Министерство общего и профессионального
образования Российской Федерации

Алтайский государственный технический
Университет им. И.И. Ползунова

Е.Г. Боровцов

ОРГАНИЗАЦИЯ ЭВМ

Барнаул 2009

УДК 681.3

Боровцов Е.Г. Организация ЭВМ: Учебное пособие.Изд. 3-е перераб. и доп./ Алт. госуд. технич. ун-т им. И.И. Ползунова.- Барнаул: 2009.-172 с.

Данное учебное пособие предназначено для дистанционного изучения дисциплины “Организация ЭВМ”. Здесь рассмотрены вопросы общей организации компьютеров, организации процессоров и их системы команд. Представлен лабораторный практикум, содержащий подробные иллюстрации к выполнению лабораторных работ по изучаемому курсу. Приложения содержат подробное описание программного обеспечения, используемого при изучении курса и выполнении лабораторных работ. Приложение 5 содержит график сдачи отчетов по лабораторным работам.

Рецензент: Е. Н. Крючкова - доцент кафедры Прикладной математики АлтГТУ

1. СТРУКТУРНАЯ ОРГАНИЗАЦИЯ КОМПЬЮТЕРОВ

1.1 ПОНЯТИЕ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ. ПРИНЦИПЫ ПРОГРАММНОГО УПРАВЛЕНИЯ

Любой компьютер является достаточно сложно организованной системой, представляющей собой совокупность двух неразрывно связанных компонент - аппаратных средств и программного обеспечения. Эти средства в совокупности составляют ***вычислительную систему.***

Аппаратные и программные средства тесно связаны (в современных компьютерах зачастую весьма сложно различить какие функции выполняют программные средства, а какие берет на себя аппаратура), и в равной степени важны, поскольку использование аппаратных средств без программных лишено смысла, а использование программных средств без аппаратуры просто невозможно.

Современные вычислительные машины строятся на основе принципов программного управления, предполагающих следующее:

* исходные данные, промежуточные и конечные результаты кодируются в той или иной (обычно двоичной) форме и разделяются на единицы информации, называемые словами;
* правила вычислений задаются с помощью операторов двух типов - преобразователей и распознавателей; *преобразователи* обеспечивают собственно выполнение операций над элементами информации; *распознаватели* обеспечивают управление порядком выполнения операторов путем анализа элементов информации;
* каждый оператор кодируется управляющим словом - командой; команда представляет собой указание на то, какие действия необходимо выполнить (какую операцию) и где расположены данные, над которыми должна быть выполнена операция, представляющая собой действие, выполняемое на аппаратном уровне; данные, над которыми выполняется операция, называются операндами; последовательность команд с явно или неявно заданным порядком выполнения представляют собой программу;
* разнотипные слова информации (данные и команды) различаются по способу использования, но не по способу кодирования; иными словами, одни и те же элементы информации в различные моменты времени и в различном контексте могут рассматриваться и как команды, и как данные;
* слова информации размещаются в ячейках памяти и идентифицируются номерами ячеек, называемых адресами;

* реализуемая компьютером функция целиком определяется хранимой в памяти программой; замена программы полностью изменяет функцию компьютера.

Принципы программного управления, основные положения которых представлены выше, были сформулированы в 1945г. Дж. фон Нейманом и по сегодняшний день находят практическое применение при построении вычислительных систем.

1.2 ФУНКЦИОНАЛЬНЫЕ БЛОКИ КОМПЬЮТЕРА, И ИХ НАЗНАЧЕНИЕ И ВЗАИМОСВЯЗЬ

Из принципов программного управления однозначно определяется функциональный состав компьютера, включающий устройства трех основных классов:

* операционные, предназначенные для выполнения операций над информацией;
* запоминающие, предназначенные для хранения информации;
* устройства ввода-вывода, предназначенные для связи вычислительной системы с внешней средой.

Важнейшими устройствами, без которых невозможна реализация программного управления, является центральный процессор (**ЦП**) и оперативная память (**ОП** или **ОЗУ** - *оперативное запоминающее устройство*). Эти устройства часто называются *центральными*. Внешние запоминающие устройства (**ЗУ**) и устройства ввода/вывода (**УВВ**) называются *периферийными* устройствами (**ПУ**).

Центральный процессор (Central Processor Unit - **CPU**) служит для выполнения арифметических и логических операций над данными, а также для управления всеми компонентами вычислительной системы. Одной из важнейших характеристик процессора является его архитектура. Выделяют следующие архитектуры процессоров:

- **CISC** – процессоры;
- **RISC** – процессоры;
- **VLIW** – процессоры;

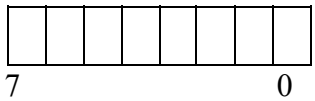
CISC – процессоры – это процессоры со сложной системой команд (**C**omplex **I**nstruction **S**et **C**omputer) . Система команд такого процессора имеет инструкции, существенно различающиеся внутренним форматом, методами адресации, длиной и временем выполнения. Таким образом, система команд процессора получается достаточно громоздкой, но гибкой.

RISC – процессоры (**R**educed **I**nstruction **S**et **C**omputer) – процессоры с сокращенным набором команд, наоборот, имеют инструкции унифицированного формата,

совпадающие по длине, методам адресации операндов и времени выполнения. Это позволяет, с одной стороны, упростить схемотехнику процессоров, и с другой, реализовать возможность автоматического распараллеливания операций в отдельных фрагментах программы.

VLIW – процессоры – процессоры с очень длинной инструкцией (**V**ery **L**ong **I**nstruction **W**ord) – являются компромиссом между CISC и RISC системами, имея, с одной стороны, длинные инструкции, различающиеся форматами, длиной и временем выполнения. С другой стороны, эти длинные инструкции при обработке разбиваются на совокупность RISC инструкций, интерпретируемых внутренним RISC-ядром процессора.

Оперативная память (называемая также **RAM** - Random Access Memory - память с произвольным доступом) - устройство, способное работать в темпе процессора. Служит для хранения оперативной информации и выдачи блокам машины этой информации по их требованию. ОП состоит из ячеек - байтов. Каждый байт состоит из восьми двоичных разрядов (битов - образовано от английского **binary digit**). Биты в байте нумеруются справа налево, начиная с нуля:



Байт является *минимальной* единицей информации, с которой центральные устройства компьютера оперирует, как с единым целым. Местоположение каждого байта в памяти компьютера характеризуется адресом. Адреса памяти начинаются с нуля для первого байта и последовательно возрастают на единицу для каждого последующего байта.

Максимальное количество байтов памяти, имеющейся в компьютере, принято называть объемом памяти. Для измерения объема памяти вычислительных систем наряду с основной единицей - байтом - используются следующие производные единицы :
Кбайт = 2^{10} байт; Мбайт = 2^{10} Кбайт; Гбайт = 2^{10} Мбайт; Тбайт = 2^{10} Гбайт.

Байты организуются в слова. В отличие от байта слово - *максимальная* единица информации, с которым центральные устройства компьютера оперируют, как с единым целым. В зависимости от типа и организации компьютера слово может иметь длину 1, 2, 4 или 8 байт. Кроме того, существуют еще две производных единицы хранения информации в памяти компьютера - полуслово и двойное слово. Местоположение слова в памяти также определяется адресом. В зависимости от длины слова адреса слов могут быть кратны 1, 2, 4 или 8.

Наряду с блоками оперативной памяти, допускающей оперативное изменение хранимой информации, в состав центральных устройств компьютера практически всегда входят блоки постоянных запоминающего устройства (**ПЗУ**). Память данного типа

предназначена только для считывания хранимой в ней информации (Read Only Memory или **ROM** - память). В такой памяти обычно хранятся наиболее часто используемые программы и данные, например, программы обслуживания устройств ввода/вывода и внешних запоминающих устройств (**BIOS** в IBM PC – **B**asic **I**nput **O**utput **S**ystem – базовая система управления вводом/выводом), а также программы, необходимые для начальной загрузки вычислительной системы. Объем подобной памяти значительно меньше, чем основного ОЗУ. Ее быстродействие также соизмеримо с быстродействием процессора, однако несколько ниже, чем у обычной оперативной памяти.

Еще одной разновидностью памяти является кэш-память (**Cache - memory**), называемая иногда сверхоперативной памятью. Кэш-память обладает наиболее высоким быстродействием и располагается между центральным процессором и оперативной памятью. Эта память предназначена для хранения данных, чаще всего используемых в ходе работы той или иной программы. Кэш-память имеет ассоциативную или частично ассоциативную организацию. В ходе работы процессор для выборки данных в первую очередь организует обращение в кэш-память, затем, если данные в кэш-памяти не найдены, осуществляется обращение в обычную оперативную память. Ассоциативная организация кэш-памяти наряду с ее повышенным быстродействием позволяет существенно (в среднем, в три-четыре раза) повысить скорость выборки данных, что, в свою очередь, существенно повышает быстродействие вычислительной системы в целом.

Внешние запоминающие устройства предназначены для длительного хранения больших объемов информации. К ВЗУ относятся накопители на магнитных дисках различных типов, накопители на магнитных лентах, накопители на оптических и магнитооптических дисках.

Устройства ввода/вывода - предназначены для связи компьютера с окружающей средой, в том числе и с человеком. Типичными устройствами ввода/вывода являются клавиатура компьютера, дисплей, печатающее устройство того или иного типа, устройство управления курсором - мышь, устройство для вычерчивания чертежей - графопостроитель, устройство для кодирования или оцифровки чертежей - дигитайзер, устройство кодирования графической информации - сканер и ряд других.

Любой современный компьютер имеет модульную структуру, т.е. строится из набора модулей, каждый из которых реализует законченную функцию и является относительно независимым от других модулей. Данный принцип обеспечивает унификацию узлов, а также позволяет изменять конфигурацию оборудования компьютера даже в процессе его функционирования (процесс носит название **HotPlug** - “горячее” подключение) - подключать дополнительные модули, заменять одни модули на другие, менять конфигурацию модулей, а это, в свою очередь, существенно повышает гибкость и

универсальность вычислительной системы. Связь между отдельными модулями осуществляется посредством **интерфейса**. Интерфейсом называется совокупность аппаратных средств, системы сигналов и набора алгоритмов, определяющих порядок обмена информацией между устройствами.

1.3 ВАРИАНТЫ СТРУКТУР КОМПЬЮТЕРОВ

Существуют различные варианты объединения устройств компьютера в единую вычислительную систему. Рассмотрим два варианта, являющиеся базовыми.

1. Структура высокопроизводительного компьютера с канальной архитектурой представлена на рисунке ниже.

В этой структуре к ОП подключены ЦП и каналы ввода/вывода.

Каналы ввода-вывода представляют собой специализированные устройства (фактически специализированные процессоры), обеспечивающие обмен информацией между ОП и ПУ. Каналы могут быть селекторные (выделенные) и мультиплексные (коммутируемые). Селекторные каналы предназначены для обмена с быстродействующими ВУ (например, магнитными дисками). Мультиплексные каналы обеспечивают обмен с группой низкоскоростных устройств, переключаясь в ходе работы от обслуживания одного устройства из группы к другому. Основное свойство канала состоит в том, что он работает по специальной канальной программе и освобождает ЦП от выполнения трудоемких операций ввода-вывода. Таким образом, процессор только инициирует операцию обмена, все остальное канал делает сам. Процессор в это время может выполнять другие действия.

Контроллер ПУ - устройство управления ПУ, выполняет следующие функции:

- преобразование последовательности сигналов интерфейса в последовательность сигналов, управляющих работой ПУ;
- обеспечивает синхронизацию работы ПУ с другими устройствами, в первую очередь, с процессором;

- обеспечивает буферизацию информации, т.е. временное ее запоминание и хранение в процессе обмена.

Каждый тип ПУ практически всегда требует специфичного контроллера.

В данной структуре имеется четыре типа интерфейса:

1. интерфейс оперативной памяти (ОП-ЦП-КВВ);

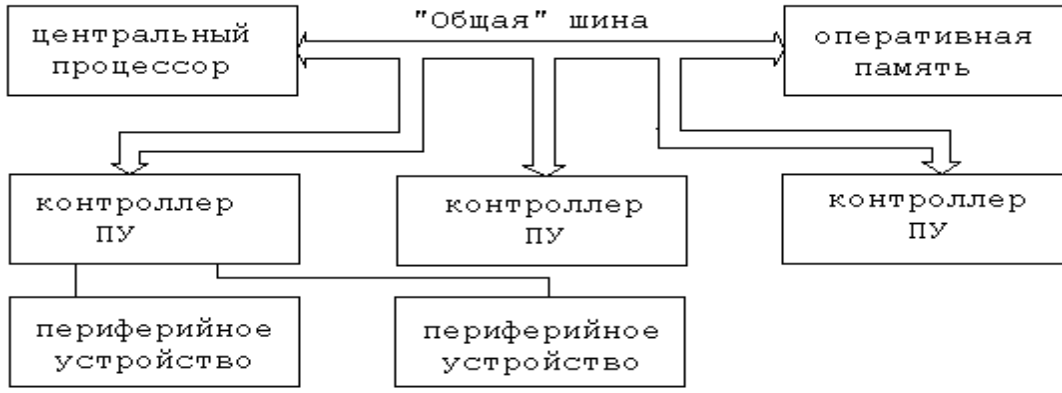
2. интерфейс каналов (управляющие сигналы от ЦП к КВВ и обратно);

3. интерфейс контроллеров (КВВ - КП);

4. интерфейс периферийного устройства (КП - ПУ).

Структура достаточно сложная и требует больших аппаратных затрат, однако обеспечивает высокую производительность подсистемы ввода/вывода и максимальную загрузку процессора, что, в свою очередь, обеспечивает высокую эффективность обработки больших объемов информации. С использованием данного структурного решения строятся мощные высокопроизводительные системы - суперкомпьютеры, узловые компьютеры сетей, файловые сервера, высокопроизводительные графические рабочие станции.

Второй вариант структурного решения - с использованием интерфейса «общая шина» - используется в мини-, микрокомпьютерах и в большинстве персональных систем. Графическая интерпретация данного варианта структурного решения приведена на следующем рисунке.



В данном случае в системе присутствует единый глобальный тип интерфейса - так называемая “общая шина”. Все устройства, подключаемые к общей шине, являются равноправными. В каждый момент времени по общей шине может обмениваться только пара устройств, причем одним устройством из этой пары является процессор, который полностью берет на себя функции управления обменом.

Достоинством данного варианта является простота аппаратной реализации, ведущая к резкому снижению себестоимости конструкции; единый интерфейс для

оперативной памяти, процессора и контроллеров всех типов ПУ. Однако, с другой стороны, отвлечение процессора на выполнение операций обмена значительно снижает общую производительность системы и повышает простои процессора.

1.4 КОДИРОВАНИЕ И ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В КОМПЬЮТЕРЕ

1.4.1. ***Двоичная, восьмеричная и*** *шестнадцатеричная* ***системы***

Двоичная система счисления является позиционной системой счисления с основанием **2**. Для представления любого разряда двоичного числа используется всего две цифры - 0 и 1. Применение двоичной системы счисления в компьютере логически вытекает из того, что электронные схемы, из которых состоит аппаратура компьютера, способны различать только два состояния - наличие сигнала (1), или его отсутствие (0). Кроме того, двоичная система счисления обладает и другими достоинствами, делающими удобным ее применение для представления информации в компьютерах, а именно :

- простота выполнения арифметических и логических операций и, как следствие, простота устройств, реализующих эти операции;
- возможность использования аппарата алгебры логики для анализа и синтеза операционных устройств.

Для представления числа в двоичной системе счисления воспользуемся тем, что каждому разряду двоичного числа соответствует определенная степень двойки (вес разряда). Тогда, например, такое двоичное число, как 11011_2 можно представить следующим образом:

$$\mathbf{11011_2 = 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0, \text{ или } \quad 1*16 + 1*8 + 0*4 + 1*2 + 1*1 = 27_{10}}$$

(Здесь и в дальнейшем справа внизу от числа записывается основание системы счисления, в которой представлено число. Это позволяет различать, например, числа $1001_2 = 9_{10}$ и 1001_{10}.)

Аналогичным образом в десятичной системе счисления можно представить и действительное двоичное число, например:

$$\mathbf{1011.111_2 = 1*8 + 0*4 + 1*2 + 1*1 + 1/2 + 1/4 + 1/8 = 11.875_{10}}$$

Правила выполнения арифметических операций для двоичной системы счисления остается таким же, как и для десятичной, что иллюстрирует приводимые ниже примеры:

а) сложение

$$\mathbf{1101_2 = 13_{10}}$$

$+\underline{1010_2 = 10_{10}}$

$10111_2 = 23_{10}$

↑

при сложении двух единиц старших разрядов происходит перенос единицы в высший разряд;

б) вычитание

при заеме единица старшего разряда равна двум единицам младшего;

↓

$11001_2 = 25_{10}$

$-\ \underline{1011_2 = 11_{10}}$

$1110_2 = 14_{10}$

в) умножение

$101_2 = 5_{10}$

$\bullet\underline{110_2 = 6_{10}}$

000

+101

+101

$11110_2 = 30_{10}$

г) деление

$26 = 11010 \ \big|\ \underline{10_2 = 2_{10}}$

- 10 $\quad 1101_2 = 13_{10}$

10

- 10

010

- 10

0

<u>Восьмеричная и шестнадцатеричная системы счисления</u>

Двоичная система счисления является достаточно громоздкой по сравнению с десятичной при использовании ее человеком, что создает определенные неудобства (длинная запись числа, необходимость преобразования из десятичной формы в двоичную и наоборот). Поэтому для представления информации в компьютерах используют и другие системы счисления, позволяющие записывать числа в более компактной форме, а именно, восьмеричную и шестнадцатеричную системы счисления (в дальнейшем ***с/с***). Эти системы счисления относятся к двоично-кодированным системам, у которых основание системы счисления кратно некоторой степени двойки: 2^3 - для восьмеричной и

2^4 - для шестнадцатеричной систем счисления. В восьмеричной системе счисления для изображения чисел используется восемь цифр: **0 ... 7**, в шестнадцатеричной - 16: **0 ... 9, A, B, C, D, E, F**. Изображения чисел в восьмеричной и шестнадцатеричной с/с вместе с их двоичными и десятичными эквивалентами представлены ниже:

основание системы счисления			
10	**8**	**16**	**2**
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	10	8	1000
9	11	9	1001
10	12	A	1010
11	13	B	1011
12	14	C	1100
13	15	D	1101
14	16	E	1110
15	17	F	1111

Рассмотрев таблицу, можно легко заметить, что для представления любой восьмеричной цифры в двоичной системе счисления достаточно **трех** двоичных разрядов, а для представления шестнадцатеричной цифры - **четырех**. Таким образом, если мы в двоичном представлении числа заменим каждую группу из трех или четырех цифр соответствующей восьмеричной или шестнадцатеричной цифрой, то получим представление исходного двоичного числа в восьмеричной или шестнадцатеричной системе соответственно.

