Федеральное агентство по образованию Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

А.А. Шелупанов, В.Н. Кирнос

ИНФОРМАТИКА. БАЗОВЫЙ КУРС

Учебник
(в четырех частях)

Часть 1. Общие вопросы информатики и программирование на Ассемблере

Допущено Сибирским региональным отделением Учебно-методического объединения Вузов по образованию в области информационной безопасности для межвузовского использования в качестве учебного пособия по специальности 090105 «Комплексное обеспечение информационной безопасности автоматизированных систем»

Томск – 2007

УДК 681.3

Данное издание является первой частью учебника «Информатика. Базовый курс» указанных авторов. В данной части учебника рассматриваются общие вопросы информатики, краткие основы организации компьютера, порядок работы в операционной системе MS DOS, основы программирования на языке Ассемблера.

Для студентов специальности 090105, а также других специальностей, связанных с информационной безопасностью.

ISBN

СОДЕРЖАНИЕ

Глава 1. ОБЩИЕ ВОПРОСЫ ИНФОРМАТИКИ.

1. ИНФОРМАЦИЯ, ЗНАНИЯ И НАУКА «ИНФОРМАТИКА»

Понятие «информация» относится к числу фундаментальных, подобных понятиям «время» или «энергия», для которых не существует строго научного определения. Тем не менее, мы рассмотрим некоторые описательные определения понятия «Информация».

Информация – это совокупность сигналов, воспринимаемых нашим сознанием, которые отражают те или иные свойства объектов и явлений окружающей действительности. Природа данных сигналов подразумевает наличие принципиальных возможностей по их сохранению, передаче и трансформации (обработке).

Подчеркнем, что это не строго научное, а пояснительное определение. Другой подход к определению информации – *это представление о её передаче от источника к приемнику; при их взаимодействии и возникает информация*. Клод Шеннон (1916 – 2001), один из основоположников теории информации, определил информацию как *снятую неопределенность*. Есть и такое простое определение: *информацией называются любые данные или знания об окружающем нас мире*.

Как видим, имеются разные подходы к определению понятия «информация».

Существуют различные формы представления информации:

- символьно-текстовая,
- графическая,
- звуковая.

В одном ряду с понятием информации стоит понятие «знания». *Знания – это информация, на основе которой путем логических рассуждений могут быть получены определенные выводы.*

Наука «Информатика», к изучению которой мы приступаем, и сложилась, как наука по изучению законов работы с информацией и знанием. Существуют два таких наиболее популярных определения для этой науки.

Определение 1. *Информатика – это наука об описании, представлении, интерпретации, формализации и применении знаний, накопленных с помощью вычислительной техники, с целью получения новых знаний.*

Определение 2. *Информатика – это наука, которая изучает общие законы хранения, обработки и передачи информации с помощью компьютера.*

На протяжении всей своей истории человечество, так или иначе, занималось хранением, обработкой и передачей информации. На этом пути можно выделить такие этапы.

Речь. Только появление устной речи сделало возможным обработку информации человеком. Однако общение посредством устной речи не всегда является надежным – информация искажается, легко теряется, передача возможна только при непосредственном общении. Но, однако, устная речь играет немалую роль в обществе. Вспомним, что устные объяснения преподавателя часто ценнее (и понятнее) написанного в учебнике.

Письменность. Появление письменности позволило решить очень важную проблему – надежность хранения информации и передачу ее во времени. Так, благодаря тому, что в Древнем Египте существовала письменность, мы знаем о его истории достаточно много. А, например, культура древних народов, живших на территории современной Сибири, нам почти неизвестна, так как собственной письменности у них не было.

Книгопечатание. В 1436 г. в Майнце Иоганн Гутенберг напечатал первую в истории человечества книгу с печатных форм. С этого момента начался действительно новый этап в истории человечества. Стало возможным быстро и надежно распространять информацию. Благодаря этому, например, появились газеты, которые до сих пор являются одним из главных способов распространения информации.

Компьютеры. С появлением в середине нашего века первых компьютеров удалось принципиально решить проблему хранения, передачи и обработки информации. Но именно в наше вре-

мя, когда компьютеры стали общедоступными и, главное, достаточно мощными, возникла современная информатика.

Современные компьютеры обладают большой емкостью для хранения информации. Стандартная дискета, к примеру, способна вместить информацию целого учебника. Быстродействие компьютеров сейчас таково, что просмотреть большой объем информации и найти, например, нужное слово в учебнике возможно за доли секунд. Современные компьютеры соединяются в сети, и это делает возможным быстро и надежно передавать информацию не только своему коллеге в этом же здании, но даже в другой город или страну.

2. СТРУКТУРА СОВРЕМЕННОЙ ИНФОРМАТИКИ

В современной информатике (в англоязычных странах науку "информатика" называют **"computer science"** – компьютерная наука) можно выделить три части: ***Устройство вычислительной техники, Программирование, Алгоритмы и теоретические методы решения задач на компьютере*** (рис. 1).

Рис. 1 Структура современной информатики.

Устройство – это аппаратура, из которой состоит компьютер. В англоязычной литературе называют **Hardware** (дословно – "железо"). ***Программирование*** (более широко – ***программное обеспечение***) – совокупность всех программ, которые используются на компьютере. Именно они "оживляют" компьютер. Чтобы уметь общаться с компьютером, надо освоить имеющееся **Software** и научиться самому составлять программы.

Алгоритмы – то, что нужно знать и уметь делать для решения той или иной задачи. Компьютер не может домыслить за человека и поэтому указания ему нужно давать строго, как говорят, алгоритмически.

3. ФУНКЦИИ КОМПЬЮТЕРА

Любой компьютер, даже, на первый взгляд, самый простой, представляет собой сложную систему, состоящую из сотен тысяч или даже миллионов простейших электронных компонентов. Описание компьютерной системы начнем с главных ее компонентов, рассмотрим их структуру и функции, а затем перейдем на следующий уровень иерархии и начнем рассматривать структуру и функции составных элементов этих главных компонентов.

Структура и функции компьютера, если рассматривать их на самом верхнем уровне абстракции, по существу довольно просты. В самом общем смысле таких функций всего четыре:

- обработка данных;
- хранение данных;
- перемещение данных;
- управление.

Компьютер, естественно, в первую очередь обязан *обрабатывать данные,* которые могут принимать самые разные формы, а диапазон выполняемых операций по их обработке также очень широк. Однако, как будет показано ниже, все разнообразие операций может быть сведено к немногим базовым типам или методам обработки.

Существенное место занимает и функция *хранения данных.* Даже если компьютер обрабатывает данные на ходу, т.е. по мере их поступления из операционной среды, причем результат также немедленно отправляется получателю, компьютер должен обладать способностью хотя бы временно хранить промежуточные результаты и фрагменты данных, которые обрабатываются в теку-

щий момент времени. Таким образом, система должна выполнять функцию хранения данных хотя бы и на короткое время. Но в большинстве случаев этого недостаточно. От компьютера чаще всего требуется выполнение функции долговременного хранения файлов данных, которые могут обрабатываться или обновляться по мере необходимости.

Компьютер должен также обладать способностью *перемещать данные,* причем в обе стороны, т.е. получать первичные данные из операционной среды и отправлять результаты обработки внешним абонентам. Среда, в которой "живет" компьютер, состоит из устройств, играющих либо роль источников данных, либо роль приемников информации. Процесс перемещения данных между компьютером и операционной средой принято называть процессом *ввода-вывода,* а устройства, входящие в состав операционной среды, — *периферийными* устройствами. Когда данные передаются на большое расстояние, т.е. выполняется обмен данными с *удаленными* устройствами, этот процесс принято называть *передачей данных.*

И, наконец, все эти три функции должны выполняться в определенной последовательности, т.е. от компьютера требуется еще и выполнение функции *управления.* В конечном счете, функция управления в основном ложится на плечи того, кто снабжает компьютер последовательностью команд — программой. В самой же компьютерной системе функция управления сводится к распределению ресурсов и "дирижированию" выполнением других функций в процессе отработки команд, заданных программой.

4. СТРУКТУРА КОМПЬЮТЕРА

Компьютер является объектом, способным некоторым образом взаимодействовать с внешней по отношению к нему средой через связи, которые можно разделить на две группы — связи с локальным периферийным оборудованием и связи для передачи данных на большое расстояние, но в основном, наше внимание будет сосредоточено на внутренней структуре компьютера.

На верхнем уровне иерархии эта структура выглядит так, как показано на рис. 2. На ней вы видите четыре основных компонента компьютера.

- *Центральный процессор* ***(ЦП)*** управляет функционированием всей системы и выполняет функции обработки информации. (Очень часто в наименовании этого компонента прилагательное "центральный" опускается.)
- *Оперативная память* хранит исходные данные и всю информацию, необходимую для их обработки.
- *Устройства ввода-вывода* перемещают данные между компьютером и окружающей средой в обе стороны.
- *Системные внутренние связи* представляют собой некоторый механизм, обеспечивающий обмен информацией между остальными компонентами — ЦП, оперативной памятью и устройствами ввода-вывода.

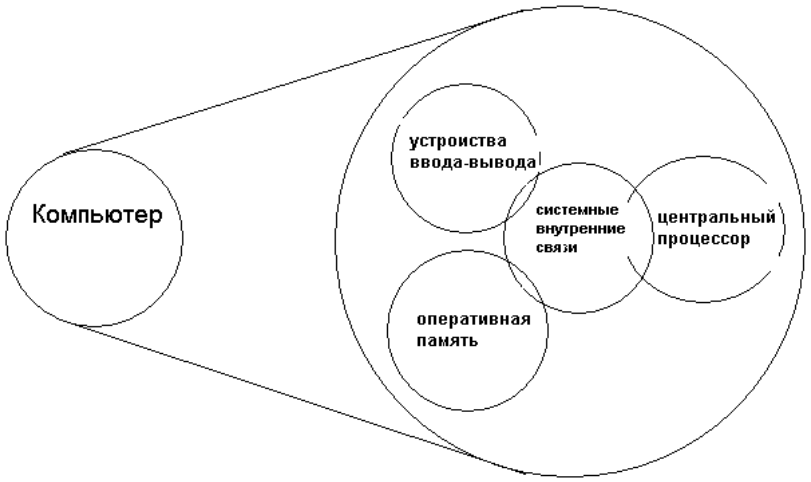


Рис.2 Верхний уровень структурной организации компьютера

В состав конкретного компьютера могут входить один или несколько компонентов каждого типа. Как правило, в компьютере имеется один ЦП, но в последние годы прослеживается тенденция включать в состав единой компьютерной системы несколько процессоров.

Каждый из означенных компонентов подробно рассмотрим позднее. Сейчас же для нас основной интерес представляет самый сложный компонент компьютера — *центральный процессор*. Его структура (это уже следующий, второй уровень иерархии) представлена на рис. 3

В состав ЦП входят:

- *устройство управления,* на которое возлагаются функции управления прочими компонентами ЦП и, следовательно, всем компьютером;
- *арифметическое и, логическое устройство (АЛУ),* которое выполняет все операции, связанные с содержательной обработкой информации;
- *регистры,* которые хранят оперативную информацию во время выполнения процессором текущей операции;
- *внутренние связи ЦП* — некоторый механизм, обеспечивающий совместную работу трех прочих компонентов ЦП.

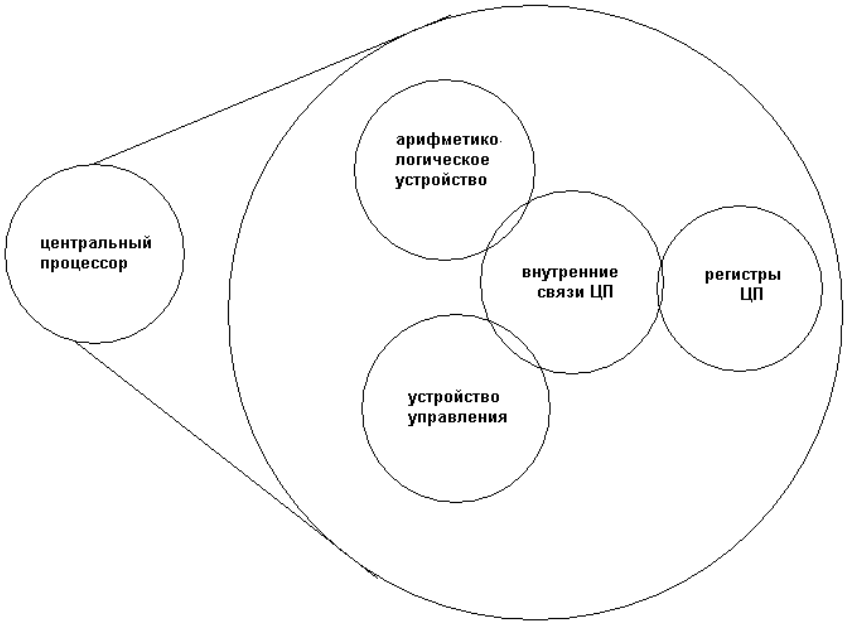


Рис.3 Структура центрального процессора

5. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ОРГАНИЗАЦИИ КОМПЬЮТЕРА

Компьютер – это электронное цифровое устройство. ***Электронное устройство*** потому, что внутри компьютера любая информация передается, хранится, обрабатывается и выводится только через изменение состояния электронов. ***Цифровое устройство*** потому, что любая информация представляется с помощью чисел.

Для того чтобы записывать числа, нужно использовать какую-нибудь систему счисления. ***Система счисления*** показывает, по каким правилам мы записываем числа и выполняем над ними действия (сложение, умножение и т.п.).

Обычно мы используем десятичную систему счисления. В этой системе любое число записывается с помощью десяти цифр: 0,1,2,3,4,5,6,7,8,9.

Количество цифр в системе счисления называется ее ***основанием***. Основание десятичной системы равно 10. Десятичная система возникла потому, что в древности люди использовали для счета десять пальцев на двух руках.

Существовали и другие системы счисления. Например, *пятиричная* с основанием 5 (пять пальцев на одной руке) или *двенадцатиричная* с основанием 12 (десять пальцев и еще две руки). Известна также *римская* система счисления, в которой, например, мы до сих пор подсчитываем столетия (например, XX век). В древнем Вавилоне люди использовали шестидесятиричную систему счисления с основанием 60. Эта система до сих пор используется нами для измерения углов и времени: 60 мин в часе и 6 раз по 60 градусов в полном круге.

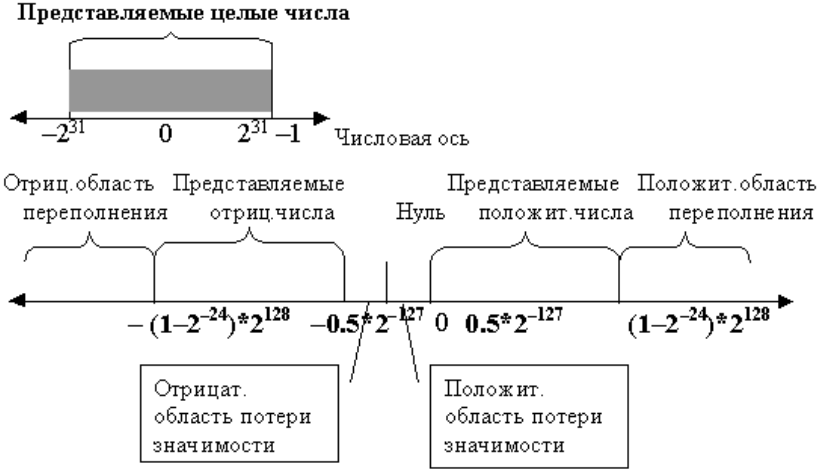
На компьютере для записи чисел используется ***двоичная система счисления***. В этой системе всего две цифры: 0 и 1, основание системы равно 2. Двоичная система для компьютера выбрана не случайно. Дело в том, что вводимая информация проходит в компьютере в виде изменения состояния электронов. Эти изменения либо есть, либо их нет. Удобно отсутствию изменения сопоставить цифру нуль (0), а наличию – цифру один (1). Для примера приведем соответствие первых пятнадцати чисел десятичной системы числам двоичной системы счисления (даны в табл. 1).

Таблица 1. Соответствие первых десятичных чисел двоичным.

Система счисления		Система счисления	
десятичная	двоичная	десятичная	двоичная
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

В проблематике компьютерной арифметики основной интерес вызывают два вопроса – способ представления чисел в двоичной системе счисления и алгоритмы выполнения над ними четырех базовых арифметических операций (сложения, вычитания, умножения и деления). Это относится как к целым числам, так и к вещественным

Вещественные числа представляются, как правило, в формате с плавающей точкой, т.е. в виде числа (мантиссы), которое умножается на основание, возведенное в целую степень (порядок). Такой формат позволяет представить на ограниченной разрядной сетке как очень малые, так и очень большие числа.



5.1 Двоичная система счисления

В повседневной жизни мы используем десятичную систему счисления. В ней, как известно, используется 10 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. С их помощью и записывают любые цифры, при этом позиция цифры в числе означает количество единиц, десяток, сотен и т.д. 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:

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

В двоичной системе счисления числа представляются с помощью комбинации единиц и нулей, знака минус и знака разделяющей точки между целой и дробной частью числа.

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:

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

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

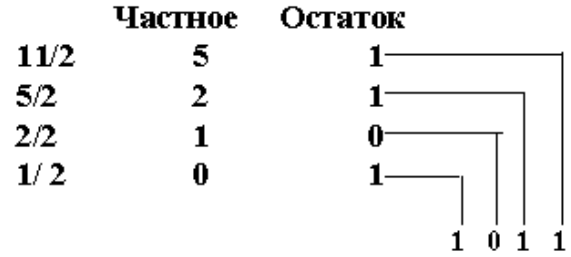
При представлении дробной части числа используются отрицательные степени двойки:

$1001.101_2 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9.625_{10}$.

Но как преобразовать из десятичной в двоичную систему?

Целая и дробная части представления преобразуются раздельно.

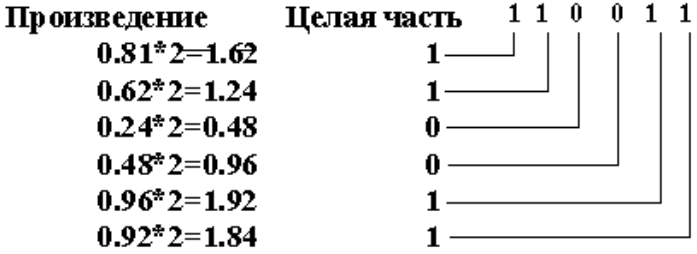
Целую часть последовательно делим на 2 и выписываем остатки в обратном порядке. Это и даст нам двоичное представление числа.



Таким образом, $11_{10} = 1011_2$.

Процедура преобразования дробей представляет собой последовательное умножение на 2. На каждом шаге дробная часть предыдущего произведения вновь умножается на 2, а цифры, представляющие целую часть произведений (они всегда равны либо 0, либо 1), выписываются как последовательные (слева направо) цифры двоичного представления.

Например, преобразуем число 0.81_{10} в двоичную систему.



Таким образом, $0.81_{10} = 0.110011_2$ (приблизительно).

Видим, что процедура перевода дробей не всегда дает точный результат. В данном случае оборвали на шестой цифре (разряде). Обычно в таких случаях заранее оговаривают, какой разрядности результат следует получить.

Например, десятичное число -1.3125_{10} в двоичном виде будет выглядеть как -1001.0101_2 . Но в компьютере мы не можем хранить и обрабатывать символы знака и разделяющей точки – для машинного представления могут использоваться только двоичные цифры (0 и 1). Если операции выполняются только с неотрицательными числами, то формат представления очевиден. В ячейке памяти из 8 разрядов можно представить число в интервале от 0 до 255:

00000000 = 0
00000001 = 1

……..
00101001 = 41
…
10000000 = 128
…
11111111 = 255

Порядок работы с отрицательными числами рассмотрим в разделе 5.3.

5.2 Шестнадцатиричная система счисления

Поскольку современная электронно-вычислительная техника принципиально построена на двоичных элементах, любые данные в ней представляются только двоичными кодами. Но пользователю применять начертание кода в виде длинных последовательностей нулей и единиц очень неудобно. Поэтому специалисты используют для работы с двоичными кодами «на бумаге» более компактное их представление. Такую компактную форму дает *шестнадцатиричная* система счисления. Для преобразования из двоичного представления в 16-ричное двоичные цифры группируются по четыре (как говорят, разбивают на *тетрады*), и каждая четырехзначная комбинация заменяется шестнадцатиричной цифрой (см. табл. 2)

Таблица 2. Соответствие первых десятичных чисел двоичным и 16-ричным.

Система счисления			Система счисления		
десятичная	двоичная	16-ричная	десятичная	двоичная	16-ричная
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

При этом разбиение целой части двоичного числа на тетрады идет справа налево (и впереди, если необходимо, дополняют крайнюю тетраду до четырех цифр нулями). Дробную же часть разбивают на тетрады справа налево (и также, несли необходимо, в конце дописывают нули).

Например, как записано выше $255_{10} = 11111111_2$. Здесь имеем две тетрады: 1111 1111. Значит, $255_{10} = 11111111_2 = FF_{16}$. Проверим правильность – переведем в десятичную систему из 16-ричной: $FF_{16} = F*16^1 + F*16^0 = 15*16 + 15*1 = 240 + 15 = 255_{10}$,т.е. все верно.

