8
Указатель команд 9
Представление команд 10
§1. ВЫЧИСЛЕНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ 11
1.1 Понятие об арифметических операциях 11
1.2 Оформление программы на Ассемблере 12
1.3 Исполнение программы 13
Практические задания к § 1 16
§ 2. ПЕРЕХОДЫ И ВЕТВЛЕНИЯ НА АССЕМБЛЕРЕ 17
Практические задания к §2 24
§ 3 ЦИКЛЫ СО СЧЕТЧИКОМ 25
Практические задания к § 3 29
§4. МАССИВЫ 29
4.1 Одномерные массивы 29
4.2 Двумерные массивы 31
Практические задания к §4 33
§5. ПРОЦЕДУРЫ 37
5.1 Понятие о процедуре 37
5.2 Передача параметров в процедуру 38
5.3 Рекурсивные процедуры 41
Практические задания к §5 42
§ 6. ВЫВОД ЧИСЕЛ НА ЭКРАН 45
Практические задания к § 6 47
§ 7. ВВОД И ВЫВОД ЧИСЕЛ В ПРОГРАММЕ НА АССЕМБЛЕРЕ .. 48
Практические задания к §7 54
§ 8. РАБОТА С ФАЙЛАМИ НА АССЕМБЛЕРЕ 55
Практические задания к § 8 62
§ 9. РАБОТА С ВЕЩЕСТВЕННЫМИ ЧИСЛАМИ 70
Практические задания к § 9 90
ЛИТЕРАТУРА 93
ПРИЛОЖЕНИЯ 94
Приложение 0. Двоичная и шестнадцатиричная системы счисления. 94
П 0.1 Двоичная система счисления 94
П 0.2 Шестнадцатиричная система счисления 95
П0.3 Представление отрицательных чисел в двоичной и 16-ричной системах 97
Приложение 1. Far Manager 98
Приложение 2. Отладчик DEBUG 103
Приложение 3. Кодировка символов 105

ВВЕДЕНИЕ

Данное пособие представляет собой задания для лабораторных работ при изучении программирования на языке Ассемблера с краткими теоретическими пояснениями к ним и рекомендациями по их выполнению.

Для выполнения данных работ желательно уже иметь представление об основах организации компьютера.

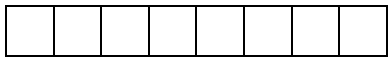
В пособии рассматривается язык Ассемблера для процессора фирмы Intel 8086.

Напомним краткие сведения по организации компьютера.

Оперативная память.

Оперативная память персонального компьютера (ПК) делится на ячейки размеров в 8 разрядов. Их принято называть *байтами* (byte). Разряды байта нумеруются справа налево от 0 до 7:

Байт



7 6 5 4 3 2 1 0

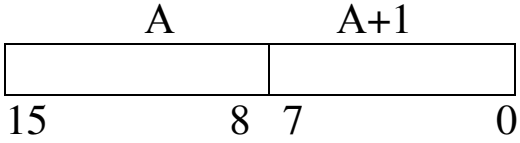
При этом правые разряды (с меньшими номерами) называются младшими, а левые разряды – старшими. В каждом разряде может быть записана величина 1 или 0, такую величину принято называть *бит* (bit). Таким образом, содержимое любого байта – это набор из 8 битов, из 8 нулей и единиц.

Ради краткости договоримся в дальнейшем записывать содержимое ячеек не в двоичной системе, а в 16-ричной, указывая в конце букву h (hexadecimal - шестнадцатиричный), чтобы отличать такие числа от десятичных. Например, если содержимым байта является (в двоичной системе) 00010011, то будем записывать его как 13h (десятичное 19).

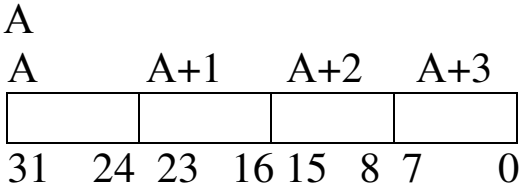
Байты нумеруются начиная с 0, порядковый номер байта называется его адресом. Объем оперативной памяти ПК – 2^{20} байтов (1 Мб), поэтому для ссылок на байты памяти нужны 20-разрядные адреса – от 00000h до FFFFFh.

Байт – это наименьшая адресуемая ячейка памяти. Но в ПК имеются и более крупные адресуемые ячейки – слова и двойные слова.

Слово (word) – это два соседних байта. Размер слова – 16 разрядов. Они нумеруются, если рассматривать слово как единое целое, справа налево от 0 до 15. Адресом слова считается по определению адрес первого его байта (с меньшим адресом).



Двойное слово (double word) – это четыре соседних байта или, что то же самое, два соседних слова. Размер двойного слова – 32 разряда, они нумеруются справа налево от 0 до 31. Адрес двойного слова – адрес первого из его байтов (с наименьшим адресом).



ПК может работать как с байтами, так и со словами и двойными словами, т.е. в ПК имеются команды, в которых ячейки этих размеров рассматриваются как единое целое. В то же время слова и двойные слова можно обрабатывать и побайтно.

Отметим, что адрес ячейки еще не однозначно определяет ячейку, поскольку с этого адреса может начинаться ячейка размеров в байт, ячейка размером в слово и в двойное слово. Как указывать размер ячейки – рассмотрим позднее.

Зачем введены ячейки разного размера? Для хранения данных разного типа. Например, байты используют для хранения небольших целых чисел (типа счетчиков) и символов. В виде же слов представляются обычные целые числа и адреса. Двойные слова используются для хранения больших чисел.

Регистры

Помимо ячеек оперативной памяти для хранения данных (правда, кратковременного) можно использовать и *регистры* – ячейки, расположенные в центральном процессоре и доступные

из машинных команд. Доступ к регистрам осуществляется намного быстрее, чем к ячейкам памяти, поэтому использование регистров заметно уменьшает время выполнения программ.

Все регистры имеют размер слова (16 разрядов), за каждым из них закреплено определенное имя (AX, SP и др.). По назначению и способу использования регистры можно разбить на следующие группы:

- регистры общего назначения (AX, BX, CX, DX, SI, DI, BP, SP);
- сегментные регистры (CS, DS, SS, ES);
- указатель команд (IP);
- регистр флагов (Flags).

Регистры общего назначения

К этой группе относятся следующие 8 регистров:

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

SI	
DI	
BP	
SP	

Приведем расшифровку этих названий:

AX accumulator, аккумулятор
BX base, база (РЕГИСТРЫ ДАННЫХ)
CX counter, счетчик
DX data, данные

(буква X – от слова eXtended, расширенный: в процессоре 8080 были байтовые регистры A,B,C,D, но затем их расширили до размера слова)

SI source index, индекс источника
DI destination index, индекс приемника
BP base pointer, указатель базы (РЕГИСТРЫ УКАЗАТЕЛЕЙ)
SP stack pointer, указатель стека

Особенностью всех этих регистров является то, что их можно использовать в любых арифметических, логических и т.п. машинных операциях. Например, можно сложить число из регистра DI с числом из регистра SI или вычесть из содержимого регистра BP содержимое регистра CX.

В то же время каждый из этих регистров имеет определенную специализацию: некоторые команды требуют, чтобы их операнд или операнды находились в определенных регистрах. Например, команда деления требует, чтобы первый операнд (делимое) находился в регистре AX или в регистрах AX и DX (в зависимости от размера операнда), а команды управления циклом используют регистр CX в качестве счетчика цикла.

В ПК используется так называемая *модификация адресов*. Если в команде операнд берется из памяти, тогда сослаться на него можно, указав некоторый адрес и некоторый регистр. В этом случае команда будет работать с так называемым исполнительным адресом, который вычисляется как сумма адреса, указанного в команде, и текущего значения указанного регистра. Именно из ячейки с таким адресом команда и будет брать свой операнд. Выгода от такого способа задания операнда заключается в том, что, меняя значение регистра, можно заставить одну и ту же команду работать с разными ячейками памяти, что, например, полезно при обработке массивов. Замена адреса, указанного в команде, на исполнительный адрес называется *модификацией* адреса, а используемый для этого регистр называется *модификатором*. В ПК модификаторами могут быть регистры BX, BP, SI, DI.

Регистр SP используется при работе со стеком. *Стек – это хранилище информации, функционирующее по правилу: первым из стека всегда считывается элемент, записанный в стек последним*. Стек полезен, например, при реализации процедур. В ПК имеются команды, поддерживающие работу со стеком. В этих командах предполагается, что регистр SP указывает на ячейку стека, в которой находится элемент, записанный в стек последним. Более подробно работу со стеком рассмотрим позднее.

Замечание. Регистр SP использовать в арифметических операциях (см. § 1) не следует, именно потому, что он используется для работы стека и изменение SP может привести к непредсказуемым последствиям в работе программы.

Регистры AX, BX, CX, DX позволяют осуществлять доступ к их старшей и младшей половине: каждый из регистров состоит из двух байтовых регистров. Обозначают их буквами H (High – выше, старший) и L (Low – ниже, младший) и первой буквой из названия регистра: AH и AL – в AX и т.д. В итоге, например, в AX можно записать слово (из 16 битов), а затем считать левую половину (байт из AH).

Сегментные регистры

Вторую группу регистров образуют следующие 4 регистра:

CS		SS	
DS		ES	

Их названия расшифровываются так:

CS code segment, сегмент команд
DS data segment, сегмент данных
SS stack segment, сегмент стека
ES extra segment, дополнительный сегмент.

Ни в каких арифметических, логических и т.п. операциях эти регистры не могут участвовать. Эти регистры используются для сегментирования адресов, которое является разновидностью модификации адресов. Более подробно сегментирование адресов рассмотрим позднее, а пока отметим следующее.

В ПК размеры сегментов памяти не должны превышать 64 Кб (2^{16} = 65536), поэтому смещения здесь – это 16-разрядные адреса. Поскольку сегментирование адресов применяется в отношении всех команд, операнды которых берутся из памяти, то в командах явно указываются только 16-разрядные адреса (смещения), а не «длинные» 20-разрядные адреса. Кроме того, в ПК принят ряд соглашений, которые позволяют во многих командах не указывать явно сегментные регистры. В связи с этим во многих программах можно ни разу не встретить 20-разрядные адреса и сегментные регистры, и создается впечатление, что ПК – это ЭВМ с 16-разрядными адресами. Одновременно договоримся термином «адрес» обозначать 16-разрядные адреса, и лишь позже вспомним, что адреса все-таки 20-разрядные. 16-разрядные

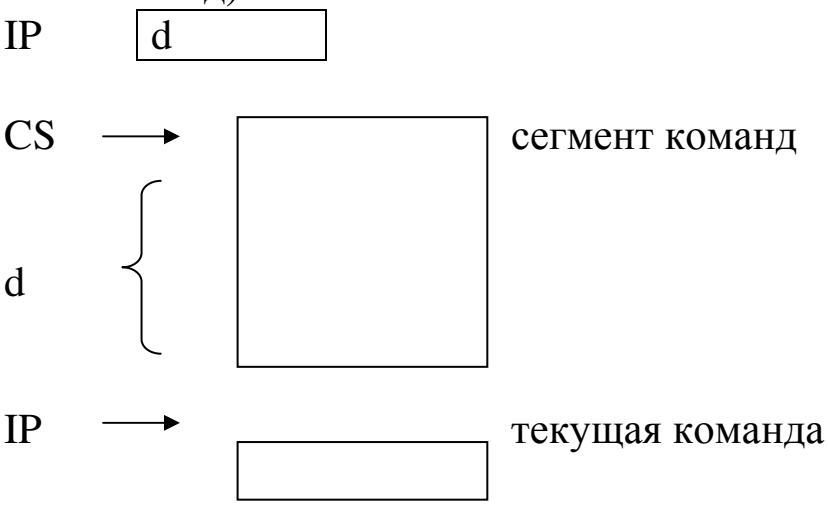
адреса меняются от 0000h до FFFFh, а 20-разрядные (абсолютные) адреса – от 00000h до FFFFFh.

Что касается принятых соглашений, то суть их в следующем:

- в регистре CS - начальный адрес сегмента команд (той области памяти, где находятся команды программы),
- регистр DS – указывает на начало сегмента данных, в котором размещаются данные программы,
- регистр SS – указывает на начало области памяти, отведенной под стек.

Указатель команд

Еще один регистр ПК – это регистр IP (instruction pointer, указатель команд):



В этом регистре всегда находится адрес команды, которая должна быть выполнена следующей. Более точно, в IP находится адрес этой команды, отсчитанный от начала сегмента команд, на начало которого указывает регистр CS. Поэтому абсолютный адрес этой команды определяется парой регистров CS и IP. Изменение любого из этих регистров есть ни что иное, как переход. Поэтому содержимое регистра IP (как и CS) можно менять только командами перехода.

Представление команд

Машинные команды ПК занимают от 1 до 6 байт. Код операции (КОП) занимает один или два первых байта команды. В ПК столь много различных операций, что для них не хватает 256 различных кодов, которые можно представить в одном байте. Поэтому некоторые операции объединяются в группу и им дается один и тот же КОП, во втором же байте этот код уточняется. Кроме того, во втором байте указываются типы операндов и способы их адресации. В остальных байтах команды указываются ее операнды.

Однако программировать в машинных кодах сложно. Поэтому придумали средство, упрощающее составление машинных программ. Это язык *ассемблера*, представляющий собой фактически символьную форму записи машинного языка: в нем вместо цифровых кодов операций выписывают привычные нам знаки операций или их словесные названия. Вместо адресов – применяют имена, а константы записывают в десятичной системе. Программу, записанную в таком виде, вводят в ПК и подают на вход транслятору, называемому ассемблером, который переводит ее на машинный язык, и далее полученную машинную программу выполняют.

Отметим, что для любой ЭВМ можно придумать различные языки Ассемблера. Мы будем рассматривать язык, который создан фирмой Microsoft и называется макроассемблером (MASM). У него имеется несколько версий. Мы будем рассматривать версию 6.12. Заметим, что для работы с транслятором MASM этой версии достаточно иметь пять файлов: masm.exe, masm.grp, ml.exe, ml.err, link.exe. Их следует просто скопировать в ту папку, где будете размещать свои файлы с программами на ассемблере.

§1. ВЫЧИСЛЕНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

1.1 Понятие об арифметических операциях

На Ассемблере существуют следующие основные команды для арифметических вычислений:

Команда сложения

ADD *приемник, источник*

Выполняется она так*: приемник* складывается с *источником* и результат помещается в *приемник*

Имеются также команды сложения:

INC *источник* –увеличение *источника* на 1

DEC *источник* – уменьшение *источника* на 1

Команда вычитания:

SUB *приемник, источник*

Выполняется так: из *приемника* вычитается *источник* и результат помещается в *приемник*

Команда умножения:

MUL *источник*

Действует она так:

а) при умножении *байтов*:

Содержимое регистра AL умножается на *источник* и *результат* помещается в AX (AH – старший байт результата, AL – младший байт результата)

б) при умножении *слов*:

Содержимое регистра AX умножается на *источник* и *результат* помещается в DX (старшее слово результата) и в AX (младшее слово)

Команда деления:

DIV *источник*

а) при делении на *байт* (в операнде *источнике)*

Делимое берется из AX и *частное* возвращается в регистре AL, а остаток – в регистре AH

б) при делении на *слово* (в операнде *источнике)*

Делимое берется из DX и AX и *частное* возвращается в регистре AX, а остаток – в DX

Мы пока не касаемся особенностей арифметических операций со знаковыми числами и не рассматриваем случаи переполнения при умножении и делении. Заметим также, что команды умножения и деления не позволяют использовать соответственно умножение и деление на непосредственное число, т.е. значение источника для умножения и деления следует предварительно загрузить в регистр или ячейку памяти.

1.2 Оформление программы на Ассемблере

При оформлении программы на Ассемблере следует отвести сегмент для команд (для собственно программы)

```
CSEG SEGMENT    ;заголовок сегмента программы
…
CSEG ENDS       ; конец сегмента программы
```

где CSEG – имя сегмента (оно может быть и иным).

Комментарии в программе на Ассемблере можно писать в той же строке, где и команда, но отделять их «точкой с запятой».

Внутри обязательна ссылка на регистр сегмента :

```
ASSUME CS: CSEG
```

которая говорит о том, что адрес начала сегмента программы находится в регистре CS.

Кроме того, требуется указать начальный адрес всей программы с 100-й (в 16-ричной системе счисления) ячейки:

```
ORG 100h
```

Еще нужна и точка входа в программу. Как правило, ее оформляют в виде метки

```
BEGIN:
```

Соответственно в завершение всей программы нужна строка

```
END BEGIN
```

Пример 1-1. *Составить программу вычисления выражения (1 + xy)/ (x – y) для заданных x и y (задавать x>y)*

```
; вычисление выражения (1+xy)/(x-y)
CSEG segment
ASSUME CS:CSEG
ORG 100H
```

```
BEGIN:
    MOV BX,10   ; задаем x
    MOV CX,5    ; задаем y
    MOV AX,BX   ; заносим в AX значение x
    MUL CX      ; вычисляем x*y
    ADD AX,1    ; вычисляем 1+x*y
     SUB BX,CX  ; вычисляем РАЗНОСТЬ x-y
    DIV BX      ; вычисляем (1+xy)/(x-y)
CSEG    ENDS
END BEGIN
```

1.3 Исполнение программы

Для исполнения программы на ассемблере необходимо пройти следующие этапы:

1) ввести команды программы в компьютер с помощью редактора
2) оттранслировать программу с помощью Ассемблера
3) преобразовать результат работы Ассемблера в исполняемый модуль с помощью загрузчика (провести компоновку)
4) выполнить (вызвать на исполнение полученный исполняемый модуль)

Ввод исходного текста программы

Запустим, например, оболочку **FAR** Manager (порядок работы с ним см. в Приложении 1), перейдем в каталог **MASM** (там имеются нужные нам служебные программы[1]), нажмем клавиши **Shift**-**F4**, зададим название для программы (например **PR1-1.ASM** – расширение **ASM** обязательно!) и введем текст этой программы. Сохраним ее, нажав клавишу **F2** и выйдем клавишей **F10**.

Трансляция, компоновка и выполнение

Проведем трансляцию программы с использованием **MASM**:
MASM PR1-1.ASM;

[1] Напомним, что для использования MASM версии 6.11 и старше достаточны (минимально) следующие файлы: ML.EXE, MASM.EXE, LINK.EXE

(точка с запятой нужна, чтобы транслятор не задавал уточняющих вопросов). В результате (если нет синтаксических ошибок) получим объектный файл **PR1-1.OBJ**

Проведем компоновку

LINK PR1-1.OBJ;

В результате получим исполняемый файл **PR1-1.EXE**. Исполнять его простым запуском мы не будем, хотя бы потому, что мы не организовали вывода результатов. Поэтому воспользуемся отладчиком **DEBUG** . В командной строке просто наберем

DEBUG PR1-1.EXE

Появится приглашающий символ отладчика "–" (дефис). Здесь для просмотра текста программы дадим команду отладчика **U** (основные команды отладчика **DEBUG** см. в Приложении 2)

–U

Получим текст программы (с «точки зрения» отладчика)[2]:

```
1A60:0100 BB0A00      MOV     BX,000A
1A60:0103 B90500      MOV     CX,0005
1A60:0106 8BC3        MOV     AX,BX
1A60:0108 F7E1        MUL     CX
1A60:010A 83C001      ADD     AX,+01
1A60:010D 2BD9        SUB     BX,CX
1A60:010F F7F3        DIV     BX
1A60:0111 5E          POP     SI
```

Обратим внимание, что команды программы размещаются в памяти с 100-го адреса (смотрим числа после двоеточия), поскольку мы указали `ORG 100h`. За последней командой программы идет некая команда с адресом 111. Т.е. выполнять в данном случае надо до 111-го адреса.