Например:

$101\ 110\ 110_2$

5 6 6

т.е. **$101110110_2 = 566_8$**.

Для представления этого же двоичного числа в шестнадцатеричной системе счисления добавим слева три незначащих нуля, получим:

$\underline{0001}\ \underline{0111}\ \underline{0110}_2$

1 7 6, или **$101110110_2 = 176_{16}$**

и наоборот:

$A5_{16} = \underline{1010}\ \underline{0101}_2$

A 5

$245_8 = \underline{010}\ \underline{100}\ \underline{101}_2$

2 4 5

т.е. **$A5_{16} = 245_8 = 10100101_2$**.

Из приведенных примеров видно, что запись восьмеричного числа в три раза, а запись шестнадцатеричного числа в четыре раза короче, чем запись того же числа в двоичном виде. В то же время восьмеричное или шестнадцатеричное представление числа эквивалентно двоичному, поскольку каждая восьмеричная или шестнадцатеричная цифра получены просто заменой двоичных разрядов, сгруппированных по три или по четыре, соответствующей восьмеричной или шестнадцатеричной цифрой.

1.4.2 Перевод чисел из одной системы счисления в другую.

Алгоритм перевода целого числа из десятичной системы счисления в двоичную, шестнадцатеричную или восьмеричную системы заключается в следующем: исходное десятичное число делится на основание системы счисления, в которую осуществляется перевод; если частное больше основания системы счисления, в которую осуществляется перевод, то оно опять делится на основание системы счисления; процесс деления продолжается до тех пор, пока частное, полученное в результате очередного деления, не получится меньше основания системы счисления, в которую осуществляется перевод. Получившееся число представляется в виде упорядоченной последовательности остатков деления в порядке, обратном их получения. Старшую цифру числа дает последнее частное. Ниже приведены примеры, иллюстрирующие перевод чисел.

а) перевод числа 113 из 10 с/с в 2 с/с

```
113   !   2
10    ———————
 13       56   !   2
-12       4    ———————
  1       16       28   !   2
```

```
-16      2   ————————
  0      08      14  !   2
        - 8      -14  ————————
         0        0      7   !   2
                       - 6   ————————
                         1       3   !   2
                             - 2   ————————
                               1       1
```

<——————————————————————————————— направление записи числа

В итоге получили двоичное число 1110001_2.

Для того, чтобы убедиться в том, что перевод осуществлен правильно, проделаем обратное преобразование:

$\mathbf{1110001_2 = 1*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 64 + 32 + 16 + 1 = 113_{10}}$.

б) перевод числа 113 из 10 с/с в 16 с/с

```
113  !  16
112  ———
  1      7
```

<————————— направление записи числа

Получили $\mathbf{113_{10} = 71_{16}}$.

Обратное преобразование : $\mathbf{71_{16} = 7*16^1 + 1*16^0 = 112 + 1 = 113_{10}}$.

В правильности преобразования можно убедиться, преобразовав 71 в двоичную форму:

$\mathbf{71_{16}}$ = **0111 0001**$_2$

7 1

Получили тот же результат, что и в предыдущем случае.

в) перевод числа 113 из 10 с/с в 8 с/с.

```
113   !   8
 8    ———
 33       14   !   8
 32        8   ———
  1        6       1
```

<————————————————— направление записи числа

Получили, что $\mathbf{113_{10}}$ = **001 110 001**$_2$

1 6 1

Алгоритм перевода десятичной дроби несколько иной и заключается в следующем. Исходная десятичная дробь умножается на основание системы счисления, в которую осуществляется перевод; получившаяся целая часть (если таковая действительно получилась) отбрасывается и запоминается; оставшаяся дробная часть опять множится на

основание системы счисления; процесс повторяется до тех пор, пока либо не будут получены все цифры, достаточные для представления числа с заданной точностью, либо в результате очередного умножения дробная часть числа не получится равной нулю. Ниже представлены примеры, иллюстрирующие перевод дроби в различные системы счисления.

а) перевод числа 0.632 в 2 с/с

направление		**0.632**
		2
записи	**1**	**.264**
		2
числа	**0**	**.528**
		2
	1	**.056**
		2
	0	**.112**
		2
	0	**.224**
		2
	0	**.448**
		2
	0	**.896**
		2
	V 0	**.792**

Получили, что 0.632 приблизительно равно 0.10100001_2 с точностью до восьмой двоичной цифры после точки.

б) перевод 0.632 в 16 с/с

	0.632
	16
10	**.112**
	16
1	**.792**

Получили, что 0.632 приблизительно равно шестнадцатеричному 0.A1 с точностью до второй шестнадцатеричной цифры после точки. Аналогичный результат можно получить и из двоичного представления дроби:

$0.\underline{1010}\ \ \underline{0001}_2 = 0.A1_{16}$

A 1

в) перевод 0.632 в 8 с/с

	0.632
	8
5	**.056**
	8

0 .448

8

3 .584

Получили, что 0.632 приблизительно равно восьмеричному 0.503 с точностью до третьей восьмеричной цифры после точки.

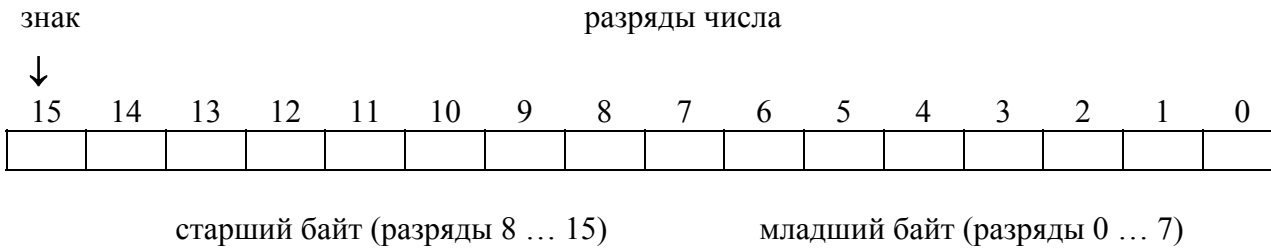
Следует заметить одну существенную деталь, а именно - при переводе получившаяся дробь может оказаться бесконечной, хотя исходная дробь являлась конечной. Именно этим фактором объясняется появление погрешностей при переводе действительных чисел в компьютере, поскольку максимальное число хранимых двоичных разрядов ограничивается разрядной сеткой той или иной машины.

Для чисел, имеющих как целую, так и дробную части, перевод из одной системы счисления в другую осуществляется отдельно для целой и дробной частей, причем каждая часть переводится в соответствии с принятыми для нее правилами.

1.4.3 Представление целых чисел в памяти

Положительные и отрицательные числа в памяти компьютера представляются в виде двоичных с фиксированной точкой (2 сфт) и, в зависимости от типа компьютера и требуемого диапазона представления, могут занимать два или четыре байта. При размещении в двух байтах целое число может принимать значения в диапазоне **-32768 ... 32767**. При размещении в четырех байтах диапазон представления целых чисел составляет **-2147483648 ... 2147483647**.

Формат двоичного числа с фиксированной точкой при представлении его в двух байтах имеет следующий вид:



Таким образом, старший разряд старшего байта содержит знак числа (0 - число положительное, 1 - отрицательное), а оставшиеся пятнадцать разрядов - модуль числа. Представление целого числа в четырех байтах выглядит аналогичным образом, т.е. старший разряд старшего байта так же содержит код знака, а оставшийся 31 разряд содержат модуль числа. Таким образом, например, число 131 , представленное как двоичное с фиксированной точкой в 16 разрядах, будет иметь следующий вид: ($131_{10} = 10000011_2$):

0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1

↑|_____|

Знак модуль числа

или, используя шестнадцатеричную или восьмеричную форму записи

0 0 8 1 → в 16 с/с ;

0 0 0 2 0 3 → в 8 с/с.

В отличие от положительных чисел, представляемых в прямом коде, отрицательные числа в памяти ЭВМ представляются в дополнительном коде.

Дополнительный код двоичного числа образуется следующим образом:

1. В знаковый разряд записывается единица.
2. Все разряды числа, исключая знаковый, инвертируются, т.е. единицы в них заменяются нулями, а нули - единицами.
3. К получившемуся числу добавляется двоичная единица, которая дает окончательный результат.

В качестве примера рассмотрим представление числа **-131** в виде двоичного с фиксированной точкой в дополнительном коде. Двоичный код модуля числа - $\mathbf{10000011_2}$. При представлении в шестнадцати двоичных разрядах прямой код числа будет иметь вид: $\mathbf{1000000010000011_2}$. Сделаем инверсию, получим двоичное число **1111111101111100**. Добавим двоичную единицу:

$$\begin{array}{ll} & \mathbf{1111\ 1111\ 0111\ 1100_2} \\ + & \underline{\qquad\qquad\qquad\qquad \mathbf{1_2}} \\ & \mathbf{1111\ 1111\ 0111\ 1101_2} \end{array}$$

Окончательно, используя 8 с/с или 16 с/с, будем, соответственно, иметь :

$\mathbf{177575_8}$ **или** $\mathbf{FF7D_{16}}$.

Для того, чтобы найти ответ на вопрос, зачем нужен дополнительный код числа, выполним сложение двоичного представления числа $\mathbf{131_{10}}$ в прямом коде и числа $\mathbf{-131_{10}}$ в дополнительном коде:

$$\begin{array}{ll} & \mathbf{1000\ 0000\ 1000\ 0011_2} \\ + & \underline{\mathbf{1111\ 1111\ 0111\ 1101_2}} \\ & \mathbf{0000\ 0000\ 0000\ 0000_2} \end{array}$$

Принимая во внимание то, что при сложении в дополнительных кодах перенос из знакового разряда не учитывается, получаем 0. Таким образом, применение

дополнительных кодов позволяет для выполнения операций сложения и вычитания использовать одно устройство (сумматор), но оперирующее с дополнительными кодами чисел.

1.4.4 ***Представление действительных чисел в памяти компьютера***

Действительные числа в памяти компьютера представляются в формате двоичного с плавающей точкой (**2спт**) и занимают, как минимум четыре байта. В зависимости от типа машины и требуемой точности внутреннее представление числа в формате 2спт может быть различным. В любом случае, во внутреннем представлении числа с плавающей точкой выделяются **три** поля: **поле кода знака** (“0” - плюс, “1” - минус), **поле характеристики**, которая является преобразованным значением порядка числа, и **поле мантиссы**. Рассмотрим, каким образом кодируются эти поля.

* Старший бит старшего байта - код знака числа;
* следующие семь битов старшего байта и один бит следующего за ним байта (всего восемь битов) - характеристика. В данном случае характеристика представляет собой порядок двоичного представления числа, сдвинутый на 128_{10} (т.е. порядок двоичного числа + 128);
* следующие двадцать три двоичных разряда хранят мантиссу числа. Мантисса числа хранится в нормализованном виде, при этом точка располагается после второй значащей двоичной цифрой мантиссы. Поскольку первая значащая двоичная цифра всегда двоичная единица, то эта двоичная единица во внутреннем представлении числа *не хранится* **(!)** (так называемый “скрытый” бит).

Рассмотрим, например, преобразование во внутреннее представление числа $\mathbf{76.625_{10}}$. Преобразуем число в двоичную с/с, получим: $\mathbf{76.625_{10} = 1001100.101_2}$. Теперь сдвинем точку так, чтобы она оказалась после второй двоичной цифры числа, одновременно при этом модифицируем порядок. После этой операции будем иметь $\mathbf{1001100.101_2 = 10.01100101_2 * 2^5}$. Вычислим характеристику числа: **5+128=133**. Первые девять битов числа (знак и характеристика) будут следующими: $\mathbf{010000101_2}$. С учетом скрытой единицы мантисса будет иметь вид: $\mathbf{00110010100000000000000_2}$. Общее представление числа, записанное с использованием 16 с/с примет вид: $\mathbf{42994000_{16}}$. Представление отрицательного числа будет отличаться от представления положительного числа только старшим битом старшего байта, установленным в единицу, и для числа **-76.625** будет иметь вид $\mathbf{C2994000_{16}}$.

2. ЛОГИЧЕСКАЯ СХЕМА ФУНКЦИОНИРОВАНИЯ КОМПЬЮТЕРА

В предыдущем разделе настоящего пособия мы определили состав компьютера и назначение основных его компонентов, выяснили, что такое команда, операция, операнды, программа. Перейдем к более детальному обсуждению процесса функционирования компьютера при выполнении хранимой в памяти программы.

Итак, вспомним, что операция представляет собой действие, выполняемое процессором на аппаратном уровне. Каждой операции соответствует некоторый код, который размещается в поле операции команды и при выполнении команды перобразуется в последовательность электрических сигналов, управляющих работой блоков процессора.

Количество двоичных разрядов, кодирующих поле операции и операндов в команде, так же, как и состав поля операндов, определяется *форматом* команды. Исходя из определения команды можно предположить, что формат команд процессора может иметь следующий вид:

КОП	**А1**	**А2**	**А3**	**АСК**

Здесь **КОП** - код операции, **А1, А2** - адреса первого и второго операндов, **А3** - адрес результата, **АСК** - адрес следующей команды.

Однако такой формат имеет существенный недостаток - он избыточен. Поэтому в реальных процессорах фактически присутствует меньшее число адресов, хотя функционально их остается по прежнему четыре. Сокращение фактического числа адресов достигается за счет неявной адресации. В частности, в большинстве процессоров результат выполнения операции помещается на место одного из операндов, а адрес следующей команды вычисляется как адрес выполняемой команды плюс длина команды.

Таким образом, фактическим форматом двухоперандной команды будет следующий:

КОП	**А1**	**А2**

Рассмотрим состав процессора и назначение отдельных блоков, обеспечивающих выполнение программы компьютером.

В состав процессора компьютера входят следующие основные функциональные блоки:

* блок управляющих регистров;
* арифметико-логическое устройство;
* блок регистров общего назначения.

Блок управляющих регистров включает в себя:

- счетчик адреса команд **(СЧАК**), называемый так же программным счетчиком (**PC** - Program Counter), или указателем инструкции (**IP** - Instruction Poiner). Счетчик адреса команд содержит адрес ячейки памяти, в которой размещается очередная команда;
- регистр команд(**RC**), содержащей код выполняемой в текущий момент времени команды;
- регистр признаков(**F**), отражающий результат операции, выполненной процессором, а также признаки некоторых особых ситуаций;
- указатель стека(**SP** - Stack Pointer) - специальный регистр, с помощью которого осуществляется доступ к стековой памяти. Стековая память относится к специальным видам памяти (безадресная память). Для чтения записи доступна только одна ячейка, называемая вершиной стека. Практически реализуется на основе обычной памяти и специального регистра-указателя стека. Указатель стека содержит адрес ячейки стековой памяти доступной в настоящий момент для чтения или записи. После записи в верхушку стека указатель стека автоматически изменяется так, чтобы он указывал на следующую ячейку. Обычно стек располагается так, что он растет от старших адресов к младшим. В этом случае после выполнения записи в вершину стека указатель стека автоматически уменьшается, а перед чтением из стека автоматически увеличивается.

Таким образом, блок управляющих регистров содержит управляющую информацию, необходимую для выполнения программы. Следует отметить, что состав блока управляющих регистров может различаться для различных типов машин, однако тип хранимой и обрабатываемой в нем информации практически не изменяется. Например, в System/370 отсутствовали отдельные регистры для хранения программного счетчика, указателя стека и признаков выполнения операций, однако процессор данного семейства машин в составе блока управляющих регистров имел регистр слова состояния программы (PSW) - восьмибайтовое поле, в котором хранится и программный счетчик, и признак результата и, кроме того, другая немаловажная информация. Аналогичная ситуация наблюдается для машин семейства PDP-11. У этих машин информация о признаке результата операции входит в состав регистра состояния процессора (PS), а роль программного счетчика выполняет один из регистров общего назначения.

Арифметико-логическое устройство (АЛУ) процессора обеспечивает выполнение всех арифметических и логических операций над данными в ходе выполнения программы. АЛУ является достаточно сложным блоком и, помимо блока арифметики с фиксированной точкой, являющегося обязательным для всех машин, может включать в себя блоки арифметики с плавающей точкой и десятичной арифметики. Все арифметические операции в АЛУ выполняются сумматором - специальным блоком, обеспечивающим сложение кодов чисел. Для выполнения логических операций в составе АЛУ имеются и другие блоки (например, блок инверсии кодов, схемы логического И, ИЛИ, схемы сдвигов и др.).

Блок регистров общего назначения (**РОН**) предназначен для оперативного хранения данных, наиболее часто используемых в ходе выполнения программы. Разрядность регистров общего назначения совпадает с разрядностью процессора. Доступ к данным, хранящимся в регистрах общего назначения, осуществляется значительно быстрее, чем к данным в оперативной памяти, однако емкость регистровой памяти очень мала. Например, в составе процессора персонального компьютера IBM PC/XT/AT имеется восемь шестнадцатиразрядных регистров общего назначения, в состав процессора компьютеров семейства PDP - восемь шестнадцатиразрядных регистров, причем один из них используется в качестве программного счетчика (**R7**), и еще один - в качестве указателя стека (**R6**); в составе процессора System/360 и System/370 регистров общего назначения шестнадцать. Если в составе АЛУ имеется блок арифметики с плавающей точкой, то в блок регистров общего назначения дополнительно включаются регистры с плавающей точкой.

Помимо названных блоков в состав процессора могут входить и другие, однако в данном случае, для понимания логики функционирования компьютера, они не имеют решающего значения, и поэтому здесь не рассматриваются.

Процесс функционирования центральных узлов компьютера при выполнении программы можно упрощенно представить следующей последовательностью шагов:

1. Из оперативной памяти по адресу, записанному в PC, выбирается команда и записывается в регистр команд.

2. Команда дешифруется, т.е. из нее выделяются поля операции и операндов. Одновременно модифицируется PC, так, чтобы он указывал на следующую команду (в случае команд переменной длины алгоритм изменения PC может быть несколько иным).

3. Производится выборка операндов из оперативной памяти и/или регистров общего назначения; код операции и операнды подаются в АЛУ.

4. Выполняется операция, причем если операция арифметическая или логическая, то результат получается в регистрах АЛУ.