Другой пример: 10 1111 1000 0101 1001. Видно, что впереди необходимо добавить два нуля: 0010 1111 1000 0101 1001.

2 F 8 5 9

Тогда имеем: $10\ 1111\ 1000\ 0101\ 1001_2 = 2F859_{16}$. Проверим, пересчитав в 10-й системе:

$10\ 1111\ 1000\ 0101\ 1001_2 = 2^{17}+2^{15}+2^{14}+2^{13}+2^{12}+2^{11}+2^6+2^4+2^3+1 = 194649_{10}$.

$2F859_{16} = 2*16^4 + F*16^3 + 8*16^2 + 5*16^1 + 9 = 2*65536+15*4096+8*256+89 = 194649_{10}$.

Видим, что все верно.

Рассмотрим пример числа с дробной частью.

10011110001,11001100001. Здесь необходимо приписать нуль впереди целой части и нуль позади дробной части (для получения тетрад):

0100 1111 0001,1100 1100 0010. В результате имеем $4F1,CC2_{16}$.

Ясно, что перевести целое число из 10-й системы в 16-ричную можно также последовательным делением десятичного числа на 16 и выписыванием остатков в обратном порядке (уже в 16-ричном виде). Дробь же переводится последовательным умножением на 16 и выписыванием целых частей произведений в правильном порядке (т.е. все так же, как и для двоичных чисел).

Практические задания «Системы счисления»

Зад.1 Получить десятичный эквивалент следующих двоичных чисел
Зад.2 Получить десятичный эквивалент следующих 16-ричных чисел
Зад.3 Перевести числа из десятичной системы в двоичную
Зад.4 Перевести числа из десятичной системы в 16-ричную
Зад.5 Перевести числа из двоичной системы в шестнадцатиричную

При выполнении заданий 3 – 5 следует проводить проверку с использованием десятичной системы счисления.

Вариант	Задание 1	Задание 2	Задание 3	Задание 4	Задание 5
1	1001010	15B	43	7501	111011110011,10011
2	110010011	CD1	24	3765	0,110111001
3	111000	16	416	908	10011110001,11001100001
4	111011110011,10011	4AD	253,2	9854	1001010
5	101111100001011001	0,81	794,15	77659	110010011
6	0,110111001	998C	6,4	4591	111000
7	10011110001,11001100001	FF77	439	692,54	11001100001
8	110111001	DA0774	8419	7428,41	10011110001
9	10011110001	B4AA	692,54	196,7591	111011110011
10	11001100001	FF9A3	7428,41	794,15	101111100001011001

5.3 Арифметические операции в двоичной системе счисления

5.3.1 Прямой код

Существует несколько соглашений о едином формате представления как положительных, так и отрицательных чисел. Все их объединяет то, что старший разряд (самый левый) является знаковым разрядом (0 – это плюс, 1 – это минус). Все последующие биты слова составляют значащие разряды числа.

Простейшим форматом, который использует знаковый разряд является *прямой код*. Например, при 8-разрядном представлении

+18 = 00010010
–18 = 10010010

Однако, прямой код неудобен при использовании в вычислениях. Во-первых, сложение и вычитание положительных и отрицатель-

ных чисел выполняется по-разному, а потому требуется анализировать знаковые разряды операндов (слагаемых). Во-вторых, в прямом коде числу 0 соответствует две комбинации:

+0 = 00000000

–0 = 10000000

Это также неудобно, поскольку усложняется анализ результата на равенство нулю, а такая операция в программах встречается часто.

По этим причинам прямой код практически не используется при реализации в АЛУ современных компьютеров. Вместо него применяют так называемый дополнительный код.

5.3.2 Обратный и дополнительный код

В обратном, как и в прямом коде, старший разряд отводится для представления знака числа, но остальные разряды интерпретируются в зависимости от того, положительное число или отрицательное.

Обратный код положительного числа остается неизменным. А обратный код отрицательного двоичного числа формируется дополнением модуля исходного числа нулями до самого старшего разряда модуля, а затем поразрядной заменой всех нулей числа на единицу и всех единиц на нули. В знаковом разряде обратного кода у положительных чисел будет 0, а у отрицательных – 1. В общем случае обратный код является дополнением модуля исходного числа до наибольшего числа без знака, помещенного в разрядную сетку.

Например, при 8-разрядном представлении числа $+28_{10}$ имеем:

$+28_{10} = 00011100_2$

Если же возьмем -28_{10} , то в обратном коде имеем:

$-28_{10} = 11100011_2$

Однако на основе обратного кода строится так называемый *дополнительный код*, который позволяет существенно облегчить арифметические операции над числами с разными знаками.

Дополнительный код строится следующим образом. Сначала формируется обратный код, а затем к младшему разряду добавляют 1. При выполнении арифметических операций положительные числа представляются в прямом коде, а отрицательные числа – в дополнительном.

Например, -5_{10} преобразуем в дополнительный код.
5 в двоичном виде дает (при 4-разрядном двоичном представлении) 0101, тогда –5 даст 1101 (впереди 1 – знак минус). Получаем обратный код: 1010, затем добавляем в младший разряд единицу: 1011 – это и есть дополнительный код –5. Итак, $-5_{10} = 1011_2$

Итак, рассмотрим n-разрядное двоичное число А в дополнительном коде. Если заданное число А *положительное,* то в значащих разрядах представляется абсолютная величина числа точно так же, как и в прямом коде:

$$A = \sum_{i=0}^{n-2} 2^i a_i$$

Если же число *отрицательное*, то для него соблюдается соотношение

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Если рассмотреть конкретно 4-разрядные целые числа, то представление их в прямом и дополнительном коде можно записать в виде таблицы 3:

Таблица 3. Представление чисел в прямом и дополнительном коде.

Десятичное представление	Прямой код	Дополнительный код
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
–0	1000	–
–1	1001	1111
–2	1010	1110
–3	1011	1101

–4	1100	1100
–5	1101	1011
–6	1110	1010
–7	1111	1001
–8	–	1000

Видим, что в дополнительном коде, по крайней мере, исчезла проблема двух нулей.

Иногда возникает необходимость записать число с большим числом разрядов. Если число представлено в прямом коде, то достаточно перенести знаковый разряд в крайний левый бит нового представления числа, а остальные дополнительные биты заполнить нулями.

Например,

+18 в прямом коде на 8 разрядов 00010010
+18 в прямом коде на 16 разрядов 00000000 00010010
–18 в прямом коде на 8 разрядов 10010010
–18 в прямом коде на 16 разрядов 10000000 00010010

Но с отрицательными числами в дополнительном коде такая схема не дает правильного результата. Преобразование дополнительного кода при расширении разрядной сетки выполняется следующим образом: нужно скопировать значение знакового разряда во все дополнительные биты. Если исходное число было положительным, то все дополнительные биты заполнятся нулями, а если отрицательным – единицами. Эта операция называется расширением знака. Применим это правило к тому же примеру:

+18 в дополнительном коде на 8 разрядов 00010010
+18 в дополнительном коде на 16 разрядов 00000000 00010010
–18 в дополнительном коде на 8 разрядов 11101110
–18 в дополнительном коде на 16 разрядов 11111111 11101110

Формально справедливость этого правила доказывается так. Рассмотрим *n*-разрядную последовательность двоичных цифр $a_{n-1}\, a_{n-2}\ldots a_1\, a_0$, которая интерпретируется как представление в дополнительном коде числа *A*:

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Сразу видно, что если число *A* положительно, правило справедливо. Если е число А отрицательно, нужно сформировать m-разрядное его представление (m>n), такое, что

$$A = -2^{m-1} a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i$$

Поскольку значения обоих представлений должны быть равны, то

$$-2^{m-1} a_{m-1} + \sum_{i=0}^{m-2} 2^i a_i = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Т.к. число отрицательное, крайние левые разряды равны единице

$$a_{m-1} = a_{n-1} = 1$$

Тогда

$$-2^{m-1} + \sum_{i=0}^{m-2} 2^i a_i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$-2^{m-1} + \sum_{i=n-1}^{m-2} 2^i a_i = -2^{n-1} \text{ или } 2^{n-1} + \sum_{i=n-1}^{m-2} 2^i a_i = 2^{m-1}$$

$$1 + \sum_{i=0}^{n-2} 2^i + \sum_{i=n-1}^{m-2} 2^i a_i = 1 + \sum_{i=0}^{m-2} 2^i$$

$$\sum_{i=n-1}^{m-2} 2^i a_i = \sum_{i=n-1}^{m-2} 2^i$$

Отсюда следует, что

$$a_{m-1} = a_{m-2} = \ldots = a_n = a_{n-1} = 1$$

В каждом из приведенных соотношений соблюдается условие неизменности младших *n–1* разрядов представления. Последнее соотношение справедливо только в том случае, когда коды во всех разрядах от *n–1* до *m–2* равны 1. Тем самым подтверждается справедливость сформулированного выше правила расширения знака.

5.3.3 Арифметические операции с целыми числами

Рассмотрим алгоритмы выполнения основных арифметических операций над целыми числами, представленными в дополнительном коде.

Сложение и вычитание

Рассмотрим эти операции на примерах.

а) –7 = 1001 б) –4 = 1100
+5 = + 0101 +4 = + 0100
1110 = –2 10000 = 0

в) +3 = 0011 г) –4 = 1100
+4 = + 0100 –1 = + 1111
0111 = 7 11011 = –5

д) +5 = 0101 е) –7 = 1001
+4 = + 0100 –6 = + 1010
1001 переполнение 10011 переполнение

Первые четыре примера демонстрируют успешное выполнение операций. Если результат операции должен быть положительным, получается код положительного числа в дополнительном коде, а если отрицательным – код отрицательного числа в дополнительном коде (сравни со значениями в Таблице 3). Обратим внимание, что в примере г) формируется перенос из старшего (знакового) разряда, который игнорируется.

При выполнении сложения чисел с одинаковыми знаками результат может оказаться таким, что не вмещается в используемую разрядную сетку, т.е. получается число, которое выходит за диапазон представления – происходит *переполнение*. В этом случае на схему АЛУ возлагается функция выявить переполнение и выработать сигнал, который должен воспрепятствовать использованию в дальнейшем полученного ошибочного результата. Существует следующее правило обнаружения переполнения:

> Если знаки слагаемых совпадают, то переполнение возникает в том и только в том случае, когда знак суммы, полученной по правилам сложения в дополнительном коде, отличается от знака слагаемых.

Примеры д) и е) иллюстрируют появление переполнения при сложении положительных и отрицательных чисел. Обратим внимание, что переполнение может проявиться и в том случае, когда возникает перенос из знакового разряда и когда перенос не возникает.

Рассмотрим операцию *вычитания*. Она выполняется по следующему правилу:

Для вычитания одного числа (*вычитаемого*) из другого (*уменьшаемого*) необходимо предварительно получить дополнительный код для уменьшаемого, а затем сложить результат с уменьшаемым по правилам сложения в дополнительном коде.

Проиллюстрируем также на примерах. В них символом M обозначено уменьшаемое, а символом S – вычитаемое.

а) M= 2 = 0010 б) M= 5 = 0101
S = 7 = 0111 S = 2 = 0010
–S = 1001 –S = 1110

0010 0101
+ 1001 + 1110
1011 = –5 10011 = 3

в) M= –5 = 1011 г) M= 5 = 0101
S = 2 = 0010 S= –2 = 1110
–S = 1110 –S = 0010

1011 0101
+ 1110 + 0010
11001 = –7 0111 = 7

д) M= 7 = 0111 е) M= –6 = 1010
S = –7 = 1001 S = 4 = 0100
–S = 0111 –S = 1100

0111 1010
+ 0111 + 1100
1110 переполнение 10110 переполнение

Последние два примера иллюстрируют возможность появления переполнения при вычитании.

5.3.4 Арифметические операции с вещественными числами

В формате короткого вещественного представляются числа с плавающей точкой, старший разряд является знаковым, один байт отводится под смещенный порядок (смещение $N=2^7 = 128$ прибавляется к порядку числа, что упрощает действия над поряд-

ками), а двоичная мантисса занимает 23 разряда; числа в памяти хранятся в нормализованном виде, т.е. в старшем порядке мантиссы записана 1:

знак	Смещенный порядок	Двоичная мантисса
31	30 23	22 0

Практические задания «Арифметические операции в двоичной системе»

Ниже предложены варианты заданий (см. табл.). Каждое задание состоит из трех этапов.

А) Числа X и Y представлены в форме короткого целого. Вычислить X+Y

Б) Вычислить X – Y

В) Числа A и B в формате короткого вещественного слова. Выполнить операцию A+B

Номер варианта	Числа			
	X	Y	A	B
1	10236	-18758	724,71	-10,83
2	-3876	14932	116,03	-494,4
3	-19392	24076	-376,47	21,001
4	-13704	-23800	-19,865	119,1
5	-21005	12037	-275,5	24,75
6	876	-14731	325,11	-36,55
7	-734	18005	610,44	-3,175
8	16538	-11010	327,93	-1,172
9	-12709	25068	505,4	-26,43
10	-10736	19805	99,031	-17,666

Методические рекомендации.

В формате короткого целого число состоит из 32 двоичных разрядов, их нумерация начинается с нуля, тогда крайний слева (31-й разряд) является знаковым (0 – плюс, 1 – минус).

Как показано ранее, для перевода десятичного числа в двоичную систему счисления целесообразно вначале перевести число в 16-ричную систему (методом деления), а затем в двоичную. Это упрощает процедуру перевода, так как для перевода из 16-ричной системы в двоичную достаточно записать каждую цифру 16-ричного числа в виде соответствующей двоичной тетрады (четырехзначного двоичного числа). Кроме того, необходимо помнить, что положительные числа хранятся в памяти компьютера в прямом коде, а отрицательные – в дополнительном.

Этапы А и Б.

Пример. *Показать изображение чисел X=18730 и Y=–16273 в формате короткого целого и выполнить над ними действия X+Y и X–Y*

$X = (18730)_{10} = (492A)_{16} = (100\ 1001\ 0010\ 1010)_2$

$Y = (-16273)_{10} = (-3F91)_{16} = (-11\ 1111\ 1001\ 0001)_2$

Представим эти же числа в виде двоичных слов (заданного формата):

X_2 =0000 0000 0000 0000 0100 1001 0010 1010 – прямой код

Y_2= 1111 1111 1111 1111 1100 0000 0110 1111 – дополнительный код

При выполнении операций сложения операнды складываются в тех кодах, в которых они хранятся в памяти:

X = 0000 0000 0000 0000 0100 1001 0010 1010

\+

Y = 1111 1111 1111 1111 1100 0000 0110 1111

X+Y = 0000 0000 0000 0000 0000 1001 1001 1001 или 1001 1001 1001

Сделаем проверку:

$(X+Y)_{10} = (18730)_{10} + (-16273)_{10} = (2457)_{10}$

А мы получили в двоичном виде: $X+Y = (1001\ 1001\ 1001)_2$. Выполним перевод в десятичную систему счисления:

$X+Y = 1*2^{11} + 1*2^8 + 1*2^7 + 1*2^4 + 1*2^3 + 1*2^0 = 2048 + 256 + 128 + 16 + 8 + 1 = 2457$. Т.е. все верно.

При выполнении операций вычитания знак второго операнда меняется на противоположный. Для этого необходимо инвертировать все разряды вычитаемого и к младшему разряду прибавить 1, после чего производить сложение операндов.

Код вычитаемого Y после указанного действия:

–Y = 0000 0000 0000 0000 0011 1111 1001 0001

Тогда

X = 0000 0000 0000 0000 0100 1001 0010 1010

\+

–Y = 0000 0000 0000 0000 0011 1111 1001 0001

X – Y = 0000 0000 0000 0000 1000 1000 1011 1011 или

Результат получен в прямом коде: $(X - Y)_2 = (1000\ 1000\ 1011\ 1011)_2$

Проверка:

$(X - Y)_{10} = (18730)_{10} - (-16273)_{10} = (35003)_{10}$

Нами же получено: $(X - Y)_2 = (1000\ 1000\ 1011\ 1011)_2 = 1*2^{15} + 1*2^{11} + 1*2^7 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^1 + 1*2^0 = 32768+2048+128+32+16+8+2+1 = 35003$, т.е. все верно.

Этап В.

Пример. *Представить в форме с плавающей точкой числа A=314,51, B= –16,22 и выполнить операцию A+B*

Переведем эти числа в двоичную систему счисления (целая часть переводится методом деления на 2, дробная – методом умножения на 2):

A = 100111010,10000010100011*2^0 = 0,10011101010000010100011*2^9

B=–10000,00111000010100011*2^0 = –0,10000001110000101000111*2^5

Смещенный порядок числа A будет равен

$P^*_A = P_A + N = 9 + 128 = 137_{10} = 1000\ 1001_2$,

Смещенный порядок числа B будет равен

$P^*_B = P_B + N = 5 + 128 = 133_{10} = 1000\ 0101_2$

Запишем числа в заданном формате

A: 0.10001001.10011101010000010100011

B: 1.10000101.10000001110000101000111

Для выполнения операции сложения необходимо выровнять порядки чисел, т.е. принять порядок меньшего числа (B) равным порядку большего числа (A), уменьшив мантиссу меньшего числа путем сдвига вправо на число разрядов, равное разности порядков чисел ($P_A – P_B = 4$) :

B: 1.10001001.00001000000111000010100

Мантиссу отрицательного числа M_B представляем в дополнительном коде, положительного M_A – в прямом. Тогда

M_A =0.10011101010000010100011

+

M_B =1.11110111111000111101100

$M_A + M_B$ =0.10010101001001010001111

Результат положительный (единица переноса из знакового разряда при использовании дополнительного кода отбрасывается), мантисса нормализованная. Запишем результат с учетом порядка в разрядной сетке заданного формата:

A+B : 0.10001001.10010101001001010001111

Проверка.

Рассчитаем порядок. $10001001 = 2^7 + 2^3 + 2^0 = 128 + 8 + 1 = 137_{10}$ Или с учетом смещения 137 – 128 = 9.

Тогда, т.к. мантисса имеет вид: 0.10010101001001010001111, то с учетом порядка имеем: 100101010.01001010001111 (сдвинули дес.точку на 9 позиций)

Значит, целая часть результата:

$2^8 + 2^5 + 2^3 + 2^1 = 256+32+8+2 = 298$

Дробная часть результата:

$2^{-2} + 2^{-5} + 2^{-7} + 2^{-11} + 2^{-12} + 2^{-13} + 2^{-14} = 0{,}28997802734375 \cong 0{,}29$

Эти же числа сложим в десятичной системе:

314,51+ (–16,22) =298,29

Видим, что все верно.

6. ЛОГИЧЕСКИЕ ОСНОВЫ ОРГАНИЗАЦИИ КОМПЬЮТЕРА

Как известно, информация в компьютере подвергается не только арифметической, но и логической обработке. Основу логических схем и устройств ЭВМ составляет специальный математический аппарат, называемый алгеброй логики или исчислением высказываний. При этом под высказыванием понимается любое утверждение, о котором можно сказать, что оно истинно или ложно. Одно и то же высказывание не может быть одновременно истинным и ложным или не истинным и не ложным.

Если высказывание истинно, то считают, что его значение равно единице; если высказывание ложно, то считают, что его значение равно нулю. Таким образом, значение высказываний можно рассматривать как переменную величину, принимающую только два дискретных значения: 0 или 1. Это приводит к полному соответствию между логическими высказываниями в математической логике и двоичными цифрами в двоичной системе счисления, что позволяет описывать работу логических схем компьютера, проводить их анализ и синтез с помощью математического аппарата алгебры логики.

Иное название алгебры логики – *булева алгебра*.

Булева алгебра является основным математическим аппаратом, который широко используется при проектировании и анализе схем обработки двоичной информации в цифровых компьютерах и других цифровых устройствах. Данный раздел математики назван в честь английского математика *Джорджа Буля (Georg Boole),* изложившего основные положения этой алгебры в своем трактате "Исследование законов мышления с помощью математических теорий логики и вероятностей", опубликованном в 1854 году. В 1937 году *Клод Шеннон (Claude Shannon),* в то время ассистент электротехнического факультета МИТ, предположил, что алгебру Буля можно использовать для решения проблем проектирования релейных переключающих схем. Предложенные Шенноном методы в дальнейшем были использованы при анализе и проектировании электронных цифровых схем. В компьютерной науке ма-

тематический аппарат булевой алгебры применяется в двух областях:

- *анализ* — исследование поведения цифровых логических цепей;
- *синтез* — при проектировании цифровых устройств с помощью методов булевой алгебры отыскивается наиболее рациональный способ реализации заданной логической функции.