Запустим программу на выполнение командой отладчика **G**

G 111

В результате получим содержимое регистров по окончании работы программы. Напомним, что результат размещается в AX (целая часть результата) и в DX (остаток от деления):

---

[2] Команда U отладчика DEBUG выдает одновременно на экран только 12 строк. Чтобы просмотреть следующие 12 строк, следует снова дать команду U и т.д. Обратите внимание, что после конца программы будут еще какие-то команды – то, что случайно оказалось в данном фрагменте памяти.

```
AX=000A  BX=0005  CX=0005  DX=0001
```

(на значение остальных регистров пока не обращаем внимание).

Обращаем внимание, что результаты даются в 16-ричной системе. Т.е. имеем целую часть результата равной 16-ричному числу **A** (это 10 в 10-й системе) и остаток равным **1**. Нетрудно проверить результат в десятичной системе:

(1+10*5)/(10–5) = 10 и остаток 1

Попробуем изменить исходные значения. Например, зададим x=12h (т.е. 18 десятичное) и y=10h (т.е. 16 десятичное):

–A 100

```
MOV BX, 12
MOV CX, 10
```

Не забывайте в конце строки нажимать клавишу Enter и после ввода второй строки Enter нажать 2 раза (для выхода из режима правки текста).

(Напоминаем, что командой отладчика **A** можно изменять текст программы на Ассемблере с указанного адреса)

Изменим адрес старта программы (напомним, что он задается регистром IP):

-R IP

```
IP 0111
:100
```

И снова запустим программу на выполнение.

В результате получим:

-G 111

```
AX=0090  BX=0002  CX=0010  DX=0001
```

Итак, целая часть результата 90h (144 в десятичной), остаток 1h. Если проверить в 10-й системе, то получаем:

(1+18*16)/(18 –16) = 144 и остаток 1.

Иногда полезно сделать трассировку в процессе выполнения программы. Для этого достаточно задавать в **DEBUG** команду **T**. Каждое нажатие **T** – выполнение очередной команды программы. Трассировка может понадобится для поиска ошибок времени выполнения.

Практические задания к § 1

Пользуясь приведенными ниже формулами составить программу на Ассемблере для вычисления суммы (для заданного **n).**

Замечание. При выполнении некоторых заданий результат может оказаться отрицательным числом, поэтому *пока* следует ограничиться такими значениями **n**, при которых результат положительный.

1-1. $1+2+3+\ldots+n = n(n+1)/2$
1-2. $p+(p+1)+(p+2)+\ldots+(p+n) = (n+1)(2p+n)/2$
1-3. $1 + 3 + 5 + \ldots + (2n - 1) = n^2$
1-4. $2 + 4 + 6 + \ldots + 2n = n(n+1)$
1-5. $1^2 + 2^2 + 3^2 + \ldots + n^2 = n(n+1)(2n+1)/6$
1-6. $1^3 + 2^3 + 3^3 + \ldots + n^3 = n^2(n+1)^2/4$
1-7. $1^2 + 3^2 + 5^2 + \ldots + (2n - 1)^2 = n(4n^2 - 1)/3$
1-8. $1^3 + 3^3 + 5^3 + \ldots + (2n - 1)^3 = n^2(2n^2 - 1)$
1-9. $1^4 + 2^4 + 3^4 + \ldots + n^4 = n(n+1)(2n+1)(3n^2 + 3n - 1)/30$
1-10. $1 + 4 + 7 + \ldots + 3n - 2 = n(3n - 1)/2$
1-11. $-1 + 2 - 3 + \ldots + (-1)^n n = (-1)^n [(n+1)/2]$ (здесь [ ] – означает целую часть)
1-12. $-1^2 + 2^2 - 3^2 + \ldots + (-1)^n n^2 = (-1)^n (n(n+1)/2)$
1-13. $-1^3 + 2^3 - 3^3 + \ldots + (-1)^n n^3 = (1/8)*(1 - (-1)^n (1 - 6n^2 - 4n^3))$
1-14. $-1^4 + 2^4 - 3^4 + \ldots + (-1)^n n^4 = (-1)^n (n^4 + 2n^3 - n)/2$
1-15. $1^5 + 2^5 + 3^5 + \ldots + n^5 = (1/12)*n^2(n+1)^2(2n^2 + 2n - 1)$
1-16. $1^5 - 2^5 + 3^5 - \ldots + (-1)^{n-1} n^5 = (1/4)(1 + (-1)^n (5n^2 - 5n^4 - 2n^5 - 1))$
1-17. $1 + 3 + 5 + \ldots + (2n+1) = (n+1)^2$
1-18. $1 - 3 + 5 - 7 + \ldots + (-1)^n(2n+1) = (-1)^n (n+1)$
1-19. $1^2 + 3^2 + 5^2 + \ldots + (2n+1)^2 = (1/3)(n+1)(2n+1)(2n+3)$
1-20. $1^2 - 3^2 + 5^2 - \ldots + (-1)^n(2n+1)^2 = (-1)^n 2(n+1)^2 - (1 + (-1)^n)/2$
1-21. $1^3 + 3^3 + 5^3 + \ldots + (2n+1)^3 = (n+1)^2(2n^2 + 4n + 1)$
1-22. $1*2 + 2*3 + \ldots + n(n+1) = (1/3)n(n+1)(n+2)$
1-23. $1*2*3 + 2*3*4 + \ldots + n(n+1)(n+2) = (1/4)n(n+1)(n+2)(n+3)$
1-24. $1*4*7 + 2*5*8 + \ldots + n(n+3)(n+6) = (1/4)\, n\,(n+1)(n+6)(n+7)$
1-25. $1*5*9 + 2*6*10 + \ldots + n(n+4)(n+8) = (1/4)\, n\,(n+1)(n+8)(n+9)$
Провести проверки для различных значений **n.**

§ 2. Переходы и ветвления на ассемблере

В программах на Ассемблере, кроме сегмента команд можно разместить и *сегмент данных*, в котором размещают данные для работы программы.

Простейшим образом значения для переменных можно придать с помощью так называемого псевдооператора '='. Например, нужно задать значения переменных *x=13* и *y =72* . Тогда сегмент данных будет иметь вид:

```
DSEG segment
X=13
Y=72
DSEG ENDS
```

Для организации переходов в программе на Ассемблере, во-первых, в программе размещают метки в виде символьно-цифровых имен (первый символ – буква), во-вторых, размещают команды сравнения и перехода.

Команда сравнения имеет вид:

CMP op1, op2

Работает она так: просто сравнивает значения **op1** и **op2**, и в зависимости от результата работает следующая за ним команда перехода. *Команд перехода* имеется довольно много. Отметим только некоторые:

JA Label - переход на метку Label, если op1>op2
JB Label - переход на метку Label, если op1<op2
JE Label - переход на метку Label, если op1=op2
JNE Label - переход на метку Label, если op1<>op2
JBE Label - переход на метку Label, если op1<=op2
JAE Label - переход на метку Label, если op1>=op2

Кроме того, имеется и безусловный переход

JMP Label

Он выполняется без команды сравнения.

Замечание. Эти команды нетрудно запомнить, учитывая, что *больше*, *выше* по-английски **Above**, *ниже*, *меньше* – **Below,** *равно* **– Equal.**

Пример 2-1. *Даны два целых числа x и y. Найти из них большее и сохранить в регистре DX .*

Алгоритм здесь понятен – сравниваем эти числа и если первое больше второго, то его помещаем в регистр DX, иначе – второе помещаем в регистр DX.

Программа имеет вид:

```
; вычисление max(x,y)
CSEG segment
ASSUME CS:CSEG
ORG 100H
BEGIN:
        MOV AX,10     ; задаем x
        MOV BX,5      ; задаем y
        CMP AX,BX     ; сравнение x и y
        JA X          ; x>y  --> метку X
        MOV DX,BX     ; Нет. В DX поместить y
        JMP EN       ; на выход из программы
X:      MOV DX,AX    ; Да. В DX поместить x
EN:
        MOV AH,4Ch   ; выход d DOS
        INT 21h
CSEG    ENDS
END BEGIN
```

В начале исходного текста программы задали конкретные значения переменным *x=10* и *y=5* .

Здесь в завершение кода программы вставили так называемое прерывание `INT 21h` – для нормального выхода из программы (о прерываниях подробнее – в других параграфах). Этому прерыванию должно предшествовать задание функции 4Ch в старшей части регистра AX – регистре AH.

Обратите внимание на комментарии. В данной программе значение *x* помещаем в регистр AX, а значение *y* – в BX. Результат (наибольшее из них) помещается в конце работы программы в регистре DX.

Проведите трансляцию и компоновку данной программы, как и в §1. Посмотрите результат с помощью **DEBUG**, а также задайте иные значения для *x* и *y*.

Увидим, например, при просмотре в **DEBUG**, что команды перехода преобразовались в команды с указанием конкретного адреса программы:

-U

```
1A70:0100 B80A00          MOV     AX,000A
1A70:0103 BB0500          MOV     BX,0005
1A70:0106 3BC3            CMP     AX,BX
1A70:0108 7704            JA      010E
1A70:010A 8BD3            MOV     DX,BX
1A70:010C EB02            JMP     0110
1A70:010E 8BD0            MOV     DX,AX
1A70:0110 B44C            MOV     AH,4C
1A70:0112 CD21            INT     21
```

Запустим на выполнение командой

-G 112

и увидим:

```
AX=4C0A  BX=0005  CX=0114  DX=000A
```

Т.е. в DX действительно большее из этих чисел (Ah = 10 в десятичной системе счисления) .

Задайте иные значения для *x* и *y* (в исходном тексте программы), снова проведите трансляцию и компоновку и с помощью DEBUG просмотрите результаты

Но иногда требуется отвести под хранение данных ячейки памяти определенного типа. Определение таких данных можно сделать с помощью *псевдооператоров* вида:

<*имя*> DB <*выражение*> - переменная с указанным именем типа ‘байт’ (8 разрядов)

<*имя*> DW <*выражение*> - переменная с указанным именем типа ‘слово’ (16 разрядов)

<*имя*> DD <*выражение*> - переменная с указанным именем типа ‘двойное слово’ (32 разряда)

В качестве выражения может выступать и константа (числовая или символьная). Если сразу значение переменной не задает-

ся, то на месте выражения пишут знак ‘?’. Например, сегмент данных может иметь вид:

```
    DSEG  segment
    X DW 13
    Y DW 72
    Z DW ?
    DSEG  ENDS
```

В этом случае потребуется дополнительно загрузить адрес начала сегмента данных в регистр DS :

```
    MOV CX,DSEG
    MOV DS,CX
```

Пример 2-1а. *Даны два целых числа x и y типа ‘слово’. Найти из них большее и сохранить в регистре DX, а также разместить это большее число в переменной z.*

Текст программы имеет окончательно вид:

```
; вычисление max(x,y)
;сегсент данных
DSEG segment
X DW 13
Y DW 72
Z DW ?
DSEG ENDS
; сегмент команд
CSEG segment
ASSUME CS:CSEG, DS:DSEG
ORG 100H
BEGIN:
    MOV CX,DSEG; загрузим адрес начала сегмента
    MOV DS,CX  ; в регистр DS
    MOV AX,X   ; заносим x в AX
    MOV BX,Y   ; заносим y  в BX
    CMP AX,BX  ; сравнение x и y
    JA XM      ; x>y  переход на  метку XM
    MOV DX,BX   ; Нет. В DX поместить y
    JMP EN      ; на выход из программы
```

```
XM: MOV DX,AX    ; Да. В DX поместить x
EN: MOV Z,DX
    MOV AH,4CH
    INT 21h
CSEG    ENDS
END BEGIN
```

Если этот пример оттранслировать и скомпоновать, а затем открыть с помощью DEBUG, то получим:

-U

```
1A71:0100 B9701A        MOV     CX,1A70
1A71:0103 8ED9          MOV     DS,CX
1A71:0105 A10000        MOV     AX,[0000]
1A71:0108 8B1E0200      MOV     BX,[0002]
1A71:010C 3BC3          CMP     AX,BX
1A71:010E 7704          JA      0114
1A71:0110 8BD3          MOV     DX,BX
1A71:0112 EB02          JMP     0116
1A71:0114 8BD0          MOV     DX,AX
1A71:0116 89160400      MOV     [0004],DX
1A71:011A B44C          MOV     AH,4C
1A71:011C CD21          INT     21
```

Обратите внимание, что теперь значатся адреса ячеек памяти (в квадратных скобках), а не значения переменных.

Запустим на выполнение

-G 11C

и получим:

AX=4C0D BX=0048 CX=1A70 DX=0048 SP=0000 BP=0000 SI=0000 DI=0000

DS=1A70 ES=1A60 SS=1A70 CS=1A71 IP=011C NV UP EI NG NZ NA PE CY

1A71:011C CD21 INT 21

Т.е. видим, что для заданных значений переменных *x=13 (Dh)* и *y =72 (48h)* получили в регистре DX правильный результат.

Отметим, что работа с *отрицательными* (точнее говоря, *знаковыми*) числами требует особого подхода.

Во-первых, команды перехода при сравнении **CMP op1, op2** знаковых чисел имеют иной вид:

- вместо **JA Label** (переход на метку Label, если op1>op2) используется команда **JG Label (Great –** *больше* по-английски);
- вместо **JB Label** (переход на метку Label, если op1<op2) используется команда **JL Label (Less –** *меньше* по-английски).

Во-вторых, напомним, что отрицательные числа хранятся в памяти компьютера в дополнительном коде. Соответственно в регистре будет показываться дополнительный код отрицательного числа (в 16-ричном виде). Т.к. в регистрах помещаются 4-значные числа, то для получения прямого кода отрицательного числа следует из 10000_{16} вычесть число, находящееся в регистре. Тем самым и получим модуль отрицательного числа, находящегося в регистре.

В-третьих, команды умножения (**MUL**) и деления (**DIV**) заменяются соответственно на **IMUL** и **IDIV**.

Пример 2-2. *Даны целые (не обязательно положительные) числа A и B. Меньшее из них заменить суммой, большее - произведением этих чисел.*

```
DSEG SEGMENT
        A=2         ; задаем первое число
        B= –5       ; задаем второе число
DSEG ENDS
CSEG SEGMENT
ASSUME CS:CSEG, DS:DSEG
ORG 100H
BEGIN:
        MOV AX, DSEG
        MOV DS, AX
        MOV AX, A       ; в регистре AX – первое число
        MOV BX, B       ; в регистре BX – второе число
        CMP AX, BX
        JG ABOL
; если второе число, т.е. B (оно в BX) больше
        MOV DX,0        ; очищаем старшую часть 1-го сомножителя
```

```
                    ; (младшая часть находится в AX)
        IMUL BX     ; умножаем на второе число
        MOV BX, AX  ; переписываем результат в BX
                    ; (на место 2-го числа)
        MOV AX, A
        ADD AX, B   ; в AX будет сумма чисел A и B
        JMP EN
ABOL:
; если первое число, т.е. A (оно в AX) больше
        IMUL BX
        ADD BX, A
EN:
        MOV AH, 4CH ; выход в DOS
        INT 21H
CSEG ENDS
END BEGIN
```

После трансляции и компоновки получим EXE-файл, который откроем с помощью DEBUG и увидим, что последние строки имеют вид:

```
0BE3:0120 83C302      ADD    BX,+02
0BE3:0123 B44C        MOV    AH,4C
0BE3:0125 CD21        INT    21
```

Запустим на исполнение до команды выхода в DOS (т.к. она нам испортит значение регистра AX):

-G 123

и получим:

AX=FFF6 BX=FFFD

Т.е. на месте первого числа (в регистре AX), которое является большим, находится отрицательное число в дополнительном коде. Переведем в прямой код: $10000_{16} - FFF6_{16} = 10_{10}$, значит, в AX оказалось –10, что верно, т.к. 2*(–5)= –10. А на месте второго числа (в регистре BX), которое здесь является меньшим, имеем $10000_{16} - FFFD_{16} = 3_{10}$, что тоже верно, т.к. 2+ (–5) = –3.

Практические задания к §2

2-1. Определить, является ли данное целое число четным (в каком-либо регистре поместить число 1 – если четное и число 2 – если нечетное).

2-2. Дано натуральное число *n* (*n*>99). Определить число сотен в нем.

2-3. Дано натуральное число *n* (*n*<=100). Сколько цифр в числе *n*?

2-4. Даны целые числа K, L. Если числа не равны, то заменить каждое из них большим среди них, а если равны, то заменить числа нулями.

2-5. Даны целые *x,y,z*. Вычислить *max(x,y,z).*

2-6. Даны целые *x,y,z*. Вычислить *min(x,y,z).*

2-7. Даны целые *x,y,z*. Вычислить *max(x+y+z, xyz).*

2-8. Даны целые *x,y,z*. Вычислить *min(x+y+z/2, xyz)+1.*

2-9. Даны два целых числа *x, y*. Меньшее заменить полусуммой, а большее – удвоенным произведением.

2-10. Даны целые *x,y*. Вычислить *z* :

$$z = \begin{cases} x - y, & \textit{если} \quad x > y \\ y - x + 1, & \textit{иначе} \end{cases}$$

2-11. Даны целые *x,y*. Вычислить *z* :

$$z = \begin{cases} \max(x, y), & \textit{если} \quad x < 0 \\ \min(x, y), & \textit{иначе} \end{cases}$$

2-12. Даны целые *x,y*. Вывести номер координатной четверти плоскости, в которой находится точка с координатами *x* и *y* .

2-13. Даны четыре целых числа a1, a2, a3, a4, причем одно из них отлично от трех других, равных между собой. Указать порядковый номер этого числа.

2-14. По номеру заданного года (в виде 4-хзначного числа) указать номер его столетия. Учесть, например, что началом 21-го столетия является не 2000, а 2001 год.

2-15. Даны целые *x,y,z*. Вычислить (*max(x,y,z)*(min(x,y))+5.*

2-16. Даны целые *x,y,z*. Вычислить *min(x+y,y–z)*(max(x,y)*

2-17. Даны целые *x,y,z*. Вычислить (*max(x,y,z)–(min(x,y))*2.*

2-18. Даны целые *x,y,z*. Вычислить *max(x+y+z, xyz)*(min(x+y+z, xyz)*

2-19. Даны целые *x,y,z*. Вычислить *max(min(x,y), z)*3.*

2-20. Даны целые *x,y,z*. Вычислить *min(max(x,y), max(y,z))*max(y,z).*
2-21. Даны целые *x,y,z*. Вычислить *max(min(x,5), max(y,0))*5.*
2-22. Даны целые *x,y*. Вычислить *max(min(x–y,y–x),0).*
2-23. Даны целые *x,y*. Вычислить $max^2(max(x*y,x+y),0)$.
2-24. Даны целые *x,y*. Вычислить $(min(0,x)–min(0,y))*max^2(y,x)$.
2-25. Даны целые *x,y,z*. Вычислить |*(min(x,y,z)–(max(x,y)|*2*

§ 3 Циклы со счетчиком

Для организации цикла со счетчиком служит команда

```
LOOP  метка
```

Причем *метка* должна быть расположена в начале цикла. Здесь используется в качестве счетчика регистр **CX** (и только он). До начала цикла в этот регистр помещают целое положительное число – количество оборотов цикла. Затем при каждом выполнении тела цикла счетчик уменьшается на единицу и значение счетчика сверяется с нулем, и если **CX** не равен нулю, то цикл повторяется. Отметим, что изменение счетчика происходит автоматически.