5. В зависимости от результата операции (положительный, отрицательный, нулевой, переполнение разрядной сетки и др.) устанавливается признак результата в регистре признаков.

6. Результат записывается в соответствующее место оперативной памяти или регистр общего назначения.

7. Происходит возврат к П.1.

В зависимости от того, какого типа команда выполняется, отдельные шаги могут опускаться или немного видоизменяться, однако общая схема функционирования, в основном, соответствует приведенной выше.

В целях наглядности рассмотрим процесс функционирования компьютера на примере простейшей функциональной модели (подробное описание модели см. Приложение 1), обеспечивающей иллюстрацию изложенной выше схемы. Наша модель, так же, как и любой компьютер, имеет процессор, включающий АЛУ; два регистра общего назначения и блок управляющих регистров; оперативную память из шестнадцати ячеек и подсистему ввода/вывода, включающую канал ввода и канал вывода.

Состояние модели в ходе ее функционирования отображается в виде специальной таблицы, содержащей значения объектов процессора и ячеек памяти в двоичном виде. Внешний вид таблицы представлен ниже на рисунке 2.1

<table>
<tr><th>адрес</th><th>память</th><th colspan="4">регистры</th><th>ввод</th></tr>
<tr><td>0000
0001</td><td>000000000000</td><td colspan="2">0</td><td colspan="2">1</td><td></td></tr>
<tr><td>0010
0011</td><td>000000000000</td><td colspan="2">000000000000</td><td colspan="2">000000000000</td><td>вывод</td></tr>
<tr><td>0100
0101</td><td>000000000000</td><td colspan="2">рег. команд</td><td colspan="2">000000000000</td><td></td></tr>
<tr><td>0110
0111</td><td>000000000000</td><td colspan="3">счетчик команд</td><td>0000</td><td rowspan="5">e -редактор
s -пошаговое
выполнение
r -непрерыв-
ное выполне-
ние
q - конец
работы</td></tr>
<tr><td>1000
1001</td><td>000000000000</td><td colspan="4">регистр признаков</td></tr>
<tr><td>1010
1011</td><td>000000000000</td><td>n</td><td>z</td><td>p</td><td>v</td></tr>
<tr><td>1100
1101</td><td>000000000000</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1110
1111</td><td>000000000000</td><td colspan="4">сумматор</td></tr>
<tr><td colspan="2"></td><td colspan="4">000000000000</td></tr>
</table>

Формат команд модели достаточно прост: первые четыре бита кодируют операцию, оставшиеся восемь бит кодируют один или два операнда. Попробуем построить простейшую программу, обеспечивающую, например, ввод двух различных чисел через канал ввода, их сложение и вывод через канал вывода. Для этого, во-первых определим последовательность действий, которые необходимо выполнить:

Ввод из канала ввода первого числа и помещение его в одну из ячеек памяти, например, в ячейку с адресом 15;

Ввод из канала ввода второго числа и помещение его в другую ячейку памяти, например, в ячейку с адресом 14;

Сложение содержимого первой ячейки памяти со второй;

Вывод содержимого ячейки памяти, содержащей результат, в канал вывода;

Останов.

Далее, каждому из вышеописанных действий поставим в соответствие машинную команду, закодировав соответствующим образом поля кода операции и операндов. Рассмотрим каждое действие из указанной последовательности отдельно. Первый пункт нашей последовательности предполагает выполнение операции ввода. В качестве первого операнда указывается адрес канала ввода (он в нашей модели всегда равен 2), в качестве второго операнда - адрес ячейки памяти, в которую помещается вводимое число. Учитывая то, что код операции ввода - 0, а вводимое число помещается по адресу 15 (1111_2) , получим следующее двоичное представление для первой команды:
$0000\ 0010\ 1111_2$. Аналогичным образом получим представление для второй команды - $0000\ 0010\ 1110_2$. Третьим действием в нашей последовательности является выполнение арифметической операции над содержимым ячеек, в которые мы ввели значения. Учитывая то, что код операции сложения содержимого двух ячеек памяти - 7 (0111_2), первый операнд размещается в ячейке с адресом 15, а второй операнд - в ячейке с адресом 14, получим следующее двоичное представление команды: $0111\ 1111\ 1110_2$. Теперь, учитывая то, что результат у нас будет размещаться на месте первого операнда, то есть в ячейке памяти с адресом 15, построим команды вывода результата в канал вывода и останова. Двоичными кодами этих команд будут $0000\ 0001\ 1111_2$ и 000000000000_2 соответственно. Теперь окончательно запишем получившуюся последовательность команд в двоичном виде:

0000 0010 1111	- ввод числа из канала ввода и помещение его по адресу 15
0000 0010 1110	- ввод числа из канала ввода и помещение его по адресу 14
0111 1111 1110	- сложение содержимого ячеек 15 и 14, результат в ячейке 15
0000 0001 1111	- вывод содержимого ячейки 15 в канал вывода
0000 0000 0000	- останов

Теперь занесем коды программ в память нашей модели в ячейки с адресами 0 - 4 соответственно. Таблица, отображающая состояние нашей модели, примет вид:

адрес	память	регистры				ввод
0000	0000001011111	0		1		
0001	0000001011110					
0010	0111111111110	0000000000000		0000000000000		вывод
0011	0000000111111					
0100	0000000000000	рег. команд		0000000000000		
0101	0000000000000					
0110	0000000000000	счетчик команд		0000		e -редактор s -пошаговое выполнение r -непрерывное выполнение q - конец работы
0111	0000000000000					
1000	0000000000000	регистр признаков				
1001	0000000000000	n	z	p	v	
1010	0000000000000					
1011	0000000000000	0	0	0	0	
1100	0000000000000					
1101	0000000000000	сумматор				
1110	0000000000000					
1111	0000000000000	0000000000000				

Обратим внимание на состояние объектов процессора в начальное состояние - программный счетчик указывает на ячейку памяти с адресом 0, в регистре команд находится 0, все биты регистра признаков (регистра флагов) сброшены. Запустим нашу модель в режим пошагового выполнения хранимой в памяти программы. Поскольку в регистре программного счетчика хранится 0, то содержимое нулевой ячейки памяти будет помещаться в регистр команд, дешифроваться и выполняться. После выполнения первой команды нашей простейшей программы процессор и память модели будут иметь следующее состояние

адрес	память	регистры				ввод
0000→	0000001011111	0		1		0000000000010
0001	0000001011110					
0010	0111111111110	0000000000000		0000000000000		вывод
0011	0000000111111					
0100	0000000000000	рег. команд		0000001011111		
0101	0000000000000					
0110	0000000000000	счетчик команд		0001		для продолжения нажмите любую клавишу
0111	0000000000000					
1000	0000000000000	регистр признаков				
1001	0000000000000	n	z	p	v	
1010	0000000000000					
1011	0000000000000	0	1	0	0	
1100	0000000000000					
1101	0000000000000	сумматор				
1110	0000000000000					
1111	0000000000010	0000000000000				

Из таблицы видно, что в регистре команд находится код команды из ячейки памяти с адресом 0, программный счетчик содержит адрес следующей команды - 1, в ячейку

памяти с адресом 15 помещены данные, введенные через канал ввода. Бит Z регистра признаков (регистра флагов) установлен в соответствии с содержимым сумматора. Продолжим выполнение нашей программы, выполнив следующую команду. При этом состояние объектов модели будет следующим

адрес	память	регистры				ввод
0000	000000101111	0		1		000000000011
0001→	000000101110					
0010	011111111110	000000000000		000000000000		вывод
0011	000000011111					
0100	000000000000	рег. команд		000000101110		
0101	000000000000					
0110	000000000000	счетчик команд		0010		для продолжения нажмите любую клавишу
0111	000000000000					
1000	000000000000	регистр признаков				
1001	000000000000	n	z	p	v	
1010	000000000000					
1011	000000000000	0	1	0	0	
1100	000000000000					
1101	000000000000	сумматор				
1110	000000000011					
1111	000000000010	000000000000				

Обратим внимание на то, что у нас опять изменилось состояние регистра команд, счетчика команд, и содержимое памяти (в ячейку памяти с адресом 14 помещено число 3). Следующей выполняемой командой будет команда сложения содержимого ячеек с адресами 15 и 14 соответственно. После ее выполнения состояние модели будет следующим:

адрес	память	регистры				ввод
0000	000000101111	0		1		
0001	000000101110					
0010→	011111111110	000000000000		000000000000		вывод
0011	000000011111					
0100	000000000000	рег. команд		011111111110		
0101	000000000000					
0110	000000000000	счетчик команд		0011		для продолжения нажмите любую клавишу
0111	000000000000					
1000	000000000000	регистр признаков				
1001	000000000000	n	z	p	v	
1010	000000000000					
1011	000000000000	0	0	1	0	
1100	000000000000					
1101	000000000000	сумматор				
1110	000000000011					
1111	000000000101	000000000101				

Опять изменилось состояние регистра команд и регистра программного счетчика. Кроме того, поскольку была выполнена арифметическая операция, изменилось состояние

регистра сумматора АЛУ, и, соответственно, изменилось состояние регистра флагов - флаг Z(флаг нулевого результата) был сброшен, а флаг P(флаг положительного результата) был установлен, поскольку содержимое регистра сумматора теперь не ноль, а $0000\ 0000\ 0101_2$ или 5_{10}. Кроме того, изменилось состояние ячейки памяти с адресом 15 - она теперь хранит результат операции сложения. Следующей будет выполнена команда вывода информации из ячейки памяти с адресом 15 в канал вывода. Состояние процессора модели опять изменится и примет вид

адрес	память	регистры				ввод
0000	000000101111	0		1		
0001	000000101110					
0010	011111111110	000000000000		000000000000		вывод
0011→	000000011111					
0100	000000000000	рег. команд		000000011111		000000000101
0101	000000000000					
0110	000000000000	счетчик команд		0100		для продолжения нажмите любую клавишу
0111	000000000000					
1000	000000000000	регистр признаков				
1001	000000000000	n	z	p	v	
1010	000000000000					
1011	000000000000	0	0	1	0	
1100	000000000000					
1101	000000000000	сумматор				
1110	000000000011					
1111	000000000101	000000000101				

Теперь у нас изменились состояния регистра команд, программного счетчика и канала вывода. Состояние регистра сумматора АЛУ и, соответственно, состояние регистра флагов не изменились (поскольку выполнялась операция ввода/вывода, а не арифметическая). Также не изменилось и состояние ячеек памяти. Последней командой нашей программы будет выполнена команда останов, переводящая процессор модели в состояние останова.

Заинтересованный читатель может сам закодировать и пронаблюдать выполнение данной программы, воспользовавшись программной моделью, находящейся на прилагаемой к данному пособию дискете в файле MICRO.EXE и документацией на модель, приведенной в приложении 1.

3. СИСТЕМА КОМАНД ПРОЦЕССОРА

Процессоры различных типов компьютеров имеют индивидуальные наборы машинных команд, различающихся как внутренним форматом, так и количеством. Например, в System360 машинный язык насчитывает 143 команды, компьютеры семейства “Электроника”, ДВК, PDP-11 без процессора плавающей арифметики имеют 64 основные команды, процессор IBM PC Original имеет набор, включающий около 300 машинных команд. Однако, несмотря на различие в количестве и внутренних форматах, наборы машинных команд процессоров различных типов компьютеров можно классифицировать по единым признакам и выделить в них родственные группы команд. Классификация может вестись по различным признакам, таким, как длина команды, тип данных, с которыми работает команда, количество операндов в команде, методы адресации, используемые для доступа к операндам, функциональное назначение команды. Следует заметить, что классификация по различным признакам часто бывает взаимосвязанной, например, длина команды зависит как от методов адресации, так и от количества операндов; с другой стороны количество операндов может определяться, например, функциональным назначением команды и, опять же, методом адресации. Наиболее информативной является классификация по функциональному назначению и методам адресации, поэтому остановимся на указанных типах классификации более подробно.

По функциональному назначению набор машинных команд процессора любого компьютера можно условно разбить на *четыре* основные группы:

* команды пересылки;
* арифметико-логические команды;
* команды переходов;
* команды управления состоянием процессора.

3.1 ГРУППА КОМАНД ПЕРЕСЫЛКИ

Группа команд пересылки включает команды, обеспечивающие пересылку данных из одной ячейки памяти в другую, из ячейки памяти в регистры и обратно, а также из регистра в регистр. Кроме того, в эту группу входят команды обмена с внешними устройствами и команды работы со стеком.

Выполнение команд данной группы с точки зрения процессора сводится к пересылке информации из одного места в другое, т.е. отсутствует обращение к сумматору

и этап пересылки значения из регистра сумматора по адресу результата. В некоторых процессорах в зависимости от пересылаемого значения может устанавливаться признак результата (**не во всех! - это скорее исключение**).

3.2 ГРУППА КОМАНД АРИФМЕТИЧЕСКИХ И ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Группа команд арифметических и логических операций включает команды, обеспечивающих выполнение над данными арифметических и логических операций (логическое И, ИЛИ, НЕ, исключающее ИЛИ), а так же операций арифметических и логических сдвигов.

Подгруппа команд арифметических операций реализует *бинарные* операции (т.е. операции над двумя операндами) “+”, “-“,”*”, “/” (“сложить”, “вычесть”, “перемножить”, “разделить”). Если с операциями сложения и вычитания все достаточно просто, то реализация операций умножения и деления требует дополнительных пояснений. В частности, *большинство процессоров вычислительных систем реализуют операцию умножения таким образом, что результат операции имеет двойную длину, то есть, если выполняется операция перемножения данных в формате слова, то результат будет иметь длину двойного слова. Например, если мы перемножаем содержимое регистра и ячейки памяти, то результат будет помещаться в паре регистров - старшая часть результата в одном регистре, младшая - в другом. В какой паре и каким образом - определяется архитектурой конкретного процессора. Аналогичная ситуация наблюдается и в случае операции деления - делимое обычно размещается в паре регистров, результат так же занимает два регистра - в одном размещается частное, в другом - остаток от деления*. Объясняется это особенностями аппаратной реализации мультипликативных операций в процессорах.

Подгруппа логических операций реализует бинарные операции И(**AND**), ИЛИ(**OR**), ИСКЛЮЧАЮЩЕЕ ИЛИ(**XOR**), а также унарную операцию НЕ(**NOT**). Операции реализуются побитно, то есть выполняются попарно над соответствующими битами операндов. Правила выполнения операций можно проиллюстрировать следующими табличками:

бинарная операция И(AND)

операнд1	**операнд2**	**Результат**
0	0	0

1	0	0
0	1	0
1	1	1

Исходя из этого 0101 0011 AND 0001 1101 будет равно 0001 0001

бинарная операция ИЛИ(OR)

операнд1	**операнд2**	**результат**
0	0	0
1	0	1
0	1	1
1	1	1

Таким образом, 0101 0011 OR 0001 1101 будет равно 0101 1111

бинарная операция ИСКЛЮЧАЮЩЕЕ ИЛИ(XOR)

операнд1	**операнд2**	**результат**
0	0	0
1	0	1
0	1	1
1	1	0

Получили, что 0101 0011 XOR 0001 1101 будет равно 0100 1110

унарная операция НЕ(NOT)

операнд	**результат**
0	1
1	0

Исходя из этого NOT 0101 0011 будет равно 1010 1100

Подгруппа команд сдвигов обеспечивает побитные сдвиги содержимого операнда влево или вправо на один или несколько разрядов. В зависимости от метода реализации различается сдвиг влево, сдвиг вправо арифметический или логический, сдвиг влево или вправо циклический с учетом или без учета содержимого флага переноса.

Сдвиг влево сдвигает биты операнда влево. Старший бит сдвигается во флаг переноса, а младший очищается. Например:

до сдвига флаг переноса=0 операнд = 1010 1010

после флаг переноса=1 операнд = 0101 0100

Нетрудно заметить, что если рассматривать содержимое байта как целое без знака, то сдвиг влево на один двоичный разряд эквивалентен умножению на 2. Действительно, возьмем, например число $86_{10} = 0101\ 0110_2$, сдвинем двоичное представление числа на один разряд влево, получим $1010\ 1100_2 = 172_{10}$.

Сдвиг вправо арифметический сдвигает биты операнда вправо. Старший бит рассматривается, как знаковый, и остается неизменным, младший бит сдвигается во флаг переноса. Например:

до сдвига флаг переноса=0 операнд = 1010 0010

после флаг переноса=0 операнд = 1101 0001

Если рассматривать содержимое байта, как целое со знаком, то сдвиг вправо на один двоичный разряд эквивалентен делению на 2 числа со знаком. Действительно, в вышеприведенном примере число $1010\ 0010_2 = -\ 94_{10}$, в результате сдвига получим $11010001_2 = -\ 47_{10}$.

Сдвиг вправо логический так же сдвигает биты операнда вправо. Но при этом старший бит сбрасывается в 0 , младший бит сдвигается во флаг переноса. Например:

до сдвига флаг переноса=0 операнд = 1010 0010

после флаг переноса=0 операнд = 0101 0001

Если рассматривать содержимое байта, как целое без знака, то сдвиг вправо на один двоичный разряд эквивалентен делению на 2 числа без знака. Действительно, в вышеприведенном примере число $1010\ 0010_2 = 162_{10}$, в результате сдвига получим $01010001_2 = 81_{10}$.

Циклические сдвиги (называемые иногда вращениями) отличаются от вышеописанных тем, что при сдвиге влево на один разряд выдвигающийся старший бит помещается в младший, а при сдвиге вправо - наоборот, например:

циклический сдвиг влево:

до сдвига флаг переноса=0 операнд = 1010 0010

после флаг переноса=1 операнд = 0100 0101

циклический сдвиг вправо:

до сдвига флаг переноса=0 операнд = 1010 0011

после флаг переноса=1 операнд = 1101 0001

С точки зрения процессора при выполнении группы арифметико-логических команд реализуется полная последовательность шагов, рассмотренная в разделе 2. Все

команды данной группы воздействуют на установку признака результата в регистре признаков.

3.3 ГРУППА КОМАНД ПЕРЕХОДОВ

Группа команд переходов может быть условно поделена на несколько подгрупп :

* подгруппа команд безусловных переходов;
* подгруппа команд условных переходов;
* подгруппа команд организации циклов;
* подгруппа команд организации подпрограмм;
* подгруппа команд работы с прерываниями.

Подгруппа команд безусловных переходов в простейшем случае может состоять из единственной команды безусловного перехода, обеспечивающей нарушение естественного хода выполнения программы. Команда является одноопернадной, в качестве операнда тем или иным способом указывается адрес перехода. Выполнение команды с точки зрения процессора сводится к записи операнда команды в программный счетчик (PC).