6.1 Основные логические операции

Как и любая другая, булева алгебра использует переменные и операции над ними. В данном случае переменные и операции являются логическими переменными и логическими операциями. Переменные могут принимать только два значения — ИСТИНА (1) и ЛОЖЬ (0). Базовые логические операции — *конъюнкция* (AND), *дизъюнкция* (OR) и *инверсия* (NOT), — которые символически представляются знаками точки, плюса и надчеркиванием:

$$A \quad AND \quad B \quad = \quad A \bullet B$$

$$A \quad OR \quad B \quad = \quad A + B$$

$$NOT \quad A \quad = \quad \overline{A}$$

Результатом операции AND будет 1 (ИСТИНА) в том и только в том случае, если оба операнда имеют значение 1. Операция OR дает результат 1 в том случае, если любой из операндов (или оба вместе) имеют значение 1. Унарная операция NOT инвертирует значение операнда.

При отсутствии скобок существует приоритет операции *AND* над операцией *OR*. При обозначении операции *AND* символ точки часто опускается, если это не порождает двусмысленного толкования. Выражения

$$A + B \bullet C$$

$$A + BC$$

эквивалентны и означают, что нужно выполнить операцию *AND* над *B* и *C*, а затем – операцию *OR* над результатом и переменной *A*.

В таблице 4 представлены базовые логические операции в форме *таблиц истинности*, т.е. перечислены значения результата операции

при всех возможных комбинациях значений операндов. Помимо уже упомянутых операций *AND*, *OR* и *NOT*, в табл.4 включены еще три часто используемые операции – *XOR*, *NAND* и *NOR*.

Операция *XOR*, которую называют *суммой по модулю два* или *исключительным или*, дает результат 1 в том случае, если *только один* из ее операндов имеет значение 1.

Операция *NAND*, которую называют еще *стрелкой Пирса*, представляет собой инверсию операции AND:

$$A \quad NAND \quad B = NOT \quad (A \quad AND \quad B) = \overline{AB}$$

Операция *NOR* (штрих Шеффера) представляет собой инверсию операции *OR*:

$$A \quad NOR \quad B \quad = \quad NOT \quad (A \quad OR \quad B) = \overline{A + B}$$

Эти три новые функции активно используются при проектировании цифровых схем.

Таблица 4. Булевы логические операции.

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P NAND Q	P NOR Q
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

В таблице 5 перечислены основные тождества булевой алгебры. Два столбца таблицы отражают двойственность и взаимодополняемость операций AND и OR. Существуют два класса тождеств: базовые, или *постулаты*, которые принимаются без доказательств, и производные от постулатов.

Таблица 5. Основные тождества булевой алгебры

Постулаты		
$A \bullet B = B \bullet A$	$A + B = B + A$	Коммутативные законы
$A \bullet (B + C) =$ $(A \bullet B) + (A \bullet C)$	$A + (B \bullet C) =$ $(A + B) \bullet (A + C)$	Дистрибутивные законы
$1 \bullet A = A$	$0 + A = A$	
$A \bullet \overline{A} = 0$	$A + \overline{A} = 1$	
Производные тождества		
$0 \bullet A = 0$	$1 + A = 1$	
$A \bullet A = A$	$A + A = A$	
$A \bullet (B \bullet C) =$ $(A \bullet B) \bullet C$	$A + (B + C) =$ $(A + B) + C$	Ассоциативные законы
$\overline{A \bullet B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \bullet \overline{B}$	Формулы де Моргана

Следует обратить внимание на один их двух дистрибутивных законов, поскольку он отличается от привычного дистрибутивного закона обычной алгебры:

$$A + (B \bullet C) = (A + B) \bullet (A + C)$$

Два последних тождества в табл. 5 называются правилами де Моргана. Их можно записать в следующем виде:

$$A \quad NOR \quad B = \overline{A} \quad AND \quad \overline{B}$$

$$A \quad NAND \quad B = \overline{A} \quad OR \quad \overline{B}$$

Нетрудно проверить правильность выражений, представленных в табл.5, подставив комбинации значений 0 и 1 переменных A, B и C.

6.2 Логические переключающие элементы

Базовыми «строительными блоками» любых схем цифровых устройств являются логические переключающие элементы или *вентили*. Любые, сколь угодно сложные логические функции реализуются соответствующим соединением правильно подобранных вентилей.

Вентиль представляет собой электронную схему, которая формирует выходной сигнал в соответствии с простой булевой

операцией преобразования сигналов, поданных на его входы. Как правило, в качестве функций преобразования простейших вентилей используются элементарные функции AND, OR, NOT, NAND и NOR.

Ниже приведены условные обозначения вентилей разных типов[1] (см. рис.4), логические выражения, реализуемые ими, и таблицы истинности. Обратите внимание на то, что в условных обозначениях инвертирование обозначается кружочком на выходе элемента.

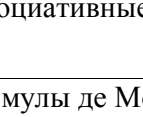
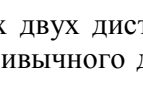
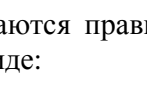
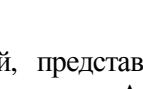
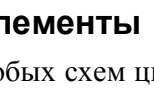
Наименование	Условное графическое обозначение	Реализуемая функция	Таблица истинности
AND		F=A • B или F = AB	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		F = A + B	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ Или F=A'	A | F 0 | 1 1 | 0
NAND		$F = \overline{(AB)}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{(A + B)}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Рис. 4 Базовые логические элементы

[1] Условные обозначения приведены в соответствии с международным стандартом *IEEE Standart 91*

Помимо логических вентилей с двумя входами, представленных выше, существуют и элементы с тремя, четырьмя и т.д. Так логическое выражение X+Y+Z может быть реализовано одним вентилем *OR* с тремя входами.

Как правило, при проектировании сложных логических схем используются не все типы представленных вентилей, а только один или два типа. Поэтому весьма важно выяснить, из каких элементарных функций можно построить любую, сколь угодно сложную булеву функцию. Ниже перечислены некоторые функционально полные множества логических функций (и соответственно, множества логических элементов):

- *AND, OR, NOT*
- *AND, NOT*
- *OR, NOT*
- *NAND*
- *NOR*

Набор *AND*, *OR* и *NOT* является *функционально полным*. Тогда и множество *AND* и *NOT* также является функционально полным, поскольку функцию *OR* можно сформировать из этих двух функций, пользуясь одним из правил де Моргана:

$$A + B = \overline{\overline{A} \bullet \overline{B}}$$

Аналогично, но с помощью другой формулы де Моргана, можно показать, что набор *OR*, *NOT* является функционально полным, поскольку с помощью этих двух функций можно реализовать функцию *AND*.

На рисунке 5 показано, как с помощью вентилей *NAND* можно реализовать логические функции *AND*, *OR* и *NOT*

На рисунке 6 – показано, как можно реализовать логические функции *AND*, *OR* и *NOT* с помощью вентилей *NOR*.

Из всего этого следует вывод: логическую схему, реализующую любую логическую функцию, можно построить из вентилей только одного типа, что часто используется на практике.

Рассматривая простейшие логические элементы, мы вышли на самый нижний уровень проблем, которые составляют предмет исследований в компьютерных науках. Далее идут вопросы реализации вентилей с помощью электронных компонентов — транзисторов, резисторов и т.д., — а это уже предмет не столько компьютерных наук, сколько электроники и электротехники.

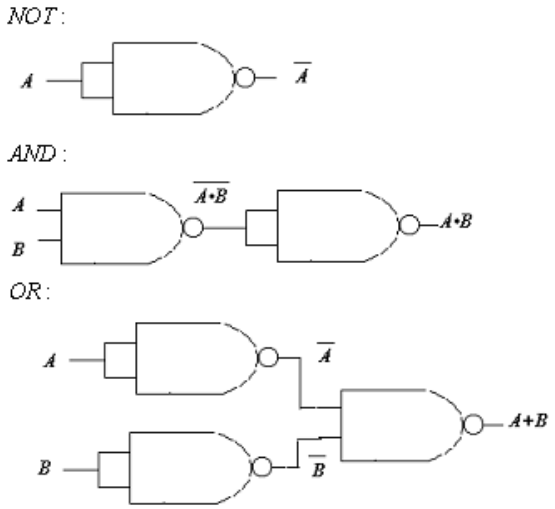


Рис 5. Использование вентилей NAND

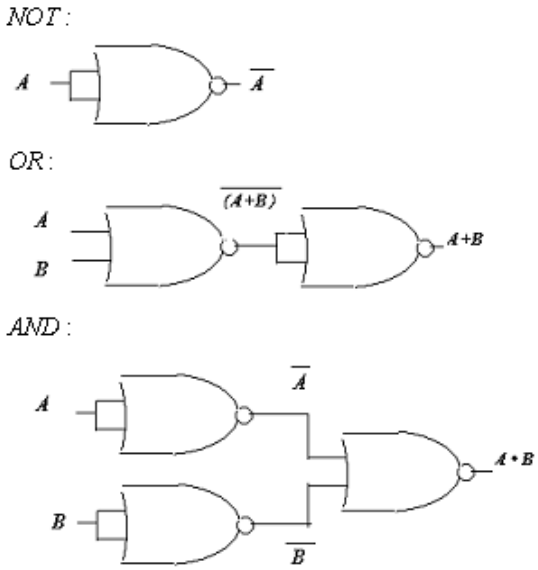


Рис 6. Использование вентилей NOR

6.3 Комбинационные схемы

Комбинационная логическая схема представляет собой объединение множества вентилей, и значения выходных сигналов такой схемы в каждый момент времени зависят только от значений сигналов на входах и никак не связаны с предысторией.

В общем случае комбинационная схема имеет *n* двоичных входов и *m* двоичных выходов. Функциональные возможности конкретной комбинационной схемы можно описать тремя способами:

- ***Посредством таблицы истинности***, в которой будут представлены значения набора из m выходных сигналов для каждой из 2^n возможных комбинаций входных сигналов.
- ***Посредством электрической функциональной схемы***, на которой будут изображены вентили и их соединения
- ***С помощью булева выражения***, точнее набора булевых выражений для каждого из *m* выходов, операндами которых будут *n* входов.

Любую булеву функцию можно реализовать в виде соединения вентилей. Как правило, одну и ту же таблицу истинности могут иметь разные булевы выражения, или, другими словами, одна и та же таблица истинности может быть по-разному представлена в виде комбинации элементарных операций.

Например, таблица истинности для трех переменных (табл.6):

Таблица 6. Таблица истинности функции трех переменных

A	*B*	*C*	*F*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Самый простой способ представить булеву функцию в виде комбинации элементарных функций — просуммировать конъюнктивные члены, обращающие функцию в 1:

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C} \qquad \textbf{(1)}$$

Такая форма представления называется *дизъюнктивной нормальной формой* (ДНФ). Вот как реализуется полученная ДНФ с помощью вентилей *AND*, *OR* и *NOT* :

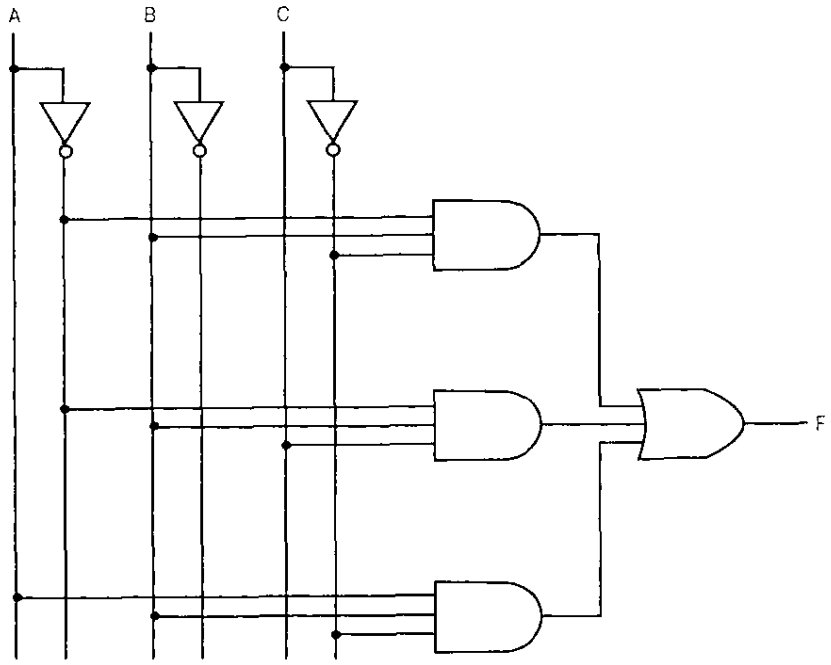


Рис.7 Реализация дизъюнктивной нормальной формы логической функции

Другая форма представления также может быть получена непосредственно из таблицы истинности. ДНФ выражает тот факт, что значение функции будет равно 1, если любой из членов дизъюнкции равен 1. Но можно также сказать, что значение

функции будет равно 1, если ни один из членов *конъюнкции* не равен 0. Следовательно:

$$F = (\overline{\overline{A}\overline{B}\overline{C}}) \bullet (\overline{\overline{A}\overline{B}C}) \bullet (\overline{\overline{A}B\overline{C}}) \bullet (\overline{A\overline{B}\overline{C}}) \bullet (\overline{ABC})$$

Это выражение можно переписать, воспользовавшись правилами де Моргана:

$$\overline{X \bullet Y \bullet Z} = \overline{X} + \overline{Y} + \overline{Z}$$

Тогда

$$F = (\overline{\overline{A}} + \overline{\overline{B}} + \overline{\overline{C}}) \bullet (\overline{\overline{A}} + \overline{\overline{B}} + \overline{C}) \bullet (\overline{\overline{A}} + \overline{\overline{B}} + \overline{C}) \bullet (\overline{A} + \overline{\overline{B}} + \overline{C}) \bullet (\overline{A} + \overline{B} + \overline{C}) =$$
$$(A + B + C) \bullet (A + B + \overline{C}) \bullet (\overline{A} + B + C) \bullet (\overline{A} + B + \overline{C}) \bullet (\overline{A} + \overline{B} + \overline{C}) \qquad \textbf{(2)}$$

Такая форма представления логической функции называется *конъюнктивной нормальной формой* (КНФ). Ее реализация с помощью элементарных вентилей показана на рис.8 ниже. На схеме не показаны вентили NOT, и предполагается, что доступны как истинные значения входных сигналов, так и их инверсии.

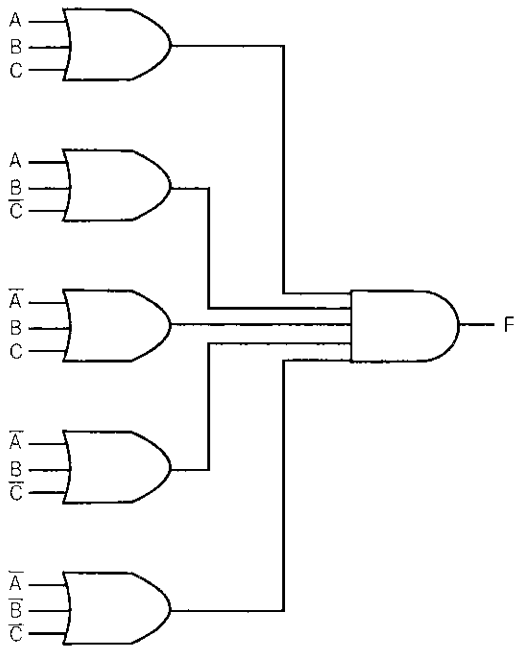


Рис .8 Реализация конъюнктивной нормальной формы логической функции

Таким образом, мы пришли к выводу, что любая логическая функция может быть представлена либо в виде ДНФ, либо в виде КНФ. На первый взгляд кажется, что выбор между этими двумя формами представления следует делать на основании подсчета количества значений 0 и 1 в выходной колонке таблицы истинности — количество членов ДНФ соответствует количеству значений 1, а количество членов КНФ — количеству значений 0. Но чаще при проектировании логических схем принимают во внимание другие соображения.

- В общем случае возможно получить более простое выражение, чем то, которое дает и ДНФ, и КНФ.
- Иногда предпочтительнее искать выражение, которое можно реализовать с помощью вентилей только одного типа.

В первом случае следует ожидать, что более простое логическое выражение будет и реализовано меньшим количеством вентилей. Существует несколько методов упрощения логических выражений, из которых мы рассмотрим два:

- алгебраический;
- с помощью карт Карно;

6.3.1 Алгебраические методы упрощения логических выражений

Алгебраические методы основаны на тождествах булевой алгебры, приведенных в *табл. 5.* Вернемся к выражению *(1).* Читателю несложно самостоятельно прийти к выводу, что эквивалентное выражение будет иметь вид:

$$F = \overline{A}B + B\overline{C} \qquad \textbf{(3)}$$

Но и его можно упростить:

$$F = B(\overline{A} + \overline{C})$$

На *рис.9* показана схема, реализующая это выражение

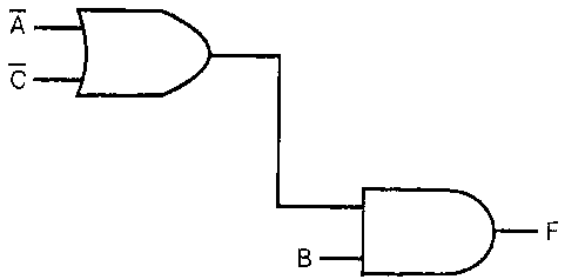


Рис 9. Упрощенная реализация функции табл 6.

Конечно, такое упрощение функций основано, скорее, на опыте и интуиции проектировщика, чем на систематической методике, а потому данный метод применяется только в самых простых случаях.

6.3.2 Карты Карно

С помощью карт Карно довольно удобно упрощать булевы функции небольшого числа переменных (теоретически — не более шести, а практически — четырех). Карта для задания функции, зависящей от *n* переменных это таблица из 2^n клеток, в которых представлены значения функции для всех возможных комбинаций значений переменных. На *рис. 10а* показана карта из четырех клеток для представления функции двух переменных. Удобно перечислять комбинации значений переменных в виде двоичных чисел 00, 01, 10, 11. Комбинации обычно записываются на внешнем поле карты. Карта для представления функции трех переменных состоит из восьми клеток (*рис. 10, б*), и значения одной из переменных записываются слева от поля карты, а двух других — над картой. Для функции четырех переменных понадобится 16 клеток, как показано на *рис. 10 в*.

Булева функция представляется в карте Карно следующим образом. Каждая клетка, в которой проставлено значение 1, соответствует определенному конъюнктивному члену в ДНФ функции. В этот конъюнктивный член входят переменные, имеющие значение 1 в комбинации для этой клетки, и инверсии переменных, имеющих в комбинации для этой клетки значение 0. Так, конъюнкция $A\ \overline{B}$ соответствует четвертой клетке в карте, показанной на *рис. 10а*. Полностью функция двух переменных, заданных этой картой, имеет вид

$$A\overline{B} + \overline{A}B$$

Первоначально карта заполняется на основании имеющейся таблицы истинности синтезируемой функции. Карта, показанная на *рис.10,б*, составлена в соответствии с таблицей истинности, представленной в *табл.6*.

Если необходимо заполнить карту для функции, заданной некоторым булевым выражением, то сначала это выражение нужно привести к *канонической* форме, т.е. к форме, в каждом члене которой будут присутствовать все переменные, а уже затем приступить к заполнению клеток карты Карно.

Маркировка карты, показанная на *рис. 10,г*, подчеркивает соответствие между переменными и столбцами и строками карты. Клетки двух строк, объединенных меткой A, будут содержать значения функции на тех наборах значений переменных, в которых переменная A равна 1, а клетки двух других строк будут содержать значения функции на тех наборах значений переменных, в которых переменная A равна 0.

После того как карта будет заполнена на основании таблицы истинности или аналитического выражения, можно приступить к формированию упрощенного выражения функции. Для этого нужно проанализировать расположение клеток, заполненных символами 1.

Основной принцип состоит в следующем. В карте Карно две соседние клетки, помеченные символами 1, всегда соответствуют конъюнктивным членам ДНФ, отличающимся только одним сомножителем. Поэтому их можно слить, причем отличающийся сомножитель при слиянии поглощается. Например, в карте, показанной на *рис.11а*, две соседние клетки соответствуют двум конъюнктивным членам —

$$\overline{A}B\overline{C}D \quad и \quad \overline{A}BCD$$

Тогда в результате слияния этих членов сформируется единственный член

$$\overline{A}B\overline{C}D + \overline{A}BCD = \overline{A}BD$$

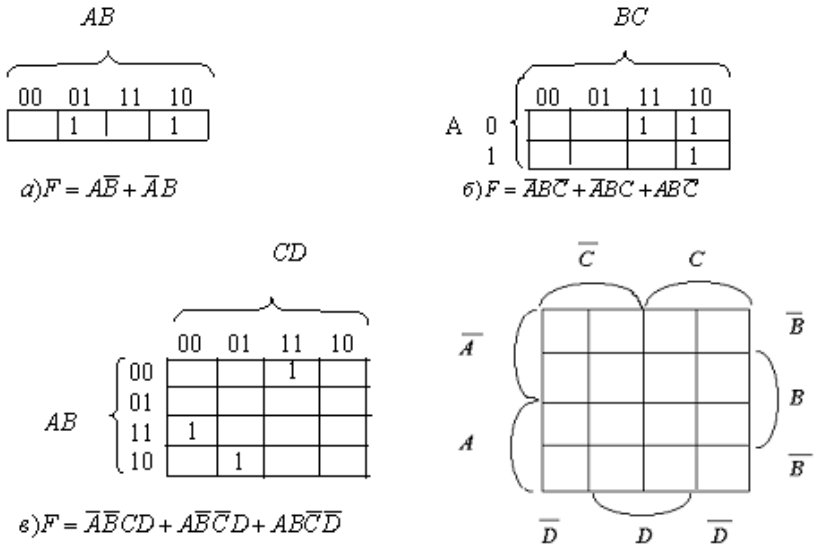


Рис. 10 (а,б,в,г) Использование карт Карно для представления булевых функций

Процесс слияния и поглощения можно расширять разными способами. Во-первых, следует учитывать, что противоположные края карты условны, т.е. клетки на противоположных краях можно объединять точно так же, как если они находились в средней части карты. Это показано на *рис.11,б* и *в*. Во-вторых, можно сливать не только 2, но и 4, 8 и т.д. соседних клеток. В следующих трех примерах, представленных на *рис.11*, демонстрируется слияние четырех клеток. В этом случае поглощаются уже две переменные. Последние три примера на этом рисунке иллюстрируют слияние восьми клеток, что позволяет поглотить три переменные в соответствующих конъюнктивных членах.