Пример 3-1. *Для заданного N рассчитать величину*

*N! = 1*2*3*…*N*

```
; вычисление факториала заданного N
DSEG segment
N = 8
DSEG ends
CSEG segment
ASSUME CS:CSEG, DS:DSEG
ORG 100H
begin:
        MOV AX,DSEG
        MOV DS, AX
        MOV AX,1     ;AX:=O!
        MOV CX,N     ;CX:=N как счетчик цикла
        MOV SI,1     ; i:=1
F:      MUL SI       ; AX:=AX*i
        INC SI       ; i:=i+1
```

```
        LOOP F
        MOV BX,AX  ; итог переносим в BX(т.к.
                   ; AX дальше понадобится)
        MOV AH,4CH
        INT 21h
CSEG ends
end begin
```

Небольшое пояснение к программе. Первоначально в регистр **AX** помещаем 1 (это факториал нуля). Значение счетчика цикла **CX** задаем равным N. В регистр **SI** помещаем единицу и далее в цикле значение этого регистра увеличивается на 1 командой `INC SI`. Таким образом, в цикле (от строки с меткой **F:** до строки `LOOP F`) значение регистра **AX** последовательно домножается на **SI** командой `MUL SI`. Цикл проработает N раз, в итоге в регистре **AX** и будет факториал числа N.

Также проведем трансляцию и компоновку, а затем просмотрим полученный EXE-файл с помощью **DEBUG**

-U

```
1A70:0100 B8701A    MOV     AX,1A70
1A70:0103 8ED8      MOV     DS,AX
1A70:0105 B80100    MOV     AX,0001
1A70:0108 B90800    MOV     CX,0008
1A70:010B BE0100    MOV     SI,0001
1A70:010E F7E6      MUL     SI
1A70:0110 46        INC     SI
1A70:0111 E2FB      LOOP    010E
1A70:0113 8BD8      MOV     BX,AX
1A70:0115 B44C      MOV     AH,4C
1A70:0117 CD21      INT     21
```

Проверим результат работы программы:

-G 117

```
AX=4C80   BX=9D80   CX=0000   DX=0000   SP=0000
BP=0000
```

Видим, что при заданном N=8 получаем в регистре **BX** значение 9D80h, а это 16-ричное число есть 40320 в десятичной системе. Нетрудно убедится (например с помощью инженерного калькулятора), что N! = 40320. Т.е. программа работает правильно.

Замечание. Как отмечалось ранее, операция **MUL** сохраняет результат умножения в двух регистрах: в **DX** – старшая часть результата, в **AX** – младшая часть (в примере 2-1 результат помещался в один регистр, т.е. в **AX**). Поэтому, например, при N больших 8 факториал уже не поместится в **AX** и уже придется смотреть и в **DX**. Но при N>12 для сохранения результата будет недостаточно уже и двух регистров!

Проверьте работу программы при иных значениях N (в том числе и при больших 12). Для этого достаточно (в том же режиме **DEBUG**) изменить строку

```
    1A70:0003 B90A00     MOV      CX,0008
```

А именно, например, задав

A 0003

```
    1A70:0003   MOV      CX,000A
```

тем самым зададим N=10 и получим результат уже в двух регистрах **AX** и **DX.**

Рассмотрим более сложный пример – сочетание цикла и ветвления.

Пример 3-2. *Рассчитать величину N!!*

$$N!!=\begin{cases}1*3*5*...*N & если \quad N-нечетное\\ 2*4*6*...*N & если \quad N-четное\end{cases}$$

В данной задаче сначала следует выяснить, каким является заданное N (четным или нечетным), и в зависимости от этого организовывать цикл с последовательным умножением соответственно четных или нечетных чисел вплоть до заданного N.

```
; вычисление дважды факториала заданного N
DSEG segment
N = 7               ; задаем число N
DSEG ends
CSEG segment
ASSUME CS:CSEG, DS:DSEG
ORG 100H
begin:
    MOV AX,DSEG   ; загрузим адрес начала сегмента
```

```
    MOV DS,AX      ; данных в регистр DS
     MOV AX,N      ; заносим N
     MOV BX,2      ;делитель – для проверки на четность
     DIV BX        ; делим для проверки на четность
     CMP DX,0 ; сравниваем остаток от деления (в DX) с 0
     JNE NECH ; если не равен нулю, то – на нечетные
     MOV CX,AX; счетчик цикла (целая часть от дел. на 2)
     MOV AX,1      ; AX:=1
     MOV SI,2   ; нач. знач. для индекса (=2,т.к.четные)
CHET:    MUL SI   ; Перемножение четных чисел
     ADD SI,2     ;получение следующего значения индекса
     LOOP CHET    ; цикл
     MOV DI,AX    ; занесение результата в DI
     JMP EN       ; выход в конец программы
NECH:
     MOV CX,AX    ; счетчик цикла (для нечетных)
     MOV AX,1     ; AX:=1
     MOV SI,3  ;нач.знач. для индекса(=3, т.к. нечетные)
N2:  MUL SI       ; перемножение нечетных чисел
     ADD SI,2    ; получение следующего значения индекса
      LOOP N2 ; цикл
      MOV DI,AX    ; занесение результата в DI
EN:  MOV AH,4CH
     INT 21h      ; выход в DOS
CSEG ends
end begin
```

Обратим внимание, что в данном случае счетчик цикла равен целой части при делении заданного N на 2. Это число N проверяем на четность делением на 2 и проверкой остатка при таком делении (для четных – остаток равен нулю).

Также оттранслируем и скомпонуем программу, а затем просмотрим полученный EXE-файл с помощью **DEBUG**. Увидим, что при N=7 результат (в регистре **DI**) равен 69h (т.е. 105). Если установить значение N =8, то получим 180h (=384). Нетрудно убедится, что все верно.

Практические задания к § 3

Выполните те же примеры, что и в заданиях к **§1**, но кроме вычисления результата по формуле, вычислите *в цикле* указанные суммы. Убедитесь в равенстве этих чисел. Вывести оба этих числа (сумму и значение по формуле) в двух различных регистрах.

Замечание. Здесь следует рассмотреть самые разные значения **n**, в том числе и такие, при которых результат может быть отрицательным.

§4. Массивы

4.1 Одномерные массивы

Описание массива делается с помощью директивы данных. Например,

```
X DW 100 DUP (?)
```

описывает массив X[0..99] , состоящий из двухбайтовых слов.

Можно при описании задать значения элементов

```
X DW 1,2,3,4,5,6,7,8,9,10
```

В результате адрес ячейки памяти нулевого элемента есть X, адрес следующего (первого по номеру) есть X+2 (элементы – двухбайтовые слова, поэтому адреса сдвигаются на два) и т.д.

При работе с массивом, таким образом, потребуется вычислять индексы. Индекс очередного элемента массива будем вычислять в регистре (например, в **SI**), а обращаться к нужному элементу будем с помощью модификации адреса, т.е. записи его в квадратных скобках рядом с именем массива: `X[SI]`.

Далее при решении задач будем дополнительно выводить поясняющую строку – где смотреть результат. Такую строку также будем формировать в сегменте данных с помощью директивы данных. Строка текста обязательно должна завершаться символом $. Для ссылки на эту строку будем использовать команду

```
LEA DX, STROKA
```

Такая команда указывает, что в **DX** помещается адрес начала строки `STROKA`. И соответственно при выполнении прерывания

INT 21h с указанием функции 09h будет выводится данная строка, т.е. содержимое переменной STROKA.

Пример 4-1. *Задан массив из 10 элементов. Вычислить сумму его элементов.*

```
; вычисление суммы элементов массива из 10
; элементов-слов (результат см. в BX)
DSEG SEGMENT
X DW 1,2,3,4,5,6,7,8,9,10
STROKA DB 'СМ. РЕЗУЛЬТАТ В BX  $ '
DSEG ENDS
SSEG SEGMENT STACK
DB 256 dup (?)
SSEG ENDS
CSEG SEGMENT
ASSUME CS:CSEG, DS:DSEG, SS:SSEG
ORG 100H
BEGIN:
    MOV AX, DSEG ; адрес сегмента данных в AX
    MOV DS, AX   ; AX в DS
    MOV AX,0     ; начальное значение суммы
    MOV CX,10    ; счетчик цикла
    MOV SI,0     ; нач. знач. (удвоенного) индекса
L:  ADD AX,X[SI] ; AX:=AX+X[I]
    ADD SI,2     ;  следующий индекс
    LOOP  L      ; цикл 10 раз
    MOV BX, AX   ; сохраняем результат в BX
                 ;(AX понадобится далее)
     MOV AH,09H
    LEA DX, STROKA  ; вывод сообщения
    INT 21H
    MOV AX,4C00H    ; выход в DOS
    INT 21H
CSEG ends
END BEGIN
```

Как обычно, проведем трансляцию, компоновку и просмотрим EXE-файл с помощью **DEBUG**. Нетрудно увидеть, что **DX** = 37h (=55 в десятичной), т.е верный результат. Обратите внимание, что на экране появится и поясняющая строка: `СМ. РЕЗУЛЬТАТ В BX`.

4.2 Двумерные массивы

В Ассемблере допускается использование и двумерных массивов. В этом случае в качестве индексных регистров нужно использовать два индексных регистра, причем один из них обязательно должен быть **BX** или **BP**, а другой – регистром **SI** или **DI** (модифицировать по парам **BX** и **BP** или **SI** и **DI** нельзя). Применим это на примере.

Пример 4-2. *Пусть имеется матрица (двумерный массив) из байтовых элементов размером 5x4:*

1 2 3 1
2 0 2 2
3 4 5 6
4 5 6 7
5 6 7 5

Требуется подсчитать количество таких строк этой матрицы, в которых начальный элемент строки встречается еще раз.

При расположении элементов матрицы в памяти по строкам (первые 4 байта – начальная строка матрицы, следующие 4 байта – вторая строка и т.д.) адрес элемента A[i,j] равен A+4*i+j. Для хранения величины 4*i отведем регистр **BX**, а для хранения j – регистр **SI**. Тогда `A[BX]` – это начальный адрес I-й строки матрицы, а `A[BX][SI]` - адрес j-го элемента этой строки.

Программа имеет вид:

```
; количество строк в матрице, в которых есть элементы,
; равные первому элементу строки
; результат см. в BX
```

```
DSEG SEGMENT
; задаем матрицу из 5 строк и 4 столбцов
   A DB  1,2,3,1,
         2,0,2,2,
         3,4,5,6,
         4,5,6,7,
         5,6,7,5
STROKA DB 'результат см. в BL  $'
DSEG ENDS
SSEG SEGMENT STACK
DB 256 DUP (?)
SSEG ENDS
CSEG SEGMENT
ASSUME CS:CSEG, DS:DSEG, SS:SSEG
ORG 100H
BEGIN:
     MOV AX, DSEG ; устан-ем начало сегмента данных
     MOV DS, AX   ; на регистр DS
     MOV AL,0     ; количество искомых строк
; внешний цикл по строкам
     MOV CX,5     ; счетчик внешнего цикла
     MOV BX,0     ; смещение от A до нач. строки (20*I)
L:   MOV AH, A[BX]; AH – нач. Элемент строки
     MOV DX, CX   ; запоминание CX внешнего цикла
; внутренний цикл по столбцам
     MOV CX, 3    ; задание счетчика внутреннего цикла
     MOV SI,0     ; индекс элемента внутри строки (j)
L1:  INC SI       ; j:=j+1
     CMP A[BX][SI], AH     ; A[i,j] =AH ?
     LOOPNE L1    ; цикл, пока A[I,j]<>AH, но не более 3 раз
     JNE L2       ; AH не повторился -> L2
     INC AL       ; учет строки
;конец внутреннего цикла
L2:  MOV CX, DX   ; восстановить CX для внешнего цикла
     ADD BX, 4    ; на начало следующей строки матрицы
     LOOP L       ; цикл 5 раз
     MOV BL, AL
     LEA DX, STROKA    ; вывод поясняющего
```

```
    MOV AH, 09H   ; сообщения
    INT 21H
    MOV AX, 4C00H     ; выход
    INT 21H           ; в DOS
CSEG ENDS
END BEGIN
```

Обратите внимание, что здесь создан и сегмент стека

```
SSEG SEGMENT STACK
DB 256 DUP (?)
SSEG ENDS
```

Его размер равен 256 байтовым ячейкам.

Здесь дополнительно использован вариант команды цикла LOOPNE, которая осуществит выход из цикла не только при равенстве вышесравниваемых значений, но и при счетчике (**CX**) равном нулю.

Проверим программу в работе и убедимся, что число таких строк (в данном случае) равно 3.

Практические задания к §4

Раздел А

А4-1. Дан массив из *N* элементов. Найти сумму элементов с четными номерами и отдельно – с нечетными номерами.

А4-2. Дан массив из *N* элементов (натуральные числа). Определить, количество элементов, которые являются нечетными числами.

А4-3. Дан массив из *N* элементов (натуральные числа). Определить количество элементов, кратных 3.

А4-4. Дан массив из *N* элементов (натуральные числа). Определить количество элементов, не кратных 5.

А4-5. Дан массив из *N* элементов (натуральные числа). Определить количество элементов, имеющих четные порядковые номера и являющихся нечетными числами.

А4-6. Дан массив из *N* элементов (натуральные числа). Определить сумму тех элементов, которые кратны 5.

А4-7. Дан массив из *N* элементов (целые числа). Определить сумму тех, которые нечетны и отрицательны.
А4-8. Дан массив из *N* элементов (натуральные числа). Определить среднее арифметическое кратных 5, но не кратных 10.
А4-9. Дан массив из *N* элементов (натуральные числа). Определить произведение элементов, кратных заданному числу P.
А4-10. Дан массив из *N* элементов (натуральные числа). Определить количество элементов, кратных 3, но не кратных 5.
А4-11. Дан массив из *10* элементов (натуральные числа). Вычислить сумму элементов, индексы которых есть степени двойки (1, 2, 4, 8).
А4-12. Дан массив из *10* элементов (натуральные числа). Вычислить сумму элементов, индексы которых есть полные квадраты (1, 4, 9).
А4-13. Дан массив из *10* элементов (натуральные числа). Вычислить сумму элементов, индексы которых есть числа Фибоначчи (1, 2, 3, 5, 8).
А4-14. Дан массив из *10* элементов (натуральные числа). Подсчитать, сколько элементов отличается от последнего.
А4-15. Дан массив из *10* элементов (натуральные числа). Подсчитать, сколько элементов отличается от первого.
А4-16. Дан массив из *10* элементов (натуральные числа). Подсчитать сумму элементов, отличающихся от заданного числа P.
А4-17. Дан массив из *10* элементов (натуральные числа). Подсчитать количество элементов, отличающихся от заданного числа P.
А4-18. Дан массив из *10* элементов (натуральные числа). Подсчитать сумму элементов, меньших заданного числа P.
А4-19. Дан массив из *10* элементов (натуральные числа). Подсчитать количество элементов, меньших заданного числа P.
А4-20. Дан массив из *10* элементов (натуральные числа). Подсчитать сумму элементов, больших заданного числа P.
А4-21. Дан массив из *10* элементов (натуральные числа). Подсчитать количество элементов, больших заданного числа P.
А4-22. Дан массив из *N* элементов (целые числа). Определить сумму тех, индексы которых есть простые числа.
А4-23. Дан массив из *N* элементов (целые числа). Определить количество тех, которые являются простыми числами.

А4-24. Дан массив из N элементов (целые числа). Определить количество тех, которые являются степенями двойки.
А4-25. Дан массив из N элементов (целые числа). Определить количество тех, которые являются полными квадратами (полный квадрат – квадрат какого-либо натурального числа).

Раздел Б

Б4-1. Дана матрица 5х5. Найти сумму элементов главной и побочной диагоналей.

Б4-2. Дана матрица 5х5. Найти наименьшее из значений элементов побочной диагонали и двух соседних с ней линий.

Б4-3. Дана матрица 5х5. Для данного натурального M найти сумму тех элементов матрицы, сумма индексов которых равна M.

Б4-4. Дана матрица 5х5. Выяснить, верно ли, что наибольшее из значений элементов главной диагонали больше, чем наименьшее из значений элементов побочной диагонали.

Б4-5. Результаты соревнований по прыжкам в длину представлены в виде матрицы 5х3 (5 спортсменов по 3 попытки у каждого). Указать, какой спортсмен и в какой попытке показал наилучший результат.

Б4-6. Результаты соревнований по пятиборью представлены в виде матрицы 5х5 (5 спортсменов и 5 видов соревнований), в которых указаны места, занятые каждым спортсменом в данном виде. Найти лучшего спортсмена (наименьшая сумма мест).

Б4-7. Дана матрица 5х5. Найти количество строк, у которых все элементы нули.

Б4-8. Дана матрица 5х5. Найти количество строк, в каждой из которых все элементы не равны нулю.

Б4-9. Дана матрица 5х5. Найти количество строк, в каждой из которых все элементы одинаковы.

Б4-10. Дана матрица 5х5. Найти количество строк, в каждой из которых все элементы четны.

Б4-11. Дана матрица 5х5. Найти количество строк, в каждой из которых все элементы нечетны.
Б4-12. Дана матрица 5х5. Найти количество строк, в каждой из которых элементы образуют монотонную последовательность.
Б4-13. Дана матрица 5х5. Найти количество строк, в каждой из которых элементы образуют симметричную последовательность (палиндром).
Б4-14. Дана матрица 5х5. В строках с нулевым элементом на главной диагонали найти сумму элементов такой строки.
Б4-15. Дана матрица 5х5. В строках с нулевым элементом на главной диагонали найти наибольший из элементов.
Б4-16. Дана матрица 3х3. Найти сумму элементов для каждого из столбцов.
Б4-17. Дана матрица 3х3. Найти сумму элементов для каждой из строк.
Б4-18. Дана матрица 3х3. Найти наименьший элемент в каждой строке.
Б4-19. Дана матрица 3х3. Найти наибольший элемент в каждой строке.
Б4-20. Дана матрица 3х3. Найти наименьший элемент в каждом столбце.
Б4-21. Дана матрица 3х3. Найти наибольший элемент в каждом столбце.
Б4-22. Дана матрица 5х5. Найти сумму элементов над главной диагональю.
Б4-23.Дана матрица 5х5. Найти сумму элементов под главной диагональю.
Б4-24. Дана матрица 5х5. Найти сумму элементов над побочной диагональю.
Б4-25.Дана матрица 5х5. Найти сумму элементов под побочной диагональю.

§5. Процедуры

5.1 Понятие о процедуре

Часто в программах приходится несколько раз решать одну и ту же подзадачу. И поэтому приходится многократно выписывать одинаковую группу команд, решающую эту подзадачу. Такую группу команд рекомендуется оформить отдельной *процедурой* и ее просто вызывать в нужных местах основной программы.

Размещают процедуры, как правило, в конце сегмента команд (за последней командой основной программы). Причем оформляют ее специальным образом:

<*имя процедуры*> PROC <*параметры*>

<тело процедуры>

<*имя процедуры*> ENDP

При этом <*имя процедуры*> своего рода метка, именно по этому имени и вызывается процедура. Среди <*параметров*> процедуры два основных:

`NEAR` – близкий; это значит, что к процедуре можно обращаться только из того сегмента команд, где она описана

`FAR` – дальний ; к такой процедуре можно обращаться из любых сегментов команд

Обращаться к процедуре следует по команде

`CALL <имя процедуры>`

Завершаться тело процедуры должно командой возврата

`RET`

По команде вызова запоминается в стеке адрес возврата (команда следующая в основной программе за вызовом `CALL`). По команде возврата `RET` и происходит переход на этот адрес возврата.

Если вызываемая процедура является *ближней*, то меняется только указатель команд **IP** и не меняется сегментный регистр **CS**.

Но если вызываемая процедура будет *дальней* (расположена в другом сегменте команд), то в этом случае должны меняться оба регистра.

Если через AB обозначить адрес возврата – адрес команды, следующей за CALL, то действия команды CALL P, где P – имя процедуры, можно описать так:
близкий возврат: AB → стек, IP:= offset P
дальний возврат:CS→стек, AB→стек, CS:=seg P, IP:=offset P
Действия для двух вариантов команды RET таковы:

близкий возврат: стек→ IP:= offset P
дальний возврат: стрек→ IP, стек →CS

Ясно, что команды CALL и RET должны действовать согласованно: при близком вызове процедура и возврат должны быть близкими, а при дальнем вызове и возврат должен быть дальним.