Подгруппа команд условных переходов обеспечивает ветвления в программе в зависимости от состояния регистра флагов. Выполнение процессором команд ветвления заключается в следующем. После дешифровки команды проверяется установка соответствующих битов в регистре признаков, например, если в регистр команд процессора поступает команда “переход по отрицательному результату”, то проверяется состояние флага N (флаг отрицательного результата). Если соответствующий бит в регистре флагов установлен в единицу, то осуществляется переход по заданному операндом команды адресу, т.е. выбирается адрес, закодированный в поле операндов в команде, и помещается в программный счетчик. Если соответствующий бит (ы) в регистре флагов не установлен (ы), то естественный ход программы не нарушается, PC изменяется обычным образом и указывает на команду, следующую за командой перехода. Проиллюстрируем это простым примером, используя систему команд уже известной нам функциональной модели компьютера (Приложение 1). Только теперь вместо кодов операций будем использовать их мнемонические обозначения, а адреса операндов и ячеек памяти будем кодировать в шестнадцатеричной системе счисления. Попробуем построить фрагмент программы для вычисления выражения вида:

$$Y:=\begin{cases} a+b, a\le b \\ a-b, a>b \end{cases}$$

Предположим, что значения переменных A и B у нас хранятся в ячейках с адресами $0F_{16}$ и $0E_{16}$ соответственно, а результат (Y) мы хотим поместить в ячейку с адресом $0D_{16}$. Поскольку наша простейшая модель не имеет команд сравнения двух операндов, обеспечивающих установку содержимого регистра флагов в зависимости от результата сравнения, воспользуемся для этой цели командой вычитания. Фрагмент программы с учетом высказанных замечаний примет вид:

```
0    MOVR 0, 0F  загрузить в регистр 0 содержимое ячейки памяти с адресом 15
1    SRM  0, 0E  вычесть из содержимого регистра 0 содержимое ячейки 14
2    JP   06     в случае, если  флаг P=1, перейти к вып. команды с адресом 6
3    MOVR 0,0F   загрузить в регистр 0 содержимое ячейки памяти с адресом 15
4    ARM  0, 0E  сложить содержимое регистра 0 и ячейки памяти с адресом 14
5    JMP  08     перейти к выполнению команды с адресом  8
6    MOVR 0, 0F  загрузить в регистр 0 содержимое ячейки памяти с адресом 15
7    SRM  0, 0E  из содержимого регистра 0 вычесть содержимое ячейки 14
8    MOVM 0, 0D запомнить содержимое регистра 0 в ячейке с адресом 13
9    STOP
```

При выполнении данной программы в случае, если A>B, последовательно будут выполнены команды с адресами 0,1,2,6,7,8, в противном случае последовательность адресов выполняемых команд будет другой - 0,1,2,3,4,5,8. В этом легко можно убедиться, преобразовав последовательность команд в коды модели, загрузив коды в модель, и запустив ее в режиме пошагового выполнения. Читателю предлагается проделать это самостоятельно.

Особого внимания заслуживают команды организации цикла, из которых наибольшее распространение получила команда организации цикла со счетчиком (имеется в наборе команд System 360/370, PDP, IBM PC, VAX и ряда других). В качестве операндов в данной команде указывается один из регистров общего назначения и адрес перехода. Выполнение ее процессором реализуется следующим образом:

* из содержимого регистра, указанного в качестве операнда в команде, вычитается единица и результат записывается в тот же регистр;
* опрашивается регистр признаков; если признак нулевого результата не установлен, то выбирается адрес, закодированный вторым операндом команды и записывается в PC; если результат операции равен нулю (т.е. в результате вычитания единицы из регистра получился нуль), то PC изменяется обычным образом и указывает на команду, следующую за командой организации цикла.

Таким образом, команда организации цикла по счетчику эквивалентна двум командам: “вычитание из содержимого регистра 1” и “условный переход по ненулевому флагу результата”. Следует заметить, что регистр-счетчик в качестве операнда в команде может быть и не указан, в этом случае предполагается что регистр счетчик определяется однозначно архитектурой процессора и стандартными соглашениями о связях. Рассмотрим использование команды организации цикла опять же на примере нашей модели. Попробуем построить программу для вычисления выражения $Y := A^X$, причем операцию возведения в степень представим, как последовательное умножение 1 на A X раз. Так же, как и в предыдущем случае, операнд A разместим в ячейке памяти с адресом $0F_{16}$, операнд X разместим в ячейке $0E_{16}$, а результат запишем в ячейку $0D_{16}$. Кроме того, в ячейку с адресом $0C_{16}$ поместим константу 1. Обратим внимание на то, что команда организации цикла в нашей модели предполагает, что счетчик цикла всегда размещается в регистре 0, поэтому в команде в качестве операнда регистр-счетчик не указывается. Фрагмент программы будет выглядеть следующим образом.

```
0   MOVR 1,0C   запишем в регистр 1 содержимое ячейки 12 - константу 1
1   MOVR 0,0E   запишем в регистр 0 содержимое ячейки 14 - показатель степени
2   MRM  1,0F   выполним очередную операцию умножения регистра на A
3   LOOP 02     замкнем цикл с началом по адресу 2.
4   MOVM 1,0D   запишем содержимое регистра 1 - результат - по адресу 0D
5   STOP        останов
```

Так же, как и в предыдущем случае, работу программы на модели читателю предлагается проверить самостоятельно.

Выполнение команды перехода к подпрограмме вызывает, во-первых, запоминание текущего значения PC либо в специально отведенных ячейках памяти, либо в одном из регистров общего назначения, либо в верхушке стека, и, во-вторых, засылку в PC адреса первой выполняемой команды подпрограммы. Команда возврата из подпрограммы извлекает запомненный командой перехода к подпрограмме адрес возврата и записывает его в PC. Особенности использования этих команд наряду с особенностями организации подпрограмм мы обсудим несколько позже.

Аналогичным образом действуют команды организации программного прерывания и возврата из прерывания; существенное их отличие от команд организации подпрограмм состоит в том, что дополнительно запоминается состояние регистра признаков, которое при возврате из прерывания восстанавливается. Их мы также обсудим в дальнейшем отдельно.

3.4 ГРУППА КОМАНД УПРАВЛЕНИЯ СОСТОЯНИЕМ ПРОЦЕССОРА

Группа команд управления состоянием процессора включает команды останова процессора, перевода процессора в состояние ожидания внешнего прерывания, разрешения и запрещения прерываний, сброса и установки битов регистра признаков. Обычно команды этой группы не влияют на содержимое ячеек памяти и регистров общего назначения процессора, модифицируя только биты регистра признаков и внутренние флаги процессора.

4. МЕТОДЫ АДРЕСАЦИИ ПАМЯТИ

При доступе к операндам, размещенным в регистрах, памяти и портах ввода/вывода выделяются следующие методы адресации:

* прямой (абсолютной адресации);
* относительной адресации;
* индексной адресации;
* непосредственной адресации;
* косвенной адресации;
* неявной адресации.

Прямая (абсолютная) адресация предполагает, что в поле операндов в команде задан абсолютный (физический) адрес ячейки памяти, в которой расположен операнд. Данный способ адресации является самым простым и достаточно универсальным, однако его применение вызывает ряд трудностей, а именно:

* сложность, а иногда и невозможность перемещения программы в памяти;
* низкая эффективность обработки массивов данных и данных сложной структуры.

Вследствие этого данный метод адресации находит ограниченное применение - либо в простых компьютерах с ограниченным набором команд и небольшим объемом ОП (управляющие микроЭВМ, например), либо при написании системных программ, использующих служебные области памяти с фиксированными адресами, непосредственно работающих с внешними устройствами, имеющими фиксированные адреса управляющих и информационных регистров (например, драйверы внешних устройств, подпрограммы обработки прерываний и др.). Кроме того, в качестве частного случая прямой адресации можно рассматривать регистровую адресацию, поскольку в этом случае в поле операндов команды указывается абсолютный адрес регистра в блоке регистров общего назначения.

Относительная адресация обеспечивает перемещение программного модуля в памяти компьютера, т.е. позволяет создавать программы, способные работать в любых адресах памяти. В связи с этим относительная адресация широко распространена в различных типах вычислительных систем. Идея относительной адресации достаточно проста и заключается в том, что в поле операндов команды указывается не абсолютный адрес операнда, а его ***смещение*** относительно начала или некоторой точки программы, называемой **базовой точкой**. Адрес этой точки программы загружается в специальный регистр (его роль может выполнять один из регистров общего назначения), называемый *базовым* регистром. Адрес (номер) этого регистра также указывается в поле операндов команды. На этапе выполнения абсолютный адрес операнда вычисляется как сумма

содержимого базового регистра B и смещения D, т.е. **A = (B) + D**, где (B) обозначает, что берется содержимое некоторого базового регистра B.

Пусть, например, программа хранится в памяти, начиная с адреса 1000. В программе имеется команда, обращающаяся к операнду по адресу 1400. Базовый регистр B будет содержать адрес начала программы, т.е. (B) = 1000, а смещение D будет составлять 1400-1000 = 400. В поле операндов команды будет храниться только номер базового регистра B и смещение 400. Т.е. в поле операндов не будет храниться абсолютный адрес, и при переносе программы в другую область памяти содержимое адресных полей команды модифицировать не надо, поскольку величина смещения не зависит от адреса начала программы, также как и адрес используемого базового регистра. Необходимо только позаботиться о том, чтобы к моменту выполнения команды в базовый регистр был занесен адрес начала программы. Обычно эта операция возлагается на программиста. В общем случае, различные фрагменты программы могут использовать различные базовые регистры (в этом случае их может быть несколько), либо различную базовую точку (в этом случае при достижении определенных точек требуется перезагрузка базового регистра соответствующим базовых адресом).

В качестве базового регистра может использоваться и программный счетчик. В этом случае мы имеем достаточно интересную форму адресации, когда базовая точка “плывет” по программе синхронно с изменением программного счетчика. Это позволяет нам сэкономить базовый регистр, однако вынуждает работать не только с положительной, но и с отрицательной величиной смещения, что, к сожалению, архитектурой подсистемы адресации некоторых процессоров просто не допускается.

Индексная адресация применяется в работе с элементами массивов. При индексной адресации адрес операнда вычисляется как сумма адреса начала массива и **индексного смещения**, определяющего смещение элемента массива относительно первого элемента, т.е.

A = AN + (X), где (X) - содержимое индексного регистра.

Индексное смещение хранится в специальном индексном регистре. В качестве индексных регистров могут использоваться как регистры общего назначения (System360/370, PDP) так и специальные индексные регистры (IBM PC - регистры SI, DI). В команде выделено специальное поле для хранения адреса индексного регистра.

Для доступа к элементам массива индексное смещение, хранимое в индексном регистре рассчитывается на основе индекса (индексов) элемента массива по следующим формулам:

а) одномерный массив:

X = (i - 1) * Lэл, где i - индекс элемента массива, а Lэл - длина элемента в байтах.

б) двумерный массив:

X = ((i - 1) * Jmax + j - 1) * Lэл, где

i - номер строки, j - номер столбца, Jmax - максимальное количество столбцов; при этом предполагается, что двумерный массив хранится в памяти по строкам, то есть так, что сначала последовательно располагаются элементы первой строки, затем второй и т.д.

в) трехмерный массив:

X = ((i - 1) * Jmax * Kmax + (j - 1) * Kmax + k-1) * Lэл, где

i, j, k - индексы элемента.

В реальных условиях часто используется комбинированная *индексно-относительная* адресация, при которой адрес элемента массива вычисляется как сумма трех величин **A = (B) + (X) + D**.

При этом во внутреннем формате команды выделяются поле базового регистра, поле индексного регистра и поле смещения.

В программах очень часто необходимо выполнять операции, в которых один из операндов является константой, не изменяющейся в ходе выполнения программы. Можно записать эту константу в ячейку памяти и адресовать ее обычным образом, однако, существует и другой способ, а именно - *непосредственная адресация*. В случае непосредственной адресации операнд записывается прямо в команде вместо адреса операнда. Подобный способ адресации значительно ускоряет выполнение команды, поскольку не требуется вычисление адреса операнда и его выборка из оперативной памяти.

В случае *косвенной адресации* в поле операндов команды указывается не адрес операнда, а адрес *указателя* на операнд, т.е. адрес регистра или ячейки памяти, содержащей адрес операнда. В цепочке указателей может быть более одного указателя, т.е. команда может содержать адрес указателя; указатель содержит адрес следующего указателя и т.д.; последний указатель содержит адрес операнда.

Количество указателей в цепочке называется **кратностью** косвенной адресации. Адрес операнда, хранимый в указателе, называется **косвенным адресом**. В ходе выполнения программы адрес указателя может оставаться постоянным, а косвенный адрес может изменяться командами самой программы. Таким образом обеспечивается, например, обработка динамических данных сложной структуры, т.е. данных, память под которые запрашивается динамически, в процессе выполнения программы. В этом случае, на этапе разработки программы память под элементы данных не выделяется, следовательно адреса элементов данных просто неизвестны, и программу можно строить, оперируя косвенными адресами.

Команды косвенной адресации выполняются дольше за счет дополнительного обращения к памяти по цепочке указателей, однако, во многих случаях их использование позволяет резко повысить эффективность программы.

Автоинкрементная и автодекрементная формы адресации являются дальнейшим развитием косвенной адресации и отличается от нее тем, что в случае автоинкрементной адресации после выполнения команды указатель автоматически увеличивается на определенную величину (слово или байт); в случае автодекрементной - уменьшается перед выполнением операции. В этом смысле адресацию к верхушке стека с использованием регистра-указателя стека можно рассматривать, как частный случай автоинкрементной (при записи в стек) и автодекрементной (при чтении из стека) адресации.

Неявная адресация имеет место в тех случаях, когда операнд в поле операндов не указывается, а его использование в той или иной команде определяется самой архитектурой процессора вычислительной системы. Например, архитектура процессора нашей первой простейшей модели компьютера предполагает, что в качестве регистра-счетчика всегда используется нулевой регистр, поэтому в команде организации цикла LOOP в качестве операнда явно он не указывается, но его содержимое при выполнении команды используется.

5. ОРГАНИЗАЦИЯ ПОДПРОГРАММ И ПРЕРЫВАНИЙ

5.1 ПОНЯТИЕ ПОДПРОГРАММ

В настоящее время при разработке программ принято придерживаться концепции *модульного программирования*, в соответствии с которой любая достаточно сложная программа разбивается на отдельные модули, каждый из которых невелик по объему, достаточно автономен и выполняет одну или ограниченное число строго определенных функций по обработке данных. Применяется эта концепция исходя из следующих соображений.

Во-первых, появляется возможность выделить в программе одинаковые фрагменты и оформить их отдельным модулем, используя его каждый раз, когда требуется выполнить соответствующие вычисления. Это позволяет значительно сократить объем программы.

Во-вторых, в подавляющем большинстве случаев гораздо легче спроектировать и отладить десяток небольших модулей, реализующих достаточно простые функции, а затем связать их между собой с помощью опять же небольшой управляющей программы, чем проектировать и отлаживать большую программу, созданную единым модулем, часто со сложной, сетевой структурой.

В-третьих, появляется возможность коллективной разработки крупных проектов программ коллективами программистов, при этом каждый из них выполняет отдельный, функционально автономный и самостоятельный модуль в соответствии с полученными спецификациями.

С точки зрения подчиненности модули подразделяются на главную (вызывающую) программу и вызываемую программу, или подпрограмму. Механизм использования подпрограмм следующий. Главная программа, когда ей требуется произвести те или иные действия, обращается к подпрограмме. Подпрограмма выполняет требуемые действия, после чего продолжает выполняться главная программа, причем выполнение ее продолжается с точки, следующей за точкой обращения к подпрограмме. При работе с подпрограммами принято использовать следующую терминологию:

* место (адрес) команды, которая обеспечивает обращение к подпрограмме, называется *точкой вызова*;

* первая выполняемая программа (оператор) подпрограммы называется *точкой входа* (соответствующий адрес - адрес входа в подпрограмму);

* команда или оператор, непосредственно следующий за точкой вызова подпрограммы, называется *точкой возврата*; соответственно, адрес, по которому расположена команда, следующая за командой обращения к подпрограмме, называется адресом возврата.

5.2 МЕХАНИЗМЫ ПЕРЕДАЧИ И ВОЗВРАТА УПРАВЛЕНИЯ

Рассмотрим внутренние механизмы передачи управления подпрограмме и возврата управления подпрограммой.

Очевидно, что для того, чтобы начала выполняться последовательность команд, реализующая алгоритм подпрограммы, необходимо в программный счетчик загрузить адрес точки входа в подпрограмму, а для того, чтобы обеспечить возврат из подпрограммы, соответствующий адрес возврата необходимо где-то сохранить.

В составе процессоров всех компьютеров имеются специальные команды, реализующие эти действия, т.е. загрузку в программный счетчик адреса точки входа и запоминание адреса возврата. Реализация этих команд определяется архитектурой процессора и принятыми в данной архитектуре соглашениями о связях. Например, в System360/370 команда перехода к подпрограмме запоминает адрес точки возврата в одном из регистров общего назначения процессора, указанном в качестве операнда команды(в соответствии с используемыми в этих системах соглашениями о связях обычно в регистре 14). В команде возврата из подпрограммы в качестве операнда должен быть указан тот же самый регистр. При наличии в архитектуре процессора стека вполне естественно его использование в качестве места временного хранения адреса возврата.

Например, архитектура процессоров семейства PDP для передачи управления подпрограмме предусматривает использование команды **JSR R, ADR**, которая обеспечивает выполнение следующих действий:

R↓(SP) - содержимое регистра R, называемого регистром связи, помещается в стек;

PC→R - содержимое программного счетчика(в момент выполнения команды содержит адрес возврата) помещается в регистр связи;

ADR→PC - адрес точки входа в подпрограмму, указанный в команде, помещается в программный счетчик.

При возврате управления подпрограммой с помощью команды RTS R выполняется обратная последовательность действий:

R→PC - содержимое регистра связи, содержащее адрес возврата, помещается в программный счетчик;

(SP)↑R - содержимое вершины стека извлекается в регистр связи.

Аналогичным образом осуществляется передача управления подпрограмме и возврат в процессорах Intel (Intel8080, Intel8085, семейство Intel80x86, Pentium). Для передачи управления используется команда **CALL ADDR** , которая обеспечивает, во-первых, запись текущего значения программного счетчика (в момент выполнения команды он содержит адрес возврата) в стек - PC↓(SP), во-вторых, запись адреса точки входа в подпрограмму (ADDR) в программный счетчик - ADDR→PC.