Суммируя все сказанное, сформулируем правила упрощения булевых функций с помощью карты Карно.

1. Среди всех маркированных клеток (т.е. клеток, соответствующих наборам переменных, на которых функция имеет значение 1) следует отыскать такие, которые совместно образуют наибольшую область размером 2, 4 или 8, и окружить эту область контуром.

2. Необходимо выбрать дополнительные блоки из маркированных клеток, которые имеют максимальный размер, причем количество таких блоков должно быть минимальным, но каждая маркированная клетка должна войти хотя бы в один блок. Результат такого выделения для некоторых функций может быть неоднозначным. При оконтуривании групп разрешается одну и ту же маркированную клетку включать более чем в одну группу.

На *рис. 12,а* представлена карта Карно, построенная в соответствии с *табл. 6*, которая иллюстрирует случай, когда одна маркированная клетка входит в две группы. Если какая-либо из групп полностью накрывается другими группами, то ее можно игнорировать. В карте, показанной на *рис. 12,б*, горизонтальная группа является избыточной, поскольку обе входящие в нее клетки уже вошли в другие группы. Такую избыточную группу можно не учитывать при составлении минимизированного выражения для булевой функции.

Следует отметить еще два важных свойства карт Карно. В некоторых случаях функция задается таким образом, что для определенных комбинаций значений входных переменных значение функции неизвестно или несущественно (например, конструкция реального объекта, описываемого этой функцией, такова, что в нем принципиально не могут возникнуть определенные сочетания значений входных переменных). В таком случае определенные клетки карты можно произвольно маркировать или не маркировать. Если при группировании такая клетка позволяет сформировать группу большего размера, ее следует считать маркированной и в результате получить более простое выражение.

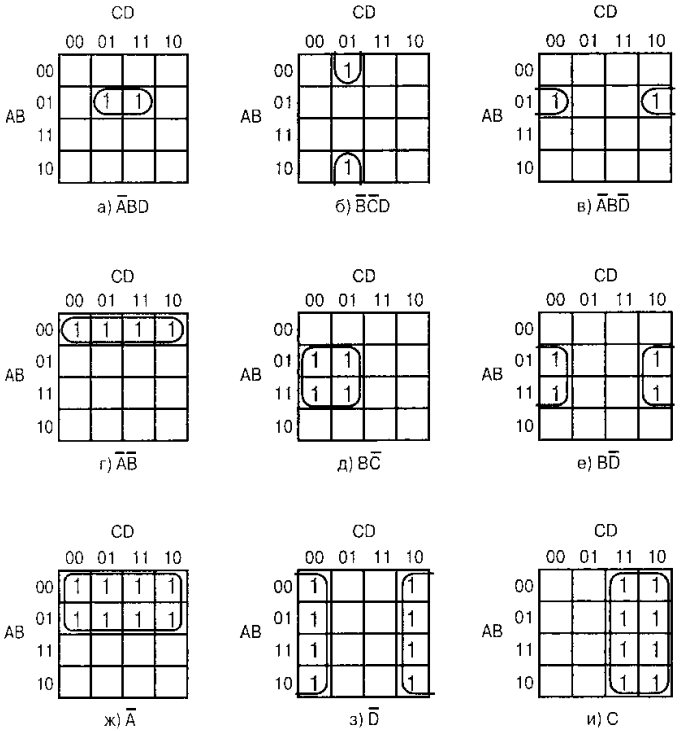
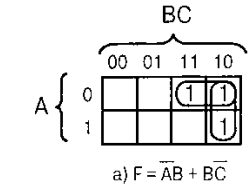


Рис. 11 Слияние клеток в картах Карно



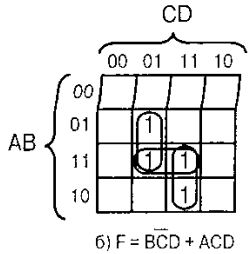


Рис.12 Перекрывающиеся группы

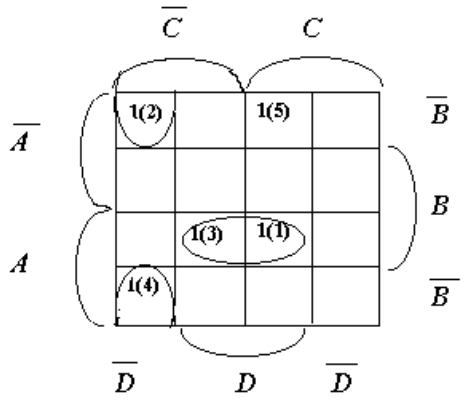
Практические задания «Комбинационные логические схемы»

Вариант	Задание 1	Задание 2
	Построить таблицу истинности для следующих булевых выражений	*Минимизировать заданное булево выражение с помощью карт Карно*
1	$ABC + \overline{ABC}$	$a\bar{b}cd + ab\bar{c}d + abcd + a\bar{b}\bar{c}\bar{d} + \bar{a}bcd$
2	$ABC + A\overline{BC} + \overline{ABC}$	$\bar{a}b\bar{c}d + a\bar{b}\bar{c}d + abc\bar{d} + abcd + a\bar{b}cd$
3	$A(B\bar{C} + \overline{BC})$	$abc\bar{d} + \bar{a}b\bar{c}d + a\bar{b}cd + \bar{a}\bar{b}c\bar{d} + abcd$
4	$(A+B)(A+C)(\overline{A} + \overline{B})$	$\bar{a}bcd + ab\bar{c}\bar{d} + a\bar{b}cd + \bar{a}bc\bar{d} + ab\bar{c}d$
	Упростите следующие булевы выражения, пользуясь коммутативным законом	
5	$A\cdot\bar{B} + \bar{B}\cdot A + C\cdot D\cdot E + \overline{C}\cdot D\cdot E + \overline{E\cdot C\cdot D}$	$a\bar{b}c\bar{d} + \bar{a}bc\bar{d} + abc\bar{d} + abcd + \bar{a}b\bar{c}d$
6	$A\cdot B + A\cdot C + B\cdot A$	$\bar{a}b\bar{c}d + \bar{a}bcd + abc\bar{d} + abcd + a\bar{b}\bar{c}d$
7	$(L\cdot M\cdot N)(A\cdot B)(C\cdot D\cdot E)(M\cdot N\cdot L)$	$\bar{a}\bar{b}\bar{c}\bar{d} + abc\bar{d} + abcd + a\bar{b}cd + \bar{a}\bar{b}cd$
8	$F\cdot(K+R)+S\cdot V+W\cdot\overline{X}+V\cdot S+\overline{X}\cdot W + (R+K)\cdot F$	$ab\bar{c}d + abc\bar{d} + a\bar{b}\bar{c}d + \bar{a}bc\bar{d} + \bar{a}b\bar{c}d$
	Примените формулы де Моргана к следующим булевым выражениям	
9	$F = \overline{V+A+L}$	$abcd + \bar{a}b\bar{c}d + abc\bar{d} + \bar{a}bc\bar{d} + a\bar{b}c\bar{d}$
10	$F = \overline{A+B} + \overline{C} + \overline{D}$	$a\bar{b}\bar{c}d + abcd + \bar{a}bcd + ab\bar{c}d + a\bar{b}c\bar{d}$

Рассмотрим для примера:

$$F = abcd + \bar{a}\bar{b}\bar{c}\bar{d} + ab\bar{c}d + a\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}cd$$

Построим карту Карно. В ней, для наглядности, в клетки будем заносить единицы с указанием номера конъюнктивного члена данной функции.



Видим, что 1-й и 3-й член объединяются с поглощением С. Объединяются также 2-й и 4-й с поглощением А. В итоге имеем:

$$F = abd + \overline{b}\,\overline{c}\,d + \overline{a}\,\overline{b}\,cd$$

Проверим результат, построив таблицы истинности как для исходной функции, так и для минимизированной. Для одних и тех же значений *a,b,c,d* должен быть один и тот же результат.

Таблица истинности ДО минимизации:

a	**b**	**c**	**d**	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	1	1

После минимизации:

a	**b**	**c**	**d**	**F**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	1	1

7 АППАРАТНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРА

7.1 История развития вычислительных средств

Основные этапы развития вычислительных средств представлены в таблице.

Этап	**Период развития**
Ручной	не установлен
Механический	с середины XVII в
Электромеханический	с 90-х годов XIX в
Электронный	с 40-х годов XX в

Рассмотрим основные моменты каждого этапа.

Первым массовым вычислительным средством был ***абак*** – счетное устройство с поразрядным представлением чисел. Нам он известен от древних римлян (еще с V–го века до н.э.). В нем использовались разгороженные канавки, в которые клали камешки в количестве, нужном для представления того или иного числа (рис. 13).

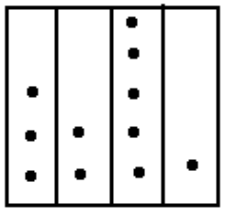


Рис.13 Представление числа 3251 на абаке.

Для этой цели использовали камешки из известняка. По латыни (на языке древних римлян) **calculi (*калькули*)** – известняк. Отсюда пошел латинский глагол **calculare (*калькуляре*)** – бросать камешки, позднее – считать, а затем и существительное **calculator (*калькулятор*)** – устройство для счета. В латинском языке был еще один глагол со значением **"считать" – computare (компутаре)**. От него через английское compute произошел термин **computer – компьютер.** Заметим, что и сейчас – в эпоху компь-

ютеров и калькуляторов – еще используется усовершенствованный абак – так называемые русские *счеты*.

В XVI веке была придумана логарифмическая функция, использование которой позволило сводить операции умножения и деления к сложению и вычитанию. И почти следом была придумана ***логарифмическая линейка***. Это, по сути, было массовое вычислительное средство. С ее помощью можно значительно быстрее, чем вручную, производить умножение и деление чисел. Однако точность результата при этом небольшая, всего лишь первые три значащие цифры. Вплоть до 70–х годов нашего века логарифмическая линейка была основным счетным орудием инженера любой специальности.

Рис.14 Блез Паскаль (1623-1662)

Первая механическая машина была построена немецким ученым **Вильгельмом Шиккардом** (предположительно в 1623 г.). Машина была реализована в единственном экземпляре и предназначалась для выполнения арифметических операций. Из-за недостаточной известности машины Шиккарда более 300 лет считалось, что первую суммирующую машину сконструировал Блез Паскаль. **Блез Паскаль** (французский математик, физик, религиозный философ и писатель) в 1642 г. изобрел механическую счетную машину, выполнявшую сложение, а в 1674 г. **Готфрид Лейбниц** расширил возможности машины Паскаля, добавив операции умножения, деления и извлечения квадратного корня. Специально для своей машины Лейбниц применил систему счисления, использующую вместо привычных для человека десяти цифр две: 1 и 0. Двоичная система счислений широко используется в современных ЭВМ.

Ни одна из этих машин не была автоматической и требовала непрерывного вмешательства человека. В 1834 г. **Чарлз Бэббидж** (Charles Babbage) первым разработал подробный проект автоматической вычислительной машины. Он так и не построил свою машину, так как в то время невозможно было достичь требуемой точности изготовления ее узлов.

Ч. Бэббидж выделял в своей машине следующие составные части:

- «склад» для хранения чисел (по современной терминологии — память);
- «мельницу» для производства арифметических действий (арифметическое устройство, процессор);
- устройство, управляющее последовательностью выполнения операций (устройство управления);
- устройства ввода и вывода данных.

В качестве источника энергии для приведения в действие механизмов машины Ч. Бэббидж предполагал использовать паровой двигатель. Бэббидж предложил управлять своей машиной с помощью перфорированных карт, содержащих коды команд, подобно тому как использовались перфокарты в ткацких станках **Жаккара**. На этих картах было представлено то, что сегодня мы назвали бы программой.

Ч. Бэббидж довольно подробно рассматривал вопросы, связанные, как мы сейчас говорим, с программированием. В частности, им была разработана весьма важная для программирования идея «условной передачи управления». Идеи Бэббиджа заложили фундамент, на котором со временем были построены ЭВМ.

Первые программы для вычислительной машины Бэббиджа создавала **Ада Лавлейс** (Ada Lovelace) — дочь известного поэта Джорджа Байрона, в честь которой впоследствии был назван один из языков программирования. Выражаясь современным языком, Лавлейс составила программу вычисления чисел Бернулли. Ада Лавлейс разработала основные принципы программирования, которые остаются актуальными до настоящего момента

времени. Ряд терминов, введенных Адой Лавлейс, используются и сейчас, например, «цикл», «рабочие ячейки».

Рис.15 Августа Ада Байрон, графиня Ловлейс (1815-1853)

Теоретические основы современных цифровых вычислительных машин заложил английский математик **Джордж Буль** (1815—1864). Он разработал алгебру логики, ввел в обиход логические операторы И, ИЛИ и НЕ. Заметим, что его дочь Э. Войнич — автор известного произведения «Овод».

Изобретение в 1879 г. петербургским ученым В.Т. Однером ***механического арифмометра*** позволило еще более ускорить процесс вычисления. Арифмометр позволял производить вычисления с точностью до 12 значащих цифр. Его конструкция была настолько совершенна, что арифмометры этого типа выпускались почти без изменений в течение ста лет. Именно поэтому принято рассматривать арифмометр как массовое вычислительное средство.

В 1888 г. **Германом Холлеритом** (Herman Hollerith) была сконструирована первая электромеханическая машина для сортировки и подсчета перфокарт. Эта машина, названная ***табулятором***, содержала реле, счетчики, сортировочный ящик. Изобретение Холлерита было использовано при подведении итогов переписи населения в США. Успех вычислительных машин с перфокартами был феноменален. То, чем за десять лет до этого занимались 500 сотрудников в течение семи лет, Холлерит сделал с 43

помощниками на 43 вычислительных машинах за 4 недели. В 1896 г. Герман Холлерит основал фирму Computing Tabulation Company. Спустя несколько лет это предприятие переименовали в известнейшую теперь фирму **International Business Machine Corporation (IBM).**

Немецкий инженер **Конрад Цузе (**Konrad Zuse) был первым, кто успешно осуществил идею создания автоматической электромеханической вычислительной машины на основе двоичной системы счисления. В 1936 г. он начал конструировать вычислительный аппарат, работающий в двоичной системе счисления, который впоследствии был назван **Zuse 1** (Z1). В 1941 г. Цузе сумел построить действующую модель Zuse 3, которая состояла из 600 реле счетного устройства и 2000 реле устройства памяти.

В 1944 г. (по другим источникам, в 1943 г.) в Англии было разработано полностью автоматическое вычислительное устройство Colossus II. Основным его назначением была дешифровка перехваченных сообщений военного противника.

Еще одна полностью автоматическая вычислительная машина, изобретенная профессором Гарвардского университета **Говардом Айкеном (**Aiken Howard, 1900—1973) при участии группы инженеров фирмы IBM, была построена в 1944 г. Она была названа **ASCC (**другое название **Mark 1)**, и была электромеханической (построена на реле), состоящей приблизительно из 750 тысяч компонентов. На умножение она тратила около 4 секунд. До знакомства с работами Цузе научная общественность считала машину ASCC первой электромеханической машиной.

В 1937 г. в США **Дж. Атанасов** начал работы по созданию электронной вычислительной машины. Им были созданы и запатентованы первые электронные схемы отдельных узлов ЭВМ. Совместно с К. Берри к 1942 г. была построена электронная машина **ABC** (**A**tanasoff-**B**erry **C**omputer).

Первый ***электронный компьютер*** был создан в США в 1946 г. в Пенсильванском университете. Его разработали **Эккерт и Маучли** (John W. Mauchly and J. Presper Eckert, Jr.). Он назывался ENIAC (Electronic Numerical Integrator And Calculator), был весьма внушительных размеров и выполнял до 5000 операций в секунду. В нем использовались электронные лампы. Все последующие компьютеры на лампах принято называть ***компьютера-***

ми первого поколения (1946 – 1955). Они имели большие размеры, потребляли много электроэнергии и обладали довольно слабыми вычислительными характеристиками см. *рис. 16* и *табл. 7*. Внешний вид электронной лампы – см. на *рис. 17*.

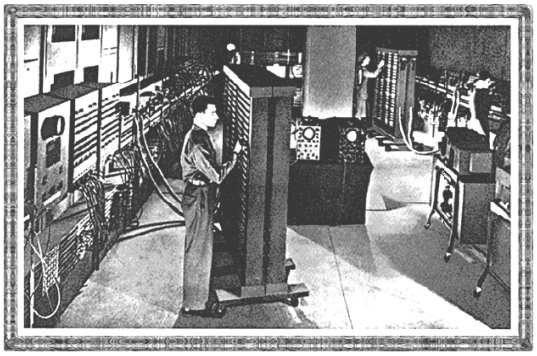


Рис. 16 Первая в истории ЭВМ (ENIAC)

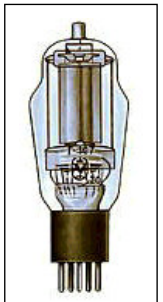


Рис.17 Электронная лампа

С 1955 г. начинает вести свой отсчет ***второе поколение компьютеров – на транзисторах*** (полупроводниках). Компьютеры стали меньше по размеру, им нужно было меньше электроэнергии, а быстродействие у них было больше – до десятков тысяч операций в секунду. В это же время начали использовать языки программирования. Внешний вид одного из самых популярных в нашей стране компьютеров второго поколения приведен на *рис. 18*.

Примерно в середине 60–х годов стали изготавливать интегральные схемы – небольшие кристаллы полупроводников, которые способны выполнять действие нескольких сотен и даже тысяч транзисторов. ***Компьютеры на интегральных схемах – компьютеры третьего поколения***. Они обладают большой памятью и высоким быстродействием (*табл. 7*).

Современные компьютеры – это компьютеры четвертого поколения. Они появились в начале 70–х в связи с созданием микропроцессора и больших интегральных схем, выполняющих работу нескольких сотен тысяч транзисторов. Изобретение микропроцессора – это революция в информатике. В результате этого и был создан ***персональный компьютер (ПК)***. Он помещается на

столе, но по объему памяти и скорости выполнения операций в десятки раз превосходит ЭВМ первого поколения см. *рис. 19* и *табл. 7*.

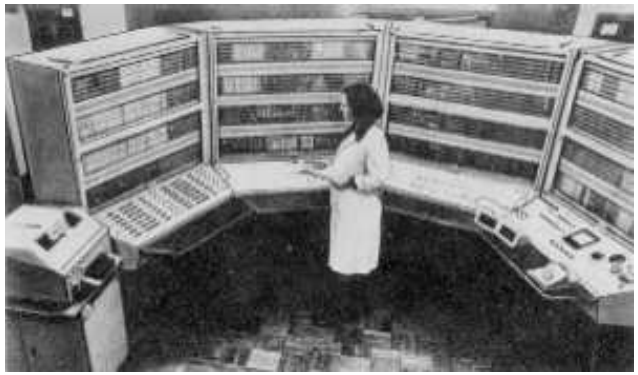


Рис. 18 ЭВМ второго поколения БЭСМ-6

Табл. 7. Характеристики компьютеров разных поколений.

Таблица характеристик различных поколений компьютеров

характеристика	1-е поколение (ЭНИАК)	2-е поколение (ЭВМ "МИР")	3-е поколение (ЕС-1022)	4-е поколение (IBM PC)
масса	30 тонн	300 кг	150 кг	20 кг
объем	170 куб.м.	2 куб.м.	6 куб.м.	0.7 куб.м
оперативная память	20 ячеек	2000 ячеек	128 Кбайт	640 Кбайт
быстродействие	5000 оп/с	8000 оп/с	80000 оп/с	2 мл н. оп/с

В настоящее время во многих странах пытаются создать ***компьютеры пятого поколения.*** Предполагается, что это будут компьютеры, воспринимающие информацию на языке, близком к естественному. Иначе говоря, когда такой компьютер появится, то люди, работающие с ним, будут давать ему задание привычным для человека способом. Как, например, преподаватель дает задание студентам.

Рис. 19 Персональный компьютер

7.2 Классификация современных ПК и его составляющих

7.2.1 Процессор

Основным устройством компьютера является процессор, который характеризуется маркой (286, 386, 486, Pentium) и быстродействием (точнее — тактовой частотой в МГц). Не менее важные характеристики компьютера — размер оперативной памяти и размер жесткого диска (винчестера).