Имеется такая тонкость. Ассемблер сам выберет правильный вариант машинной команды для CALL, только если процедура была описана раньше этой команды. В противном случае Ассемблер всегда выбирает близкий уровень. Чтобы не было такой ошибки, при обращении к дальней процедуре, которая будет описываться позже, надо в команде CALL с помощью оператора PTR явно указать, что процедура дальняя:

```
CALL FAR PTR P
```

5.2 Передача параметров в процедуру

Самый простой способ передачи параметров в процедуру и получение результатов ее работы – *через регистры*. Рассмотрим пример

Пример 5-1. *Вычислить c= max(a,b)+max(5,a–1), где все числа знаковые размером в слово*.

Вычисление *max(x,y)* оформим как процедуру, причем, *x* будем передавать через регистр **AX**, *y* через **BX**, а возвращать *max* через **AX**.

Тогда программа с процедурой выглядит так:

```
; вычислить max(a,b)+max(5,a-1), используя процедуру
DSEG SEGMENT
A = 5
B = 9
CC DW ?
```

```
STROKA DB 'результат смотри в BX $', 10, 13
        ; коды 10 и 13 – коды команд
        ; перехода на следующую строку экрана и
        ; установки на начало этой строки
DSEG ENDS
CSEG SEGMENT
ASSUME CS:CSEG, DS:DSEG
ORG 100H
BEGIN:
;основная программа
    MOV AX,DSEG
    MOV DS,AX
    MOV AX,A     ; AX:=a
    MOV BX,B     ; BX:=b
    CALL MAX     ; AX:=MAX(A,B)
    MOV DX,AX        ; спасти AX
    MOV AX,5     ; AX:=5
    MOV BX,A
    DEC BX       ;BX:=a-1
    CALL MAX     ; AX:=MAX(5,a-1)
    ADD DX, AX
    MOV CC,DX    ; C:=max(a,b)+max(5,a-1)
     MOV BX,CC

    MOV AH,09H
    LEA DX,STROKA
    INT 21H
    MOV AH,4CH
    INT 21H
; процедура : AX=max (AX,BX)
MAX PROC FAR
    CMP AX,BX
    JAE MAX1
    MOV AX,BX
MAX1: RET
MAX  ENDP
CSEG ENDS
END BEGIN
```

Проверьте эту программу в работе и посмотрите результат в регистре **BX**. При заданных значениях мы должны получить в **BX** 16-ричное число E (=14 в десятичной).

Но как быть, когда требуется передавать много параметров? На них просто может не хватить регистров. В этом случае рекомендуется передавать параметры *через стек*. Напомним, что *стек – это хранилище информации, функционирующее по правилу: первым из стека всегда считывается элемент, записанный в стек последним*. По сути, стек – совокупность выделенных ячеек памяти. В компьютере имеются команды, поддерживающие работу со стеком. В этих командах предполагается, что регистр **SP** указывает на ячейку стека, в которой находится элемент, записанный в стек последним. Для *занесения данных в стек* служит команда

```
PUSH регистр1
```

(т.е. указывается содержимое какого регистра следует поместить в стек), а для *извлечения значения из стека* используется команда

```
POP регистр2
```

т.е указывается в какой регистр следует поместить результат (*регистр1* и *регистр2* не обязательно должны совпадать!)

Перед обращением к процедуре основная программа записывает параметры в стек, а процедура их затем извлекает. Как? Как добраться к элементам стека без считывания их из стека? Для этого используют регистр **BP** – засылают в него адрес вершины стека (содержимое регистра **SP**), а затем используют выражение `[BP+i]` для доступа к параметрам процедуры. При этом, однако, если регистр **BP** используется в процедуре, то его предварительно надо спасти, пользуясь тем же стеком:

```
P    PROC
     PUSH BP      ;спасти BP
     MOV BP,SP    ;BP – на вершину стека
     …
     команды процедуры
```

Очевидно, что перед выходом из процедуры нужно восстановить старое значение BP и очистить стек от параметров :

```
     POP BP   ; восстановить старое значение BP
     RET 2*k  ;очистить стек от параметров (их k слов)
```

5.3 Рекурсивные процедуры

Процедура называется *рекурсивной*, если она обращается сама к себе либо непосредственно (*прямая рекурсия*), либо через другие процедуры (*косвенная рекурсия*).

При обращении процедуры к самой себе лучше рассматривать обращение к копии, т.е. как будто к другой такой же процедуре. Но сразу возникает вопрос: не произойдет ли зацикливания? Нет, если в ней есть нерекурсивная ветвь, т.е. такой путь вычисления, на котором рекурсивный вызов обходится. Надо также позаботится о том, чтобы рекурсивная процедура обязательно сохраняла значения всех регистров, которыми она пользуется.

Пример 5-2. *Опишем рекурсивную процедуру для вычисления чисел Фибоначчи:*

$$F(n) = \begin{cases} 1, n=0 \text{ или } n=1 \\ F(n-1)+ f(n-2), n>=2 \end{cases}$$

```
;BX=F(X) – число Фибоначчи с номером n (AL=n)
F    PROC
     CMP AL,1      ;n>1 →F1
     JA F1
; нерекурсивная ветвь
     MOV BX,1      ; n<=1 → BX=F(n)=1
     RET
; рекурсивная ветвь
F1:  PUSH AX       ;спасти AX  (будем менять)
     DEC AL        ; AL=n –1
     CALL F        ; BX=F(n–1) (AX  не изменится)
     PUSH BX       ; спасти  F(n–1)
     DEC AL        ; AL=n –2
     CALL  F       ; BX=F(n–2)
     POP AX        ;восстановить F(n–1), но уже в AX
     ADD BX, AX    ; BX=F(n)=F(n–2)+F(n–1)
     POP AX        ; восстановить исходное значение AX
     RET
F    ENDP
```

Напишите главную программу для задания числа N и вычисления числа Фибоначчи для заданного N с вызовом приведенной процедуры.

Практические задания к §5

5-1. Для заданных целых *x,y* вычислить *z=(sign(x)+sign(y))*sign(x+y),* где

$$sign(a)=\begin{cases}-1 & a<0\\ 0 & a=0\\ 1 & a>0\end{cases}$$

5-2. Даны три натуральных числа. Вычислить наибольший общий делитель, создав процедуру определения наибольшего общего делителя для двух чисел.

5-3. Даны четыре натуральных числа. Вычислить наименьшее общее кратное, создав процедуру определения наименьшего общего кратного для двух чисел.

5-4. Даны числа *a,b,c,d*. Получить *x=min(a,b), y=min(c,d), z=min(x,y).* Вычисление *min(k,m)* (меньшего из двух чисел *k,m*) оформить процедурой.

5-5. Даны целые (не обязательно положительные) *x, y* и натуральные *n , k.* Вычислить x^n, y^k. Оформить процедуру *step(a,m)* от целого *a* и натурального *m*, вычисляющую (через последовательное умножение) a^m и проверить ее в работе.

5-6. Оформить процедуру *maxmin(x,y),* которая присваивает *x* большее из целых *x* и *y*, а *y* – меньшее из них. Рассчитать с ее помощью для четырех заданных целых чисел сумму наибольших и произведение наименьших.

5-7. Даны целые *s,t*. Получить *f(t,–2s,1) + f(2,t, s–t)*, где $f(a,b,c)=(2a-b)(5+c)$

5-8. Даны целые s,t. Получить *g(1,s) + g(t, s) – g(2s–1,st),* где $g(a,b)=(a^2+b^2)(a^2+2ab+3b^2+4)$

5-9. Даны целые *a,b,c*. Получить $(\max(a,a+b)+\max(a,b+c))(1+\max(a+bc,10))$

Вычисление большего их двух чисел оформить процедурой.

5-10. Даны целые *a,b*. Получить

$u= min(a,b), v=min(ab,a+b), min (u+v^2, 3)$

Вычисление минимального из двух чисел оформить функцией.

5-11. Даны целые *a,b,c,d*. Для каждой из возможных троек чисел ответить – можно ли построить треугольник с такими сторонами. (в треугольнике большая сторона меньше суммы двух других сторон). Поиск наибольшего из трех оформить процедурой.

5-12. Дан одномерный массив из 10 элементов (натуральные числа). Получить количество элементов, являющихся простыми числами. Проверку, является ли число простым, оформить процедурой.

5-13. Дан одномерный массив из 10 элементов (натуральные числа). Получить количество элементов, являющихся полными квадратами. Проверку, является ли число полным квадратом, оформить процедурой.

5-14. Дан одномерный массив из 10 элементов (натуральные числа). Получить количество элементов, являющихся числами Фибоначчи. Проверку, является ли элемент массива числом Фибоначчи, оформить процедурой.

5-15. Даны натуральные M и N. Найти натуральные M1 и N1, не имеющие общего делителя такие, что

$\frac{M}{N}=\frac{M1}{N1}$ (т.е. сократить дробь). Поиск необходимого в данном случае наибольшего общего делителя оформить процедурой.

5-16. Написать процедуру сложения простых дробей:

$\frac{M}{N}+\frac{P}{Q}=\frac{A}{B}$, где A,B,M,N,P,Q – натуральные числа. Поиск необходимого здесь наименьшего общего кратного оформить процедурой.

5-17. Написать процедуру вычитания простых дробей:

$\frac{M}{N}-\frac{P}{Q}=\frac{A}{B}$, где A,B,M,N,P,Q – натуральные числа. Поиск необходимого здесь наименьшего общего кратного оформить процедурой.

5-18. Написать процедуру умножения простых дробей (с последующим сокращением дроби-результата). Поиск необходимо-

го в данном случае наибольшего общего делителя оформить процедурой.

5-19. Написать процедуру деления простых дробей (с последующим сокращением дроби-результата). Поиск необходимого в данном случае наибольшего общего делителя оформить процедурой.

5-20. Дан массив a1, … a5 из натуральных чисел. Получить b1,… b5 по правилу: b1 =a1; b2 = a1+a2; … b5=a1+a2+a3+a4+a5. Вычисление bi оформить процедурой.

5-21. Дан массив a1, … a5 (каждое есть натуральное число меньшее 10). Получить b1+ b2+ b3 + b4 +b5, где b1 = a1, b2= $a2^2$, … b5 = $a5^5$. Вычисление bi оформить процедурой.

5-22. Дан массив a1, … a5 (каждое есть натуральное число меньшее 10). Получить b1• b2• b3 • b4 •b5, где

$$bi = \begin{cases} 1, если \quad ai - нечетное \\ 2, если \quad ai - четное \end{cases}$$ Вычисление bi оформить процедурой.

5-23. Дан массив a1, … a5 (каждое есть натуральное число меньшее 10). Получить b1+ b2+ b3 + b4 +b5, где

$$bi = \begin{cases} 1, если \quad ai - нечетное \\ 0, если \quad ai - четное \end{cases}$$ Вычисление bi оформить процедурой.

5-24. Дано 5 различных целых чисел. Найти два числа, модуль разности которых имеет наибольшее значение. Поиск наибольшего из двух чисел оформить процедурой.

5-25. Дано 5 различных целых чисел. Найти два числа, модуль разности которых имеет наименьшее значение. Поиск наименьшего из двух чисел оформить процедурой.

5-26. Составить программу, которая позволит вычислить число сочетаний из N по M по следующей формуле:

$$C_N^0 = C_N^N = 1; \quad C_N^M = C_{N-1}^M + C_{N-1}^{M-1} \quad при \quad 0 < M < N$$

Собственно вычисление числа сочетаний оформить в виде рекурсивной функции.

§ 6. Вывод чисел на экран

До сих пор мы с вами полученные результаты программы просматривали в режиме отладчика DEBUG – смотрели содержимое регистров (причем, видели числа в 16-ричной системе).

Конечно, требуется научиться выводить результаты работы программы на экран, причем, в привычной нам 10-й системе.

Такой вывод на экран можно сделать по одной цифре. Иначе говоря, будем разбивать число на отдельные цифры путем последовательного целочисленного деления на 10. Остатки от такого деления и будут давать цифры (справа налево). Например, пусть некоторое число равно A=5678.

A0:=A mod 10 даст 8
A:=A div 10 даст 567 (т.е. уже без последней цифры)
A1:=A mod 10 даст 7
A:=A div 10 даст 56
A2:=A mod 10 даст 6
A:=A div 10 даст 5
A3:=A mod 10 даст 5
A:=A div 10 даст 0

(здесь операцией **mod** обозначили получение остатка от деления, а **div** – целочисленное деление). Процесс деления продолжаем пока частное (очередное полученное число A) не станет равным нулю. При этом также следует считать, сколько цифр числа мы получили (для организации цикла). Полученные остатки (в данном примере числа A0, A1, A2, A3) будем последовательно помещать в стек, а затем также последовательно, но уже в обратном порядке доставать оттуда.

Кроме того, надо иметь в виду, что в памяти компьютера числа хранятся в виде цифр-символов. И у каждого символа, как известно, имеется ASCII-код. Причем, код цифры 0 равен 30, 1 – код 31, … 9 – 39 (см. таблицу ASCII- кодов в Приложении 3). Поэтому, чтобы получить код цифры-символа, нужно к этой цифре прибавить 30 и таким образом выводить полученный символ. Для вывода символа можно использовать функцию **02h** прерывания **INT 21**. Значение 02h помещаем в **AH**. При этом код символа

должен быть размещен в регистре **DX** (именно так работает функция вывода символа).

Оформим вывод десятичного числа на экран *процедурой*. Для ее вызова организуем главную программу, в которой зададим это десятичное число, вызовем процедуру вывода и просто выйдем в DOS.

Пример 6-1. *Вывести десятичное число на экран.*

```
; программа задает десятичное число и
; выводит его на экран
Code_SEG  SEGMENT
    ASSUME CS:Code_SEG
    ORG 100H
BEGIN:  MOV DX,1234      ; задание десятичного числа
    CALL Write_dec   ; вызов процедуры вывода
                     ; десятичного числа
    MOV AH, 4CH      ;выход в DOS
    INT 21H
; процедура вывода 10-ичного числа
Write_dec    PROC NEAR
    mov cx,0
    mov ax,dx    ; перенос пришедшего извне DX в AX
    mov dx,0
    mov bx,10   ; задаем делитель 10
del:inc cx       ; увеличиваем CX на 1
                 ;(считаем кол-во цифр в числе)
    div bx       ; делим AX на BX, в AX будет
                 ; частное, в DX - остаток
    push dx       ; запоминаем в стек остаток
                 ;(очередную цифру числа)
    mov dl,0
    cmp ax,0 ; проверяем AX на равенство нулю
    jnz del  ; если нет - возвращаемся на del
             ;(продолжить деление)
vyv:pop dx   ; вытаскиваем их стека послед.
             ; занесенную туда цифру
    add dl,30h   ; прибавим к ней 30 - получаем
                 ; код цифры-символа
```

```
outt:
    mov ah,2      ; функция вывода символа
    int 21h
    loop vyv      ; цикл для вывода след. цифры
                  ; (в CX – счетчик кол–ва цифр)
    ret
Write_dec    ENDP
CODE_seg ENDS
    END   BEGIN
```

Внимательно проанализируйте текст программы, комментарии в ней. Процедуру вывода числа нужно будет использовать в дальнейшем.

После трансляции и компоновки программы с выводом числа, как обычно, будет создан EXE-файл, и его достаточно просто запустить на выполнение (без использования DEBUG). В результате мы просто получим число на экране (в данном случае, 1234).

Практические задания к § 6

Доработать программы своего варианта практических заданий § 2 и § 3 так, чтобы результат выводился на экран (с необходимым текстовым пояснением).

Замечание 1. При выводе результатов заданий §3 должно быть выведено *два* числа (*сумма* – левое выражение равенства и *формула* – правое выражение равенства). Лучше всего их вывести в разных строках с соответствующими пояснениями в строке.

Замечание 2. Обратите внимание, что числа-результаты могут быть и *отрицательными*. Для вывода отрицательных чисел придется получить их модуль (например, домножив на –1) и вывести на экране перед модулем этого числа символ “–“.

§ 7. Ввод и вывод чисел в программе на ассемблере

До сих пор исходные данные для программ мы задавали сразу в тексте программы (в сегменте данных). Но было бы удобнее (например, для многократного использования программы с другими исходными данными) вводить их в процессе выполнения программы.

Для ввода символа с клавиатуры имеется функция **01h** прерывания **INT 21**. При вводе чисел следует дополнительно анализировать нажимаемый символ. Нужно проверять, не нажата ли иная клавиша (кроме цифровой) и если это так, то попросить нажать именно цифровую клавишу.

Таким образом, следует организовать:
- процедуру ввода символа
- процедуру проверки нажатого символа

Процедура ввода отдельного символа имеет вид:

```
INPUT PROC
    MOV AH,01H
    INT 21H
    RET
INPUT ENDP
```

Сам символ при этом помещается в AL.
Процедура анализа введенного символа приводится ниже. Напомним, что коды символов-цифр находятся в интервале от 30h до 39h, т.е. если нажат иной символ, то на экране будет предупреждающее сообщение и потребуется вводить число сначала. Здесь же, в этой процедуре будем накапливать и вводимое из цифр число. Напомним, что мы имеем дело с позиционной системой счисления, а это значит, например, при вводе 3-значного числа, что первая цифра – это число сотен (10 в степени 2), вторая цифра – это число десятков (10 в степени 1) , а третья цифра – число единиц. Соответственно до первого обращения к процедуре нужно задать значение **CL** равным максимальной (для данного

числа) степени 10. Если это 3-значное число, то **CL** надо придать значение 100.

```
_test proc
    Cmp al, 30h
    Jb error
    Cmp al, 39h
    Ja error
    Sub al, 30h
    Mul cl  ; домножаем на степень 10 (чтобы цифра
            ; заняла нужную позицию в числе)
    Add dl, al; накапливаем все вводимое число в dl
    mov al,cl
    mov ch,10
    div ch      ; уменьшаем степень 10-ки
    mov cl,al   ; в регистре CL
    Ret
error:
    Mov ah, 09h  ; вывод сообщения об ошибке
    Lea dx, meserror
    Int 21h
    stc       ; установка флага CF (признак ошибки)
    Ret
_test endp
```

Здесь мы использовали флаг **CF** – признак переноса в арифметических операциях. С помощью команды **STC** мы *устанавливаем* этот флаг (делаем значение его равным 1) в случае ошибочности введенного значения (нажали символ, не являющийся цифрой). И далее будем осуществлять переход при установленном этом флаге с помощью команды **JC** (см. ниже)

В главной программе в сегменте данных должны быть сообщение вида

```
meserror db ‘ Вводите правильно  (цифру) $’
```

Окончательный текст программы ввода 3-значного десятичного числа приводим ниже.

Пример 7-1. *Введите трехзначное число и сохраните его в DL*

```
D_seg  segment
message db  13,10,' Вводите число $'
Meserror db 13,10,' Введите правильно (цифру) $'
d_seg  ends
c_seg segment
assume cs:c_seg, ds:d_seg
org 100h
begin:  mov ax, d_seg
    mov ds, ax
    mov ah, 09
    lea dx, message ; сообщение с приглашением ввода
    int 21h
    mov dx,0
    mov bl,04    ; количество цифр числа
    mov cx, 0003
input1: call Input
    dec bl
    call _test   ; вызов процедуры проверки
                 ; правильности ввода цифры
    jc begin     ; если была ошибка ввода,
                 ; то начать ввод сначала
    cmp bl,0     ; ввели все цифры числа?
    ja input1    ; Нет. Продолжить ввод
    mov ah, 4ch  ; Выход в DOS
    int 21h
;Процедура ввода отдельного символа:
Input proc
    Mov ah,01h
    Int 21h
    Ret
Input endp
; процедура проверки введенного  символа
_test proc
    Cmp al, 30h
    Jb error
```

```
    Cmp al, 39h
    Ja error
    Sub al, 30h
    Mul cl  ; домножаем на степень 10(чтобы цифра
            ; заняла нужную позицию в числе)
    Add dl, al;накапливаем все вводимое число в dl
    mov al,cl
    mov ch,10
    div ch      ; уменьшаем степень 10-ки
    mov cl,al   ; в регистре CL
    Ret
error:
    Mov ah, 09h  ; вывод сообщения об ошибке
    Lea dx, meserror
    Int 21h
    stc        ; установка флага CX (признак ошибки)
    Ret
_test endp

c_seg  ends
end begin
```

Доработайте программу так, чтобы сначала можно было указать количество цифр в числе (не более 5, т.к. максимальное десятичное число, допустимое в регистре есть 65535).