Для возврата управления в точку возврата используется команда RET, обеспечивающая извлечение из вершины стека адреса возврата и его запись в программный счетчик - (SP)↑PC.

В общем случае, при организации подпрограмм должны выполняться следующие требования.

1. Подпрограмма может в ходе своей работы использовать регистры процессора. Для того, чтобы не испортить значения регистров основной программы, подпрограмма должна начинать свою работу с сохранения содержимого используемых ей регистров процессора, а перед возвратом в основную программу обеспечивать восстановление их содержимого. Содержимое регистров может сохраняться либо в стеке, либо в области памяти, доступной подпрограмме.

2. Для стековых машин перед возвратом управления из подпрограммы в основную программу стек должен быть *сбалансирован*, т.е. при выходе из подпрограммы он должен иметь то же состояние, что и при входе в подпрограмму.

5.3 МЕХАНИЗМЫ ПЕРЕДАЧИ ПАРАМЕТРОВ

При передаче управления подпрограмме, в подавляющем большинстве случаев, ей передаются некоторые *параметры*, т.е. конкретные значения, с которыми оперирует алгоритм подпрограммы, например, если мы в ходе каких-либо вычислений обращаемся к некоторой подпрограмме, вычисляющей Max(A,B), то в качестве параметров в этом случае мы должны передать пару значений, а подпрограмма должна вернуть максимальное значение - еще один параметр. Таким образом, параметры можно подразделить на *входные* параметры, и *выходные*. Кроме того, в подпрограмму можно передавать как конкретные значения некоторых переменных, так и адреса ячеек памяти, в которых эти значения расположены. В первом случае мы имеем дело с передачей параметров *по значению*, а во втором - *по ссылке* (или по адресу). Обратим внимание на то, что в случае передачи параметров по значению изменение их в подпрограмме не

приводит к изменению исходных значений в вызывающей программе, чего нельзя сказать в случае передачи параметров по ссылке. При рассмотрении внутренних механизмов передачи параметров можно выделить несколько способов передачи, а именно:

* передача параметров с использованием общей области памяти;
* передача параметров через регистры процессора;
* передача параметров через стек;
* передача параметров комбинированными способами, в том числе через использование таблицы адресов параметров.

Рассмотрим эти способы более подробно.

5.3.1 Передача параметров через общую область памяти

Передача параметров с использованием общей области памяти предполагает, что в оперативной памяти сформирована некоторая область памяти, доступная как главной программе, так и подпрограмме. Структура этой области жестко фиксирована и также известна программе и подпрограмме. Перед передачей управления подпрограмме главная программа помещает значения параметров в общую область, оттуда же извлекаются результаты работы подпрограммы. Подпрограмма, в свою очередь, извлекает из общей области переданные ей значения параметров и помещает в общую область результаты работы. Проиллюстрируем это небольшим примером, используя систему команд МИКРО2 (Приложение 2). Предположим, что в ходе работы некоторой программы нам неоднократно требуется вычисление значения выражения $Y:=Z^X$. Имеет смысл организовать эти вычисления в виде подпрограммы с двумя входными параметрами (Z, X) и одним выходным (Y). Таким образом, общая область у нас будет содержать три значения - значение основания (Z), значение показателя степени (X) и значение результата (Y), причем именно в том порядке, в котором они перечислены.

Фрагмент главной программы, обеспечивающей запись значений параметров в общую область памяти, примет вид:

```
. . .
load a, AAA  ;   загрузка в регистр а содержимого AAA (основание)
stor a, Z    ;   запись основания в общую область
load a, BBB  ;   загрузка в регистр а содержимого BBB (показатель степени)
stor a, X    ;   запись показателя степени общую область
call POWER ;     вызов подпрограммы возведения в степень
load a,Y     ;     загрузка в регистр а результата из общей области
...
```

Описание общей области будет выглядеть следующим образом:

```
Z:      .ds 1
X:      .ds 1
Y:      .ds 1
```

Подпрограмма будет иметь вид:

```
POWER:  push b          ; сохранить содержимое регистра B в стеке
        push a          ; сохранить содержимое регистра A в стеке
        mvi b,1         ; в регистре B будет Z^X
        load a,X        ; загрузили в регистр A значение из общей области
ICYC:   push a          ; сохранить содержимое регистра A в стеке
        mum a,Z         ; выполнить умножение на Z из общей области
        pop a           ; восстановить из стека содержимое  регистра A
        loop a,ICYC     ; организовать цикл по умножению
        stor b,Y        ; запомнить результат в общей области
        pop a           ; восстановить содержимое регистра A
        pop b           ; восстановить содержимое регистра B
        ret             ; возврат из подпрограммы
```

Передача параметров подпрограмме через общую область является достаточно простым и понятным способом, однако , с другой стороны, подобный способ передачи параметров не всегда удобен тем, что ограничивает такое свойство подпрограммы, как универсальность - подпрограмма оказывается жестко привязанной к адресам и структуре общей области, что часто бывает нежелательным.

5.3.2 Передача параметров через регистры процессора

Передача параметров через регистры процессора предполагает, что значения параметров (или их адресов) перед вызовом подпрограммы помещаются в регистры общего назначения процессора, выходные параметры подпрограмма так же передает через регистры, помещая в них значения перед возвратом управления. Ниже приведен пример, иллюстрирующий данный способ для той же подпрограммы возведения в степень.

```
. . .
load b, Z       ;   загрузка в регистр a содержимого ячейки Z (основание)
load a, X       ;   загрузка в регистр a содержимого ячейки X (показатель степени)
call POWER      ;   вызов подпрограммы возведения в степень
stor a,Y        ;   запись результата работы п/п по символическому адресу Y
. . .
;       подпрограмма вычисления Z^X
;       вход    RA - X; RB - Z; выход: RA - Z^X
```

```
POWER:  push b          ; сохранить основание в стеке
        mvi b,1
CYCL:   push a          ; сохранить количество умножений в стеке
        mum a,2(sp)     ; выполнить очередное умножение
        pop a           ; восстановить счетчик цикла
        loop a, CYCL    ; замкнуть цикл
        mov a,b         ; переслать результат в регистр a
        pop b           ; сбалансировать стек
        ret             ; возврат в основную программу
```

В данном примере основание и показатель степени передаются в подпрограмму через регистры процессора B и A соответственно. Результат возвращается через регистр A.

Следует заметить, что передача параметров через регистры является стандартным способом в случае подпрограмм-функций, имеющих единственный входной и единственный выходной параметры. Таким образом реализуются, например, стандартные встроенные функции систем программирования PASCAL, C и ряда других. Это такие функции, как Sin(x), Cos(x), Abs(x), ln(x) и другие.

5.3.3 Передача параметров через стек

Передача параметров через стек предполагает, что вызывающая программа перед обращением к подпрограмме помещает параметры в стек, при этом резервируется место и для выходных параметров. После возврата управления подпрограммой главная программа обеспечивает очистку стека, извлекая из него значения входных и выходных параметров.

Подпрограмма оперирует со значениями параметров либо непосредственно в стеке, используя адресацию относительно вершины стека, либо выталкивает значения параметров в регистры процессора, а затем использует. Для нашего случая подпрограммы возведения в степень фрагмент главной программы и подпрограмма примут вид:

```
. . .
push a      ;  резервирование в стеке места для выходного параметра - Y
load a, Z   ;  загрузка в регистр a содержимого ячейки Z (основание)
push a      ;  запись параметра в стек
load a, X   ;  загрузка в регистр a содержимого ячейки X (показатель степени)
push a      ;  запись параметра в стек
call POWER  ;  вызов подпрограммы возведения в степень
pop a       ;  извлечение из стека значения X
pop a       ;  извлечение из стека значения Z
```

```
        pop a           ;   извлечение из стека значения Y
        stor a,Y        ;   запись результата работы п/п по символическому адресу Y
        . . .
;       подпрограмма вычисления Z^X
;       параметры передаются и возвращаются через стек
;       состояние стека при входе в подпрограмму
;   (sp+1)  | адрес возврата          |
;   (sp+2)  | значение X              |
;   (sp+3)  | значение Z              |
;   (sp+4)  | ячейка под значение Y   |
;
POWER:  push b          ; сохранить содержимое b регистра в стеке
        mvi b,1         ;
        load a,3(sp)    ; загрузить в регистр a показатель степени
CYCL:   push a          ; сохранить количество умножений в стеке
        mum a,5(sp)     ; выполнить очередное умножение
        pop a           ; восстановить счетчик цикла
        loop a, CYCL    ; замкнуть цикл
        stor b,5(sp)    ; переслать результат в стек на зарезервированное место
        pop b           ; восстановить содержимое регистра b из стека
        ret             ; возврат в основную программу
```

Передача параметров через стек в настоящее время является наиболее часто используемым способом в современных системах программирования. Если через стек передаются сами значения параметров, то реализуется передача по значению, в случае, если в стек записываются адреса параметров, реализуется механизм передачи параметров по ссылке.

5.3.4 Передача параметров через таблицу адресов параметров

В заключение рассмотрим механизм передачи параметров с использованием таблицы адресов параметров. Данный способ позволяет передать множество параметров через единственный объект (регистр или ячейку стека), кроме того, с помощью этого способа достаточно просто реализуется механизм передачи переменного числа параметров подпрограмме. Идея реализации состоит в том, что в памяти организуется таблица (одномерный массив) элементами которой являются адреса параметров. Адрес этой таблицы перед передачей управления подпрограмме помещается в один из регистров

процессора или в стек. Подпрограмма может получить доступ к параметрам, используя косвенно-регистровую адресацию кратности 2. В случае использования переменного числа параметров перед вызовом подпрограммы в первый элемент таблицы заносится количество параметров, использующееся в данном обращении к подпрограмме. Проиллюстрируем вышесказанное на примере все той же программы возведения Z в степень X.

```
        . . .
; формирование таблицы адресов параметров
        mvi a, Z            ; загрузка в регистр A адреса Z
        stor a, TABL        ; запись адреса Z в первый элемент таблицы
        mvi a, X            ; загрузка в регистр A адреса X
        stor a, TABL1       ; запись адреса X во второй элемент таблицы
        mvi a, Y            ; загрузка в регистр A адреса Y
        stor a, TABL2       ; запись адреса Y в третий элемент таблицы
; таблица адресов сформирована
        . . .
        mvi a, TABL         ; загрузка в регистр A адреса таблицы адресов параметров
        call POWER          ;  вызов подпрограммы возведения в степень
        load a,Y            ;  загрузка в регистр a результата для дальнейшей работы
        ...
; описание  области  данных
        Z:      .ds 1
        X:      .ds 1
        Y:      .ds 1
; описание таблицы адресов параметров
        TABL: .ds 1
        TABL1:.ds 1
        TABL2:.ds 1
        . . .
        ;       подпрограмма вычисления Z^X
        ;       вход    RA - адрес таблицы адресов параметров
POWER:  push b          ; сохранить регистр B в стеке
        push a          ; сохранить регистр A(адрес таблицы) в стеке
        load a,0(a)     ; в регистре a - адрес первого параметра
        load a,0(a)     ; в регистре a - первый параметр - основание
        load b,1(sp)    ; загрузить в регистр b адрес таблицы
```

```
        load b,1(b)     ; в регистре b - адрес второго параметра
        load b,0(b)     ; в регистре b - второй параметр - показатель степени
        xchg            ; обмен содержимого регистров a и b
        push b          ; сохранить основание в стеке
        mvi b,1
CYCL:   push a          ; сохранить количество умножений в стеке
        mum a,2(sp)     ; выполнить очередное умножение
        pop a           ; восстановить счетчик цикла
        loop a, CYCL    ; замкнуть цикл
        pop a           ; вытолкнуть из стека основание(оно больше не нужно)
        load a,1(sp)    ; загрузить в регистр A адрес таблицы адресов параметров
        load a,2(a)     ; загрузить в регистр A адрес третьего параметра
        stor b,0(a)     ; запомнить результат по адресу третьего параметра
        pop a           ; восстановить содержимое регистра A
        pop b           ; восстановить содержимое регистра B
        ret             ; возврат в основную программу
```

5.4 ПРЕРЫВАНИЯ

Прерывание - это временное прекращение выполнения процессором последовательности команд одной программы с целью выполнения другой, имеющей в данный момент времени более высокий приоритет.

Прерывания подразделяются на аппаратные и программные. Аппаратные прерывания, в свою очередь, подразделяются на внешние аппаратные прерывания и внутренние, называемые также исключительными ситуациями или просто исключениями.

Внешние аппаратные прерывания вызываются внешними по отношению к процессору устройствами компьютера и сигнализируют процессору о том, что он должен прервать выполнение текущей программы и заняться обслуживанием внешнего устройства. Подобная схема обслуживания внешних устройств избавляет процессор от необходимости периодически опрашивать состояние внешних устройств, непроизводительно затрачивая на это время. Достаточно иллюстративным примером подобного подхода является организация работы клавиатуры персонального компьютера - при нажатии любой клавиши возникает прерывание от клавиатуры, в ответ на которое процессор заканчивает выполнение текущей команды, выполняемой в настоящий момент программы и переключается на обработку прерывания, которая заключается в

определении кода нажатой клавиши, и помещении этого кода в специальный системный буфер. В качестве еще одного примера можно привести организацию работы системного таймера, генерирующего прерывания через определенные промежутки времени, в этом случае обработка прерывания сводится к увеличению временных счетчиков, так же хранящихся в специальных системных переменных.

Аппаратные прерывания не координируются с работой программного обеспечения. Когда процессор распознает прерывание, он заканчивает выполнение текущей команды, а затем переходит к процессу обработки прерывания, заключающемуся в выполнении специальной программы - обработчика прерывания. После того, как прерывание обработано, процессор продолжает выполнение прерванной программы с той точки, в которой она была прервана. Для реализации механизма обработки прерываний в составе компьютера имеется специальный блок - *контроллер прерываний.* Контроллер прерываний обеспечивает прием запросов на прерывание от отдельных устройств (каждое устройство или однотипная группа устройств использует отдельную линию запроса прерывания), оповещение процессора о возникшем запросе на прерывание и формирование для него кода прерывания, в случае готовности процессора обработать прерывание. На основании сформированного кода прерывания, полученного от контроллера прерываний, процессор определяет адрес программы - обработчика прерываний и передает ей управление - начинается процесс обработки прерывания. Последней командой обработчика прерывания является команда возврата из прерывания, обеспечивающая возврат управления в прерванную программу.

Для разрешения конфликтов при обработке прерываний от различных устройств обычно используется приоритетная схема. В этом случае каждой линии прерывания (фактически каждому внешнему устройству или группе устройств) присваивается определенный приоритет. В первую очередь осуществляется обработка запросов на прерывание от наиболее приоритетных устройств. Более того, если при обработке прерывания от какого-либо устройства возникает запрос на прерывание от более приоритетного устройства, то этот запрос удовлетворяется в первую очередь, иными словами, работа менее приоритетного обработчика может быть прервана с целью выполнения действий по обслуживанию более приоритетного устройства. Например, при работе какой-либо программы возникает прерывание от клавиатуры. Для обработки прерывания начинает выполняться обработчик прерывания. В процессе его работы возникает прерывание от таймера, имеющее более высокий приоритет. В этом случае работа обработчика прерывания от клавиатуры прерывается и начинает выполняться обработчик прерывания от таймера. По завершении обработки прерывания по таймеру

возобновляет работу прерванный обработчик клавиатурных событий, по завершению которого возобновляет работу прерванная программа.

Внутренние аппаратные прерывания, в отличие от внешних, генерируются аппаратурой процессора и его окружением и свидетельствуют о нарушении нормального хода вычислительного процесса и возникновении нестандартной ситуации. Подобными ситуациями являются, например, деление на ноль, переполнение, несуществующий код операции, переполнении стека, попытках нарушения защиты памяти и ряд других.

Адреса программ - обработчиков прерываний располагаются в защищенной системой области памяти, называемой областью векторов прерываний. Каждому типу прерывания соответствует свой обработчик прерывания. Сам обработчик прерывания может располагаться либо в ПЗУ (системном или периферийного контроллера), либо в оперативной памяти.

Обработка прерываний может быть запрещена. Запрещение осуществляется на двух уровнях - на уровне контроллера прерываний и на уровне процессора. Запрещение прерываний, осуществляемое на уровне процессора, запрещает обработку прерываний от *всех внешних* устройств, при этом процессор просто игнорирует запрос на обработку прерывания, поступающий от контроллера прерываний и прерывания “теряются”. Запрещение прерываний, выполняемое на уровне контроллера, позволяет запретить обработку прерываний от *отдельных внешних* устройств путем “маскирования” отдельных линий запросов на прерывание. Естественно, что операция запрещения прерываний и маскирования прерываний имеет отношение только к внешним аппаратным прерываниям, и не оказывает никакого влияния на обработку внутренних.

Программные прерывания на самом деле ничего не прерывают. Обращения к ним осуществляется практически так же, как и к подпрограммам. Они обеспечивают возможность работы с системными процедурами, адреса которых могут варьироваться в зависимости от версии системного программного обеспечения и конфигурации оборудования (например, различные типы видеоадаптеров могут иметь различный набор функций и реализующие их подпрограммы размещены по различным адресам). При модификации системного программного обеспечения и изменении конфигурации оборудования в этом случае нет необходимости изменять другие программы, использующие системные запросы, при условии, если не изменилось расположение векторов прерываний. Таким образом, механизм программных прерываний обеспечивает мобильность программного обеспечения, то есть его независимость от аппаратуры и расположения в памяти подпрограмм, реализующих системные функции.

5.5 ПОНЯТИЕ О СОПРОГРАММАХ

Во многих типах процессоров команда вызова подпрограммы допускает сложные виды адресации, например, относительно, косвенную и др. Это дает дополнительные возможности организации подпрограмм, в частности, обеспечивает организацию *сопрограмм*. Сопрограммы - это программы, которые могут поочередно передавать управление друг другу. Название “сопрограммы” подчеркивает факт их кооперации, основанной на равноправии, а не на подчинении, как в подпрограммах. Каким образом осуществляется поочередная передача управления в сопрограммах? Для ответа на этот вопрос рассмотрим следующие фрагменты текстов сопрограмм, реализованных с использованием системы команд модели 2:

Сопрограмма 1	Сопрограмма 2
…	…
MVI A, адрес сопрограммы	
CALL 0 (A)	POP A
…	CALL 0 (a)
POP A	…
CALL 0 (A)	POP A
…	CALL 0 (A)
	…

Таким образом, при передаче управления из одной сопрограммы в другую в вершине стека происходит замещение адреса вызова одной сопрограммы адресом вызова другой, а для передачи управления используются симметричные команды CALL

Сопрограммы оказываются незаменимыми, когда программа разделяется на два основных задания, которые должны быть между собой скоординированы, но при этом их слишком тесная взаимосвязь ведет к излишним усложнениям. Очень часто подобная ситуация наблюдается в системных программах, которые нередко выполняют столь переплетенные функции, что использование сопрограмм является в них наиболее естественным. В машинах, обеспечивающих вызов по косвенному адресу в верхушке стека, организация сопрограмм происходит без явного манипулирования со стеком - адреса вызовов сопрограмм замещаются автоматически, при выполнении команды CALL. Например, для PDP-11 JSR PC, (SP)+, или CALL FAR[BP] для IBM PC.

6. ОРГАНИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

6.1 КЛАССИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Под программным обеспечением (ПО) понимается совокупность программ вычислительной системы, процедур и правил вместе со всей связанной с этими компонентами документацией, позволяющая использовать компьютер для решения различных задач.

Состав и взаимосвязь различных компонентов ПО может быть представлена следующей схемой:

Базовое или общесистемное программное обеспечение представляет собой совокупность программных средств (вместе с соответствующей документацией), обеспечивающих автоматизацию трудоемких технологических этапов разработки

программного обеспечения, а также для организации и контроля вычислительного процесса на компьютере в процессе его функционирования.

Прикладное или специальное ПО – это совокупность программных средств, обеспечивающих решение той или иной задачи, либо круга задач из конкретной предметной области.

6.2 БАЗОВОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

В базовое программное обеспечение включается следующий набор компонентов:

- Операционная система и ее окружение;
- Средства контроля и диагностики;
- Системы программирования.

6.2.1 Операционная система и ее окружение.

Операционная система (ОС), по определению Г. Дейтела, специалиста в области операционных систем - это набор программ, обеспечивающих возможность использования аппаратуры компьютера. При этом аппаратные средства ЭВМ предоставляют «сырую» вычислительную мощность, а задача операционной системы заключается в том, чтобы сделать аппаратуру доступной, и по возможности удобной для пользователя.

Основное назначение операционной системы состоит в эффективном управлении ресурсами системы, основными из которых являются:

- процессор (процессоры);
- память;
- устройства ввода/вывода.

Операционная система реализует множество функций, в том числе:

- обеспечивает взаимодействие между пользователем и вычислительной системой;
- обеспечивает разделение аппаратных средств между пользователями;
- планирует доступ пользователей к общим ресурсам;
- обеспечивает эффективное планирование и выполнение операций ввода/вывода;
- осуществляет восстановление информации и вычислительного процесса в случае сбоев и ошибок.

Иными словами, ОС представляет собой как бы «программную прослойку» между пользователями и аппаратурой ЭВМ.

ОС можно классифицировать по различным признакам, основными из которых являются:

- режим работы (диалоговый или пакетный);
- количество одновременно решаемых задач и обслуживаемых пользователей.

Управление работой диалоговой (интерактивной) ОС осуществляется с помощью команд, вводимых пользователем либо с клавиатуры терминала, либо с помощью других устройств ввода (мышь, тактильный экран, устройства речевого ввода и др.). Примерами диалоговых ОС являются такие системы, как MS DOS, Windows, Mac OS, Unix и его разновидности, операционные системы мобильных систем Palm OS, Symbian OS, Linux DA и др.

Управление работой пакетной ОС осуществляется с помощью специального языка управления заданиями. Задание является внешней единицей работы такой ОС и представляет собой совокупность операторов языка управления заданиями, программы и исходных данных для программы. Совокупность нескольких заданий образует пакет (поток) заданий.

Пакет заданий поступает в систему и помещается во входные очереди. Из входных очередей задания для выполнения выбираются либо последовательно, либо на основе приоритетов. Задание может содержать один или несколько пунктов. При выборке из входной очереди очередного пункта задания и распределении ресурсов, необходимых для его функционирования, образуется задача, являющаяся внутренней единицей работы ОС.

Результаты выполнения задания помещаются в выходные очереди ОС, откуда могут быть введены на периферийные устройства.

Следует заметить, что в этом случае отсутствует непрерывное интерактивное взаимодействие пользователя и системы в процессе решения задачи, то есть конечный пользователь обосабливается от системы и процесса выполнения задачи на компьютере. Классическими примерами подобных операционных систем являются OS/360 и OS/370.

По количеству одновременно выполняемых задач и одновременно обслуживаемых пользователей можно выделить следующие разновидности ОС:

- однопользовательские однозадачные;
- однопользовательские мультизадачные;
- многопользовательские мультизадачные.

В однопользовательской однозадачной операционной среде в каждый момент времени может работать только один пользователь, причем все ресурсы принадлежат только этому пользователю. Средства разделения и защиты ресурсов в такой среде отсутствуют, поэтому в системе в каждый момент времени может работать только одна задача. Примерами таких операционных систем являются CP/M, MSDOS.

В однопользовательской мультизадачной операционной среде имеются средства диспетчеризации задач, механизмы планирования и управления доступом к разделяемым ресурсам, механизмы защиты. В каждый момент времени с такой системой может работать только один пользователь, однако у него есть возможность параллельного выполнения нескольких задач. К подобным системам относятся, например, Windows 98/ME, Windows NT/2000 Workstation, Palm OS, Windows CE.

Многопользовательские мультизадачные операционные системы дополнительно снабжены средствами многотерминальной поддержки, то есть возможностью подключения множества физических или виртуальных терминалов для работы множества пользователей. Таким образом, в такой среде одновременно может работать несколько пользователей, причем каждый пользователь имеет возможность параллельного выполнения нескольких задач. К этому классу относится семейство операционных систем UNIX, операционные системы Windows NT Server, Windows NT 2000 Server, QNX и ряд других.

При реализации мультизадачного режима работы все ресурсы вычислительной системы разделяются между множеством задачам, при этом механизмы диспетчеризации программ обеспечивают одновременное их выполнение (хотя на самом деле это далеко не так в случае однопроцессорной вычислительной системы). В общем случае в такой системе в определенные моменты времени системные ресурсы предоставляются то одной задаче, то другой и т.д. В зависимости от дисциплины обслуживания задач различают три основных подхода организации мультипрограммных систем:

- классическое мультипрограммирование. В случае классического мультипрограммирования та или иная задача сама освобождает ресурсы процессора в случае перехода к выполнению операций ввода/вывода. Освободившиеся ресурсы передаются другой задаче, ожидающей времени процессора. Такой тип мультизадачности также называют корпоративной или невытесняющей (non-preemptive) мультизадачностью. В реальных условиях на механизм переключения задач накладывается механизм выбора очередной задачи на основе системы динамических приоритетов.

Достоинством такой схемы является простота реализации и, как следствие, низкие накладные расходы на реализацию механизма переключения. Основной недостаток схемы заключается в том, что возможна ситуация монополизации времени процессора одной задачей, требующей значительных вычислительных ресурсов. В этом случае остальные задачи, находящиеся в системе, будут простаивать.

- параллельная обработка (квантование времени). Параллельная обработка предполагает, что в системе каждой задаче выделяется одинаковый промежуток времени (временной квант), по истечении которого ресурсы у задачи отбираются и передаются другой задаче. Так же, как и в предыдущем случае, на механизм переключения накладывается схема с динамическими приоритетами. В этом случае ни одна из эадач не в состоянии монополизировать системные ресурсы, однако могут возникать простои процессора в случае, если задача, получающая временной квант, переходит в режим ожидания в связи с выполнением операции ввода/вывода. Подобный тип мультизадачности так же называют вытесняющей (preemptive) мультизадачностью.
- сочетание классического мультипрограммирования с параллельной обработкой обеспечивает, с одной стороны, невозможность монополизации ресурсов какой-либо задачей, с другой стороны, минимизирует простои процессора, так как переключение задач осуществляется либо по истечении кванта времени, либо при переходе задачи в состояние ожидания в связи с выполнением операций обмена данными с периферийными устройствами. Так же, как и в двух предыдущих случаях, на механизм переключения накладывается схема динамических приоритетов.

6.2.2 Средства контроля и диагностики.

Средства контроля и диагностики представляют собой набор служебных программ, обеспечивающих контроль правильности функционирования ВС (аппаратных и программных средств), а также диагностику обнаруженных неисправностей и восстановления системы при сбоях и отказах. Примерами таких программных средств могут служить процедуры начальной проверки системы при включении питания POST (**P**ower **O**n **S**elf **T**est), программы-тесты (AdvDiag, AMIDiag, PC Test), служебные программы-утилиты (Norton Utilities, Check disk, Scandisk и др.).

6.2.3 Системы программирования.

Системы программирования обеспечивают полный цикл разработки, отладки и тестирования программ на различных языках программирования. В общем случае системы программирования могут включать в себя следующий набор компонентов:

1. язык программирования;
2. лингвистический процессор, обеспечивающий преобразование конструкций языка программирования в машинный код;
3. библиотеки стандартных функций;
4. средства компоновки программ из различных модулей (редактор связей, Linker);
5. средства отладки и тестирования программ (отладчики, профилировщики);
6. интегрированную среду разработки (IDE).

В зависимости от изобразительных средств языки программирования традиционно классифицируются следующим образом.

1. Машинные языки (машинные коды) или языки уровня 0.

2. Ассемблеры - языки, в которых коды операций процессора заменены мнемоническими обозначениями, для ссылки на данные можно использовать символические адреса (имена), имеются способы описания распределения памяти. Лингвистические процессоры для таких языков часто также называют ассемблеры. Перевод в машинный язык осуществляется по принципу “один в один” - один оператор языка в одну команду машины. Дальнейшим развитием ассемблеров являются макроассемблеры, в которых допускается применение макрокоманд, представляющих собой команду, разворачивающуюся в несколько инструкций ассемблера. Перевод, или трансляция, в этом случае осуществляется в два этапа – на первом этапе осуществляется макроподстановка – то есть макрокоманды заменяются последовательностью нескольких команд – макрорасширением, а затем осуществляется перевод по принципу “один в один”.

Ассемблеры и макроассемблеры являются машинно-ориентированными языками и ориентированы на конкретную аппаратную платформу и систему.

3. Языки высокого уровня. Языки высокого уровня являются машинно-независимыми. В основе конструкций языка высокого уровня лежит ограниченный естественный или специализированный язык (например, язык математических обозначений и символов). Трансляция осуществляется по принципу “несколько - во множество”, то есть нескольким элементам языковой конструкции соответствует множество машинных команд.

Транслятор с языка высокого уровня часто называют компилятором.

4. Языки сверхвысокого уровня или генераторы программ. В отличие от алгоритмических языков высокого уровня, описывающих процесс задачи в терминах алгоритма (то есть “как делать”), описание задачи на языке сверхвысокого уровня формулирует цель решения задачи (то есть “что делать”). По описанию задачи генератор формирует алгоритм решения и готовую рабочую программу.

По принципу работы лингвистического процессора выделяют трансляторы, интерпретаторы и лингвистические процессоры смешанного типа.

Транслятор обеспечивает однократное преобразование программы на исходном языке программирования в машинные коды. В процессе решения задачи допускается отторжение рабочей программы от средств разработки, то есть от транслятора. Иными словами, для запуска оттранслированной программы транслятор уже не нужен.

Системы интерпретирующего типа работают по-другому. Интерпретатор, в отличие от транслятора, при работе не только переводит фрагмент программы в машинные коды, но и тут же их исполняет. Следовательно, интерпретатор не допускает отторжения программы от средств разработки – разработанная один раз программа каждый раз должна выполняться в среде интерпретатора.

Системы смешанного типа допускают оба типа обработки – часть конструкций языка может транслироваться, а часть – интерпретироваться.

6.3. ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

В группе прикладного ПО можно выделить две базовых подгруппы – оригинальное программное обеспечение и пакеты прикладных программ (ППП).

Оригинальное программное обеспечение создается каждый раз, когда возникает потребность в решении той или иной задачи или узкого круга задач, а готовые программные продукты отсутствуют.

ППП общего назначения представляют собой наборы программ, позволяющих решать различные классы задач в различных предметных областях (СУБД, электронные таблицы, ИПС, графические пакеты, тестовые процессоры и др.).

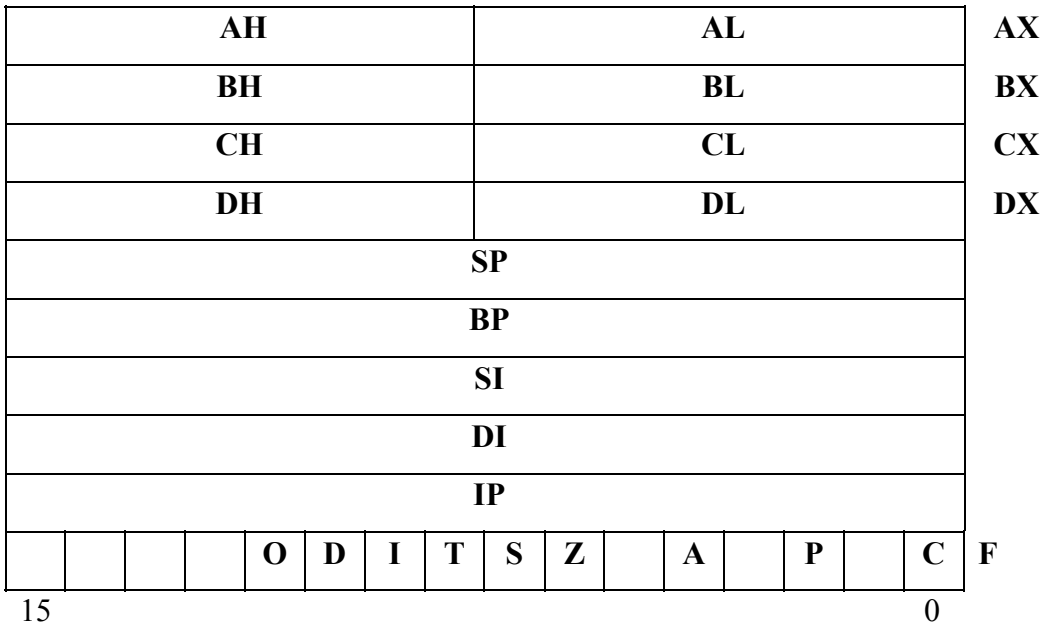
Методо-ориентированные ППП предназначены для реализации определенных методов решения задач, например, для решения задач обработки статистических данных, поиске оптимального решения методами линейного или нелинейного программирования, решении дифференцированных уравнений и др.

Проблемно-ориентированные пакеты используются для решения определенных задач в конкретной предметной области. Это, например, системы автоматизированного проектирования (САПР), предназначенные для решения задач проектирования радиоэлектронных схем и их компонентов, строительных и машиностроительных конструкций, проектирования интерьеров зданий и сооружений.

7. ОРГАНИЗАЦИЯ СИСТЕМ НА БАЗЕ ПРОЦЕССОРОВ INTEL80X86

7.1 РЕГИСТРЫ ПРОЦЕССОРА

Базовая модель микропроцессоров Intel80x86 - микропроцессор I8086/8088 имеет следующий набор регистров:



Все регистры являются шестнадцатиразрядными, однако каждый из регистров AX, BX, CX, DX может использоваться как два восьмиразрядных регистра. Регистры SP, BP, SI, DI могут использоваться только в качестве шестнадцатиразрядных. Регистры общего назначения процессора (первые восемь) могут использоваться в любых операциях с данными, кроме того, каждый из регистров имеет свои, строго определенные функции:

- AX (AL) выполняет роль, аналогичную аккумулятору процессоров I8080, I8085, Z80 - с использованием данного регистра выполняется умножение, деление, обмен с портами, преобразования, операции двоично-десятичной арифметики;
- BX выполняет роль базового регистра при использовании относительной и индексно-относительной адресации;
- CX (CL) выполняет роль счетчика в командах организации цикла и префиксах, используется при реализации сдвигов и вращений;
- DX (DL) используется в качестве расширителя регистра AX в операциях умножения и деления, а также при обмене данными с портами ввода/вывода в качестве регистра косвенной адресации;

- SP является шестнадцатиразрядным регистром указателя стека;
- BP используется в качестве базового при адресации относительно вершины стека, позволяя адресовать данные в окне стека (STACK FRAME);
- SI, DI используются в качестве индексных регистров и в строковых операциях.

Шестнадцатиразрядный регистр указателя инструкции IP используется при выборке команд из оперативной памяти и является программно-недоступным, т.е. программист не может его изменить непосредственно, с помощью каких-либо команд.

Регистр флагов F также является шестнадцатиразрядным, однако для хранения флагов, отражающих текущее состояние процессора и вычислительного процесса, используются только девять разрядов. Значения разрядов следующие:

* **O** - флаг переполнения; сигнализирует о потере старшего бита результата сложения или вычитания в связи с переполнением разрядной сетки при работе со знаковыми числами;

* **D** - флаг направления, использующийся в строковых операциях; при D=0 адреса последовательно увеличиваются, при D=1 уменьшаются;

* **I** - флаг разрешения прерываний; при I=1 внешние прерывания разрешены, при I=0 внешние прерывания запрещены;

* **T** - флаг трассировки; при установленном флаге (T=1) процессор переходит в пошаговый режим работы, при этом, после выполнения каждой команды генерируется внутреннее прерывание 1; используется отладчиками;

* **S** - флаг знака; устанавливается в единицу при отрицательном результате операции полученном путем выполнения арифметико-логических команд;

* **Z** - флаг нуля; устанавливается при нулевом результате;

* **A** - флаг вспомогательного переноса; устанавливается при наличии переноса из младшей тетрады в старшую; используется только командами десятичной арифметики;

* **P** - флаг четности; устанавливается в случае четного числа единиц результата;

* **C** - флаг переноса; фиксирует значение переноса (заема) при сложении (вычитании), а также значение выдвигаемого бита при сдвигах;

Кроме регистров общего назначения, указателя инструкции и регистра флагов в составе процессора имеется четыре сегментных регистра:

CS	сегментный регистр кода
DS	сегментный регистр данных
ES	сегментный регистр расширения
SS	сегментный регистр стека

15 0

Наличие группы сегментных регистров объясняется применением в микропроцессоре сегментации памяти, которая предполагает выделение в памяти отдельных блоков для хранения кода программ, данных и стека. Адрес начала такого блока или сегмента хранится в соответствующем сегментном регистре. Поскольку сегментные регистры так же, как и регистры общего назначения, являются шестнадцатиразрядными, то с их помощью возможна адресация 65535 различных сегментов, располагающихся на границе 16 байт. Последовательный блок из шестнадцати байт оперативной памяти в IBM PC носит название параграфа. Размер каждого сегмента фиксированный (для процессоров I8086/88 и для реального режима 80286/80386 и выше) и составляет 65535 байт или 64 Кбайт. В общем случае сегменты могут частично или полностью перекрываться. Дополнительный сегмент и соответствующий сегментный регистр ES используются при работе с большими блоками данных, когда основного сегмента данных недостаточно. Сегментные регистры являются программно-доступными, т.е. программа в ходе функционирования может изменять их значение, получая доступ в различные области физической памяти. Хотя адресные регистры процессора являются 16-ти разрядными, процессор способен адресовать 1M памяти с помощью пары шестнадцатиразрядных величин "сегмент:смещение". Вычисление полного адреса в этом случае осуществляется следующим образом:

* величина "сегмент" во внутреннем регистре шинного интерфейса процессора сдвигается влево на четыре двоичных разряда (что эквивалентно умножению этой величины на 16), при этом освободившиеся младших четыре двоичных разряда заполняются нулями; получается двадцатиразрядный физический адрес памяти, указывающий параграф, с которого начинается сегмент, к которому осуществляется доступ;
* к получившемуся двадцатиразрядному адресу добавляется величина "смещение"; в итоге получается полный двадцатиразрядный адрес байта или слова памяти.