Наиболее типичные характеристики приведены в таблице 8.

Табл. 8. Характеристики персональных компьютеров.

Марка процессора	***Быстродействие (тактовая частота), МГц***	***Оперативная память, Мб***	***Жесткий диск (винчестер), Мб***
286	8 — 20	1 — 2	20 — 80
386	40 — 60	2 — 4	40 — 200
486	66 — 100	4 — 8	80 — 500
Pentium	100 — 300	8 — 32	500 — 2000
Pentium II	300 — 450	16 — 32	1000 — 4000
Pentium III	500 — 1000	32 — 128	10000 — 40000
Pentium IV	1000 — 3400	128 — 512	10000 — 80000

В настоящее время компьютеры с 286-м, 386-м и 486-м процессорами являются устаревшими и, в основном, вышли из употребления. Действительно современными можно назвать, пожалуй, лишь компьютеры с процессором Pentium IV.

Рис. 20 Микросхема процессора Pentium II (сравни с размерами руки человека)

7.2.2 Монитор

Одним из важных элементов компьютера является *монитор* (дисплей). Мониторы бывают разных типов. Отличаются они разрешающей способностью и числом представимых на экране цветов. Сейчас применяется такая классификация – см. *табл.9*. Дисплей взаимодействует со своим адаптером, который может также называться видеокартой, видеоадаптером или контроллером. Дисплей и адаптер очень тесно связаны между собой и совместно определяют качество изображения — разрешение, количество воспроизводимых цветов, скорость регенерации (число кадров в единицу времени).

Разрешение зависит от размеров экрана и минимального элемента изображения (так называемого «зерна», равного для лучших мониторов 0,24—0,28 мм). Для 14-дюймовых мониторов разрешение обычно не более 800 x 600 элементарных точек (пикселей), для 15-дюймовых — 1024 x 768, для 21-дюймовых — 1280 x 1024 точек.

Способность адаптера выводить на экран монитора изображение с заданным разрешением и глубиной цвета (т. е. числом цветовых оттенков) определяется объемом установленной оперативной памяти на плате адаптера. Для отображения 16,7 миллиона оттенков цветов (24 бита на пиксель) нужно установить в адаптер не менее 1,37 Мбайт памяти при разрешении 800 x 600 элементарных точек, 3,75 Мбайт при разрешении 1280 x 1024.

Табл. 9. Характеристики мониторов персональных компьютеров

Тип адаптера монитора	***Разрешение (точек по горизонтали X по вертикали)***	***Число цветов***
CGA	320 x 200	16
EGA	640 x 350	64
VGA	640 x 480	256
SuperVGA	1024 x 768 и более	до 16 млн.

В настоящее время стандартом принят тип SuperVGA (SVGA).

Заметим, что по физическому принципу действия мониторы подразделяют на: э*лектронно-лучевые* мониторы, и *жидкокристаллические*.

Основным узлом электронно-лучевого монитора является электронно-лучевая трубка (ЭЛТ). Катод, анод и модулятор образуют электронный прожектор, который иногда называют электронной пушкой. Горизонтальные и вертикальные отклоняющие пластины образуют отклоняющую систему. Такая отклоняющая система называется электростатической. Существуют магнитные отклоняющие системы, в которых для изменения траектории движения электронного потока вместо пластин используют катушки. В ЭЛТ используется поток электронов, сфокусированных в узкий пучок, управляемый по интенсивности и по положению в пространстве и взаимодействующий с экраном трубки. Электронный пучок испускается электронным прожектором (точнее, катодом), а изменение положения пучка на экране производится отклоняющей системой.

Перемещение электронного луча по экрану ЭЛТ в соответствии с определенным законом называется разверткой, а рисунок, прочерченный следом пучка электронов на экране, — растром. Развертка осуществляется подачей на отклоняющую систему ЭЛТ периодически изменяющихся напряжений. В ходе развертки электронный пучок последовательно обегает по строчкам поверхность экрана ЭЛТ.

Экран покрыт люминофором, поэтому в местах падения электронного пучка появляется свечение, яркость которого пропорциональна интенсивности пучка. Интенсивность потока электронов изменяется в соответствии с сигналами, подаваемыми на

управляющий электрод — модулятор. Именно эти сигналы формируют необходимое изображение на экране дисплея.

Цветной дисплей содержит три электронные пушки с отдельными схемами управления. Экран выполняется в виде мозаичной структуры (прямоугольной матрицы), состоящей из зерен люминофора трех цветов свечения: красного (Red), зеленого (Green) и синего (Blue). Зерна расположены тройками (триадами) так, чтобы электроны каждой из трех пушек попадали только на зерна «своего» цвета. Для обеспечения этого на пути движения электронов устанавливают маски.

Принцип действия цветного дисплея базируется на физиологической особенности зрения человека. Так, при одинаковой интенсивности свечения трех разноцветных маленьких соседних зерен этот участок экрана воспринимается как белая точка. Свечение соседних красного и зеленого зерен воспринимается как желтая точка, а свечение синего и зеленого зерен дает голубую точку и т. д. Изменяя интенсивность свечения трех основных цветов (RGB), можно получить любой цвет или оттенок. Такой способ получения любых цветов является одной из систем цветопередачи и назван RGB-системой (по первым буквам соответствующих английских слов).

Принцип действия *жидкокристаллического монитора* (ЖКМ) существенно отличается от принципа действия монитора с ЭЛТ. В ЖКМ используется физический эффект изменения пространственного положения молекул кристаллов под действием электрического поля. Так же, как в ЭЛТ, в ЖКМ изображение формируется из большого числа точек (пикселей), которые образуют прямоугольную матрицу. Однако в жидкокристаллической матрице управление процессом формирования изображения идет цифровым способом. В ЖКМ одновременно изменяется свечение всех элементов целой строки матрицы (экрана). Мерцание ЖКМ принципиально меньше, чем дисплеев с ЭЛТ, так как при формировании изображения обновляются только изменяющиеся пиксели. Изображение статических картинок не требует обновления, поэтому в этих случаях мерцаний экрана ЖКМ совсем нет.

7.2.3 Принтер

Другим важным элементом ПК является *принтер* — устройство для вывода на бумагу. Принято выделять три типа принтеров (в зависимости от способа печати на бумагу) – см. *табл. 10*.

Табл. 10. Типы и характеристики принтеров

Тип	***Способ печати***	***Скорость печати (символов/сек)***
Матричный	Печатающей головкой с 9-ю (18-ю или 24-мя) иглами через красящую ленту	200- 400
Струйный	Картриджем с чернилами путем выстреливания чернил через маленькие сопла	200-500
Лазерный	Принцип подобен ксерографии: намагничивание участков бумаги лазерным лучем и прилипание к ним тонера — красящего порошка	1000 – 5000

7.2.4 Память

Память — функциональная часть ЭВМ, предназначенная для записи, хранения и выдачи информации. В ЭВМ запоминание информации происходит в ОЗУ (оперативное запоминающее устройство), ПЗУ (постоянное запоминающее устройство), ВЗУ (внешнее запоминающее устройство), кэш-памяти (недоступный для пользователя буфер), CMOS-памяти (здесь хранятся системные данные) и РОН — внутренних регистрах процессора (используются при вычислениях). ОЗУ, ПЗУ, РОН, кэш-память, CMOS-память относятся к электронной памяти, а ВЗУ — к электромеханической памяти.

Заметим, что сокращение CMOS (**C**omplement **M**etal **O**xide **S**emi-conductor — комплементарные пары металл-оксид-полупроводник, отечественная аббревиатура — КМОП) указывает на технологию изготовления данной памяти, а не на ее функциональное назначение. Точнее, ее можно было бы назвать памятью системных установок (конфигурации).

Существует еще одно понятие — видеопамять — электронная память, размещенная на видеокарте (графическом адаптере). Она используется, как правило, в качестве буфера для хранения кадров динамического изображения.

ОЗУ и ПЗУ совместно образуют так называемую **основную** память (ОП).

Объем и быстродействие памяти во многом определяют производительность всего компьютера.

При работе на компьютере в качестве носителя внешней памяти (например, для переноса информации с компьютера на компьютер) активно используются дискеты. Самая популярная дискета – диаметром в 3,5 дюйма и с объемом памяти в 1440 Кбайт. Для переноса большого объема информации используются так называемые ZIP-устройства с дискетой на 20 и даже 100 Мб информации. В настоящее время дискеты и ZIP-устройства почти не применяются. Для переноса информации теперь активно применяют так называемые Flash-устройства. Они подключаются через специальный USB–порт, имеющийся у каждого современного компьютера, и имеют память в 256 и более Мбайт.

8. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В КОМПЬЮТЕРЕ

8.1 Единицы измерения информации

Любую информацию можно разделить на небольшие, элементарные (т.е. далее уже неделимые) части. Например, текст в книге состоит из букв и других символов. Буква – это элементарная часть текстовой информации.

Как мы уже знаем, компьютер может работать только с числами (причем, записанными в двоичной системе счисления). Поэтому для обработки информации произвольного вида (в том числе и буквенной) нужно ее закодировать в виде двоичных чисел.

Минимальной единицей информации является ***бит***. Один бит информации – это одна двоичная цифра : 0 или 1 (само название ***bit*** происходит от английских слов ***bi***nary digi***t*** – двоичная

цифра). Но бит – очень маленькое количество информации, поэтому в компьютерах для обработки частей информации используют более крупную единицу – *байт*. Один байт состоит из 8 битов.

Какое наибольшее число можно записать с помощью одного байта? Это число, у которого все 8 битов равны 1. Для получения десятичного значения прибавим к этому числу 1 и вычтем ее:

$$111\ 111\ 11_2 = (111\ 111\ 11_2 + 1) - 1 =$$
$$= 100\ 000\ 000_2 - 1 = 2^8 - 1 = 255.$$

Как видим, в 1 байте можно хранить одно из 256 целых чисел: от 0 до 255. Для того чтобы закодировать буквы английского алфавита (26 больших и 26 малых букв), нужно 52 числа. Для кодирования букв русского алфавита нужно 33+33 = 66 чисел, а для кодирования десятичных цифр – еще 10 чисел. Таким образом, с помощью 1 байта можно закодировать все буквы английского и русского алфавитов, десятичные цифры и еще 256 – 52 – 66 –10 = 128 других символов (точка, запятая и другие знаки препинания, знаки арифметических действий и т.п.). Итак, ***1 байт = 1 символ.***

Один байт – это не только единица информации, это элементарная ячейка памяти компьютера. Память компьютера состоит из последовательности таких ячеек. Каждая ячейка (байт) имеет свой адрес (номер ячейки) и содержимое – двоичный код, который хранится в ней. Когда процессор обрабатывает информацию, он находит по адресу в памяти нужную ячейку, читает из нее содержимое, выполняет необходимые действия и записывает результат в другую ячейку памяти.

Память компьютера измеряется в байтах, но чаще для этого используют другие единицы: ***килобайт*** (Кбайт или Kb) и ***мегабайт*** (Мбайт или Mb) :

$$1\ Кбайт = 2^{10}\ байт = 1024\ байта$$
$$1\ Мбайт = 2^{20}\ Кбайт = 1024\ Кбайта$$

8.2 Представление целых и вещественных чисел в памяти компьютера

Но каким образом компьютер хранит числовую информацию? Это зависит от того, какой процессор используется в компьютере.

Каждый процессор предназначен для обработки двоичных кодов определенной длины. В первых персональных компьютерах использовались процессоры, которые одной командой могли обработать только один байт информации. Эти компьютеры были восьмиразрядными. Затем появились 16–разрядные и 32-разрядные компьютеры. Современные компьютеры являются 64-разрядными. В них применяют процессоры, которые одной командой могут обработать восемь байтов информации.

В 16–разрядных компьютерах для хранения и обработки целых чисел используется 2 байта памяти. Какие целые числа могут обрабатывать такие компьютеры? Вспомним, что целые числа могут быть положительными и отрицательными. Как закодировать знак числа? Для этого можно использовать один из 16 битов, например, самый левый бит. Если он равен 0, то будем считать число положительным, а если он равен 1 – отрицательным. Итак, запомним:

Для записи целого числа используется два байта (16 битов). Один бит используется для знака числа и 15 битов – для абсолютной величины числа.

По этой схеме целое число будет иметь наибольшую абсолютную величину, если все 15 битов будут равны 1:

$$(111\ 1111\ 1111\ 1111_2 + 1) - 1 = 1\ 000\ 0000\ 0000\ 0000_2 - 1 =$$
$$= 2^{15} - 1 = 32767.$$

Наибольшее целое число, которое может обработать процессор 16–разрядного компьютера, равно 32767. Если нужно обрабатывать целые числа большей величины, то для их хранения нужно больше 2 байтов памяти. Процессор не может обработать такую информацию одной командой, поэтому нужно написать специальную программу для обработки целых чисел больше 32767.

Вещественные (дробные) числа обычно занимают в памяти компьютера 4 байта, а сами числа представляются в экспоненциальной форме. Например, число –184.525 (вместо десятичной запятой используем точку !) записывается в виде – 0.184525E+3. Здесь 184525 – это мантисса числа, а 3 – порядок числа (E+3 означает "умножить на 10^3").

В ячейке из 4 байтов нужно хранить мантиссу числа со знаком и порядок числа тоже со знаком. Имеющиеся разряды (биты) распределены следующим образом: 7 битов для порядка числа (вместе с его знаком) и 25 битов для мантиссы числа (тоже со знаком).

Итак, запомним:

Для записи вещественного числа используется четыре байта (32 бита). Семь битов используется для порядка числа и 25 битов – для мантиссы числа.

По этой схеме максимальная абсолютная величина порядка числа равна $2^6 - 1 = 63$, а максимальная величина мантиссы равна $2^{24} - 1 = 16\ 777\ 215$. Итак, ***мантисса вещественного числа не может содержать больше 8 десятичных цифр.*** Компьютер при вычислениях отбрасывает лишние цифры в мантиссе, поэтому все вычисления с вещественными числами на компьютере всегда выполняются ***приближенно***.

8.4 Обработка информации в компьютере

Как известно, информация в компьютере представлена кодами. Обрабатывая информацию, компьютер производит определенные преобразования двоичных кодов. В основе устройства, реализующего эти операции, лежат законы алгебры логики (см. раздел 1.5). Напомним, что в ней чаще всего используются три простейшие логические функции (***инверсия***, ***конъюнкция*** и ***дизъюнкция***), каждая из которых принимает лишь два значения: "истина" или "ложь" (иначе говоря, 0 или 1). Рассмотрим эти функции. Заметим сразу, что значения логических функций в зависимости от аргументов принято задавать в виде таблиц (см. табл. 11), которые называются ***таблицами истинности.***

Табл. 11. Таблицы истинности для основных логических функций

Функция "НЕ"

Вход	Выход
0	1
1	0

а)

Функция "ИЛИ"

Вход1	Вход2	Выход
0	0	0
1	0	1
0	1	1
1	1	1

б)

Функция " И "

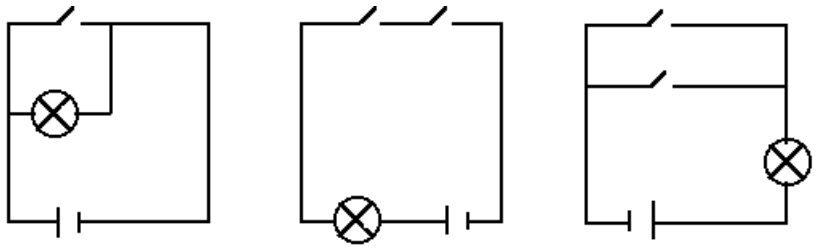
Вход1	Вход2	Выход
0	0	0
1	0	0
0	1	0
1	1	1

в)

Инверсия. Обозначается "НЕ". Функция зависит от одного аргумента и обращается в нуль в случае, если аргумент равен единице и наоборот (табл. 11,*а*). Показ действия инверсии можно осуществить на простой электрической схеме (рис. 21,*а*).

Конъюнкция (логическое умножение). Обозначается "И". Она зависит от двух аргументов и принимает как бы значение их произведения (табл. 11,*б*). Действие ее можно проиллюстрировать схемой на рис. 21,*в* .

Дизъюнкция (логическое сложение). Обозначается "ИЛИ". Функция обращается в 0 только в том случае, когда оба ее аргумента равны нулю. В остальных случаях ее значение равно 1 (табл. 11,*в*). Действие этой функции иллюстрируется схемой на рис. 21,*б* .



а) ***б)*** ***в)***

Рис. 21 Иллюстрация основных логических элементов.

На базе этих основных логических элементов строятся все логические схемы электронных цифровых машин. Обработка информации в компьютере может быть организована лишь в ре-

зультате объединения очень большого числа комбинаций логических элементов.

Посмотрим, как реализуется на основе этих логических функций сложение двоичных цифр 0 и 1 :

0 + 0 = 00, 0 + 1 = 01, 1 + 0 = 01, 1 + 1 = 10

(для единой записи результаты написаны в виде двузначных чисел), или в общем виде можно записать так:

$$A + B = C_2C_1.$$

Рассмотрим схему ***двоичного сумматора*** (рис. 22), имеющего два входа A, B и два выхода для функций C_1, C_2 . Если внимательно проанализировать эту схему, нетрудно убедиться в правильности его работы. Этот сумматор является *одноразрядным*, но на его основе нетрудно сделать многоразрядный двоичный сумматор. А это позволит обрабатывать любую информацию, закодированную в двоичном виде, поскольку любая арифметическая операция (вычитание, умножение и деление) может быть сведена к сложению.

Логические элементы "И", "ИЛИ", "НЕ" реализуются в виде физических элементов (диодов и проч.). Значит, нетрудно организовать арифметические и логические операции, нужные для обработки информации на компьютере.

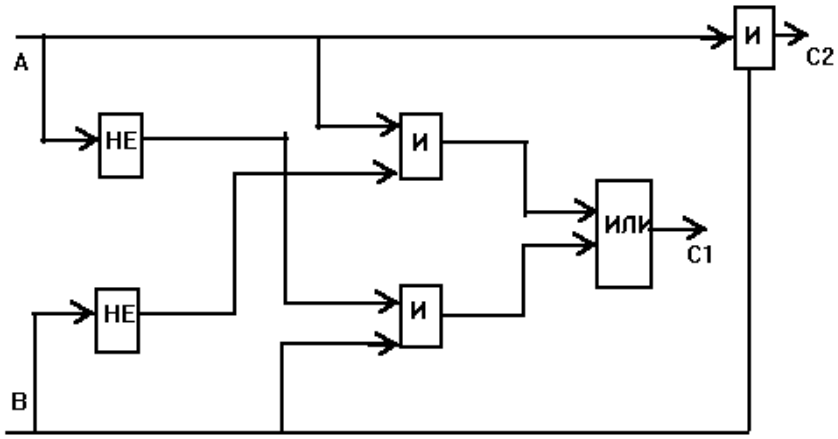


Рис. 22. Схема одноразрядного двоичного сумматора.

9. ОСНОВЫ ОПЕРАЦИОННОЙ СИСТЕМЫ MS DOS.

9.1 Программное обеспечение персонального компьютера

Как известно, компьютер "способен" общаться с человеком только при наличии программного обеспечения (ПО). Структура ПО изображена на рис. 22. Главным среди всего ПО является ***операционная система*** (ОС).

Совокупность управляющих и обрабатывающих программ, которые обеспечивают работу компьютера и согласованные действия всех его частей, называется операционной системой.

Рис. 22. Структура программного обеспечения персонального компьютера.

Проще говоря, ОС – это то, что "оживляет" компьютер. ОС обычно загружается в оперативную память при включении компьютера и только после этого компьютер готов к общению с человеком. Благодаря постоянно загруженной ОС компьютер "уме-

ет двигать руками–ногами" – управлять периферийными устройствами и понимает простейшие команды.

Но чтобы компьютеру дать указание на решение той или иной задачи, нужно составить программу на ***языке программирования***. Как известно, компьютер понимает только машинный язык, а мы пишем программу на языке высокого уровня, приближенном к естественному, человеческому языку. Для перевода с языка программирования на машинный язык служат ***программы–трансляторы***. Иначе говоря, в состав ПО должны входить программы–трансляторы с языков высокого уровня на машинный язык. Благодаря им компьютер "понимает" язык и умеет сам "говорить".

В состав ПО входят также ***прикладные программы***, ориентированные на выполнение конкретных действий на компьютере. В их числе – редакторы (текстовые, графические, музыкальные), обучающие программы и др. К изучению этой части программного обеспечения ты и приступаешь в следующей главе учебника.

Программное обеспечение – такая же важная составная часть компьютера, как и его аппаратура. Поэтому говорят: ***современный компьютер – неразрывное единство аппаратного и программного обеспечения.***

Итак, любая информация обрабатывается на компьютере с помощью программ. И главным элементом программного обеспечения ПК является операционная система (ОС). Операционная система – это посредник между человеком, работающим на компьютере (его называют ***пользователем***) и самим компьютером. Операционная система выполняет все команды пользователя и управляет различными устройствами компьютера: дисками, клавиатурой, принтером и другими устройствами.