Доработайте программу так, чтобы не появлялось предупреждающего сообщения в случае ошибки, а просто курсор продолжал быть в ожидании ввода очередной цифры (т.е. в случае ошибки одной цифры не требовалось начинать ввод сначала).

Доработайте программу ввода еще и выводом полученного числа (см. предыдущую лаб.работу).

Пример 7-2. *Доработать пример 3-1 (факториал N) вводом числа N и выводом результата*

Отметим сразу, что, как отмечалось ранее, задавать большие N не имеет смысла – факториал очень быстро растет и значение его просто не поместится в регистре. Поэтому при проверке данной программы рекомендуется ограничится числом N не более 8.

```
; ВВОД 3-значного десятичного числа N
; вычисление факториала 1*2*3*...*N (не более 8!)
;  вывод результата на экран
D_seg  segment
N dw ?
message db  13,10,' Вводите число $'
Meserror db 13,10,' Введите правильно (цифру) $'
entr db 13,10,'результат $'
D_seg  ends
C_seg segment
assume cs:c_seg, ds:d_seg
org 100h
_BEG:   mov ax, d_seg
    mov ds, ax
    call vvod4
    mov N, DX
    MOV AX,1
    MOV CX,N
    MOV SI,1
F:  MUL SI
    INC SI
    LOOP F
    mov bx, ax
    mov ah,09h
    lea dx, entr
    int 21h
    mov dx,bx
    call vyvod4
    mov ah, 4ch     ; Выход в DOS
    int 21h
vvod4 proc
begin:  mov ax, d_seg
    mov ds, ax
    mov ah, 09
    lea dx, message ; сообщение с приглашением ввода
    int 21h
    mov dx,0
    mov bl,03    ; количество цифр числа
```

```
    mov cl, 100  ; степень 10 (двузначное число!)
input1: call input
    dec bl
    call _test   ; вызов процедуры проверки
                 ; правильности ввода цифры
    jc begin     ; если была ошибка ввода,
                 ; то начать ввод сначала
    cmp bl,0     ; ввели все цифры числа?
    ja input1    ; Нет. Продолжить ввод
    ret
vvod4   endp
;Процедура ввода отдельного символа:
Input proc
    Mov ah,01h
    Int 21h
    Ret
Input endp
; процедура проверки введенного  символа
_test proc
    Cmp al, 30h
    Jb error
    Cmp al, 39h
    Ja error
    Sub al, 30h
    mul cl
    Add dl, al; накапливаем все вводимое число в dl
    mov al,cl
    mov ch,10
    div ch
    mov cl,al
    Ret
error:
    Mov ah, 09h  ; вывод сообщения об ошибке
    Lea dx, meserror
    Int 21h
    stc       ; установка флага CX (признак ошибки)
    Ret
_test endp
```

```
; вывод 10-ичного числа
vyvod4  proc
    mov cx,0
    mov ax,dx  ; перенос пришедшего извне DX в AX
    mov dx,0
    mov bx,10  ; задаем делитель 10
del:inc cx     ; увеличиваем CX на 1
               ;(считаем кол-во цифр в числе)
    div bx     ; делим AX на BX, в AX будет частное,
               ; в DX - остаток
    push dx    ; запоминаем в стек остаток
               ;(очередную цифру числа)
    mov dl,0
    cmp ax,0
    ja del       ; если нет - возвращаемся на del
                 ; (продолжить деление)
vyv:pop dx   ; вытаскиваем их стека последнюю
             ; занесенную туда цифру
    add dl,30h   ; прибавим к ней 30 - чтобы
                 ; получить код цифры (символ)
outt:
    mov ah,2     ; функция вывода символа
    int 21h
    loop vyv     ; цикл для вывода след. цифры
                 ; (в CX счетчик числа цифр)
    ret
vyvod4  ENDP
c_seg  ends
end _BEG
```

Практические задания к §7

Доработайте программы заданий §3 и §5 так, чтобы сначала извне *вводились* исходные данные, а потом, после вычислений, *выводились* результаты.

Замечание. При выполнении этих заданий может потребоваться вводить отрицательные числа. Для их ввода можно сначала потребовать указать знак числа (например, «введи 0, если число положительное или введи 1 – если число отрицательное»), а уже потом вводить модуль числа.

§ 8. Работа с файлами на Ассемблере

При работе с файлами в программе на Ассемблере также используются функции MS DOS (в паре с прерыванием INT 21h).

Приведем эти функции:

Функция для открытия файла **3Dh**:

Вход	*Выход*
AH = 3Dh	AX = номер файла
AL = тип открытия файла: 00 – только чтение, 01 – только запись 02 – и чтение, и запись	JC – ошибка (флаг CF устанавливается в 1)
DS:DX = адрес ASCII строки с именем файла	

Функция для закрытия файла **3Eh**:

Вход	*Выход*
AH = 3Eh	ничего
BX = номер файла	

Функция для чтения из файла **3Fh**:

Вход	*Выход*
BX = номер файла	AX = число считанных байт (или код ошибки, если CF=1)
CX = число байт	JC – ошибка (флаг CF устанавливается в 1)
DS:DX = адрес буфера	

Функция для записи в файл (или устройство) **40h**:

Вход	*Выход*
BX = номер файла	AX = число переданных байт (или код ошибки, если CF=1)
CX = число байт; если 0, то длина файла усекается до текущего положения указателя	JC – ошибка (флаг CF устанавливается в 1)
DS:DX = адрес буфера	

Удобство работы с описателями (номерами) файлов состоит в том, что для выбора устройства ввода/вывода достаточно указать нужный описатель:
0 – устройство ввода клавиатура
1 – устройство вывода экран
2 – устройство для вывода ошибок (тоже экран)
3 – асинхронный порт (COM1)
4 – печатающее устройство (LPT1)

Проанализируйте приведенный ниже пример. Подобную программу нужно будет разработать для выполнения индивидуального задания.

Пример 8.1 *Программа вывода данных из текстового файла на экран*

```
; программа вывода текстового файла на экран
; файл должен быть предварительно создан в текстовом
; редакторе
Dseg SEGMENT
BUFER DB 200 DUP (?)      ; буфер ввода/вывода
FILE DB 'C:\POS_ASM\PROBA.TXT',0   ; путь к файлу
HANDLE DW ?       ; переменная для хранения описателя
                  ; (номера) файла
EOF DB 0          ; если 1, то достигнут конец файла
DSEG ENDS
SSEG SEGMENT STACK
    DB 30 DUP(?)
TOP DB ?
SSEG ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DSEG, SS:SSEG
BEGIN:
; готовим регистры
    MOV AX, DSEG
    MOV DS, AX
    MOV AX, SSEG
    MOV SS, AX
    MOV SP, OFFSET SS:TOP
```

```
; открываем файл
    MOV AH, 3DH
    MOV AL,0
    LEA DX, FILE
    INT 21H          ; вызов функции открытия файла
    JC ER            ; если ошибка – закончим
    LEA DX, BUFER    ;  DX на буфер ввода/вывода
    MOV BX, AX
    MOV HANDLE, BX   ; сохраним описатель файла
    MOV CX, 200      ; читаем/пишем по 200 байт
CIKL:
; читаем в буфер
    MOV AH, 3FH
    INT 21H
    JC CLOSE         ;  если ошибка – закончить
    CMP CX, AX
    JZ NOT_EOF
; конец файла достигнут
    MOV EOF,1
    MOV CX, AX
NOT_EOF:
; пишем на экран из буфера
    MOV AH, 40H
    MOV BX,1 ;  описатель для вывода на экран
    INT 21H
    CMP EOF,1    ; не пора ли заканчивать
    MOV BX, HANDLE
    JZ CLOSE
    JMP CIKL ; продолжаем читать
CLOSE:
; закрыть файл
    MOV AH, 3EH
    INT 21H
ER:
    MOV AH, 4CH  ;выйти в ОС
    INT 21H
CODE ENDS
     END BEGIN
```

Рассмотрим теперь пример *создания* текстового файла с клавиатуры. Для этого потребуется использовать еще одну функцию.

Функция для создания файла **3Ch**:

Вход	*Выход*
AH = 3Ch	AX = номер файла
CX = атрибут файла	JC – ошибка (флаг CF устанавливается в 1)

Создавая файл, помните, что он всегда открывается как для чтения, так и для записи. В CX рекомендуется задавать 0 – отсутствие специальных атрибутов. Напомним, что у файлов бывают атрибуты вида: «только для чтения», «архивный» и т.д. Если файл не будет создан (например, диск или текущий каталог закрыты для записи), то устанавливается флаг CF.

Пример 8.2 *Программы ввода данных с клавиатуры в файл*

```
;ввод символов с клавиатуры в создаваемый файл 1.TXT
; прекращение работы программы – клавиши Ctrl+C
DATA SEGMENT
PATH DB '1.TXT',0     ; файл для вывода
DATA ENDS
STA SEGMENT STACK
    DB 100 DUP (?)
STA ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:STA
BEGIN:
    MOV AX,DATA
    MOV DS,AX
;создаем и открываем файл
    MOV AH, 3CH
    LEA DX, PATH
    MOV CX, 0
    INT 21H
    JC KONEC; если ошибка – конец
    MOV BX, AX   ; сохранить описатель
```

```
READ:
; читаем со стандартного устройства один символ
    PUSH BX
    MOV BX, 0
    MOV CX,1
    MOV AH,3FH
    INT 21H
    POP BX
    JC CLOSE ;если не удалось прочесть, то конец
    MOV SI, CX
    MOV DI,AX
    MOV CX, AX
;пишем в файл
    MOV AH,40H
    LEA DX,PATH
    INT 21H
    JC CLOSE     ; Если не удалось, то конец
    CMP CX, AX
    JNZ CLOSE
    CMP SI,DI
    JZ READ
; закрываем файл
CLOSE:
    MOV AH, 3EH
    INT 21H
KONEC:
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END BEGIN
```

Рассмотрим пример, в котором содержимое одного файла добавляется к содержимому другого файла. Здесь потребуется, в частности, устанавливать указатель файла в нужное место. Для этого служит функция **42h**, которая позволяет переместиться к любому байту файла.

Функция установки указателя в файле **42h**:

Вход	*Выход*
AL = режим установки указателя: 0 – абсолютное смещение от начала файла 1 – знаковое смещение от текущего положения указателя 2 – знаковое смещение от конца файла	DX = старшая часть значения текущей позиции указателя
BX = указатель файла	AX = младшая часть значения текущей позиции указателя
CX = старшая часть смещения	
DX = младшая часть смещения	

Пример 8.3 *Программа для перезаписи данных из одного файла в другой файл.*

```
; программа слияния двух файлов
DATA SEGMENT
PATH1 DB 'PRIMER1.TXT',0 ; ИМЯ 1-ГО ФАЙЛА
PATH2 DB 'PRIMER2.TXT',0 ; ИМЯ 2-ГО ФАЙЛА
HANDL1 DW ?              ; описатель 1-го файла
HANDL2 DW ?              ; описатель 2-го файла
BUFER DB 1000 DUP (?)
EOF DB 0     ; если 1, то достигнут конец файла
DATA ENDS
SSEG SEGMENT STACK
    DB 200 DUP(?)
SSEG ENDS
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:SSEG
BEGIN:
```

```
    MOV AX, DATA
    MOV DS,AX
; открываем первый файл
    MOV AH, 3DH
    MOV AL,0
    LEA DX, PATH1
    INT 21H
    JC EXIT
    MOV HANDL1, AX
; открываем второй файл
    MOV AH, 3DH
    MOV AL,1
    LEA DX, PATH2
    INT 21H
    JC close1
    MOV HANDL2, AX
; указатель второго файла на конец
    MOV AH,42H
    MOV BX,HANDL2
    XOR CX,CX
    XOR DX,DX
    MOV AL,2
    INT 21H
;готовы регистры
    LEA DX,BUFER
    MOV CX, 1000
;блок копирования
    LOO:
; читаем
    MOV BX,HANDL1
    MOV AH,3FH
    INT 21H
    CMP AX,CX
    JZ NORM
    MOV CX, AX   ; <1000 байт
    MOV EOF,1    ; достигнут конец файла
NORM:
; пишем
```

```
    MOV BX, HANDL2
    MOV AH,40H
    INT 21H
    CMP EOF,0    ; не достигнут ли конец
    JZ LOO
; закрываем второй файл
CLOSE2:
    MOV AH,3EH
    MOV BX, HANDL1
    INT 21H
; закрываем первый файл
CLOSE1:
    MOV AH,3EH
    MOV BX,HANDL1
    INT 21H
; выход в DOS
EXIT:
    MOV AH, 4CH
    INT 21H
CODE ENDS
END BEGIN
```

Практические задания к § 8

1) Создать программу для ввода данных в файл с клавиатуры. Имя файла задается в программе.
2) Создать программу для обработки полученного файла и результат обработки записать программой в другой файл.

Ниже приводятся конкретный вид данных для ввода в первый файл и правило для его обработки.

8-1. Дана запись с именем STUDENT, содержащая следующие поля:

- Фамилия и инициалы,
- Номер группы,
- Успеваемость (массив из пяти элементов)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 10 записей типа STUDENT, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл фамилий и номеров групп для всех студентов, у кого средний балл студента больше 4 (если таких нет – вывести об этом сообщение)

8-2. Дана запись с именем STUDENT, содержащая следующие поля:
- Фамилия и инициалы,
- Номер группы,
- Успеваемость (массив из пяти элементов)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 10 записей типа STUDENT, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5 (если таких нет – вывести об этом сообщение)

8-3. Дана запись с именем STUDENT, содержащая следующие поля:
- Фамилия и инициалы,
- Номер группы,
- Успеваемость (массив из пяти элементов)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 10 записей типа STUDENT, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2 (если таких нет – вывести об этом сообщение)

8-4. Дана запись с именем AEROFLOT, содержащая следующие поля:
- Название пункта назначения рейса,

- Номер рейса,
- Тип самолета

Написать программу, которая выполняет следующие действия:

- Ввод с клавиатуры данных из 7 элементов типа AEROFLOT, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры (если таких нет – вывести об этом сообщение)

8-5. Дана запись с именем AEROFLOT, содержащая следующие поля:

- Название пункта назначения рейса,
- Номер рейса,
- Тип самолета

Написать программу, которая выполняет следующие действия:

- Ввод с клавиатуры из 7 элементов типа AEROFLOT, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры (если таких нет – вывести об этом сообщение)

8-6. Дана запись с именем WORKER, содержащая следующие поля:

- Фамилия и инициалы работника,
- Название занимаемой должности,
- Год поступления на работу

Написать программу, которая выполняет следующие действия:

- Ввод с клавиатуры данных из 10 элементов типа WORKER, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в файл фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры (если таких нет – вывести об этом сообщение)

8-7. Дана запись с именем TRAIN, содержащая следующие поля:
- Название пункта назначения,
- Номер поезда,
- Время отправления

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа TRAIN, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о поездах, отправляющихся после введенного с клавиатуры времени (если таких нет – вывести об этом сообщение)

8-8. Дана запись с именем TRAIN, содержащая следующие поля:
- Название пункта назначения,
- Номер поезда,
- Время отправления

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 6 элементов типа TRAIN, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о поездах, отправляющихся в пункт, название которого введено с клавиатуры (если таких нет – вывести об этом сообщение)

8-9. Дана запись с именем TRAIN, содержащая следующие поля:
- Название пункта назначения,
- Номер поезда,
- Время отправления

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа TRAIN, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о поезде, номер которого введен с клавиатуры (если таких нет – вывести об этом сообщение)

8-10. Дана запись с именем MARSH, содержащая следующие поля:
- Название начального пункта назначения,
- Название конечного пункта назначения,
- Номер маршрута

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа MARSH, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о маршруте, номер которого введен с клавиатуры (если таких нет – вывести об этом сообщение)

8-11. Дана запись с именем MARSH, содержащая следующие поля:
- Название начального пункта назначения,
- Название конечного пункта назначения,
- Номер маршрута

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа MARSH, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о маршрутах, которые начинаются или заканчиваются в пункте, название которого введено с клавиатуры (если таких нет – вывести об этом сообщение)

8-12. Дана запись с именем NOTE, содержащая следующие поля:
- Фамилия, имя,
- Номер телефона,
- Дата рождения (массив из трех чисел)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа NOTE, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о человеке, номер телефона которого введен с клавиатуры (если таких нет – вывести об этом сообщение)

8-13. Дана запись с именем NOTE, содержащая следующие поля:
- Фамилия, имя,
- Номер телефона,
- Дата рождения (массив из трех чисел)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа NOTE, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры (если таких нет – вывести об этом сообщение)

8-14. Дана запись с именем NOTE, содержащая следующие поля:
- Фамилия, имя,
- Номер телефона,
- Дата рождения (массив из трех чисел)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа NOTE, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о человеке, чья фамилия введена с клавиатуры (если таких нет – вывести об этом сообщение)

8-15. Дана запись с именем ZNAK, содержащая следующие поля:
- Фамилия, имя,
- Знак Зодиака,
- Дата рождения (массив из трех чисел)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа ZNAK, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о человеке, чья фамилия введена с клавиатуры (если таких нет – вывести об этом сообщение)

8-16. Дана запись с именем ZNAK, содержащая следующие поля:
- Фамилия, имя,
- Знак Зодиака,
- Дата рождения (массив из трех чисел)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа ZNAK, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о людях, родившихся под знаком, название которого введено с клавиатуры (если таких нет – вывести об этом сообщение)

8-17. Дана запись с именем ZNAK, содержащая следующие поля:
- Фамилия, имя,
- Знак Зодиака,
- Дата рождения (массив из трех чисел)

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных в из 8 элементов типа ZNAK, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о людях, родившихся в месяц, значение которого введено с клавиатуры (если таких нет – вывести об этом сообщение)

8-18. Дана запись с именем PRICE, содержащая следующие поля:
- Название товара,
- Название магазина, в котором продается товар,
- Стоимость товара в руб.

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа PRICE, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о товаре, название которого введено с клавиатуры (если таких нет – вывести об этом сообщение)

8-19. Дана запись с именем PRICE, содержащая следующие поля:
- Название товара,
- Название магазина, в котором продается товар,
- Стоимость товара в руб.

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа PRICE, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о товарах, продающихся в магазине, название которого введено с клавиатуры (если таких нет – вывести об этом сообщение)

8-20. Дана запись с именем ORDER, содержащая следующие поля:
- Расчетный счет плательщика,
- Расчетный счет получателя,
- Перечисляемая сумма в руб.

Написать программу, которая выполняет следующие действия:
- Ввод с клавиатуры данных из 8 элементов типа ORDER, и занесение их в файл данных
- Чтение данных из файла и вывод их на экран
- вывод в другой файл информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры (если таких нет – вывести об этом сообщение)

§ 9. Работа с вещественными числами

Для работы с вещественными числами на ассемблере необходимо использовать *математический сопроцессор*. Чтобы его подключить, необходимо в начале программы написать команду вида: **.286** или **.8087** в зависимости от того, какой тип сопроцессора будем использовать.