Например, пусть мы имеем величину "сегмент: смещение" в виде $40{:}17_{16}$. Вычислим для нее физический адрес памяти. Сдвигаем величину 40_{16} на четыре двоичных разряда влево, освободившиеся разряды заполняем нулями, получаем 400_{16} - адрес параграфа памяти, с которого начинается сегмент с адресом 40_{16}. К получившемуся адресу добавляем 17_{16} - внутрисегментное смещение. В итоге получаем полный физический адрес памяти - 00417_{16}, по которому размещается требуемый байт. Следует заметить, что преобразование пары "сегмент: смещение" в физический адрес осуществляется однозначно, обратное же преобразование - нет. В частности, физическому адресу 00417_{16} могут соответствовать, например, такие пары: $40{:}17_{16}$, $20{:}217_{16}$, $30{:}117_{16}$, $0{:}417_{16}$ и др.

7.2 РАСПРЕДЕЛЕНИЕ АДРЕСНОГО ПРОСТРАНСТВА IBM/PC/XT/AT

Распределение памяти компьютера IBM в случае работы под управлением операционной системы MS DOS выглядит следующим образом:

Пояснение	тип памяти	адреса
существует только для процессоров 80286 и выше, недоступна в реальном режиме	расширенная память (extended memory)	10000 …
Объем 384 K. Содержит ПЗУ BIOS, ПЗУ адаптеров периферийных устройств, видеопамять, возможно, окно (фрейм) отображаемой (expanded) памяти, UMB	область системных ПЗУ, ПЗУ адаптеров, видеопамять и фрейм отображаемой памяти	A0000 … FFFFF
Основная оперативная память системы. Объем 640K. Доступна для всех типов систем (I8086/88, 80286 и выше)	основная память (conventional memory)	0 … 9FFFF

Основная оперативная память (conventional memory) обычно имеет объем 640 K и доступна в системах на базе любых процессоров от Intel8086 и выше. Эта память доступна всегда, вне зависимости от режима работы процессора и установленного менеджера памяти. При более детальном рассмотрении в этой памяти можно выделить несколько достаточно фиксированных областей (при работе в среде MS DOS):

- область векторов прерываний объемом 1K расположена в диапазоне адресов 00000 ... 003FF. Содержит адреса обработчиков прерываний. Адрес каждого обработчика представляется парой "сегмент:смещение" и занимает 4 байта. Таким образом, область векторов прерываний содержит 256 векторов прерываний;

- служебная область BIOS объемом 256 байт располагается с сегментного адреса 40 или с физического адреса 00400;

- служебная область DOS располагается с сегментного адреса 0050;

- ядро системы, содержащееся в файлах IBMBIO.COM (IO.SYS) и IBMDOS.COM (MSDOS.SYS) (размер зависит от версии системы. В случае, если произведена загрузка DOS в верхние адреса, эта область располагается в другом месте);

- область загружаемых драйверов устройств и системных буферов (также частично может быть перенесена в верхние адреса в системах с процессором 80286 и выше);

- резидентная часть командного процессора COMMAND.COM;

- область пользователя (размер зависит от версии и конфигурации операционной системы);

- транзитная часть командного процессора COMMAND.COM.

Область адресов памяти свыше 640 K имеет объем 384 K. В этой области также можно выделить несколько подобластей:

- область системного ПЗУ BIOS; для PC/XT объем ПЗУ BIOS составляет 8 K. BIOS располагается в диапазоне адресов FE000 ... FFFFF; кроме того, у машин данного класса в диапазоне адресов F6000...FDFFF может располагаться кассетный бэйсик, занимая при этом объем 32 K. Для машин класса PC/AT обьем ПЗУ BIOS может достигать 64K и 128K, занимая при этом диапазон адресов F0000...FFFFF;

-область видеопамяти; в зависимости от дисплейного адаптера может располагаться в различных участках памяти. Для дисплейного адаптера MDA (Monocrome Display Adapter) видеопамять располагается с адреса B0000 и имеет объем 4K. Область видеопамяти цветного графического адаптера CGA (Color Graphics Adapter) занимает диапазон адресов B8000 ... BBFFF (16K). Для улучшенного графического адаптера EGA (Enhanced Graphics Adapter) и адаптера VGA (Video Graphics Array) под видеопамять занимаются две области - для текстового режима - B8000...BDFFF (или B0000...B03FF в случае работы в монохромном текстовом режиме), для графического режима - A0000...AFFFF (64 K). Следует заметить, что в общем случае видеопамять адаптеров EGA и VGA может превышать объем 64 K, однако, поскольку эта память является страничной, то в общем адресном пространстве компьютера она все равно доступна через окно в 64 K;

- ПЗУ дополнительных адаптеров периферийных устройств в общем случае могут занимать несколько областей, например, ПЗУ BIOS EGA и VGA адаптеров обычно занимает адресное пространство в диапазоне C0000 ... C3FFF (EGA) или C0000 ... C7FFF (VGA); ПЗУ BIOS жесткого диска может занимать адресное пространство C8000 ... CBFFF или D8000 ... DBFFF; ПЗУ сетевых адаптеров располагается в диапазоне адресов DC000 ... DFFFF.

- окно отображаемой памяти (Expanded Memory) обычно располагается в диапазоне адресов E0000 ... EFFFF, занимая область размером в 64 K (четыре страницы по 16Кбайт каждая). Отображаемая память имеет страничную организацию и позволяет через окно размером в 64K путем переключения страниц получить доступ к полю памяти, превосходящему размером 1M - т.е. то адресное пространство, которое способен адресовать процессор I8086/88. Изначально спецификация отображаемой памяти была разработана для компьютеров на базе процессоров I8086/88 для расширения объема доступной оперативной памяти, однако в настоящий момент многие машины на базе процессоров I80286/386/486/Pentium поддерживают эту спецификацию для совместимости с разработанным ранее программным обеспечением;

- блоки верхней памяти UMB (Upper Memory Block) организуются на машинах с процессором 80286 и выше путем перемещения части Extended памяти в диапазон адресов до 1 Мбайта для увеличения объема памяти, доступной в реальном режиме. Эта память может быть использована для системных буферов, загружаемых драйверов устройств и резидентных программ.

Расширенная память (Extended Memory) присутствует только в системах на базе процессора I80286 и выше. Обьем расширенной памяти для процессора 80286 может достигать 15 Мбайт, для процессоров 80386 и выше - 4 Гбайт. Память доступна только в защищенном режиме работы процессора за небольшим исключением 64 Кбайт с сегментного адреса FFFF. Этот блок памяти получил название "Область высших адресов" (High Memory Area или HMA). Эта область обычно занимается ядром операционной системы.

7.3 СИСТЕМА ПРЕРЫВАНИЙ

Внутренние аппаратные прерывания (исключения) вызываются аппаратурой процессора, имеют номера в диапазоне 0 ... 31 и обеспечивают реакцию на следующие ситуации:

0 - деление на ноль; вызываются инструкциями деления DIV и IDIV в случае нулевого делителя;

1 - пошаговый режим; прерывание возникает после выполнения каждой команды при установленном флаге трассировки T;

2 - немаскируемое прерывание; генерируется процессором при поступлении сигнала на вход NMI;

3 - прерывание по точке останова; генерируется в случае, если код очередной команды - CC; используется отладчиками;

4 - переполнение; генерируется в случае, если установлен флаг переполнения O и выполняется инструкция INTO;

5 - печать экрана (Print Screen); вообще говоря, является внешним, однако традиционно имеет фиксированное назначение и располагается в области внутренних прерываний;

6 (80286 и выше) - неверный код операции;

7 (80286 и выше) - математический сопроцессор недоступен; вызывается при попытке выполнить команду сопроцессора при его отсутствии; может использоваться для эмуляции сопроцессора;

8 (80286 и выше) - обнаружена двойная исключительная ситуация; вызывается в случае возникновения более чем одной исключительной ситуации при выполнении очередной команды; в защищенном режиме вызывается при попытке обратиться за установленный предел таблицы векторов прерываний;

9 (80286 и выше) - попытка обращения к памяти за пределами границы сегмента;

A (80286 и выше) - неверный сегмент состояния задачи;

B (80286 и выше) - сегмент не найден; возникает при отсутствии требуемого сегмента в памяти в защищенном режиме;

C (80286 и выше) - стек переполнен или неверное значение указателя стека;

D (80286 и выше) - общее нарушение защиты памяти;

E (виртуальный режим 80386 и выше) - используется для реализации виртуальной памяти;

10 (80286 и выше) - ошибка сопроцессора.

Внешние аппаратные прерывания вызываются внешними по отношению к процессору устройствами. Для управления аппаратными прерываниями во всех компьютерах IBM PC используется программируемый контроллер прерываний, обеспечивающий обработку прерываний с учетом их приоритетов. IBM PC/XT имеет 8 уровней прерываний, IBM PC/AT - 16, так как использует два контроллера прерываний, включенных каскадно. Запросы на аппаратные прерывания 0 - 7 соответствуют векторам прерываний от 8 до 0F, для IBM PC/AT запросам на аппаратные прерывания соответствуют вектора прерываний 70 ... 77_{16}.

Приоритет аппаратных прерываний следующий:

0 - таймер;

1 - клавиатура;

2 - прерывания от платы расширения (PC,PC/XT);

управление вторым контроллером прерываний(AT);

8 - часы реального времени;

9 - программно переводится в 2;

10 - резерв;

11 - резерв;

12 – PS/2 Mouse;

13 - математический сопроцессор;

14 - контроллер жесткого диска EIDE/ATA I канал;

15 – контроллер жесткого диска EIDE/ATA II канал;

3 - прерывания от COM1 (COM2 для AT);

4 - прерывания от COM2 (COM1 для AT);

5 - жесткий диск (PC,PC/XT), принтер 2(LPT2)(AT) или звуковая карта (AT);

6 - контроллер гибких дисков;

7 - принтер (LPT1).

Высший приоритет имеет аппаратное прерывание 0 - прерывание от таймера, низший - 7 - прерывание от принтера.

Программные прерывания.

Программные прерывания вызываются командой процессора INT n, где n - номер программного прерывания. Программные прерывания можно условно разбить на четыре группы:

* прерывания BIOS;
* прерывания DOS;
* программные прерывания пользователя;
* программные прерывания интерпретатора BASIC;

Прерывания BIOS устанавливаются при включении машины либо при полной перезагрузке (COLD BOOT) инициализирующей процедурой BIOS(сокращенный набор прерываний BIOS приведен в приложении 4). Прерывания DOS устанавливаются при загрузке ядра операционной системы и загружаемых драйверов устройств. Прерывания интерпретатора BASIC используются кассетным БЕЙСИКом и его дисковыми версиями при их функционировании. Прерывания пользователя могут использоваться программами по усмотрению программиста. В принципе, любое прерывание, как программное, так и аппаратное, может быть дополнено, модифицировано или подменено путем замены в таблице векторов прерываний соответствующего вектора и написания собственной процедуры обработки прерывания, на которую этот вектор указывает.

7.4 ОРГАНИЗАЦИЯ ВВОДА/ВЫВОДА

Организация обмена с периферийными устройствами в IBM PC/XT/AT базируется на использовании портов ввода/вывода - специальных регистров, обеспечивающих управление периферийными устройствами и обмен данными, и имеющих адреса на общей шине. Адресное пространство портов ввода/вывода составляет 64 Кбайта, однако реально используется значительно более узкий диапазон адресов. Каждое устройство в большинстве случаев представлено несколькими портами, занимающими смежные адреса адресного пространства портов ввода/вывода, при этом порт с самым младшим адресом принято называть базовым. Использование адресного пространства портов достаточно стандартизовано, т.е. в большинстве случаев периферийные устройства имеют стандартные адреса портов, однако, следует иметь в виду, что в ряде случаев, с целью устранения конфликтов с уже установленным оборудованием, адаптер периферийного устройства может быть переконфигурирован для работы в другой области адресуемого пространства портов. Типичная карта портов ввода/вывода для IBM PC/XT/AT может выглядеть, например, следующим образом:

Устройства	Адреса портов
контроллер прямого доступа к памяти 1	0 - F
контроллер прямого доступа к памяти 2 (AT)	C0 - DF
контроллер прерываний	20 - 21
контроллер прерываний 2	A0 - A1
спикер	61 - 61
контроллер клавиатуры	60 или 64
таймер	40 - 43
регистры страниц ПДП	80 - 8F
регистр маски NMI	A0 (XT) 7F(AT)
Порт игрового адаптера	201 - 201
Звуковая карта	220 - 22F, 388-38B, 330 - 331
Адаптер локальной сети	240 -24F
Устройства	Адреса портов
COM1	3F8 - 3FF
COM2	278 - 27F
контроллер принтера	378 - 37F

видеоадаптер MDA(EGA,VGA,SVGA)	3B0 - 3BB
контроллер принтера 2	3BC - 3BF
видеоадаптер CGA(EGA,VGA,SVGA)	3D0 - 3DF
видеоадаптер EGA(VGA,SVGA)	3C0 - 3DF
видеоадаптер VGA(SVGA)	3C0 - 3DF
контроллер гибких дисков	3F0 - 3F7
контроллер жесткого диска	320 - 32F(XT), 1F0 - 1F7(AT) 170 - 177(AT)
CMOS и часы реального времени(AT)	70 – 71

7.5 СИСТЕМА АДРЕСАЦИИ РЕАЛЬНОГО РЕЖИМА

Для процессоров I8086/88 и реального режима процессоров I80286/386 и выше допустимыми являются следующие виды адресации:

* регистровая адресация;
* непосредственная адресация;
* прямая адресация;
* косвенная регистровая адресация;
* базовая адресация;
* индексная адресация;
* базовая индексная адресация;
* относительная адресация;
* неявная адресация.

Регистровая адресация предполагает, что операнд находится в одном из регистров общего назначения, причем регистр может быть как восьми -, так и шестнадцатиразрядный. Специфичных особенностей не имеется.

Непосредственная адресация предполагает задание в поле операнда команды восьми- или шестнадцатиразрядного непосредственного операнда. В случае двухоперандной команды обязательно соответствие длин операндов. Частным случаем данного вида адресации является длинная прямая адресация, при которой в поле операнда команды содержится четыре байта, указывающие абсолютный адрес памяти в формате

"сегмент: смещение", однако подобная форма адресации допустима только в командах межсегментных (далеких) переходов и вызовах подпрограмм. Другая форма прямой адресации допускается в командах обмена с портами, когда в качестве операнда указывается абсолютный адрес порта.

При косвенно-регистровой адресации адрес операнда может располагаться в одном из шестнадцатиразрядных регистров общего назначения, т.е. регистрах BX, BP, SI, DI. Разновидностью данного способа адресации является косвенная адресация портов ввода/вывода через регистр DX.

Базовая адресация предполагает, что в качестве операнда в команде указывается имя базового регистра и смещение. Исполнительный адрес в этом случае указывается как сумма базового регистра и смещения. В качестве базового регистра могут использоваться регистры BX и BP. Причем, если в качестве базового регистра указывается BX, выборка операнда осуществляется из сегмента данных, адресуемого сегментным регистром DS, в случае указания регистра BP выборка операнда осуществляется из стека.

Индексная адресация предполагает, что значение адреса вычисляется как сумма смещения и содержимого индексного регистра SI или DI, указанных в команде. Базовая индексная адресация является комбинацией чисто индексной и базовой адресации, адрес вычисляется соответствующим образом.

Относительная адресация реализуется только по отношению к регистру IP и предполагает, что адрес вычисляется как сумма содержимого регистра IP и смещения. Данная форма адресации используется в командах организации переходов, циклов и вызова подпрограмм, обеспечивая независимость соответствующих команд от места размещения в памяти.

Неявная форма адресации используется в командах умножения, деления, преобразования и перекодировки, в которых в качестве операнда всегда предполагается регистр AX (или пара DX:AX), однако в качестве операнда он не указывается. Кроме того, подобная форма адресации используется в строковых операциях, в которых для адресации применяются регистры SI и DI, также не указываемые явно в команде.