Существуют различные типы компьютеров и у каждого типа своя ОС. Мы будем рассматривать только компьютеры типа IBM и соответствующую им операционную систему **MS DOS** (MicroSoft Disk Operating System) и **Microsoft Windows**. MicroSoft – это название фирмы–разработчика этой системы. Название *дисковая ОС* означает, что система эта хранится на магнитном диске и загружается в память компьютера при включении его.

Итак, ***загрузка операционной системы происходит с диска при включении компьютера.***

Рассмотрим более подробно этот процесс. Вспомни, что компьютер имеет два типа памяти: внутреннюю память и внешнюю – память на магнитных дисках. Внутренняя память ПК подразделяется на два вида:

- **ОЗУ (Оперативное Запоминающее Устройство) или по–английски RAM (Random Access Memory – память произвольного доступа) ;**
- **ПЗУ (Постоянное Запоминающее Устройство) или по–английски ROM (Read – Only Memory – память только для чтения).**

ОЗУ – это быстрая память компьютера, в которой должны находится выполняемые программы и данные для них. В этой памяти можно и писать, и читать информацию. Напомним, что ***содержимое ОЗУ стирается при выключении компьютера.***

ПЗУ – это память, в которой постоянно хранятся записанные на заводе программы, их можно использовать, но нельзя стереть. Из этой памяти можно только читать информацию. ***Информация в ПЗУ не стирается при выключении компьютера.***

Какие же программы содержит ПЗУ? Во–первых, это тестовые программы, которые проверяют работу основных устройств компьютера. Во–вторых, это программы, которые управляют внешними устройствами. Эти программы являются частью ОС, но такой ее частью, которая полностью зависит от модели компьютера. Поэтому эти программы записываются в ПЗУ при изготовлении ПК на заводе.

При включении компьютера сразу начинают работать тестовые программы. Результат проверки оперативной памяти показывается цифрами на экране дисплея.

Если все устройства ПК работают правильно, то одна из программ ПЗУ начинает искать ОС на магнитных дисках – как правило, сначала на дискетах, а потом на жестком диске.

Дискета, на которой записана ОС, называется ***системной дискетой.***

В состав дисковой операционной системы входят три основных программы: IO.SYS, MSDOS.SYS и COMMAND.COM. При

загрузке ОС программа **COMMAND.COM** считывается с диска и заносится в оперативную память. **Эта программа находиться в ОЗУ все время, пока включен компьютер, и обрабатывает все команды, которые вводит пользователь в ответ на приглашение ОС**.

9.2 Файловая система персонального компьютера

Любая информация хранится на магнитных дисках в виде ***файлов***. Каждая программа – это тоже файл (**file**). Числовые и другие данные, которые обрабатывает программа, могут храниться в виде файла. Даже текст учебника, который вы сейчас читаете, тоже записан в виде файла на магнитном диске – это текстовый файл.

Каждый файл имеет название – **имя файла (file name**).

Имя файла состоит из двух частей, которые разделены точкой:

1) основное имя, которое может содержать от одного до восьми символов;

2) расширение имени (file name extension), которое может содержать до трех символов.

Например, имя файла главной программы ОС – это COMMAND.COM ; здесь COMMAND – основное имя, а COM – расширение имени.

Имя файла нельзя задавать символами, разделенными пробелами. Например, "FILE10" – правильное имя, а "FILE 10" – неправильное, потому что между словом FILE и номером 10 есть пробел.

Список всех файлов на диске называется *каталогом* диска.

Файловая система ОС очень похожа на библиотеку, это видно из следующего сопоставления:

ОС	Библиотека
магнитный диск	полка с книгами
файл	книга
имя файла	название книги
каталог диска	каталог, список книг

Этой аналогией будем пользоваться и дальше.

Для чего нужно расширение имени файла? Выделим два случая, когда использование расширения является удобным и полезным.

1. Скажем, учебник состоит из нескольких глав. При записи его на магнитный диск удобнее содержание каждой главы хранить в отдельном файле. Например, UCH.GL1, UCH.GL2 и т.д. – файлы имеют одинаковые имена, но разные расширения.

2. В библиотеке книги группируются по разделам науки: "математика", "история" и т.п. Подобно этому удобно файлам, которые имеют одинаковое назначение, давать и одинаковые расширения. Например, все файлы, которые являются ***исполняемыми*** непосредственно программами, имеют расширение COM или EXE (***COM – Command, команда*** или ***EXE – Execute , выполнить***). Например, программа ОС – COMMAND.COM, программа Бейсик – GWBASIC.EXE .

Все файлы, которые содержат программы, написанные на языке Бейсик, имеют расширение BAS, на языке Паскаль PAS, файлы-документы, созданные в Word – DOC.

В библиотеке на полке может оказаться книга, которая сама является каталогом, например, она может содержать список картин в музее. Точно так же на диске могут быть файлы, которые содержат списки других файлов. Такие файлы называются ***подкаталогами*** *(*или ***каталогами****)*, в системе Windows их называют папками.

Если на диске имеется очень много разных файлов, то необходимо использовать каталоги. Например, большинство программ операционной системы можно записать в каталог с именем DOS, а все программы на языке Бейсик – в каталог BASIC. В этом случае список файлов на диске будет короче и можно легко найти нужные программы и файлы.

9.3 Основные команды ОС MS DOS

Назовем самые основные команды ОС MS DOS, которые потребуются нам для работы с файлами. Все команды ОС записываются определенными словами английского языка.

Система Windows позволяет работать и "по старинке" — используя команды системы MS DOS. В некоторых случаях действительно возникает такая необходимость. Например, при исполь-

зовании программ, созданных ранее в системе MS DOS и не работающих корректно в системе Windows.

Для перехода в сеанс MS DOS достаточно дать команду ***Пуск/Программы/Сеанс MS DOS.*** (В *Windows 2000* команда ***Пуск/Программы/Стандартные/Командная строка,*** в *Windows NT* команда ***Пуск/Выполнить/cmd*** **).** Окно при этом приобретет примерно такой вид (см. *рис. 23*)

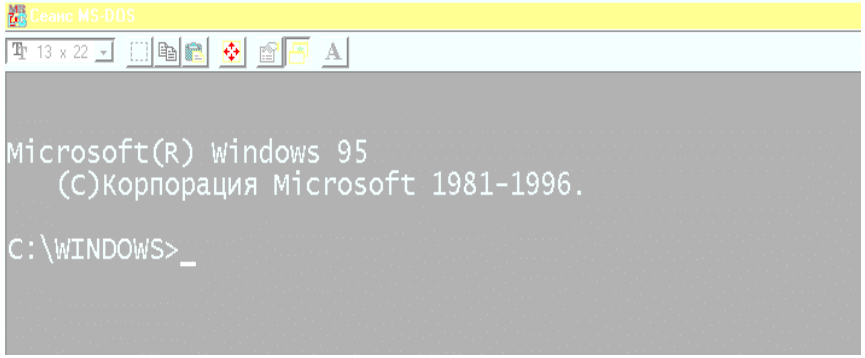


Рис. 23 Окно сеанса MS DOS

Здесь, как и во всяком окне Windows, имеется панель с инструментами, которая позволяет сменить шрифты, вырезать (а, значит, и скопировать) фрагмент экрана, развернуть окно на весь экран и др. (В *Windows 2000* в окне *командной строки* такой возможности нет). В последнем случае будет полная иллюзия работы в системе MS DOS. Теперь, чтобы вернуться к другим окнам Windows, следует нажать клавиши **Alt+Tab**. Заметим, что если мы вновь захотим вернуться в окно MS DOS, то оно теперь будет развернуто на весь экран. Чтобы вернуть его к нормальному — оконному, а не полноэкранному размеру, следует в Панели задач щелкнуть правой клавишей мыши по прямоугольничку "сеанс MS DOS", выбрать Свойства, выбрать вкладку Экран и сменить режим с полноэкранного на оконный.

После того, как мы перешли в сеанс MS DOS, можно задавать команды этой ОС. Многие из приведенных ниже команд без труда выполняются и в самой Windows . Однако может возникнуть необходимость их "ручного выполнения". Заметим, что команды MS DOS условно делятся на *внутренние* и *внешние*. Для

выполнения внутренних команд достаточно просто загрузить MS DOS — они содержатся в командном процессоре COMMAND.COM. Для выполнения внешних — нужны специальные файлы–утилиты. В данном параграфе мы будем рассматривать только внутренние команды (да и то лишь немногие).

1. Команда *просмотра оглавления* диска

DIR *диск* **(DIRectory — каталог)**

Например,

DIR A: — просмотр оглавления диска на дисководе A:

DIR C: — просмотр оглавления диска C:

Существует возможность с помощью этой команды:

- сделать *постраничную* выдачу оглавления (если весь каталог не умещается на экран и тогда всякий раз для продолжения просмотра следует дополнительно нажимать любую клавишу):
  DIR C:/P
- сделать *краткий просмотр* оглавления (не столбцом, а вытянуть список файлов по строкам):
  DIR C:/W
- сделать просмотр списка *группы файлов* по заданному шаблону (маске): DIR P*.* — выдает список файлов, наименование которых начинается с буквы P (в текущем каталоге)
  DIR A: *.EXE — выдает список файлов с расширением EXE (в корневом каталоге диска A:).

2. *Переназначение* текущего устройства

C> A:

После выполнения такой команды рабочим станет дисковод A: (соответственно на экране будет системное приглашение в виде: A>).

3. Вызов программы на *выполнение*

A> *имя файла*

где в качестве *имени файла* может быть только файл с расширением EXE, COM или BAT. Причем, можно указывать и другое устройство:

A> C:NC — запуск NC из корневого каталога диска C:

4. *Копирование* файлов

COPY *имя исходного файла имя конечного файла*

где *имя исходного файла* и *имя конечного файла* могут включать и указание устройств (A:,C: и др.).

Причем, если хотите скопировать файл под прежним именем, но в другой каталог (или даже на другой диск), необходимо указывать на месте имени конечного файла только название диска, затем (через \ ) название каталога.

Например, для копирования файла TEXPRAV из каталога LEXICON (в котором мы находимся) в каталог MY нужно дать команду вида:

COPY TEXPRAV C:\MY

Копирование можно выполнять и для *объединения группы файлов в один файл*. Например, фрагменты текста объединить в один файл:

COPY *имя 1+имя 2+...+имя K имя*

Здесь файлы *имя 1, имя 2, ..., имя K* объединяются в один файл *имя.*

Команда COPY позволяет и создать новый (текстовый) файл непосредственно с клавиатуры:

COPY con *имя файла*

затем набираем текст строку за строкой. В конце каждой строки нужно нажимать ENTER, а после ввода последней нажать F6 и ENTER.

5. *Удаление* файла

DEL *имя файла* **(DELete — удалять)**

где *имя файла* может включать и указание устройства и каталога. При удалении можно задавать шаблон (маску). Например,

DEL A:*.BAK — удаление всех файлов с расширением BAK в корневом каталоге диска A: .

6. *Переход* в другой каталог

CD *имя каталога* **(Change Directory — сменить каталог)**

7. *Переименование* файла

REN *имя исходное имя конечное* **(REName — переименовать)**

Например, REN skas skaska - переименование файла skas в файл skaska .

8. *Создание* нового каталога

MD *имя каталога* **Make Directory — создать каталог)**

Например, MD PROBCAT — создание каталога PROBCAT в текущем каталоге. Заметим, что, так как новый каталог всегда создается в текущем каталоге, то предварительно, если нужно, следует выйти в нужный каталог (или корневой каталог).

9. *Выход в корневой каталог* делается командой

CD \

Напомним, что если из одного каталога нужно перейти в другой, то предварительно надо выйти в корневой каталог, а затем войти в нужный.

10. *Уничтожение* каталога

RD *имя каталога* **(Remove Directory — удалить каталог)**

Здесь также следует помнить, что удаляется указанный подкаталог в текущем каталоге. **Внимание**: в системе MS DOS удалить каталог можно лишь, если в нем нет никаких других файлов. Иначе предварительно следует удалить эти файлы, а уж затем и каталог.

11. *Вывод файла* на экран

TYPE *имя файла*

12. Вывод файла на *печать*

COPY *имя файла* PRN

13. *Очистка экрана* дисплея

CLS **(CLear Screen — чистый экран)**

Замечание. Ранее в системе MS DOS существовало жесткое правило: файлы и каталоги должны иметь имена, заданные только латинскими буквами и не более восьми символов в имени. В сеансе MS DOS из-под WINDOWS это правило не столь жестко — допускается использование русских букв, однако, по-прежнему не более восьми символов. Напомним, что для переключения в режим русских букв в сеансе MS DOS используют **CTRL+Shift** (правые), а для обратного переключения — в латинские буквы — **CTRL+Shift** (левые).

9.4 Оболочка FAR MANAGER (FAR)

9.4.1 Общая характеристика FAR

Оболочка FAR 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. Более того, ряд операций, к которым привыкли пользователи FAR, до сих пор присутствуют только в FAR. Например, выделение группы файлов по маске (шаблону) или сравнение содержимого двух каталогов. Именно на такого рода особенности мы и будем далее обращать внимание.

Мы рассмотрим специальную версию FAR для Windows. Особенность ее в том, что она позволяет активно использовать как клавиатуру (для тех, кто привык именно ее применять в FAR), так и манипулятор «мышь».

Для запуска FAR из-под Windows, как правило, следует дать команду ***Пуск/Программы/Far Manager/Far manager***. Экран при этом приобретет примерно такой вид — см. *рис. 24*.

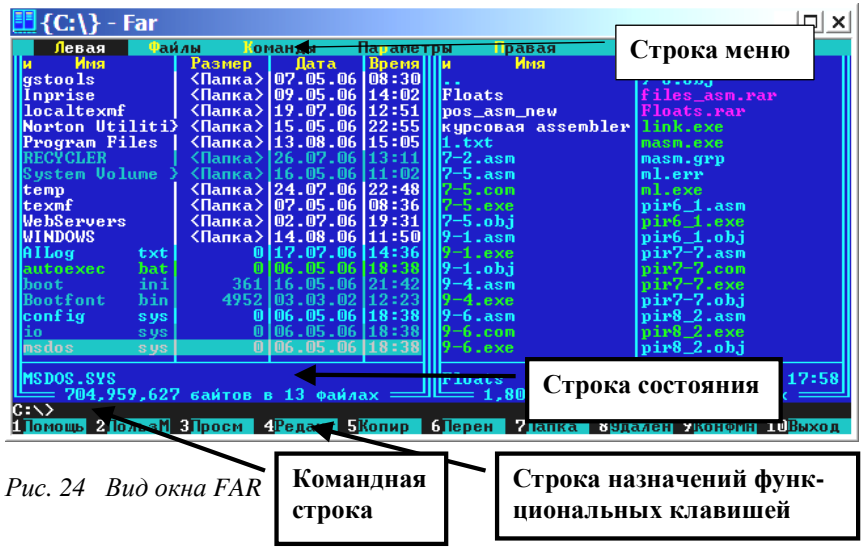


Рис. 24 Вид окна FAR

Сверху находится *строка меню*. 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 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. левую панель), так и в *кратком* (см. на рис. правую 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 (подробнее в п. 9). 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 для Windows.

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.

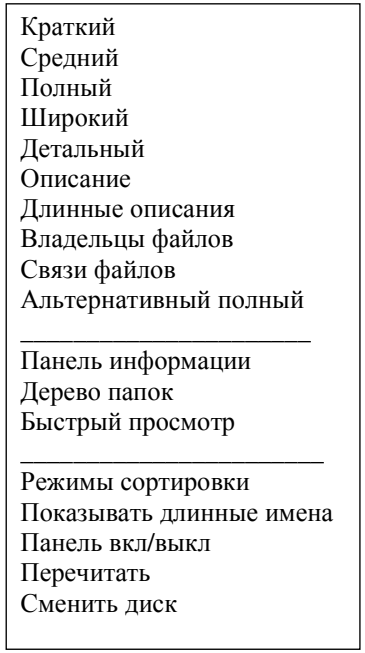
4- Редакт. Позволяет отредактировать активный файл. Опять же это можно делать только с текстовыми файлами.

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 (папки) внутри активного каталога или диска.

8- Удален. Удаление выбранного файла (или группы файлов). Заметим, что, в отличие от удаления в системе Windows, здесь



файлы не попадают в Корзину, а удаляются вовсе! (Хотя специальными настройками можно обеспечить сбрасывание в Корзину и здесь)

9- КонфМ. Выход в меню для выполнения операций в FAR.

10- Выход. Выход из FAR.

Заметим, что пункты меню, помощь и сообщения по умолчанию настроены по-английски. Для настройки русского языка достаточно дать команду **F9/Option/Languages** и выбрать **Russian.** Сохранить настройки можно командой **F9/Параметры/ Сохранить параметры**.

9.4.2 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 панель* (или *Правая панель*):

Команды первой группы (*Краткий* и др.) служат для отображения информации на панели в нужном виде;

Команды второй группы служат для отображения информации о компьютере, диске и файлах:

Дерево папок — дерево каталогов диска на другой панели,

Быстрый просмотр — вывод содержимого файла с *другой* панели, на котором находится курсорная рамка; это имеет смысл делать только для файлов-документов.

Команды третьей группы служат для упорядочения (сортировки) файлов др.

9.4.3 Файловые операции в FAR

Для пользователя, знакомого с системой Windows, операции просмотра или редактирования файла не вызывает затруднений (для этого в FAR служат соответственно клавиши **F3** и **F4**). Поэтому далее рассмотрим только отличительные особенности FAR. Одна из таких особенностей – операции копирования и переноса файла (группы файлов).

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 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). При удалении – просто сбрасывание в Корзину. Можно указать фильтры (если щелкнуть по кнопке *Фильтры*) для отделения файлов определенного вида (по имени, дате создания и проч.)

Напомним, что *смена активного диска на панели* может быть произведена командой ***Левая панель (или Правая панель)/ Сменить диск…*** и выбором соответствующего диска или клавишами:

Alt – F1 – смена диска на левой панели,

Alt – F2 – смена диска на правой панели.

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, 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 OK. 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*** — появится окно диалога,
- указать нужный атрибут (поставить "крестик" клавишей «Пробел» в нужном месте блока "Установить"),
- 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/Настройки панели***, появится окно диалога ,
- пометить или *снять* "крестить" у "Показывать скрытые и системные файлы",
- щелкнуть Продолжить

9.4.4 Архивация файлов в FAR

Одной из особенностей рассматриваемой версии FAR является возможность проведения так называемой архивации файлов. Что это такое?

Поскольку для программ рекомендуется иметь страховочную копию, то копию лучше хранить на гибких дисках в виде так называемых *архивных файлов*. Существуют различные способы архивации файлов. В настоящее время наиболее распространенной является архивация типа ZIP. Она позволяют существенно (в среднем на 50 процентов) сократить объем файлов, причем в один архивный

файл можно поместить сразу группу совместно используемых файлов. Это позволит гораздо легче разобраться в архиве файлов.

Итак, **порядок архивации**:

- перейти в нужный каталог и выделить нужные для включения в архив файлы
- дать команду ***Файлы/Архивировать***
- в возникшем окне диалога указать *Имя архива* (обязательно дописать каталог, в который следует поместить архивный файл и имя для архивного файла)
- указать *Архиватор* (рекомендуется ZIP)
- указать (если необходимо) *Пароль* и нажать *Добавить*
- пойдет процесс архивации (может длиться довольно долго — зависит от размеров и количества архивируемых файлов) и в результате создается архивный файл (с расширением zip)
- архивируемые файлы (т.е. включенные в архив) можно удалить, а созданный архивный файл — скопировать (или перенести) на гибкий магнитный диск

Порядок разархивации:

- перейти в каталог, где находится архивный файл или создать отдельный каталог, в который поместить требуемый для разархивации файл (с расширением zip) и установить курсор на архивный файл
- дать команду ***Файлы/Распаковать***
- в возникшем окне диалога указать *Куда* (обязательно дописать каталог, в который следует поместить файлы, "добытые" из архивного файла)
- пойдет процесс разархивации (если был установлен пароль при архивации, то здесь потребуется его ввести)
- чтобы не загромождать, архивный файл следует удалить

9.5 Архивация файлов программой WinZip

В системе Windows имеется целый ряд специальных программ для архивации файлов. Рассмотрим наиболее популярную программу WinZip (версии 8.0).

Если она установлена, то для ее запуска достаточно дать команду ***Пуск/Программы/WinZip/WinZip 8.0***. При этом появится окно вида (см. *рис. 25*).