Непосредственно перед использованием сопроцессор необходимо инициализировать командой **finit**.

У сопроцессора имеется свой регистровый стек, состоящий из нескольких регистров: st(0), st(1), … st(7). Размерность каждого такого регистра 80 битов. Для управления ими также имеются специальные команды, причем отдельно для работы с целыми и вещественными числами. Мы будем рассматривать работу только с вещественными числами.

Кроме этих восьми регистров имеются еще служебные регистры, состоящие каждый из 16 битов :

swr - регистр состояния сопроцессора (аналогичен регистру флагов обычного процессора)

cwr – управляющий регистр сопроцессора (с его помощью можно управлять точностью выполнения операций, округлением и т.п.). Для нас важным является 10-й и 11-й биты этого регистра. Их значения задают порядок округления. Если обозначить за M значение в st(0), которое должно быть округлено, обозначим также A и B – наиболее близкие к M значения, причем A<M<B, тогда значение этих битов:

00 – округление к ближайшему числу A или B

01 – округление в меньшую сторону (т.е. M=A)

10 – значение M округляется в большую сторону (M=B)

11 – производится отбрасывание дробной части M (для использования в операциях целочисленной арифметики)

Для арифметических операций с использованием сопроцессора служат специальные команды. Причем все команды сопроцессора начинаются с буквы **f** (float). Вторая буква в имени операции определяет тип числа:

i – целое двоичное число,

b – целое десятичное число,

отсутствие буквы – вещественное число.

Последняя буква **p** в записи команды означает, что последним действием команды обязательно является извлечение операнда из стека.

Для записи вещественных чисел существуют различные форматы:

Формат	**Короткий**	**Длинный**	**Расширенный**
Длина числа	32	64	80
Размерность мантиссы	24	53	64
Диапазон значений	от 10^{-38} до 10^{+38}	от 10^{-308} до 10^{+308}	от 10^{-4932} до 10^{+4932}
Размерность порядка	8	11	15
Фиксированное смещение			
Директива определения числа	dd	dq	dt

Схематично представление чисел в таких форматах выглядит так:

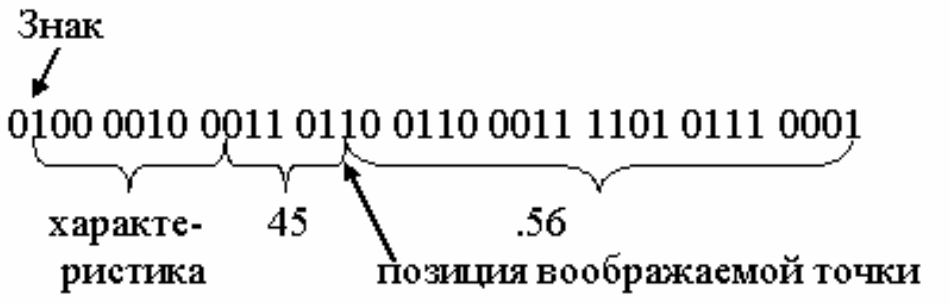
знак s	Характеристика q	Мантисса (M)	*Короткий формат*
31	24	23 0	
знак s	Характеристика q	Мантисса (M)	*Длинный формат*
63	53	52 0	
знак s	Характеристика q	Мантисса (M)	*Расширенный формат*
79	64	63 0	

В итоге числа записывают в виде:

$A = (-1)^s * N^q * M$

здесь: S – значение знакового разряда числа (0 – число положительно, 1 – число отрицательное), q = p+фиксированное смещение (p – порядок числа), M – мантисса ($|M| < 1$) , N – основание системы счисления (N=2).

В результате, например, 43,56 в формате короткого вещественного в двоичном представлении имеет вид:



Команды передачи данных

Группа команд передачи данных предназначена для организации обмена между регистрами стека, вершиной стека сопроцессора и ячейками оперативной памяти. Команды этой группы имеют такое же назначение для процесса программирования сопроцессора, как и команда MOV основного процессора. Эти команды можно разделить на три группы: передачи данных в вещественном, в целочисленном и десятичном формате. Мы будем рассматривать команды передачи данных в вещественном формате.

fld *источник* – загрузка вещественного числа из области памяти (*источник*) в вершину стека сопроцессора (st(0))

fst *приемник* – сохранение вещественного числа из вершины стека сопроцессора в память

fstp *приемник* – то же, но с выталкиванием числа из стека (т.е. очищается st(0))

fxch st(i) – обмен значений между вершиной стека и регистром стека сопроцессора.

Действие команды **fld** можно сравнить с командой **push** основного процессора.

Дополнительно упомянем и *команды загрузка констант* :

fldz – загрузка нуля в вершину стека сопроцессора,

fld1 – загрузка единицы в вершину стека сопроцессора,

fldpi – загрузка числа Pi в вершину стека сопроцессора,

Аналогичные команды для целых чисел просто содержат букву **i** после буквы **f**.

Арифметические команды

Также будем рассматривать команды сопроцессора для вещественных чисел.

Команда сложения:

fadd - сложение st(0) и st(1) и результат в st(0)
fadd *источник* - сложение st(0) и *источник*а и результат в st(0)
fadd st(i), st - сложение st(i) и st(0) и результат в st(0)
faddp st(i), st - сложение st(i) и st(0) и результат в st(i-1), а из st(0) число выталкивается

Команда вычитания

fsub – вычитает значение st(1) из значения st(0)и результат - в st(0),
fsub *источник* – вычитает значение источника из значения st(0) и результат - в st(0),
fsub st(i), st – вычитает значение st(0) из значения st(i) и результат - в st(i),
fsubp st(i), st – вычитает значение st(0) из значения st(i)и результат - в st(i-1), причем из st(0) значение выталкивается.

Команда умножения

fmul - умножает st(0) на st(1) и результат помещает в st(0),
fmul st(i) - умножает st(0) на st(i) и результат помещает в st(0),
fmul st(i), st - умножает st(0) на st(i) и результат помещает в st(i),
fmulp st(i), st - умножает st(0) на st(i) и результат помещает в st(i-1), а из st(0) значение выталкивается

Команда деления

fdiv - делит st(0) на st(1) и результат помещает в st(0),
fdiv st(i) - делит st(0) на st(i) и результат помещает в st(0),
fdiv st(i), st - делит st(0) на st(i) и результат помещает в st(i),
fdivp st(i), st - делит st(0) на st(i) и результат помещает в st(i-1), а из st(0) значение выталкивается.

Рассмотрим простейший пример.

Пример 9.1 *Поместить в регистры стека st(0) и str(1) два вещественных числа и сложить их с помещением результата в тот же регистр st(0).*

```
; СУММИРОВАНИЕ ДВУХ ВЕЩЕСТВЕННЫХ ЧИСЕЛ
```

```
; С ПОМОЩЬЮ СОПРОЦЕССОРА
.model small ; указание модели памяти
.8087
.486    ; указание типа сопроцессора
; СЕГМЕНТ ДАННЫХ
dseg segment
    f1 dq 1.1
    f2 dq 3.3
    rez dq ?
dseg ends
; СЕГМЕНТ КОДА
cseg segment
    assume ds:dseg, cs:cseg
    org 100h
begin:
    mov ax,dseg
    mov ds,ax
    finit   ; инициализация сопроцессора
    fld f1  ;занесение 1-го числа (в регистр st(0) )
    fld f2  ;занесение 2-го числа (в регистр st(1) )
    fadd ;сложение st(0) с st(1) и результат в st(0)
; ВЫХОД В DOS
mov ax,4c00h
int 21h
cseg ends
end begin
```

Запишем, например, эту программу под именем **z9_1.asm**. Как обычно ассемблируем эту программу с получением в итоге exe-файла, т.е. даем последовательно команды:

MASM Z9_1.ASM;
LINK Z9_1.OBJ;

Чтобы просмотреть результаты (или даже ход работы) такой программы нужно воспользоваться более мощным отладчиком, чем DEBUG, а именно **TurboDebugger** (TD). При этом нас интересует именно содержимое регистров сопроцессора. Для их просмотра следует вызвать окно арифметического сопроцессора. Делает-

ся это командой ***View/Numeric processor***. При этом откроется окно, где и будут видны регистры сопроцессора.

Итак, чтобы просмотреть результаты работы конкретной программы, запускаем TD, затем даем команду ***File/Open*** и выбираем нужный exe-файл, а для просмотра регистров сопроцессора даем команду ***View/Numeric processor***. В результате получим окно вида (*рис. 9.1*).

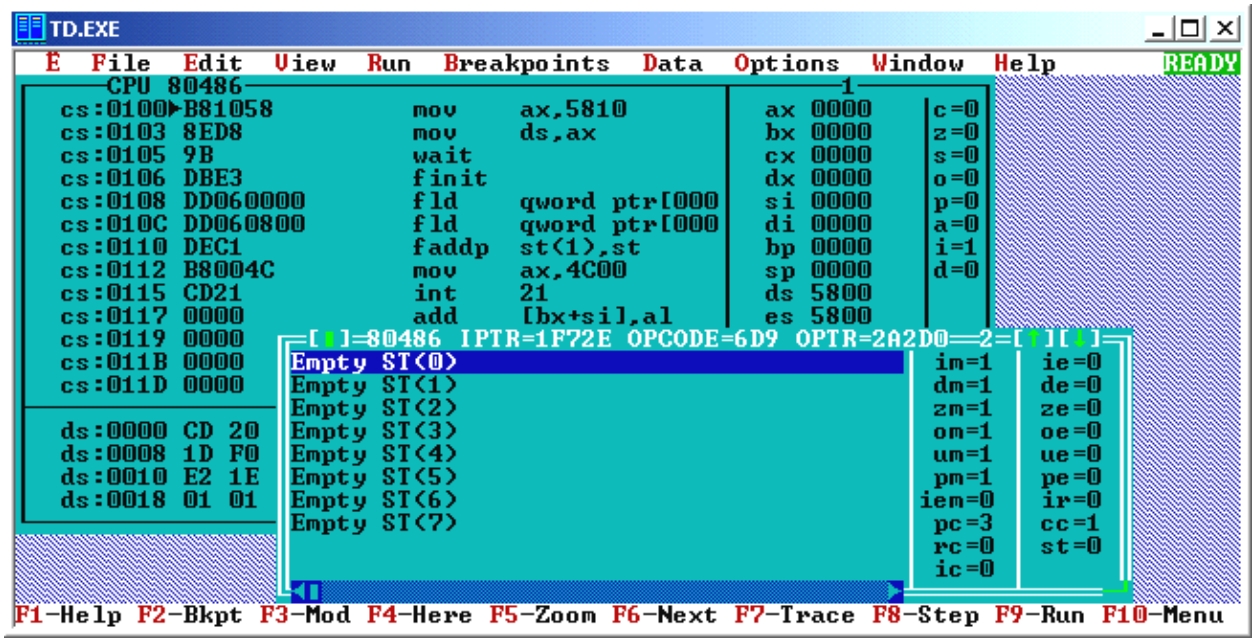


Рис. 9.1 *Окно Turbo Debugger с окошком Numeric processor*

Чтобы просмотреть работу нашей программы *примера 9.1*, нажмем клавишу **F9** (или проведем трассировку, последовательно нажимая **F7**). Нетрудно проследить за изменениями регистров в окошке сопроцессора и увидеть результат (в st(0)) равный (в данном случае) 4.3999999999999999. Напомним, что заданы числа 1.1 и 3.3, их сумма должна быть равна 4.4, но мы получили приближенное значение.

Попробуем ввести обсуждавшееся ранее указание на точность вычислений. Напомним, что для этого служит **cwr** – управляющий регистр сопроцессора (с его помощью можно управлять точностью выполнения операций, округлением и т.п.). Для нас важным является 10-й и 11-й биты. Их значения задают порядок округления. Зададим эти 10-й и 11-й биты значением 01. Если ввести переменную **cr** типа **dw**, то задание точности имеет вид:

Fstcw cr

```
Or cr, 0000010000000000b
Fldcw cr
```

Команды сравнения данных

Такие команды выполняют сравнение значений числа в вершине стека (st(0)) и операнда, указанного в команде.

fcom - сравнений значений из st(0) и st(1).

fcomp *операнд* - сравнение st(0) со значением операнда, который находится в регистре или в памяти, причем в конце результат из st(0) выталкивается.

fcompp *операнд* - аналогична предыдущей, причем в конце результат из st(0) и st(1) выталкивается.

ficom *операнд* - сравнений значений из st(0) и *целого* операнда в памяти.

ficomp *операнд* - сравнение st(0) со значением *целого* операнда, который находится в регистре или в памяти, причем в конце результат из st(0) выталкивается.

ftst – сравнение значений в st(0) со значением 00.

В результате работы команд сравнения в регистре состояния устанавливаются следующие значения бит кода условия c3,c2,c0:

- если st(0) > *операнда*, то 000 (c3c2c0)
- если st(0) < *операнда*, то 001 (c3c2c0)
- если st(0) = *операнду*, то 100 (c3c2c0)
- если операнды неупорядочены, то 111 (c3c2c0)

Для того чтобы получить возможность реагировать на эти коды командами условного перехода основного процессора (а оно реагируют на флаги в **flags**), нужно как-то записать сформированные биты условия c3,c2,c0 в регистр **flags**.

В системе команд сопроцессора существует команда **fstsw**, которая позволяет запомнить слово состояния сопроцессора в регистр **ax** или ячейке памяти. Далее значения нужных бит извлекаются и анализируются командами основного процессора. Например, запись старшего байта слова состояния, в котором находятся биты **c0 – c3**, в младший бит регистра **flags** осуществляется командой **sahf**. Эта команда записывает содержимое **ah** в младший байт регистра **flags**. После этого бит **c0** записывается на место **CF**, **c2** – на место **PF**, **c3** – на место **ZF**. Бит **c1** выпадает из общего правила, т.к. в регистре флагов на месте соответствующе-

го ему бита находится единица. Анализ этого бита нужно проводить с помощью логических команд основного процессора.

Рассмотрим более сложный пример.

Пример 9.2 *Даны два вещественных числа. Заменить первое число нулем, если оно меньше или равно второму, или оставить числа без изменения в противном случае.*

Обратите внимание, что здесь потребуется задавать числа извне, а значит, потребуется процедура ввода числа с экрана (т.е. процедура преобразования строки символов в число). Также потребуется процедура для вывода результата (т.е. преобразование вещественного числа в строку символов для вывода на экран). Проанализируйте эти процедуры, их потребуется применять при выполнении индивидуальных заданий.

```
; Даны два вещественных числа. Заменить первое число
; нулем, если оно меньше или равно второму, или
; оставить числа без изменения в противном случае
        .model  small
        .8087
        .486

STACKSG Segment para    public  'stack'
        org     100h
STACKSG ends

DATASG  Segment word    public  'DATA'
msg_inx db 0Ah, 0Dh, 'Введите действительное число x : $'
msg_iny db 0Ah, 0Dh, 'Введите действительное число y : $'
msg_ox  db      0Ah, 0Dh, 'X после вычислений : $'
msg_oy  db      0Ah, 0Dh, 'Y после вычислений : $'
x       dq      0
y       dq      0
a1      dq      0
a2      dq      0
ftemp   dq      0
c10     dw      10
c1      dw      1
```

```
c2      dw      2
cr      dw      ?
dig     dw      0
minus   db      0
buffer  db      255
buflen  db      0
bufdata db      255 dup(' ')
DATASG  ends

CODESG  Segment word    public  'CODE'
Assume  CS:CODESG, DS:DATASG, SS:STACKSG, ES:NOTHING
Start:  mov     ax, DATASG
        mov     ds, ax
        mov     ax, STACKSG
        mov     ss, ax
        finit           ; инициализация сопроцесора
        fstcw   cr
        or      cr, 0000110000000000b
        fldcw   cr
        mov     ah, 09h
        lea     dx, msg_inx
        int     21h     ; вывод строки на экран
        mov     ah, 0Ah
        lea     dx, buffer
        int     21h     ; ввод числа X
        lea     bx, x
        call asc2fp ;преобраз. числа в двоичный вид

        mov     ah, 09h
        lea     dx, msg_iny
        int     21h     ; вывод строки на экран
        mov     ah, 0Ah
        lea     dx, buffer
        int     21h     ; ввод числа y
        lea     bx, y
        call asc2fp  ;преобраз. числа в двоичный вид

        fld     x
```

```
        fcomp   y
        fstsw   ax
        sahf
        ja      @@1
        fldz
        fstp    x
@@1:    fstp    st(0)

        mov     ah, 09h
        lea     dx, msg_ox
        int     21h     ; вывод строки на экран
        lea     bx, x
        call    printfp
        mov     ah, 09h
        lea     dx, msg_oy
        int     21h     ; вывод строки на экран
        lea     bx, y
        call    printfp

        mov     ah, 4Ch ; выход
        int     21h

; процедура перевода числа из текстового
; представления в вещественное число
; параметры:
;   dx – смещение буффера
;   bx – смещение результата
asc2fp  proc    near
        fild    c10
        fldz
        inc     dx
        mov     di, dx
        xor     cx, cx
        mov     cl, [di]
        mov     minus, 0
        cmp     byte ptr [di+1], '-'
        jne     @01
        mov     minus, 1
```

```
        inc     di
        dec     cx
        ; преобразование целой части
@01:    inc     di
        xor     ax, ax
        mov     al, [di]
        cmp     al, '.'
        je      @02
        and     al, 0Fh
        mov     dig, ax
        fmul    st(0), st(1)
        fild    dig
        faddp   st(1), st(0)
        loop    @01
        jmp     @03
        ; преобразование дробной части
@02:    fstp    [bx]
        dec     cx
        xor     dx, dx
        xor     ax, ax
        fldz
@021:   inc     di
        mov     al, [di]
        and     al, 0Fh
        mov     dig, ax
        fmul    st(0), st(1)
        fild    dig
        faddp   st(1), st(0)
        inc     dx
        loop    @021
        mov     cx, dx
@022:   fdiv    st(0), st(1)
        loop    @022
        fld     [bx]
        faddp   st(1), st(0)
@03:    mov     al, minus
        test    al, al
        jz      @04
```

```
        fchs
@04:    fstp    [bx]
        fstp    st(0)
        ret
asc2fp  endp

; процедура вывода вещественного числа на экран
; bx – смещение действительного числа
printfp proc    near
        fild    c10
        fld     [bx]
        ftst
        fstsw   ax
        sahf
        jnb     @105
        mov     dl, '-'
        mov     ah, 02h
        int     21h
        fabs
@105:   fld1
        mov     cx, 7
@10:    fdiv    st(0), st(2)
        loop    @10
        faddp   st(1), st(0)
        xor     cx, cx
@11:    inc     cx
        fcom    st(1)
        fstsw   ax
        sahf
        jb      @12
        fdiv    st(0), st(1)
        jmp     @11
@12:    fist    dig
        mov     dx, dig
        or      dl, 30h
        mov     ah, 02h
        int     21h
        fisub   dig
```

```
        fmul    st(0), st(1)
        loop    @12

        fmul    st(0), st(1)
        fmul    st(0), st(1)
        fmul    st(0), st(1)
        fld1
        faddp   st(1), st(0)

        fstcw   cr
        push    cr
        and     cr, 1111001111111111b
        fldcw   cr
        frndint
        pop     cr
        fldcw   cr

        ftst
        fstsw   ax
        sahf
        jz      @14

        mov     ah, 02h
        mov     dl, '.'
        int     21h

        fdiv    st(0), st(1)
        fdiv    st(0), st(1)
        fdiv    st(0), st(1)
        mov     cx, 3

@13:    fist    dig
        mov     dx, dig
        or      dl, 30h
        mov     ah, 02h
        int     21h
        fisub   dig
        fmul    st(0), st(1)
```

```
        loop    @13

@14:    fstp
        fstp
        ret
printfp endp

CODESG  ends
        end     Start
```

Среди команд сопроцессора имеются *команды для вычисления функций.*

fsqrt – вычисление квадратного корня значения в st(0) и результат – помещается в st(0)

fabs - вычисление модуля значения в st(0) и туда же помещается результат.

frndint – округление до целого значения в st(0). Напомним, что режим округления задается значениями в двухбитовом поле RC (10-й и 11-й биты) управляющего регистра сопроцессора. При этом используются две команды **fstcwr** и **fldcwr**, которые, соответственно, записывают в память содержимое управляющего регистра сопроцессора и восстанавливают его обратно. Таким образом, пока содержимое этого регистра находится в памяти, можно установить необходимое значение поля RC .

***Пример 9**.3 Вычисление выражения* $z=(sqrt(|x|)-y)^2$

```
; вычисление выражения z=(sqrt(abs(x))-y)^2
.model small
.8087
.486
data segment
    x dd -100.0
    y dd 30.0
    z dd 0
data ends
code segment
```

```
assume cs:code, ds:data
org 100h
begin:
    mov ax, data
    mov ds, ax
    finit
    fld x
    fabs
    fsqrt
    fsub y
    fst st(1)
    fmul
    fst z
exit:
    mov ax, 4c00h
    int 21h
code ends
end begin
```

Имеются также команды и для вычисления тригонометрических функций.

fcos – команда вычисляет косинус угла, значение которого находится в вершине стека сопроцессора – регистре st(0). Результат возвращается в регистр st(0),

fsin – вычисляет синус значения в st(0) и значение возвращает в st(0).

Пример 9.4 *Вычисление COS и SIN PI радиан и размещение результата соответственно в регистрах st(1) и st(0)*

```
; Вычисление  COS и SIN  PI радиан и размещение ре-
зультата
; соответственно в регистрах st(1) и st(0)
.model small
.8087
.486
DATA SEGMENT
    X DQ ?
    Y DQ ?
DATA ENDS
```

```
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
ORG 100H
BEGIN:
     mov ax, data
     mov ds, ax
     finit
     fldpi
     FCOS
     fstp X
     fldpi
     FSIN
     fstp Y
     fld X
     fld Y
mov ax, 4c00h
int 21h
CODE ENDS
END BEGIN
```

После выполнения этой программы получим результат (*рис. 9.2*)

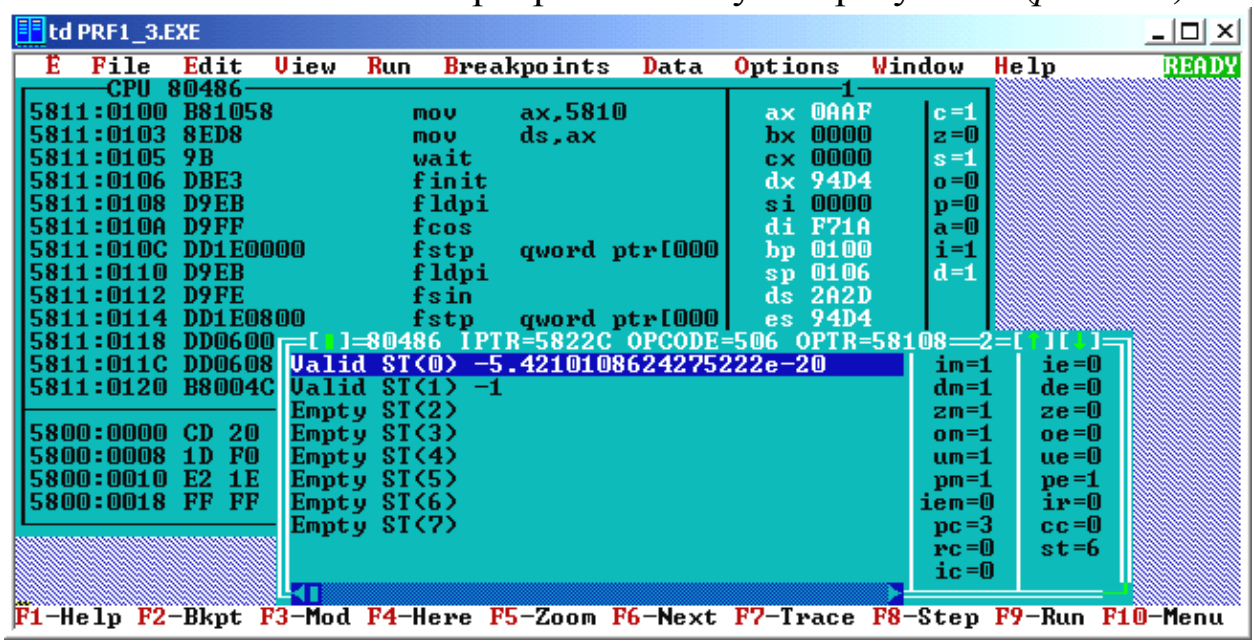


Рис. 9.2 *Результат (в Turbo Debugger) работы программы вычисления Sin и Cos величины π радиан*

Обратите внимание, что в st(0), где должен находится sin(Pi), вместо нуля имеем число порядка 10^{-20}. Это и есть нуль с машинной точностью. Если воспользоваться здесь командой **frndint**, вставив ее после вычисления косинуса, то уже получим чистый нуль.

Для реализации операции возведения вещественного числа в произвольную вещественную степень используют следующую формулу:

$$x^y = 2^{y*log2(x)}$$

где *log2(x)* – логарифм числа x по основанию 2.

На ассемблере для реализации этой операции служат следующие три команды:

f2xm1 – команда вычисления значения функции $y=2^{x-1}$; исходное значение *x* помещается в st(0) и должно лежать в диапазоне $-1<=x<=1$; результат *y* замещает *x* в st(0)

fyl2x – команда вычисления значения функции *z=ylog2(x);* исходное значение *x* помещается в st(0) и должно лежать в диапазоне $0<=x<+\infty$; исходное значение *y* помещается в st(1) и должно лежать в диапазоне $-\infty<=x<+\infty$; результат *z* помещается в st(0).

Обратите внимание, что *x* (основание) в операции $y=2^{x-1}$ должно лежать в диапазоне $-1<=x<=1$. Это значит, что эту команду надо применять, предварительно разбив *x* на целую часть и дробную часть (меньшую единицы). А для целых (т.е. 2^x, где *x* – целое со знаком) использовать команду **fscale**. Таким образом, применив формулу $2^{a+b} = 2^a \bullet 2^b$, где a – целая часть показателя степени, b – дробна часть (меньшая 1 по модулю), и сможем возводить в произвольную степень.

Обобщенный алгоритм вычисления степени произвольного числа *x* по произвольному показателю *y* следующий:

1) загрузить в стек основание степени *x*
2) загрузить в стек показатель степени *y* и проверить его знак:
   - если показатель степени *y*<0, то запомнить этот факт (см. далее шаг 10) и заменить в стеке значение *y* на его модуль: $x^{-y}=1/x^y$
   - если показатель степени y>=0, то перейти на шаг 3
3) командой **fyl2x** вычислить значение выражения *z = ylog2(x)*

4) проверить z на диапазон $-1<z<+1$:
   - если значение $|z|<1$, то обозначить его как $z2$ и перейти на шаг 5
   - если $|z|>1$, то представить z в виде двух слагаемых $z=z1+z2$, где $z1$ – целое значение, а $z2$ – значение меньше единицы. В программе это можно выполнить путем последовательного вычитания единицы из z до тех пор, пока оно не станет меньше единицы. Количество вычитаний надо запомнить, обозначим его n. Перейти на шаг 5.
5) выполнить команду **f2xml** с аргументом $z2$
6) выполнить команды **fldl** и **fadd**, компенсирующие единицу, которая была вычтена из результата функции 2^x на шаге 5 предыдущей командой **f2xml**.
7) Выполнить команду **fscale** с аргументом $z1$. Переменную $z1$ нужно инициализировать нулем, что позволит получить корректный результат, даже если исходное значение было меньше единицы.
8) Выполнить умножение результатов, полученных на шаге 6 и 7. После этого в вершине стека находится результат возведения в степень.
9) Завершить работу.
10) Шаг выполняется в случае, если показатель степени был отрицательным. В этом случае необходимо единицу разделить на результат шага 8. Для этого в стек загружаем единицу командой **fldl** и применим команду **fdiv**. Завершить работу

Реализует описанный алгоритм на конкретном примере.

Пример 9.5 *Вычислить 2^{-2}, используя вышеописанный алгоритм.*

```
.model small
.8087
.486
data segment
    flag db 0
    p1 dw 0
    y dt 2.0    ; основание степени
    x dt -2.0   ; показатель степени
data ends
```

```
code segment
    assume cs:code, ds:data
org 100h
begin:
main proc
    mov ax, data
    mov ds, ax
    finit
    fld y
    fld x
    ftst
    fstsw ax
    sahf
    jnc m1      ;переход, если x>=0
    inc flag    ; взведен flag, если x<0
    fabs
m1: fxch
    fyl2x
    fst st(1)
    fabs    ;|z|
; сравним |z| с единицей
    fld1
    fcom
    fstsw ax
    sahf
    jp exit ;операнды не сравнимы
    jnc m2  ; если |z|<1, то переход на m2
    jz m3   ; если |z|=1, то переход на m3
; если |z|>1, то приводим к формуле z=z1+z2,
; где z1 – целое, z2 – дробное и z2<1
    xor cx, cx ;счетчик вычитаний
m12:    inc cx
    fsub st(1),st(0)
    fcom
    fstsw ax
    sahf
    jp exit ; операнды не сравнимы
    jz m12
```

```
    jnc m2  ; если |z|<1, то переход на m2
    jmp m12 ; если |z|>1, то переход на m12
m3: mov p1,1
    jmp $+7
m2: mov p1, cx
    fxch
    f2xm1
    fadd    ; компенсируем 1
    fild p1 ; показатель степени для fscale
    fld1
    fscale
    fxch
    fincstp
    fmul
; проверка на отрицательную степень
    cmp flag,1
    jnz exit
    fld1
    fxch
    fdiv
exit:
    mov ax, 4c00h
    int 21h
main endp
end begin
```

После выполнения этой программы получим результат (*рис. 9.3*)

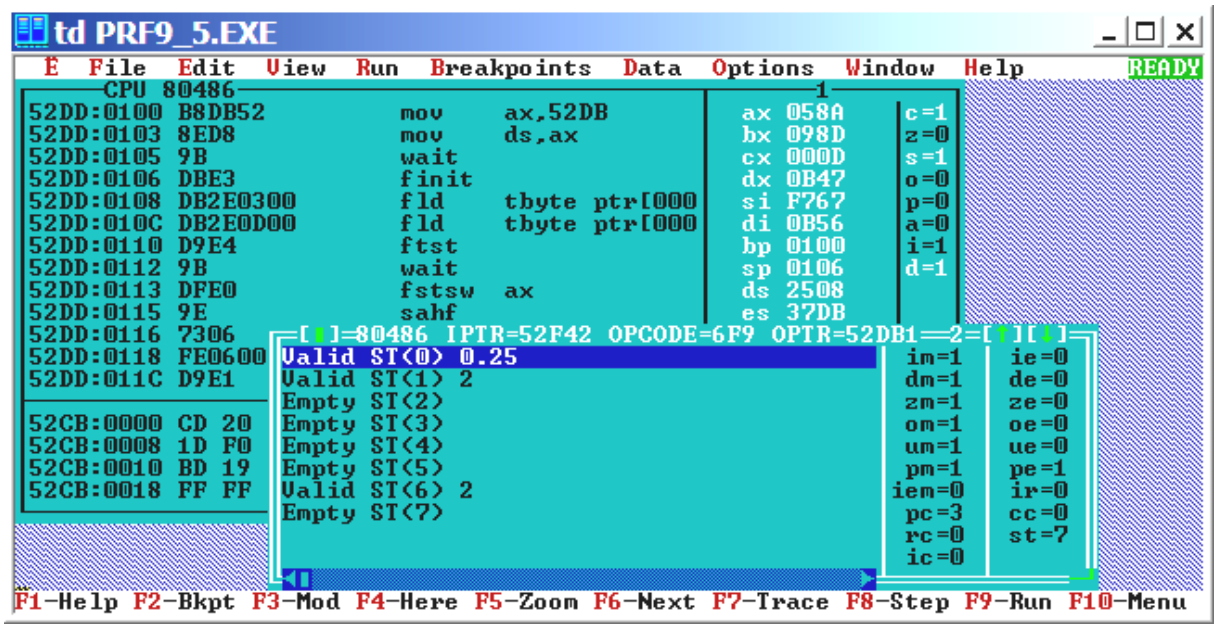


Рис. 9.3 *Результат (в Turbo Debugger) работы программы вычисления* 2^{-2}

Нетрудно видеть, что результат (в st(0)) верный: $2^{-2}=0.25$.

Практические задания к § 9

Рассчитать на Ассемблере нижеприведенные выражения. При этом результат вывести на экран.

9.1 *а)* $\dfrac{2{,}38-\sqrt{4{,}2^2+8{,}31}}{13{,}9+2{,}6\bullet 7{,}9}$ *б)* $\dfrac{1{,}79+3{,}5^{2{,}1}}{9{,}1\bullet(\sin 0{,}4+\cos 2{,}09)}$

в) $ctg(x^2+y^2)-2xy$ при x=0,01, y= – 0,1

Ответы: а) $-7{,}88\bullet 10^{-2}$ б) – 16,13 в) 99,009

9.2 *а)* $\dfrac{7{,}21-3{,}8*2{,}94}{\sqrt{4{,}1^3-7{,}4}+8{,}44}$ *б)* $\sqrt{\cos(\pi\bullet 0{,}32)}+\dfrac{9{,}43}{2{,}51}$

в) $|x|+(1-x)tg|x|$ при x= – 2,1

Ответы: а) –0,243 б) 4,4889 в) – 3,2005

9.3 *а)* $\sqrt{7{,}56+8{,}3^2}-\dfrac{6{,}25\bullet 9{,}28}{65{,}6-54{,}3}$ *б)* $\dfrac{\sin 0{,}56}{\sqrt[3]{42{,}7}}-\dfrac{6{,}5^{2{,}7}}{2{,}21}$

в) $\sqrt{4+1/t}+\sqrt[7]{1-1/t+2^t}$ при t= sin(π/2)

Ответы: а) 3,6108 б) – 70,7199 в) 4,236

9.4 *а)* $2{,}35^3-\dfrac{2{,}18+\sqrt{9{,}16}}{5{,}43}$ *б)* $\dfrac{2{,}96+\sqrt{8{,}26^2+3{,}2^2}}{16{,}7+\sin 2{,}7\bullet 0{,}43}$

в) $\sqrt{x}+\dfrac{\sqrt{x}}{\sqrt{x}-1}+\cos x$ при x= π/8

Ответы: а) 12,019 б) 0,7 в) - 0,128

9.5 *а)* $\dfrac{\sqrt{6{,}17-3{,}26}}{5{,}2\bullet 3{,}14-10{,}2}+2{,}12^2$ *б)* $\sqrt{7{,}2^3-6{,}75}+\dfrac{\sin 4{,}25}{0{,}884}$

в) $\sqrt{y+1}-(y^2+1)^2$ при y = 2/π tg (√ 3 / 2)

Ответы: а) 4,773 б) 18,1 в) – 1,1123

9.6 *а)* $\dfrac{4{,}3-\sqrt{9{,}65-2{,}81^2}}{11{,}6\bullet 0{,}32+5{,}3}$ *б)* $\dfrac{45{,}9-4{,}64}{\sin(\pi/13)}-\sqrt{arctg\,42{,}7}$

в) $99^{x/\,y-|x|}$ при x= –1,8, y = 2,01

Ответы: а) 0,33 б) 171,16 в) $4{,}1756\bullet 10^{-6}$

9.7 *а)* $\dfrac{\sqrt{29{,}3}-2{,}75^2}{16{,}5+12{,}3\bullet 0{,}65}$ *б)* $\dfrac{\sqrt{2{,}9^3-\sqrt{5{,}65}}}{\sin 0{,}14}+\dfrac{2{,}65}{3{,}17-tg\,2{,}66}$

в) $\sqrt{x^2+y^2}\bullet\sin(tgx)+\dfrac{x}{y}$ при x = – 4,87, y = 4,9

Ответы: а) $-8{,}77\bullet 10^{-2}$ б) 34,34 в) – 0,9322

9.8 *а)* $\dfrac{14{,}2^2-\sqrt{112{,}3}}{6{,}17\bullet 3{,}12-5{,}42}+3{,}7^3$ *б)* $\sin 0{,}25+\dfrac{\sin(\pi/0{,}35)}{3{,}226}$

в) $\dfrac{\sin x}{x\sqrt{x}}+\cos(x^2-1)$ при x = 4,17

Ответы: а) 64,5 б) 0,382 в) – 0,8776

9.9 *а)* $12{,}4-\dfrac{\sqrt{15{,}62-3{,}28^2}}{3{,}14-0{,}24\bullet 10{,}3}$ *б)* $\dfrac{45{,}6-ctg\,6{,}75}{2{,}65+1{,}065}-\sqrt{65{,}7}$

в) $\dfrac{1}{y(y+1)(y+2)}+tg\left(\dfrac{y}{y+1}\right)$ при $y=\sin(\pi/18)$

Ответы: а) 9,099 б) 3,64 в) 2,4065

9.10 *а)* $\sqrt{7{,}23}+\dfrac{65{,}2-43{,}8+2{,}16}{0{,}65^2+1{,}36}$ *б)* $tg\,7{,}65+\dfrac{1{,}015-0{,}27}{\sqrt[3]{65{,}9}}$

в) $\sqrt{x+1}+\dfrac{\sin x}{x-1}$ при $x=\dfrac{\pi}{12}$

Ответы: а) 15,91 б) 5,0187 в) 0,773

ЛИТЕРАТУРА

1. Л. Скэнлон. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера: Пер. с англ. – М.: Радио и связь, 1989 г. – 336 с.
2. В. Юров, С. Хорошенко Assembler: учебный курс. – СПб.: Издательство «Питер», 1999. – 672 с.
3. В. Юров Assembler: учебник. – СПб.: Издательство «Питер», 2000. – 624 с.
4. В.Н. Пильщиков Программирование на языке ассемблера IBM PC. М.: «ДИАЛОГ-МИФИ», 2003. – 288 с.

ПРИЛОЖЕНИЯ

Приложение 0. Двоичная и шестнадцатиричная системы счисления.

П 0.1 Двоичная система счисления

В повседневной жизни для записи чисел мы используем *десятичную* систему счисления. В ней, как известно, имеется десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. При этом десятичная система является *позиционной* системой, поскольку значимость цифры зависит от ее положения в записи числа.

Например, число 4728 означает:

$4728 = 4*1000 + 7*100 + 2*10 +8$ или

$4728 = 4*10^3 + 7*10^2 + 2*10^1 +8*10^0$

Можно сказать, что цифры числа (разряды) нумеруются справа налево, причем счет идет с нуля. В данном примере, 8 – нулевой разряд, 2 – первый, 7 – второй, 4 – третий.

По такому же принципу представляются и величины, имеющие дробную часть:

$472.83 = = 4*10^2 + 7*10^1 + 2*10^0 +8*10^{-1} +3*10^{-2}$

На компьютерах используется *двоичная* система счисления. Это связано с тем, что память компьютера (условно!) состоит из элементов, которые могут принимать лишь два состояния – включено (обозначим 1) или выключено (обозначим 0).

В *двоичной* системе счисления, таким образом, числа представляются с помощью комбинации единиц и нулей.