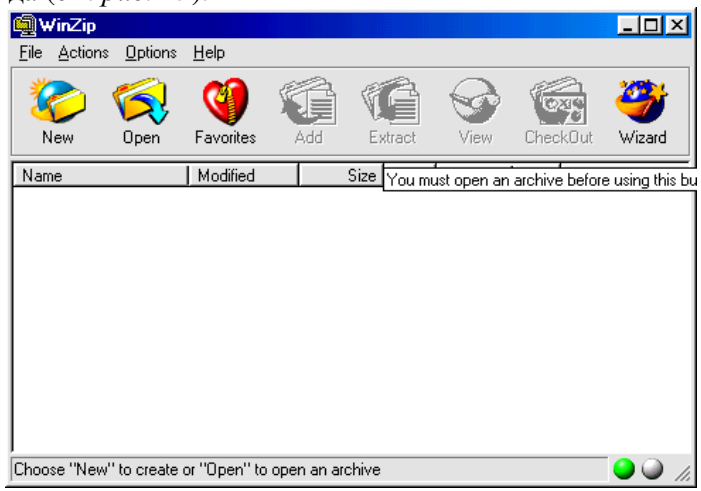


Рис. 25 Окно диалога при запуске WinZip

Для *создания архива* необходимо выбрать кнопку ***New*** (или дать команду ***File/New Archive***…), затем перейти в нужную папку и задать имя для архивного файла (т.е. то имя, под которым вы сохраните создаваемый архив). В очередном окне диалога выбрать нужные для архивации файлы (их может быть несколько – тогда используйте дополнительно клавишу **Ctrl**) и щелкнуть по кнопке ***Add*** (добавить). В указанной папке появится архивный файл.

Для *разархивации* нужно, конечно, также запустить WinZip, щелкнуть по кнопке ***Open*** (дать команду ***File/Open Archive***…), перейти в папку, где находится архив, выбрать этот архивный файл и нажать кнопку **Открыть**. В окне архиватора появится список файлов, «спрятанных» в данном архиве. Выделить нужные и щелкнуть ***Extract*** (или дать команду ***Action/Extract***). В очередном окне можно сменить папку для размещения разархивированных файлов и щелкнуть ***Extract***.

Замечание. При установке WinZip она «прописывается» и в контекстном меню. Поэтому для *архивации* достаточно перейти в нужную папку, выделить нужные файлы и, стоя на одном из них,

вызвать контекстное меню (правой клавишей мыши) и выбрать **Add to Zip**, указать имя архива и его размещение, затем щелкнуть **Add**.

Для *разархивации* – вызвать контекстное меню (стоя на файле – архиве), выбрать пункт **Extract to…** и указать путь размещения разархивируемых файлов и щелкнуть **Extract**.

9.6 Антивирусы

На современных компьютерах очень большой вред могут принести так называемые вирусы. Они могут испортить данные на диске, нарушить работу ОС и т.д. Распространяются вирусы, как правило, при копировании с дискеты на дискету и далее на винчестер. Но на сегодняшний день самый распространенный способ передачи вирусов – по компьютерным сетям (в том числе и по сети Internet). Поэтому для борьбы с вирусами необходимо регулярно проводить *антивирусную профилактику* дисков (и дискет).

9.6.1 Антивирусная программа AVP

На современных компьютерах наиболее популярной антивирусной программой является AVP Касперского (AntiViral Toolkit Pro). Работает она в среде Windows, при установке требуется регистрация и при этом она позволяет легко обновлять на Web-узле этой программы антивирусные базы[2].

Если программа AVP установлена на компьютере, то для ее запуска на проверку и лечение достаточно дать команду ***Пуск/Программы/ AntiViral Toolkit Pro/AVP Сканер***. Появится окно вида (*рис. 26*).

[2] Дело в том, что практически каждый день появляются новые вирусы и поэтому разработчики антивирусных программ регулярно пишут к ним антивирусы, добавляя их в антивирусную базу. По этой причине и требуется регулярно обновлять антивирусные базы с Web-узла антивирусной программы.

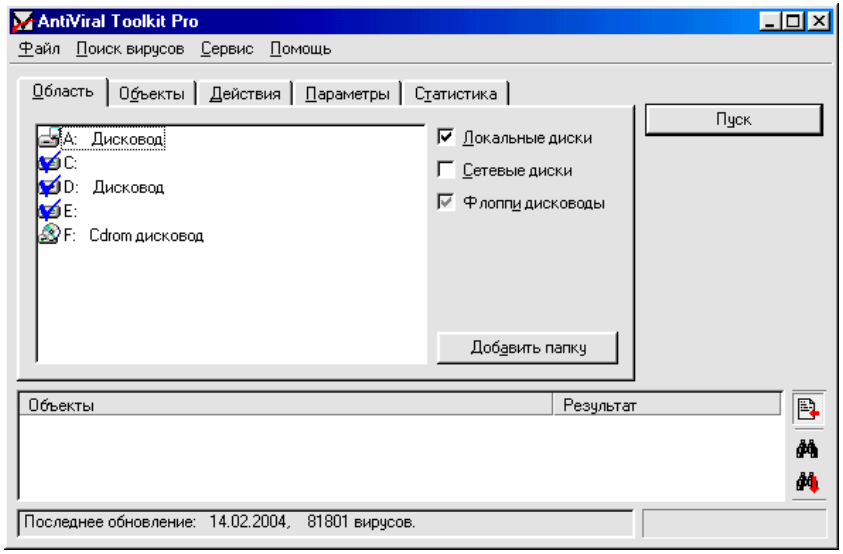


Рис. 26 Окно программы AVP Сканер

Здесь необходимо отметить те диски, для которых хотим провести антивирусную профилактику. Затем на вкладке **Объекты** следует указать *Все файлы*, а на вкладке **Действия** – *Лечить без запроса*. После чего нажать кнопку **Пуск**. Заметим, что при полной установке данной программы среди прочего установится и так называемый AVP Монитор, который постоянно «сидит» в оперативной памяти компьютера и сигнализирует при появлении (например, при копировании с дискеты) файла, зараженного вирусом. Рекомендуем, кстати, всякий раз, как собираемся копировать файлы, предварительно их проверять на вирусы. Для этого достаточно стать на проверяемый файл, вызвать контекстное меню, в нем выбрать пункт ***AntiViral Toolkit Pro.***

9.7 Создание меню пользователя в FAR

В предыдущем параграфе при изучении FAR, мы отложили рассмотрение очень важной возможности — *создание меню пользователя*, которое вызывается клавишей **F2**. Теперь, когда мы познакомились с командами MS DOS, вернемся к этому пункту. Сразу отметим, что создание такого меню позволит нам в даль-

нейшем облегчить выполнение наиболее ходовых операций. Например, быстрый переход в нужный подкаталог и автоматический запуск затем нужной нам программы.

Итак, при запущенном FAR для WINDOWS , если нажать на клавишу F2, то появится окно, в котором для ввода команды меню надо нажать клавишу INS, для удаления ранее созданной команды – клавишу Del, а для редактирования – клавишу F4

Причем следует в пункте *Горячая клавиша* указать клавишу для вызова данного пункта меню, в пункте *Метка* ввести название, а в пункте *Команды* задать команды MS DOS, которые и обеспечат выполнение нужной нам операции.

Далее следует нажать *Продолжить* и сохранить созданный пункт меню. Можно, впрочем, тут же создать еще несколько пунктов меню. Затем, когда будем вызывать данное меню, то после нажатия F2 будет появляться окно со списком всех команд и их можно выбирать и запускать клавишей *Enter*.

Замечание. Ранее в системе MS DOS существовало жесткое правило: файлы и каталоги должны иметь имена, заданные только латинскими буквами и не более восьми символов в имени. В сеансе MS DOS из-под WINDOWS это правило не столь жестко — допускается использование русских букв, однако, по-прежнему не более восьми символов. Напомним, что для переключения в режим русских букв в сеансе MS DOS используют **CTRL+Shift** (правые), а для обратного переключения — в латинские буквы — **CTRL+Shift** (левые).

ЛИТЕРАТУРА К ГЛАВЕ 1

1. Симонович С.В. Информатика. Базовый курс. Учебное пособие для студентов втузов. - СПб.: Питер, 2001. - 640 с
2. Фигурнов В.Э. «IBM PC для пользователя». М.: ИНФРА М, 1997 – 478 с.
3. Могилев А.В. и др. Информатика. Учебное пособие для студ. пед. вузов. – М.: «Академия», 2003. – 816 с.
4. Столлинг У. Структурная организация и архитектура компьютерных систем. – М.: Издательский дом «Вильямс», 2002. – 896 с.
5. Алексеев А. Информатика 2003: Учебное пособие - 3-е изд.. - М.: СОЛОН-Пресс, 2003. – 463 с.

Глава 2 ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА i8086

ВВЕДЕНИЕ

Под термином «Персональный компьютер» мы будем в дальнейшем понимать персональную ЭВМ, созданную на базе микропроцессоров семейства 80x86 фирмы Intel (8086, 80286, i386, i486, Pentium).

Первый микропроцессор (процессор, реализованный в виде одной микросхемы) появился в 1971 г. Его создала фирма Intel, которая с тех пор остается лидером в области разработки микропроцессоров. Он работал с 4-разрядгыми данными. В 1974 г эта же фирма создала микропроцессор 8080, работающий с 8-разрядными машинными словами и памятью до 64 Кб. В 1976 г появилась первая персональная ЭВМ, т.е. процессор плюс память и устройства ввода-вывода, разработанная фирмой Apple.

В 1978 г фирма Intel разработала микропроцессор нового поколения – 16-разрядный процессор 8086 с памятью до 1 Мб. В 1979 г появился его вариант – микропроцессор 8088, который также работал с 16-разрядными словами, но использовал 8-разрядную шину (в процессоре 8086 была 16-разрядная шина), что позволило воспользоваться имеющимися в то время внешними устройствами (дисководами и т.п.) с 8-разрядными соединениями. На базе этого процессора фирма IBM в 1981 г создала свой первый персональный компьютер (ПК) под названием IBM PC (personal computer). Именно с этого времени началось широкое распространение ПК в мире. Чуть позже (1983 г) фирма IBM создала усовершенствованную модель ПК – IBM XT (eXtended Technology).

В 1983 г фирма Intel разработала микропроцессор 80186, но он практически не использовался, т.к. в том же году появился более совершенный микропроцессор 80286. На его основе IBM в 1984 г построила свой очередной ПК – IBM AT (advanced technology). В процессоре 80286 предусмотрены аппаратные средства

для реализации многозадачного режима работы ЭВМ. Но реально многозадачный режим начал использоваться только с появлением 32-разрядныз процессоров. В 1987 г фирмой Intel был создан процессор i386, а в 1990 г – процессор i486. Они могут работать в двух режимах – в реальном режиме, в котором они фактически представляют собой очень быстрые варианты процессора 8086, и в защищенном режиме, позволяющем реализовать многозадачность. В 1993 г фирма Intel разработала 64-разрядный микропроцессор, получивший собственное имя Pentium.

Все указанные процессоры объединяют в семейство 80x86, поскольку в них соблюдается преемственность: программа, написанная для младшей модели, может без каких-либо изменений выполнена на любой более старшей модели. Обеспечивается это тем, что в основе всех этих процессоров лежит система команд процессора 8086, в старшие модели лишь добавляется новые команды (главным образом, необходимые для реализации многозадачного режима). Таким образом, процессор 8086 – это база, основа для изучения всех остальных моделей данного семейства. Именно эта база нас и будет интересовать (многозадачный режим мы рассматривать не будем). Поэтому в дальнейшем под сокращением ПК будем понимать персональный компьютер с процессором 8086.

Оперативная память

Напомним, что 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:

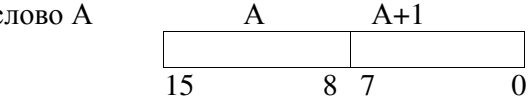
При этом правые разряды (с меньшими номерами) называются младшими, а левые разряды – старшими. В каждом разряде может быть записана величина 1 или 0, такую величину принято называть *бит* (bit). Таким образом, содержимое любого байта – это набор из 8 битов, из 8 нулей и единиц.

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).

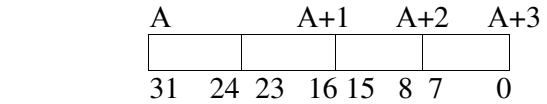
Байты нумеруются начиная с 0, порядковый номер байта называется его адресом. Объем оперативной памяти ПК – 2^{20} байтов (1 Мб), поэтому для ссылок на байты памяти нужны 20-разрядные адреса – от 00000h до FFFFFh.

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
CX counter, счетчик
DX data, данные

Эти четыре регистра есть *регистры данных.*

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

Следующие четыре регистра есть *регистры указателей*:

SI source index, индекс источника
DI destination index, индекс приемника
BP base pointer, указатель базы
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 данной главы) 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 три файла: **masm.exe, ml.exe, 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*

Еще одна команда из группы сложения и вычитания: NEG op – команда изменения знака.

И еще пара команд сложения и вычитания:

Сложение с учетом переноса (add with carry): ADC op1,op2

Вычитание с учетом переноса (subtract with borrow): SBB op1,op2

Отличие от обычных команд сложения и вычитания в следующем: в команде ADC к сумме операндов еще прибавляется значение флага переноса CF : op1:=op1+op2+CF, а в команде SBB из разности операндов еще вычитается значение этого флага: op1:=op1–op2–CF.

Зачем нужны такие команды? Например, для сложения чисел размеров в двойное слово. Пусть X=1204F003h и Y=8052300Fh. Условно разобъем каждое число на два слова. Сначала складываем младшие (правые) части их, используя команду ADD. Может получится единица переноса, которую надо учесть при сложении старших (левых) частей чисел. И поскольку единица переноса попадает в флаг CF, то далее применяем команду ADC. Если для определенности считать, что число X размещается в двух регистрах AX (старшие цифры) и BX (младшие), а число Y в регистрах CX (старшие цифры) и DX (младшие), а сумму этих двух чисел надо записать в X, т.е. надо реализовать (AX,BX):=(AX,BX)+(CX,DX), то это делается так:

```
ADD BX,DX   ; BX:=Xмл+Yмл, CF - перенос
ADC AX,CX   ; AX:=Xст +Yст+CF
```

(Отметим, что при сложении старших частей также может появиться единица переноса, однако мы ее уже не будем учитывать).

Аналогично делается и вычитание беззнаковых чисел в двойное слово. Например, вычитание (AX,BX):=(AX,BX)–(CX,DX) делается так:

```
SUB BX,DX   ; BX:=Xмл–Yмл, CF - перенос
SBB AX,CX   ; AX:=Xст –Yст–CF
```


Команда умножения:

Если сложение и вычитание беззнаковых и знаковых чисел производятся по одним и тем же алгоритмам, то умножение чисел этих двух классов выполняются по разным алгоритмам:

Умножение целых без знака (multiply): MUL *источник*

Умножение целых со знаком (integer multiply): IMUL *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)

Обратим внимание, что в командах только один операнд. Это потому, что первый сомножитель всегда находится в регистре AL (при умножении байтов) или в AX (при умножении слов). Результат записывается:

- при умножении байтов (результат будет вообще говоря, слово) заносится в регистр AX (в AH – старшие цифры произведения, в AL – младшие)

- при умножении слов результатом будет двойное слово и оно записывается в два регистра: в DX заносятся старшие цифры произведения, в AX – младшие.

Команда деления:

Как и умножение, деление чисел без знака и со знаком также реализуется двумя командами:

Деление целых без знака (divide): DIV *источник*

Деление целых со знаком (integer divide): IDIV *источник*

При делении слова на байт:

AH:=AX mod op, AL:=AX div op,

т.е. *делимое* берется из AX и *частное* возвращается в регистре AL, а остаток – в регистре AH (здесь операцией **mod** обозначили получение остатка от деления, а **div** – целочисленное деление).

При делении двойного слова на слово:

DX:= (DX,AX) mod op, AX:=(DX,AX) div op,

т.е *делимое* берется из DX и AX и *частное* возвращается в регистре AX, а остаток – в DX

Т.е. при делении слова на байт делимое обязано находится в регистре AX, а делитель должен быть байтом. При делении двойного слова на слово делимое обязано находится в двух регистрах – в DX (старшая часть делимого) и в AX (младшая часть), а делитель должен иметь размер слова.

В области целых чисел в результате деления получают неполное частное (div) и остаток (mod). Оба этих числа помещаются на место делимого: старшая часть заменяется на остаток, младшая часть – на неполное частное.

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[3]), 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;

(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

[3] Напомним, что для использования MASM версии 6.11 и старше достаточны (минимально) следующие файлы: ML.EXE, MASM.EXE, LINK.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)[4]:

```
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):

```
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

[4] 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.

```
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 разделу 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 разделе 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 задается, 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 разделу 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!!*

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

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 разделу 3

Выполните те же примеры, что и в заданиях к **разделу 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. Здесь использована конструкция повторения DUP (от английского слова *duplicate – повторять*), которая и означает, что значение нужно повторить (в данном случае 100 раз).

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 вы-

числять 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 разделу 4

Часть 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 Р.
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 Р.
А4-17. Дан массив из *10* элементов (натуральные числа). Подсчитать количество элементов, отличающихся от заданного числа Р.
А4-18. Дан массив из *10* элементов (натуральные числа). Подсчитать сумму элементов, меньших заданного числа Р.
А4-19. Дан массив из *10* элементов (натуральные числа). Подсчитать количество элементов, меньших заданного числа Р.
А4-20. Дан массив из *10* элементов (натуральные числа). Подсчитать сумму элементов, больших заданного числа Р.
А4-21. Дан массив из *10* элементов (натуральные числа). Подсчитать количество элементов, больших заданного числа Р.
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).

Часть 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. Необходимость оформления процедур еще и в том, что команд в программе может быть столь много, что они не помещаются в сегмент размером в 64 К байт (именно такой максимальный размер выделяется под отдельный сегмент). В этом случае часть программы оформляют в виде отдельной внешней процедуры.

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
```

Еще раз напомним, что в компьютере адрес команды, которая должна выполнятся следующей задается парой регистров CS, IP: регистр CS указывает на начало сегмента памяти, в котором находится эта команда, а в регистре IP находится смещение этой команды, отсчитанное от начала данного сегмента. Изменение любого из этих регистров есть ничто иное, как переход, поскольку меняется адрес команды, подлежащей выполнению.

Если меняется только IP, то это означает переход внутри сегмента. Такие переходы называют *близкими* или *внутрисегментными*. Но если в программе имеется несколько сегментов команд, тогда возникает потребность в переходах из одного такого сегмента в другой. Такие переходы называют *дальними* или *межсегментными*. При этих переходах меняется и значение регистра CS, и регистра IP: CS устанавливается на начало сегмента с меткой (CS:=C2), а в IP записывается смещение метки внутри ее сегмента (IP:= offset L).

В компьютере предусмотрены команды, реализующие такие дальние переходы, причем, все они являются только безусловными переходами (условные переходы всегда близкие), прямыми или косвенными.

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 разделу 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+v², 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 необходимого 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 разделу 6

Доработать программы своего варианта практических заданий раздела 2 и раздела 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
    ret         ; (в CX счетчик числа цифр)
vyvod4  ENDP
c_seg  ends
end _BEG
```

Практические задания к разделу 7

Доработайте программы заданий раздела 3 и раздела 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 разделу 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
Директива определения числа	dd	dq	dt

Схематично представление чисел в таких форматах выглядит так:

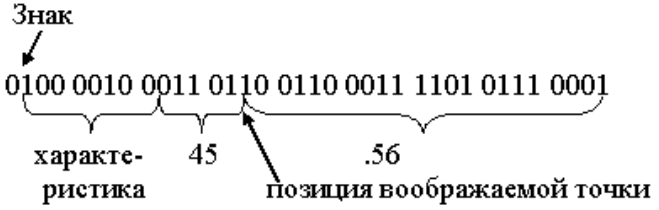
знак s	**Характеристика q**	**Мантисса (M)**	***Короткий формат***
31	24	23 0	
знак s	Характеристика q	Мантисса (M)	*Длинный формат*
63	53	52 0	
знак s	Характеристика q	Мантисса (M)	*Расширенный формат*
79	64	63 0	

В итоге числа записывают в виде:

$$A = (-1)^S \bullet N^q \bullet M$$

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 веществен-

ном, 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
```

Сохраним, 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. Делается 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. 27*).

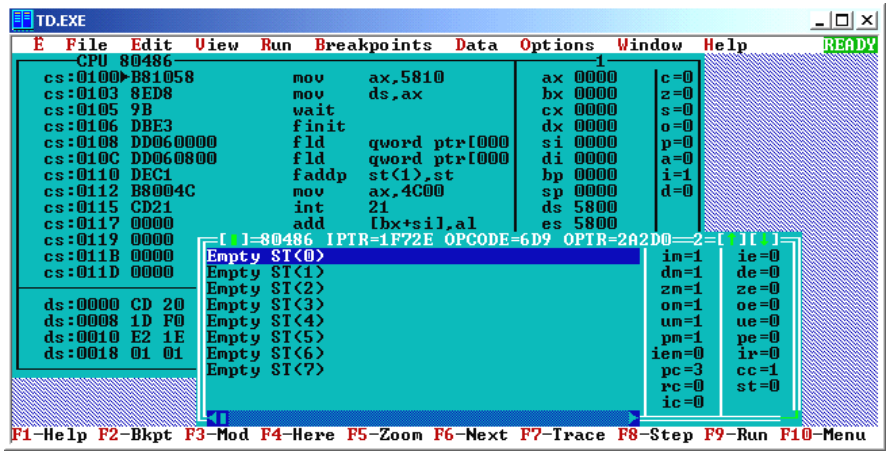


Рис. 27 Окно Turbo Debugger с окошком Numeric processor

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 соответствующего 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. 28*)

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.