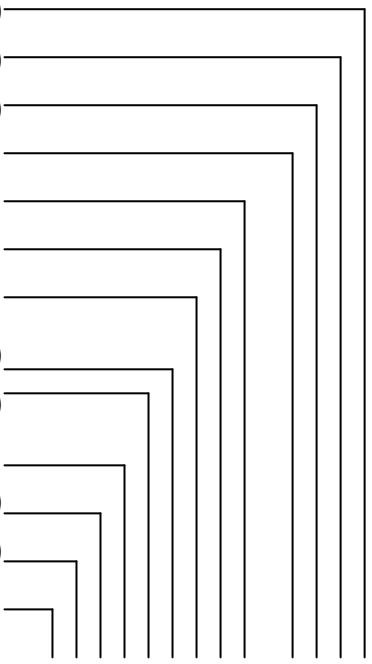
Цифры 0 и 1 в двоичной системе означают то же, что и в десятичной: $0_2 = 0_{10}$, $1_2 = 1_{10}$. И далее также используется позиционная система:

$10_2 = 1*2^1 + 0*2^0 = 2_{10}$; $11_2 = 1*2^1 + 1*2^0 = 3_{10}$; $100_2 = 1*2^2 + 0*2^1 + 0*2^0 = 4_{10}$

Но как преобразовать из десятичной в двоичную систему?

Число последовательно делим на 2 и выписываем остатки в обратном порядке. Это и даст нам двоичное представление числа.

	Частное	Остаток
4728/2	2364	0
2364/2	1182	0
1182/2	591	0
591/2	295	1
295/2	147	1
147/2	73	1
73/2	36	1
36/2	18	0
18/2	9	0
9/2	4	1
4/2	2	0
2/2	1	0
1 /2	0	1



1 0010 0111 1000

Итак, $4728_{10} = 1\ 0010\ 0111\ 1000_2$

Нетрудно проверить правильность такого перевода, учитывая, что двоичная система тоже позиционная.

$10010\ 0111\ 1000 = 1*2^{12} + 1*2^9 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 = 4096 + 512 + 64 + 32 + 16 + 8 = 4728$, т.е. все верно.

П 0.2 Шестнадцатиричная система счисления

Ясно, что работать с двоичными числами не совсем удобно. Слишком громоздкие записи, в которых легко ошибиться. Поэтому для представления чисел в компьютере используют *шестнадцатиричную* систему счисления. В состав этой системы входят 16 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Таким образом, между десятичной, двоичной и 16-ричной системами можно представить такое соответствие (см. табл. П0.1)

Таблица П0.1 Соответствие первых 16-ти чисел двоичной, десятичной и 16-ричной систем счисления.

Десятичная	Двоичная	16-ричная	Десятичная	Двоичная	16-ричная
0	0	0	8	1000	8
1	1	1	9	1001	9
2	10	2	10	1010	A
3	11	3	11	1011	B
4	100	4	12	1100	C
5	101	5	13	1101	D
6	110	6	14	1110	E
7	111	7	15	1111	F

Ясно, что переводить десятичные числа в 16-ричную систему можно также последовательным делением на 16 и выписыванием остатков в обратном порядке. Для перевода двоичных чисел в 16-ричную систему поступают проще: двоичное число разбивают на *тетрады* (группы по четыре цифры) и ставят в соответствие каждой тетраде 16-ричную цифру. Обратите внимание, что число разбивают на тетрады справа налево и впереди, если необходимо, дописывают нули (если число цифр не кратно четырем).

Например, возьмем ранее полученное число
$1\ 0010\ \ 0111\ 1000_2$. Здесь потребуется добавить впереди три нуля:
$0001\ 0010\ 0111\ 1000 = 1278_{16}$ в соответствии с вышеприведенной таблицей. Проверим правильность такого пересчета, учитывая, что 16-ричная система тоже позиционная:
$1278_{16} = 1*16^3 + 2*16^2 + 7*16^1 + 8*16^0 = 4096 + 2*256 + 7*16 + 8 = 4728_{10}$, т.е. все верно.

П0.3 Представление отрицательных чисел в двоичной и 16-ричной системах

Для представления отрицательных чисел на компьютере принято использовать так называемый *дополнительный* код записи чисел. Двоичные числа переводятся в дополнительный код заменой всех нулей на единицы, а единиц на нули и прибавлением единицы к самому младшему разряду. Например,
1 0010 0111 1000 → 0 1101 1000 0111 +1 = 0 1101 1000 1000. Это и есть обратный код числа 4728_{10}.

Обратный код можно получить и иначе – путем вычитания данного числа из ближайшего к нему большего на порядок:

```
    10 0000 0000 0000
–    1 0010 0111 1000
     0 1101 1000 1000
```

т.е. полученный дополнительный код *дополняет* данное число до ближайшего числа следующего порядка.

Заметим, что дополнительный код положительного числа – само это число.

Введение понятия дополнительного кода обусловлено, в частности, необходимостью единообразия выполнения арифметических операций в компьютере (компьютер, как известно, выполняет арифметические действия в двоичной системе). Например, вычислим сумму

```
   –9        (доп. код)  0111
+   5                    +0101
   –4                     1100 → 0100 (в прямом коде),
```

а 0100 – это и есть 4, т.е. верно

Если же вычислить сумму

```
    9        1001
+   5       +0101
   14        1110, а это и есть 14.
```

В *16-ричной* системе также используется понятие дополнительного кода при записи отрицательных чисел. Соответственно, чтобы получить модуль такого числа достаточно вычесть его из

ближайшего большего на порядок. Например, имеем число $FFE8_{16}$.

$$\begin{array}{r} 1\,0000 \\ -FFE8 \\ \hline 18 \end{array}$$

т.е. это число $-18_{16} = -24_{10}$.

Замечание. Проверить такой пересчет можно с помощью инженерного калькулятора Windows. Если в десятичной системе (кнопка Dec) задать число –24 и произвести переход в 16-ричную систему (кнопка Hex) при двухбайтовом представлении, то и получим число FFE8.

Приложение 1. Far Manager

Оболочка FAR Manager (как и подобная ей Norton Commander) появилась для удобства работы пользователя в операционной системе MS DOS, поскольку позволяла облегчить пользователю выполнение файловых операций. Однако использование FAR оказалось столь удобным, что большинство пользователей продолжает применять эту оболочку даже в системе Windows.

Для запуска FAR из-под Windows, как правило, следует дать команду ***Пуск/Программы/FAR Manager/ Manager FAR***. Экран при этом приобретет примерно такой вид — см. *рис.П.1*.

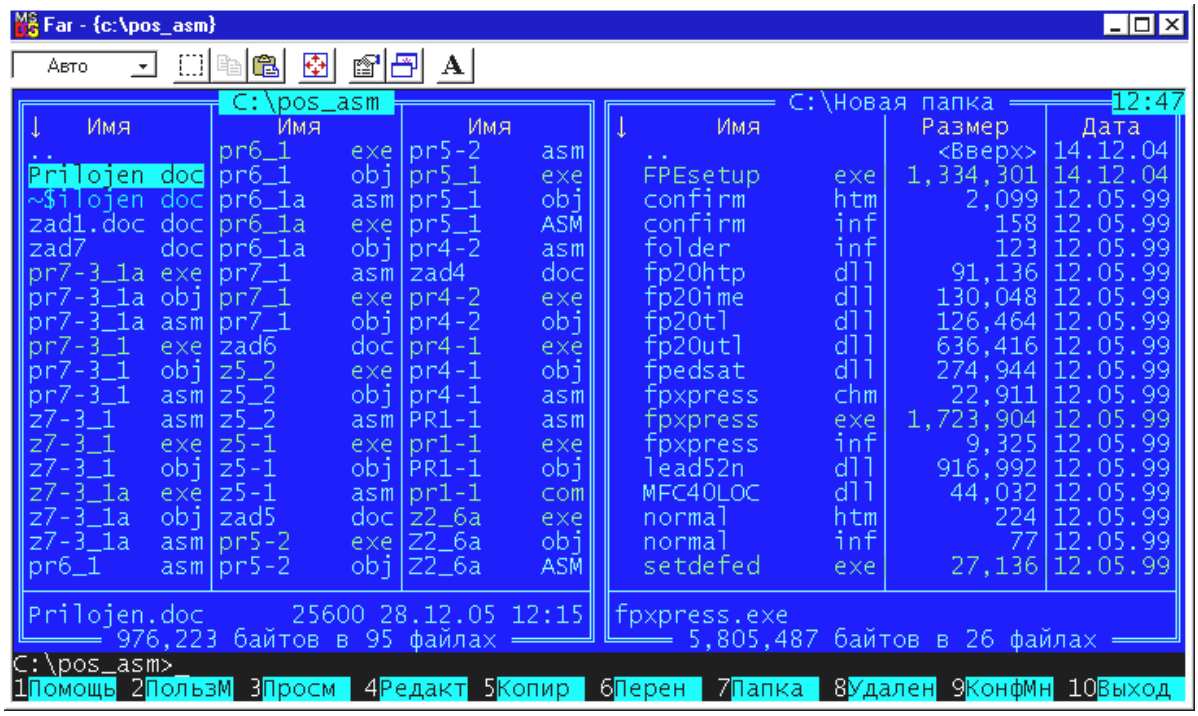


Рис. П.1 Вид окна оболочки Far

Сверху находится *строка меню* (если ее нет, то ее можно вызвать клавишей **F9**). При выходе на каждый из пунктов (клавишей **F9** или щелчком курсора мыши) появляются, как обычно, подпункты, при выборе которых выполняется определенные операции.

Ниже находятся две панели. Они нужны, прежде всего, для выполнения наиболее ходовых файловых операций – копирования и переноса файлов. (Вспомним, что в Проводнике Windows мы имели в действительности одну панель с файлами, поэтому копирование и перенос файлов там был не столь простым делом.). Обратим внимание, что информация на панелях (список файлов и каталогов) может быть представлена как в *подробном* виде (см. на рис. правую панель), так и в *кратком* (см. на рис. левую панель).

Сразу под панелями находится *строка состояния*, в которой указаны характеристики активного файла (или каталога) – его дата создания и размер (для файлов). Ниже располагается *командная строка*, в которой указано, какой каталог (папка) является активным. Здесь, в частности, можно задать команду MS DOS. И еще ниже – *строка назначений функциональных клавишей* Far.

Рассмотрим подробнее это назначение функциональных клавишей. Заметим, что 1 – означает клавишу F1, 2 – F2 и т.д.

1- Помощь Получение помощи по работе в Far.

2- ПользМ. Это вызов (или редактирование) своего меню операций. Для этого требуется знать команды MS DOS.

3- Просм. Позволяет просмотреть содержимое активного файла. Обратим внимание, что это имеет смысл делать только для текстовых файлов.

4- Редакт. Позволяет отредактировать активный файл. Опять же это можно делать только с текстовыми файлами.

5- Копир. Копирование выбранного файла (или группы файлов) на новое место – в другой каталог или на другой диск.

6- Перен. Перенос выбранного файла (или группы файлов) на новое место – в другой каталог или на другой диск.

7- Папка. Создание нового каталога внутри активного каталога или диска.

8- Удален. Удаление выбранного файла (или группы файлов).

9- КонфМн. Выход в меню для выполнения операций в Far.

10- Выход. Выход из Far.

Управление панелями в Far

Будем называть панель, на которой находится курсорный прямоугольник, активной панелью. Сделать активной ту или иную панель нетрудно — достаточно нажимать клавишу Tab (или просто щелкать курсором мыши в нужной панели). Управление информацией на панелях в Far также не вызовет затруднений. Например,

- для входа в нужный каталог достаточно установить на него курсорную рамку и нажать клавишу Enter (или дважды щелкнуть мышью по его имени);
- для выхода из каталога — стать на две точки вверху каталога и нажать Enter (или дважды щелкнуть курсором мыши по этим двум точкам);
- для запуска на исполнение файла-программы достаточно установить на него курсорную рамку и нажать Enter (или дважды щелкнуть курсором мыши по этому файлу).

Для изменения информации на панелях служат пункты меню ***Левая*** (или ***Правая***)
Подпункты в этих пунктах, в основном, не вызывают затруднений.

Отметим основные комбинации клавишей для управления панелями.

Alt – F1 – смена диска на левой панели,
Alt – F2 – смена диска на правой панели.
Ctrl – F1 – скрыть/восстановить левую панель,
Ctrl – F2 – скрыть/восстановить правую панель,
Ctrl – O – скрыть/восстановить обе панели

Файловые операции в Far

Для *просмотра* или *редактирования* файла в Far служат соответственно клавиши **F3** и **F4**.

Копирование и *перенос* (а также *удаление*) одного файла в Far также не вызывает затруднений. Для этого достаточно:

- установить нужные каталоги на панелях (на одной панели – *откуда* и на другой панели – *куда* копировать или переносить)
- установить курсор на нужный файл,
- дать соответствующую команду:
  - для копирования – нажать **F5**
  - для переноса – **F6**
  - для удаления – **F8**

В каждом случае появится дополнительный запрос для подтверждения операции. В частности, для копирования и переноса можно будет изменить место, куда хотим скопировать (перенести).

Но значительно эффективнее выполнение файловых операций для выделенной группы файлов. Для *выделения группы файлов* в Far имеются специальные возможности:

- Нажать клавишу «серый плюс» (на правой части клавиатуры) и ввести соответствующий шаблон и л и
- Дать команду **F9/*Файлы/Пометить группу*** и также ввести нужный для группы шаблон.

Например, для выделения всех файлов текущего каталога достаточно задать шаблон ***.***. Для выделения, например, всех файлов с расширением DOC следует ввести шаблон ***.doc***. Если же хотим выделить все файлы, имя которых начинается на букву D, то следует ввести шаблон **D*.*** и т.д.

Затем с выделенной группой также можно проделать операции:
копирования — клавишей **F5**
переноса — клавишей **F6**
удаления — клавишей **F8**
Если же требуется снять выделение с группы файлов, то также можно поступить двояко:

- Нажать клавишу «серый минус» (на правой части клавиатуры) и ввести соответствующий шаблон и л и
- Дать команду **F9/*Файлы/Снять пометку*** и также ввести нужный для группы шаблон.

Возможно выделение файлов и "вразброс". Это также можно делать двояко: становясь на нужный файл курсорной рамкой и щелкать клавишей Insert (INS).

Создание нового каталога (папки) в Far делается при помощи клавиши **F7**. Следует стать в нужном каталоге, нажать **F7**, в возникшем окне диалога ввести название нового каталога и нажать ОК. Далее этот каталог можно использовать как обычно: копировать (переносить) в него файлы, удалять оттуда файлы, создавать в нем подкаталоги и т.п.

Управление атрибутами файлов и настройками в Far

Файлы в ОС Windows могут иметь специальные атрибуты:

- *скрытый* (при определенных настройках экрана — невидимый в списке на экране)
- *системный* (файл, входящий в состав ОС),
- *только для чтения* (доступный для просмотра, но не для исправлений),
- *архивный* (хранение файла в определенном, сжатом виде — см. п. 8.5).

Установка файлу тех или иных атрибутов позволит уменьшить риск, связанный с удалением или порчей этого файла.

Задание атрибутов делается так:

- выделить нужные файлы (те, у которых хотим сменить атрибуты),
- дать команду **F9/*Файлы/Атрибуты файла***
- в возникшем окне указать нужный атрибут (поставить "крестик" клавишей Пробел в нужном месте),
- закрыть окно диалога (щелкнуть по кнопке Установить).

Обратная операция — *снятие* того или иного *атрибута* — делается аналогично, но отличие в том, что требуется снять "крестик" в нужном месте.

Когда установлены специальные атрибуты, в частности, можно сделать, чтобы файлы с атрибутом "скрытый" были *не видны* на экране. Для этого следует:

- дать команду **F9/*Параметры/Конфигурация***, появится окно диалога
- в блоке "Панель" *снять* "птичку" у "Скрытые файлы" (если она там стояла),
- щелкнуть ОК

Если потребуется вновь сделать скрытые файлы видимыми, то поставить упомянутую "птичку" у слов "Скрытые файлы" во вкладке Экран.

Заметим, что с помощью той же команды ***Параметры/ Конфигурация*** можно сделать некоторые дополнительные настройки FAR.

Приложение 2. Отладчик DEBUG

На начальном этапе программирования на Ассемблере рекомендуется активно использовать *отладчик DEBUG* (специальная программа, входящая в состав MS DOS).

Для его запуска достаточно в командной строке MS DOS или Norton Commander (или FAR Manager) набрать DEBUG и нажать Enter. Сразу же появляется приглашение отладчика к работе – знак дефис “–“. Здесь можно вводить команды отладчику. Приведем основные команды отладчика в виде таблицы:

Команда	Действие	Примечание
D адрес	Изобразить содержимое ячеек памяти	
E адрес	Изменить содержимое ячеек памяти, начиная с указанного адреса	
G [адрес]	Исполнить программу. Значения адреса задают точки остановки с выводом содержимого регистров	См. ниже команду T
Q	Выйти из отладчика и вернуться в DOS	
R [имя регистра]	Изобразить содержимое одного или всех регистров	Если изображено содержимое одного регистра, то R позволяет изменить его
T [число команд]	Исполнить заданное число команд и изобразить содержимое регистров на каждом шаге	См. выше команду G
U [адрес]	Преобразовать содержимое ячейки памяти в команду на языке ассемблера	

Примечание: В квадратных скобках заключены необязательные элементы команд.

С помощью DEBUG можно просматривать (проводить *дизассемблирование*) исполняемые файлы (типа COM или EXE), для этого достаточно вместе с вызовом DEBUG указать сразу и имя файла. Например, для просмотра файла PR1-1.EXE дают команду:

DEBUG PR1-1.EXE

и далее, после появления приглашающего дефиса, командой U просматривают файл и т.д.

Приложение 3. Кодировка символов

В компьютере данные хранятся в двоичном виде. Для этого каждому символу ставится в соответствие некоторое неотрицательное число, называемое кодом символа, и это число записывается в память в двоичном виде. Конкретное соответствие между символами и их кодами называется системой кодировки.

Как правило, используются 8-разрядные коды символов. Это позволяет закодировать 256 различных символов, чего вполне достаточно для представления символов, используемых на практике. Поэтому для кода символа достаточно выделить один байт. В связи с этим коды символом принято записывать в 16-ричной системе счисления.

В ПК обычно используется кодировка ASCII (American Standart Code for Information Interchange – американский стандартный код для обмена информацией). Конечно, в ней не предусмотрены коды для букв русского алфавита, поэтому в нашей стране используются варианты этой системы кодировки, в которые включают буквы русского алфавита. Чаще всего, пожалуй, используется вариант, известный под названием «Альтернативная кодировка ГОСТ». Отметим основные особенности этой кодировки.

- Код пробела меньше кода любой буквы и цифры и вообще меньше кода любого графически представимого символа.
- Коды цифр упорядочены по возрастанию и идут без пропусков. Поэтому из неравенства код(‘0’)<=код(C)<=код(‘9’) следует, что C – цифра, и поэтому справедливо равенство код(i)= код(‘0’)+i , где i – число от 0 до 9. Отметим также, что код(‘0’)<>0.
- Коды больших латинских букв упорядочены согласно алфавиту и также идут без пропусков.
- То же самое верно и для малых латинских букв.
- В альтернативной кодировке ГОСТ коды русских букв (больших и малых) упорядочены согласно алфавиту, но если коды больших букв идут без пропусков, то между кодами малых букв ‘п’ и ‘р’ вклиниваются коды иных символов.

Сведем все это в виде таблицы (см. ниже). В ней для получения кода (16-ричного!) символа нужно брать номер строки и номер столбца. Например, латинская буква ‘N’ имеет код 4E, а русская буква ‘б’ имеет код A1.

Таблица кодов ASCII (альтернативная)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	
1	►	◄							↑	↓	→	←		↔	▲	▼
2		!	“	#	$	%	&	‘	(	)	*	+	,	–	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
9	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
A	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
B	▒	▒	▓		┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
C		┴	┬	├	─	┼	╞	╟	╚	╔						
D																
E	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
F	Ё	ё														

В пропущенных клетках находятся малоупотребительные символы и здесь не приводятся.