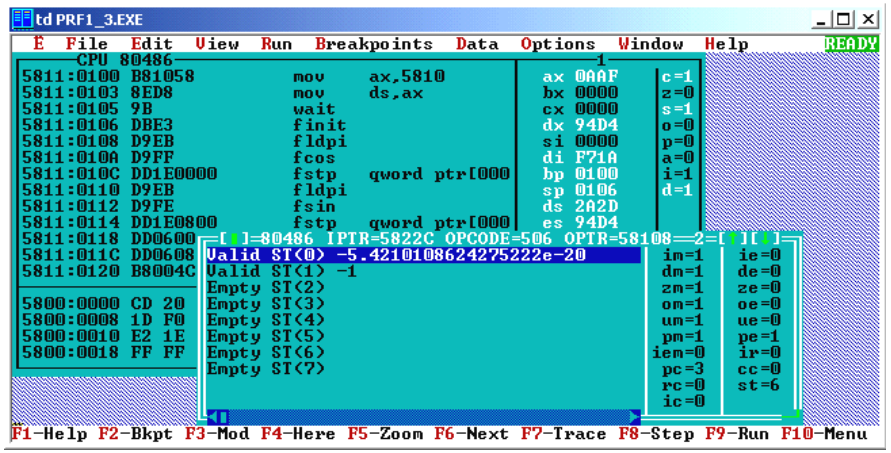


Рис. 28 Результат (в Turbo Debugger) работы программы вычисления Sin и Cos величины π радиан

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

$$x^y = 2^{y \bullet \log_2 x}$$

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. 29*)

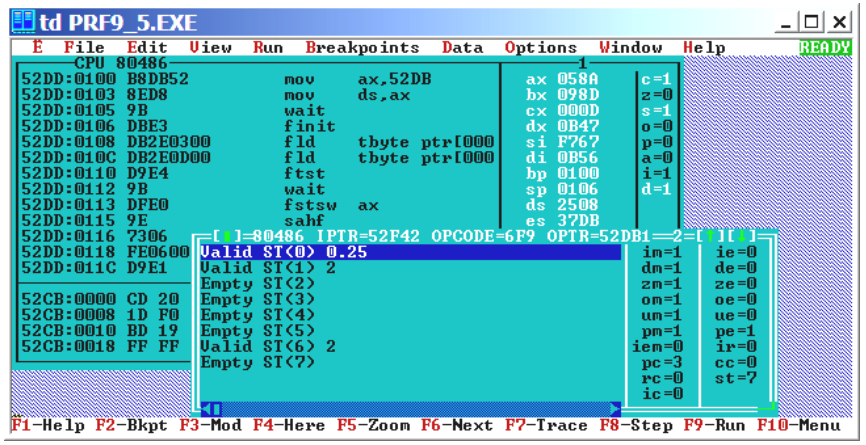


Рис. 29 Результат (в Turbo Debugger) работы программы вычисления 2^{-2}

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

Практические задания к разделу 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) $-7{,}88\cdot 10^{-2}$ 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 К ГЛАВЕ 2

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 1. 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 2. 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.

ТЕСТЫ

Тесты по общим вопросам информатики[5]

1. Какой вид памяти отсутствует у персонального компьютера?
A) оперативная B) наружная C) внешняя
D) кэш-память E) постоянная

2. Что не входит в состав аппаратного обеспечения персонального компьютера?
A) клавиатура B) процессор C) операционная система
D) монитор E) мышь

3. Что не принадлежит к устройствам ввода персонального компьютера?
A) клавиатура B) мышь C) сканер
D) линия связи E) монитор

4. Что принадлежит к устройствам вывода персонального компьютера?
A) клавиатура B) мышь C) сканер
D) принтер E) CD-ROM

5. Что не входит в состав программного обеспечения персонального компьютера?
A) жесткий диск B) операционная система
C) трансляторы с языков программирования
D) текстовые редакторы E) табличные процессоры

6. Сколько значений может принимать 1 бит?
A) одно B) два C) восемьD) шестнадцать E) 256

7. Сколько бит содержит 1 байт?
A) один B) два C) восемьD) шестнадцать E) 256

[5] Тесты с 35 по 60 данного раздела взяты с сайта www.fepo.ru

*8. Число 16 десятичной системы запишется в двоичной системе:
A) 32 B) 8 C) 10000 D) 10 E) 1A

*9. Сколько будет $4_{10} + 6_{10}$ в двоичной системе?
A) 1010 B) 10 C) 10000 D) A E) 8

*10. Сколько будет $10_2 + 110_2$ в десятичной системе?
A) 120 B) 8 C) 100 D) 11010 E) 10

*11. Сколько будет $1001_2 + 111_2$ (в той же двоичной системе)?
A) 1112 B) 10001 C) 11111 D) 0 E) 10000

*12. Число 16 десятичной системы запишется в 16-ричной системе:
A) A B) B C) C D) 10 E) 11

*13. Сколько будет $4_{16} + 10_{16}$ в десятичной системе:
A) 20 B) 14 C) 6 D) E E) 104

*14. Сколько будет $4_{10} + 6_{10}$ в 16-ричной системе?
A) 10 B) B C) C D) D E) A

*15. Сколько будет $1001_2 + 111_2$ в 16-ричной системе:
A) 10 B) B C) C D) D E) 16

16. О чем говорит расширение EXE в наименовании файла?
A) это – файл-документ B) это исполняемый файл
C) это командный файл D) это программный файл
E) это файл с рисунком

17. О чем говорит расширение BMP в наименовании файла?
A) это – файл-документ B) это исполняемый файл
C) это командный файл D) это программный файл
E) это файл с рисунком

18. О чем говорит расширение COM в наименовании файла?
A) это – файл-документ B) это исполняемый файл

C) это командный файл D) это программный файл
E) это файл с рисунком

19. О чем говорит расширение DOC в наименовании файла?
A) это – файл-документ B) это исполняемый файл
C) это командный файл D) это программный файл
E) это файл с рисунком

20. Дана структура:

```
                  |— ПАПКА10
C:ПАПКА1 ——       |— ПАПКА11    |—— ПАПКА12А
                  |— ПАПКА12 —|—— ПАПКА12Б
```

Указать правильную форму записи пути к файлу PROBA.TXT, находящемуся в ПАПКЕ11
A) C:\ПАПКА11\PROBA.TXT
B) C:ПАПКА11\PROBA.TXT
C) C:/ПАПКА1/ПАПКА11/PROBA.TXT
D) C:/ПАПКА11/PROBA.TXT
E) C:\ПАПКА1\ПАПКА11\PROBA.TXT

21. Дана структура:

```
                  |— ПАПКА10
C:ПАПКА1——— |— ПАПКА11    |—— ПАПКА12А
                  |— ПАПКА12 —|—— ПАПКА12Б
```

Указать правильную форму записи пути к файлу TEXT21.DOC, находящемуся в ПАПКЕ12Б

A) C:\ПАПКА1\ПАПКА12\ПАПКА12Б\ TEXT21.DOC
B) C:/ПАПКА1/ПАПКА12/ПАПКА12Б/ TEXT21.DOC
C) C:\ПАПКА1\ПАПКА12\ПАПКА12А\ПАПКА12Б\ TEXT21.DOC
D) C:/ПАПКА1/ПАПКА12/ПАПКА12А/ПАПКА12Б/ TEXT21.DOC
E) C:\ПАПКА12Б\TEXT21.DOC

22. Сколько бит содержится в двух байтах?
A) 8 B) 16 C) 32 D) 2^2 E) 8^2

23. Сколько байт в одном килобайте?
A) 2^9 B) 1000 C) 1024 D) $2^{10} - 1$ E) 10000

24. Сколько килобайт в одном мегабайте
A) 2^{20} B) $2^{10} - 1$ C) 1000 D) 1024 E) 1000000

25. Чем отличается 866 кодовая страница от 1251-й:
A) разной кодировкой латинских букв
B) разной кодировкой русских букв
C) отсутствием латинских букв
D) отсутствием русских букв
E) нет отличий в этих кодовых страницах

26. Где преимущественно используется 866 кодовая страница (для работы с русскими текстами):
A) в системе Windows 9x
B) в системе Windows 2000
C) в системе MS DOS
D) в системе UNIX
E) в системе Linux

27. Где преимущественно используется 1251 кодовая страница (для работы с русскими текстами):

A) в системе Windows 9x
B) в системе Windows 2000
C) в системе MS DOS
D) в системе UNIX
E) в системе Linux

28. Чем отличается 1251 кодовая страница от 866-й:
A) разной кодировкой латинских букв
B) разной кодировкой русских букв
C) отсутствием латинских букв
D) отсутствием русских букв
E) нет отличий в этих кодовых страницах

29. Что, как правило, служит для переноса информации с компьютера на компьютер?
A) монитор B) компакт-диск C) винчестер
D) жесткий диск E) дискета

30. Что у современного компьютера является более емким по размеру памяти?
A) компакт-диск B) жесткий диск C) дискета
D) оперативная память E) кэш-память

31. Какой тип принтера отсутствует у современного персонального компьютера?
A) матричный с 9 иголками B) струйный
C) лазерный D) карманный E) матричный с 24 иголками

32. Какова стандартная емкость дискеты 3,5 дюйма
A) 1500 Кб B) 1200 Кб C) 1440 Кб
D) 1,4 Мб E) 1,5 Мб

33. Какова стандартная емкость компакт-диска:
A) 600 Мб B) 700 Мб C) 1 Мб
D) 40 Гб E) 36,6 Гб

34. У какого принтера принцип работы подобен ксерографии:
A) матричного B) струйного C) с широкой кареткой
D) лазерного E) с узкой кареткой

35. Разрешение принтера – это…
A) максимальный размер печатного листа
B) число цветов, используемых для печати
C) число точек, которые способен напечатать принтер на одном дюйме
D) число листов, которое принтер печатает за минуту

36. По реализации пользовательского интерфейса операционные системы подразделяются на…
A) программные и аппаратные B) локальные и глобальные
C) общие и частные D) графические и неграфические

37. Методы и средства взаимодействия человека с аппаратными и программными средствами называют интерфейсом…
A) программным B) аппаратно-программным
C) пользовательским D) аппаратным

38. Совокупность способностей, знаний, умений и навыков, связанных с пониманием закономерностей информационных процессов в природе, обществе и технике – это…
A) информационная культура
B) компьютерная грамотность
C) необходимость современной жизни
D) образованность

39. Скорость выполнения арифметических операций зависит от…
A) типа дискеты B) процессора
C) типа монитора D) наличия модема

40. С помощью одного бита можно запомнить _____ число различных состояний
A) 1 B) 8 C) 2 D) 256

41. В вычислительной технике используется в качестве основной _____ система счисления
A) восьмеричная B) шестнадцатиричная
C) десятичная D) двоичная

42. Компакт-диск (CD) – это…
A) сменный магнитный диск малого размера
B) диск после выполнения операции сжатия информации
C) оптический диск, информация с которого считывается лазерным лучом
D) магнитный диск с высокой плотностью записи информации

43. Для хранения на диске слова ИНФОРМАТИКА в системе кодирования ASCII необходимо _____ бит
A) 1 B) 11 C) 176 D) 88

44. Даны числа, записанные в двоичной системе счисления как 101 и число, записанное в десятичной системе счисления как 55. Их сумма в десятичной системе счисления записывается в виде…
A) 60 B) 58 C) 111100 D) 156

45. Постоянное запоминающее устройство (ПЗУ) предназначено для…
A) временного хранения данных
B) хранения пользовательских файлов
C) хранения программ начальной загрузки компьютера
D) хранения прикладного программного обеспечения

46. Зарегистрированные сигналы – это …
A) информация B) символы
C) сведения D) данные

47. Пикселом называется …
A) минимальный размер шрифта
B) размер напечатанного изображения
C) объект в векторном изображении
D) размер точки изображения

48. В основные функции операционной системы не входит …
A) управление ресурсами компьютера
B) обеспечение диалога с пользователем
C) организации файловой структуры
D) разработка программ для ЭВМ

49. При отключении компьютера данные НЕ СОХРАНЯЮТСЯ …
A) в постоянной памяти (ПЗУ) B) дискете
C) в оперативной памяти (ОЗУ)
D) на жестком диске (винчестере)

50. Закодировать 8 битами можно _____ различных символов
A) 8 B) 1024 C) 64 D) 256

51. При включении компьютера процессор в первую очередь обращается к …
A) в постоянной памяти (ПЗУ) B) принтеру
C) гибкому диску D) компакт-диску

52. К основным характеристикам процессора относится …
A) объем ПЗУ B) тактовая частота
C) объем оперативной памяти D) емкость винчестера

53. Одна из проблем развития современной вычислительной техники - это …
A) хранение данных B) разработка алгоритма
C) формализация задач D) совершенствование памяти ЭВМ

54. Ядро операционной системы можно отнести к _____ программному обеспечению
A) системному B) прикладному
C) базовому D) служебному

55. В кодируемом английском тексте используется только 26 букв латинского алфавита и еще 6 знаков пунктуации. В этом случае текст, содержащий 1000 символов можно гарантированно сжать без потерь информации до размера …
A) 1000 байт B) 5 Кбит C) 32 Кбита D) 5000 бит

56. Основной задачей информатики НЕ ЯВЛЯЕТСЯ …
A) накопление и обработка информации с целью получения новых знаний
B) систематизация приемов и методов работы с программными средствами вычислительной техники
C) систематизация приемов и методов работы с аппаратными средствами вычислительной техники
D) анализ и исследование физических параметров источников информации

57. Число в десятичной системе счисления имеет вид 11. Следующее за ним целое число в двоичной системе записывается в виде …
A) 1100 B) 1011 C) 1010 D) 12

58. Функциональной частью компьютера, предназначенной для приема, хранения и выдачи данных, НЕ ЯВЛЯЕТСЯ …
A) память B) процессор C) графопостроитель D) ПЗУ

59. Именованная область внешней памяти произвольной длины с определенным количеством информации - это …
A) файл B) слово C) атрибут D) программа

60. «Пользовательский интерфейс» - это …
A) определенный операционной системой набор операций
B) методы и средства взаимодействия человека с аппаратными и программными средствами
C) технические средства ввода и вывода информации
D) монитор, клавиатура и мышь

Тесты по языку Ассемблера

1. Сколько бит в одном байте
A) 1 B) 2 C) 7 D) 8

2. Как запишется число 10011101_2 в 16-ричной системе счисления:
A) 9D B) 913 C) D9 D) 139

3. Сколько байтов в 1 Мбайте:
A) 1000 B) 1000000 C) 1024 D) 2^{20}

4. Сколько байтов одном слове:
A) 16 B) 2 C) 32 D) 4

5. Сколько байтов в двойном слове:
A) 16 B) 2 C) 32 D) 4

6. Какая группа регистров отсутствует в центральном процессоре:
A) регистры общего назначения
B) регистры машинного назначения
C) сегментные регистры
D) регистр флагов

7. Регистры AX, BX, CX, DX относятся к:
A) регистрам данных
B) сегментным регистрам
C) регистрам флага
D) регистрам указателей

8. Регистры SI, DI, BP, SP относятся к:
A) регистрам данных
B) сегментным регистрам
C) регистрам флага
D) регистрам указателей

9. Какой из регистров есть указатель стека:
A) AX B) CX C) BP D) SP

10. Какой из регистров служит для организации счетчика:
A) AX B) CX C) BP D) SP

11. Какой из регистров служит для накопления произведения в арифметических операциях:
A) AX B) CX C) BP D) SP

12. Какой из регистров не имеет старшей и младшей части:
A) AX B) BX C) DI D) CX

13. Если в регистре DX поместили число $F5E4_{16}$, то что находится в полурегистре DH?
A) F5 B) E4 C) 5E D) F4

14. Если в регистре DX поместили число $F5E4_{16}$, то что находится в полурегистре DL?
A) F5 B) E4 C) 5E D) F4

15. Укажите регистр сегмента команд:
A) CS B) DS C) SS D) ES

16. Укажите регистр сегмента данных:
A) CS B) DS C) SS D) ES

17. Укажите регистр сегмента стека:
A) CS B) DS C) SS D) ES

18. Укажите регистр сегмента команд:
A) CS B) DS C) SS D) ES

19. Каков максимальный размер сегмента памяти, используемой в языке Ассемблер:
A) 16 Кб B) 64 Кб C) 128 Кб D) 256 Кб

20. Что находится в регистре указателей IP?
A) адрес команды, которая должна выполняться следующей
B) адрес команды, которая только что была выполнена
C) только начальный адрес программы
D) только конечный адрес программы

21. Какая из команд не относится к командам сложения:
A) ADD B) INC C) MUL D) DEC

22. Какая из команд является командой умножения:
A) ADD B) INC C) MUL D) DEC

23. Какая из команд является командой вычитания:
A) ADD B) SUB C) MUL D) DIV

24. Какая из команд является командой деления:
A) ADD B) SUB C) MUL D) DIV

25. Что будет находиться в регистре AX после выполнения фрагмента программы:

```
MOV AX, 100
MOV BX, 10
MOV DX, 0
MUL BX
```

A) 1000 B) 3E8 C) 64 D) A

26. Что будет находиться в регистре AX после выполнения фрагмента программы:

```
MOV AX, 100
MOV BX, 10
MOV DX, 0
DIV BX
```

A) 10 B) 64 C) 3E8 D) A

27. Что будет находиться в регистре DX после выполнения фрагмента программы:

```
MOV AX, 100
MOV BX, 10
MOV DX, 0
DIV BX
```

A) 0 B) 64 C) 3E8 D) A

28. Какая из команд отладчика DEBUG позволяет исполнить программу:

A) E B) G C) Q D) R

29. Какая из команд отладчика DEBUG позволяет изменить содержимое ячеек памяти, начиная с указанного адреса:

A) E B) G C) Q D) R

30. Какая из команд отладчика DEBUG позволяет выйти из отладчика и вернуться в DOS:

A) E B) G C) Q D) R

31. Какая из команд отладчика DEBUG позволяет отобразить содержимое одного или всех регистров:

A) E B) G C) Q D) R

32. Что будет находится в регистре CX после выполнения фрагмента программы:

```
MOV CX, 5
MOV BX, 6
ADD CX, BX
```

A) 11 B) B C) 56 D) 65

33. Что будет находится в регистре CX после выполнения фрагмента программы:

```
MOV CX, 5
MOV BX, 6
SUB CX, BX
```

A) 11 B) B C) 56 D) 65

34. При выполнении пары команд вида:

```
CMP AX, BX
JB LAB
```

когда будет осуществлен переход на метку LAB

A) в случае, если содержимое AX > BX
B) в случае, если содержимое AX < BX
C) в случае, если содержимое AX >= BX
D) в случае, если содержимое AX <= BX

35. При выполнении пары команд вида:

```
CMP AX, BX
JBE LAB
```

когда будет осуществлен переход на метку LAB

A) в случае, если содержимое AX > BX
B) в случае, если содержимое AX < BX
C) в случае, если содержимое AX >= BX
D) в случае, если содержимое AX <= BX

36. При выполнении пары команд вида:

```
CMP AX, BX
JA LAB
```

когда будет осуществлен переход на метку LAB

A) в случае, если содержимое AX > BX
B) в случае, если содержимое AX < BX
C) в случае, если содержимое AX >= BX
D) в случае, если содержимое AX <= BX

37. При выполнении пары команд вида:

```
CMP AX, BX
JAE LAB
```

когда будет осуществлен переход на метку LAB
A) в случае, если содержимое AX > BX
B) в случае, если содержимое AX < BX
C) в случае, если содержимое AX >= BX
D) в случае, если содержимое AX <= BX

38. Какая из команд Ассемблера служит для помещения данных в регистре?
A) MUL B) MOV C) DIV D) ADD

39. Какая из команд Ассемблера служит для выполнения операции умножения?
A) MUL B) MOV C) DIV D) ADD

40. Какая из команд Ассемблера служит для выполнения операции деления?
A) MUL B) MOV C) DIV D) ADD

41. Какая из команд Ассемблера служит для выполнения операции сложения?
A) MUL B) MOV C) DIV D) ADD

42. Какой регистр может выступать в качестве счетчика при организации цикла в программе на ассемблере?
A) AX B) BX C) CX D) DX

43. Какой из регистров может выступать в качестве делимого при выполнении операции деления на байт в программе на ассемблере?
A) AX B) BX C) CX D) DX

44. Какой из регистров служит для хранения частного при делении на байт в программе на ассемблере?
A) AX B) BX C) CX D) DX

45. Какой из регистров служит для хранения остатка при делении на байт в программе на ассемблере?
A) AX B) BX C) CX D) DX

46. Какая из директив данных служит для описания переменной в виде двухбайтовых слов:
A) DB B) DW C) DD D) DX

47. Какая из директив данных служит для описания переменной в виде однобайтовых слов:
A) DB B) DW C) DD D) DX

48. Какая из директив данных служит для описания переменной в виде четырехбайтовых слов:
A) DB B) DW C) DD D) DX

49. Какой путь передачи параметров в процедуру отсутствует в языке Ассемблер?
A) через регистры
B) через стек
C) используя сегмент данных
D) через заголовок

50. Какой тип процедуры отсутствует в программе на Ассемблере:
A) ближняя
B) дальняя
C) рекурсивная
D) скрытая