7.6 БАЗОВАЯ СИСТЕМА КОМАНД ПРОЦЕССОРА

Система команд процессоров I80x86 построена таким образом, что набор инструкций процессора I8086 является базовым для всех последующих моделей. Следовательно, программа, написанная с использованием команд процессора I8086/88 может функционировать на всех последующих моделях. Ниже кратко рассмотрен базовый

состав команд процессоров с использованием мнемонических кодов ассемблера для обозначения операций. В приведенной сводке команд использованы следующие обозначения:

dest - операнд-приемник; первый операнд команды и местонахождение результата;

src - операнд-источник; второй операнд команды;

src16 - шестнадцатиразрядный операнд-источник;

src8 - восьмиразрядный операнд-источник;

count - счетчик; может задаваться либо непосредственным операндом, либо с помощью регистра CX(CL);

reg16 - шестнадцатиразрядный регистр;

mem16 - шестнадцатиразрядное слово памяти;

acc - аккумулятор - регистр AX(AL);

port - адрес порта ввода/вывода, задаваемый непосредственно;

addr - адрес памяти;

far_addr - "дальний адрес" - адрес в формате "сегмент:смещение";

near_addr- "близкий адрес" - адрес в формате "смещение";

short_label - "короткая метка" - метка(адрес) отстоящая от текущего адреса не более чем на +/- 127 байт, фактически всегда преобразуется в однобайтовое знаковое смещение;

7.6.1. Арифметические и логические инструкции

ADD dest,src	Сложить два операнда: dest+src->dest
ADC dest,src	Сложить два операнда с учетом переноса src+dest+cy->dest
INC dest	Увеличить содержимое операнда на 1 dest+1 -> dest
SUB dest,src	Вычесть из dest src : dest-src -> dest
SBB dest,src	Вычесть с учетом заема: dest-src-cy -> dest
DEC dest	Уменьшить на единицу содержимое dest: dest-1 -> dest
NEG dest	Изменить знак операнда: 0-dest -> dest
MUL src	беззнаковое умножение: 1.Операнды восьмиразрядные (AL)*src8 -> AX 2.Операнды шестнадцатиразрядные: (AX)*src16->

	DX:AX
IMUL src	Умножение чисел со знаком (Аналогично выше)
DIV src	Деление беззнаковое: 1.Восьмиразрядные операнды AX/src8 -> AL; AX mod src8 -> AH; 2. Шестнадцатиразрядный операнд: DX:AX / src16 -> AX; DX:AX mod src16 -> DX.
IDIV src	Деление чисел со знаком - аналогично беззнаковому
AAA	Надстройка для сложения чисел в коде ASCII. Корректирует сумму двух байтов в регистре AL. Если правые четыре бита в регистре AL имеют значение больше 9 или флаг AF=1, то команда AAA прибавляет к регистру AH единицу и устанавливает флаги AF и CY. Команда всегда очищает четыре левых бита регистра AL.
AAS	Надстройка для вычитания ASCII-чисел. Применяется после вычитания для преобразования AL в две цифры кода ASCII.
AAM	Надстройка для умножения ASCII- чисел. Делит содержимое регистра AL на 10, частное помещается в регистр AH, остаток - в регистр AL. Применяется после умножения двух чисел в коде ASCII.
AAD	Надстройка для деления ASCII . Применяется перед делением чисел в коде ASCII. Команда корректирует делимое в двоичное значение в регистре AL для последующего двоичного деления. Затем умножает содержимое регистра AH на 10, прибавляет результат к содержимому регистра AL и очищает AH.
DAA	Десятичная надстройка для сложения. Корректирует результат операции сложения двух упакованных десятичных чисел с целью получения упакованного десятичного числа.

DAS	Десятичная надстройка для вычитания. Корректирует результат вычитания двух упакованных десятичных чисел с целью получения десятичного числа.
CBW	Преобразовать БАЙТ в СЛОВО. Исходный операнд в AL, результат в AX
CWD	Преобразовать СЛОВО в ДВОЙНОЕ СЛОВО. Исходное слово в AX, результат в DX:AX
AND dest,src	Логическое И: src & dest -> dest
OR dest,src	Логическое ИЛИ: src OR dest -> dest
XOR dest,src	Исключающее ИЛИ: dest ^ src -> dest
NOT dest	Логическое НЕ: dest -> ^dest
RCL dest,count	Содержимое dest циклически сдвигается влево через флаг переноса на count двоичных разрядов. В качестве count может использоваться содержимое регистра CL, либо непосредственный операнд 1.
RCR dest,count	То же, но сдвиг вправо.
ROL dest,count	Циклический сдвиг влево
ROR dest,count	Циклический сдвиг вправо
SAL/SHL dest,count	Арифметический сдвиг влево (сдвиг на один разряд эквивалентен умножению dest на 2).
SAR dest,count	Арифметический сдвиг вправо (сдвиг на один разряд эквивалентен делению на 2)
SHR dest,count	Логический сдвиг вправо на count разрядов. Освобождающиеся разряды заполняются нулями.

7.6.2. Инструкции пересылки данных

MOV dest,src	Пересылка данных: src -> dest
XCHG dest,src	Обмен содержимым операндов: dest <-> src
LEA reg16,addr	Загрузка адреса : addr -> reg16
LDS reg16,mem	Загрузка сегментного регистра DS и reg16 содержимым памяти, при этом reg16 < [mem16];

	DS < [mem16+2]
LES reg16,mem	Загрузка сегментного регистра ES и reg16 содержимым памяти, при этом reg16 < [mem16]; ES < [mem16+2]
PUSH src	Запись операнда в стек
PUSHF	Запись регистра флагов в стек
POP src	Извлечение операнда из верхушки стека
POPF	Извлечение регистра флагов из верхушки стека
LAHF	Загрузка младшего байта регистра флагов в регистр AH. После выполнения операции регистр AH содержит: значения флагов S Z * A * P * C
SAHF	Загрузка младшего байта регистра флагов из регистра AH
MOVS MOVSB MOVSW	Пересылает данные между областями памяти размером до 64 K. Команда MOVS должна иметь операнды, которые используются ассемблером для определения типа пересылки, команды MOVSB и MOVSW используются для пересылки байтов и слов соответственно, и явных операндов не имеют. Пересылка всегда идет из области, адресуемой парой DS:SI в область, адресуемую парой ES:DI. После выполнения пересылки очередного элемента SI и DI увеличиваются (DF=0) или уменьшаются на единицу (для байтов) или на 2 (для слов). Операция обычно применяется с префиксами повторений.
LODS LODSB LODSW	Помещает в аккумулятор (AL или AX) значение операнда, адресуемого парой DS:SI, за тем, в зависимости от флага DF, адрес операнда инкрементируется или декрементируется
STOS STOSB STOSW	Операция, обратная LODS
IN acc,port (или DX)	Чтение содержимого порта, заданного адресом port или содержимым регистра DX, в аккумулятор

	(AL или AX)
OUT acc,port (или DX)	Запись в порт, заданный адресом port или содержимым регистра DX содержимого аккумулятора (AL или AX)
XLAT src	Транслитерация. Инструкция использует AL как смещение байта в 256-байтовой таблице, адресуемой парой DX:BX. Указанный байт замещает собой значение AL.

7.6.3. Инструкции переходов

JMP addr	Безусловный переход по адресу addr. Может быть далеким (межсегментным), близким (в пределах сегмента) и коротким
JCXZ short_label	Переход, если CX=0 (только короткий).
LOOP short_label	Организация цикла со счетчиком: (CX-1) -> CX , jmp short_label, если CX<>0
LOOPE short_label LOOPZ short_label	То же, что и LOOP, но переход в том случае, если CX<>0 и ZF=1
LOOPNE short_label LOOPNZ short_label	То же, что и LOOP, но переход, если CX<>0 и ZF=0

Условные переходы.

Все условные переходы - короткие (в пределах 127/-128 байт)

Мнемоника	Условие перехода	Флаги
JA/JNBE	Более/Не менее и не равно	CF or ZF =0
JAE/JNB	Более или равно/Не менее	CF=0
JB/JC	Менее/Перенос	CF=1
JE/JZ	Равно/Нуль	ZF=1

JG/JNLE	Больше/Не меньше и не равно 0	SF=0 or SF<>OF
JGE/JNL	Больше или равно/Не меньше	SF=0 or ZF=1
JL/JNGE	Меньше/Не больше и не равно	SF=1,ZF<>1
JLE/JNG	Меньше или равно/Не больше	ZF=0 or SF=1
JNC	Нет переноса	CF=0
JNE/JNZ	Не равно/Не нуль	ZF=0
JNO	Нет переполнения	OF=0
JNP/JPO	Нет четности	PF=0
JNS	Нет знака	SF=0
JO	Переполнение	OF=1
JP/JPE	Четность	PF=1
JS	Знак	SF=1

Инструкции работы с подпрограммами и прерываниями

INT type	Переход к прерыванию типа type
INTO	Прерывание по переполнению. Вызывает прерывание по вектору 4 в случае, если установлен флаг переполнения
IRET	Возврат из прерывания
CALL addr	Переход к подпрограмме
RET	Возврат из подпрограммы
RETF	Возврат из far процедуры
RET n	Возврат с удалением n элементов из верхушки стека

7.6.4. Управление состоянием процессора

CMP dest,src	Сравнить dest и src и установить регистр флагов, операнды не изменяются
TEST dest,src	dest&src и установить регистр флагов, операнды не изменяются
SCAS	Сканировать строку. Сравнивается содержимое

SCASB SCASW	аккумулятора и элемент, указываемый текущим значением пары ES:DI, соответствующим образом устанавливается регистр флагов. После чего значение DI увеличивается для выборки следующего элемента.
CLC	Очистить флаг переноса
STC	Установить флаг переноса
CLI	Запретить прерывания
STI	Разрешить прерывания
CMC	Инвертировать флаг переноса (CY)
CLD	Очистить флаг направления (установить его в 0 или UP) (для строковых операций)
STD	Установить флаг направления (установить его в 1 или DN)(для строковых операций)
HLT	Остановить процессор
WAIT	Перевести процессор в состояние ожидания внешнего прерывания или сброса
LOCK	Блокировка шины на время выполнения следующей команды
SEG:	Установить сегмент для последующей адресации (префикс)

7.6.5. Операнды инструкций ассемблера

В зависимости от используемых методов адресации в качестве операндов инструкций ассемблера могут использоваться различные выражения, примеры записи которых приведены ниже.

режим адресации	формат операнда	сегм. Регистр	примеры
регистровая	8 или 16-разрядный регистр	нет	add al,bl mov ax,cs

			sub bx,dx

непосредственная	8 или 16-разрядный операнд	нет	mov al,10
			mov ax,235h
			mov cl,9fh

Прямая	смещение	DS	mov ax,[100h] mov bx,alfa
	метка	CS	jmp Label call subroute

косвенно-регистровая	[BX]
	[BP]
	[SI]
	[DI]
		DS
		SS
		DS
		DS
			sub ax,[bx]
			add cx,[bp]
			and ax,[si]
			mul [di]

базовая адресация	смещение[BX]
	[смещение+BX]
	[BX]+смещение
	смещение[BP]
	[смещение+BP]
	[BP]+смещение
		DS
		DS
		DS
		SS
		SS
		SS
			mov ax,tabl[bx]
			mov ax,[tabl+bx]
			mov ax,[bx]+tabl
			mov ax,10[bp]
			mov ax,[10+bp]
			mov ax,[bp]+10

индексная адресация	адрес[DI] адрес[SI]	DS DS	or al,100[DI] and dl,mass[SI]

базово-индексная адресация	смещение[BX][SI]
	смещение[BX][DI]
	смещение[BP][SI]
	смещение[BP][DI]
		DS
		DS
		SS
		SS
			mov ax,[BX][SI]+10
			add dx,matr[bx][di]
			or ax,2[BP][SI]
			or ax,[BP+SI+4]

7.7 ПРИМЕР ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА

Ниже приведен исходный текст простой программы, обеспечивающей определение конфигурации оборудования компьютера. Здесь же приведен пример использования директив описания сегментов (**SEGMENT, ASSUME**) и описания данных.

```
Prg     segment para public 'code'
        assume cs:prg,ds:prg,es:prg,ss:prg
        org 100h
```

```
beg:    jmp start
messsopr  db   0ah,0dh,'сопроцессор присутствует$'
messmem   db   0ah,0dh,'объем оперативной памяти:$'
messdisk  db   0ah,0dh,'накопителей на магнитных дисках  $'
messser   db   0ah,0dh,'последовательных портов $'
messgame  db   0ah,0dh,'установлен игровой адаптер $'
messparr  db   0ah,0dh,'параллельных портов  $'
messvideo db   0ah,0dh,'начальный видеорежим  $'
messvidac db   0dh,0ah,'текущий видеорежим $'
reg01     db   'цветной 40x25 $'
reg10     db   'цветной 80x25 $'
reg11     db   'монохром$'
memkb     db   '       ','k$'
tmp       dw   ?
am        dw   offset reg01,offset reg10,offset reg11
include   bin2str.asm
include   hex_out.asm
start:    int  11h     ; получить конфигурацию оборудования
          test ax,2    ; есть сопроцессор ?
          jz   floppy  ; нет сопра, анализируем флоппы
; присутствует сопроцессор, выводим сообщение
          push ax      ; сохранить байты конфигурации оборудования
          mov ah,9     ; функция DOS вывод строки на экран
          lea dx,messsopr   ; загрузить адрес сообщения
          int 21h      ;  вывести строку на экран
          pop ax       ;  восстановить байты конфигурации
floppy:   test ax,1    ; есть флоппы ?
          jz   memory   ; нет, идем на память
; есть флоппы, анализируем сколько
          push ax      ; сохранить байты конфигурации оборудования
          mov  cl,6    ; установить счетчик сдвигов
          shr  ax,cl   ; сдиг "ненужных" разрядов
          and  ax,3    ; выделить биты количества дисков
          add  ax,31h  ; привести число в символьную форму
          push ax      ; сохранить количество флоппов в стеке
          mov  ah,9    ; функция 9 DOS - вывод сообщения на экран
```

```
        lea  dx,messdisk  ; в DX - адрес сообщения
        int  21h     ; вывести строку на экран
        pop  dx      ; восстановить количество флоппов в DL
        mov  ah,6    ; функция 6 DOS - вывод символа на экран
        int  21h     ; вывод символа в текущей позиции курсора
        pop  ax      ; восстановить в AX слово конфигурации
memory:   push ax      ; сохранить слово конфигурации
        int  12h     ; запрос количества основной памяти
        lea  bx,memkb  ; адрес стр. передается п/п преобразования
        call bin2str   ; преобразовать число в симв. форму
        mov  ah,9      ; функция вывода сообщения на экран
        lea  dx,messmem  ; выдача сообщения
        int  21h         ; "количество памяти"
        mov  ah,9        ;
        lea  dx,memkb+1  ; адрес начала симв. предст. числа
        int  21h         ; вывести объем на экран
        pop  ax          ; восстановить байты конфигурации
; videosystem
        push ax          ; сохранить байты конфигурации
        mov  cl,4        ; установить счетчик сдвигов
        shr  ax,cl       ; сдвинуть лишние биты
        and  ax,3        ; выделить код начального видеорежима
        dec ax
        push ax          ; сохранить приведенный код видеорежима
        mov  ah,9        ; задать номер функции
        lea  dx,messvideo ; загрузить в DX адрес сообщения
        int  21h         ; вывести сообщение на экран
        pop  bx          ; восстановить код видеорежима
        shl  bx,1        ; построить в BX индекс для доступа к адресу
        mov  dx, word ptr am[bx] ; в dx адрес строки нужного режима
        mov  ah,9        ; вывести
        int  21h         ;     наименование видеорежима на экран
        pop ax           ; восстановить байты конфигурации
; текущий видеорежим
        push ax          ; сохранить байты конфигурации
        mov  ah,0fh      ; функция 15 BIOS - запросить видеорежим
```

```
        int  10h        ; выполнить прерывание 10h - видеосервис
        push ax         ; сохранить в стеке код видеорежима
        mov ah,9        ; выдать
        lea dx,messvidac  ;  строку сообщения
        int 21h           ;    на экран
        pop ax            ; восстановить код видеорежима
        call hex_out      ; вывести код видеорежима на экран
        pop ax          ; восстановить байты конфигурации
; последовательные порты
        push ax         ; сохранить байты конфигурации
        mov  cl,9       ; установить счетчик сдвигов
        shr  ax,cl      ; сдвинуть содержимое AX на CL разрядов
        and  ax,7       ; установить количество портов
        add  ax,30h     ; и преобразовать цифру в формат ASCII
        push ax         ; сохранить количество портов
        mov  ah,9       ; вывести
        lea  dx,messser ;    строку сообщения
        int  21h        ;         на экран
        pop  dx         ; восстановить количество портов
        mov  ah,6       ; вывести
        int  21h        ;   количество портов на экран
        pop  ax         ; восстановить байты конфигурации
; параллельные порты (устанавливаются аналогично послед.)
        push ax
        mov  cl,0eh
        shr  ax,cl
        and  ax,3
        add  ax,30h
        push ax
        mov  ah,9
        lea  dx,messparr
        int  21h
        pop  dx
        mov  ah,6
        int  21h
        pop  ax
```

```
; игровой адаптер
        test ah,10h      ; игровой адаптер присутствует ?
        jz   f           ; нет, выходим
        lea  dx,messgame  ; иначе
        mov  ah,9         ; выводим
        int  21h          ; сообщение
f:      int 20h           ; а затем завершаем работу
prg     ends
        end beg

; процедура преобразования двоичного числа в строку символов
; вход - ax - число;
; выход - строка по адресу ds:bx
bin2str proc
      push dx
      push cx
      push bx
      push si
      push ax
      mov cx,6
fill_buff:
      mov byte ptr [bx+1],' '    ; заполнить строку пробелами
      inc bx
      loop fill_buff
      mov  si,10                 ; подготовиться к делению на 10
      or  ax,ax                  ; если в (ax) отрицательное число
      jns clr_div
      neg ax                     ; то изменить его знак
clr_div:
      sub dx,dx                  ; очистить старшую половину делимого
      div si                    ;(dx;ax)/10 -> (рез -> ax, ост->dx)
      add dx,'0'                 ; преобразовать остаток в симв. вид
      dec bx
      mov [bx+1],dl              ; запомнить очередн. цифру в строке
      or  ax,ax                  ; в регистре ax - 0?
      jnz clr_div                ; нет, повторить процедуру
```

```
        pop ax
        or ax,ax                ; исх.число отрицательно ?
        jns no_more             ; нет, тогда все
        dec bx
        mov byte ptr [bx+1],'-'    ; поместить в строку знак '-'
no_more:
        pop si
        pop bx
        mov byte ptr [bx],6             ; поместить счетчик символов
        pop cx
        pop dx
        ret
bin2str endp

; процедура вывода на экран одного символа
; входной параметр - выводимый символ - в регистре al
out_hex_1 proc near
        cmp     al,09h   ; символ в диапазоне 0 ... 9 ?
        jle     nu       ; да, идем на преобразование
;иначе считаем символ в диапазоне a ... f
        add     al,041h  ; переводим символ в код ASCII
        sub     al,0ah   ;
        jmp     outsym   ; переход к выводу символа
nu:     add     al,30h   ; переводим символ в код ASCII
outsym: mov     dl,al    ; выводимый символ в DL
        mov     ah,06h   ; функция DOS 6 - вывод символа на экран
        int     21h      ; вызов функции вывода
        ret              ; возврат
out_hex_1 endp
; вывод байта на экран в шестнадцатеричном виде
; входной параметр - выводимый байт - в регистре AL
; выходных параметров нет
; используется процедура out_hex_1
hex_out proc near
        push cx          ; сохранить
        push dx          ; используемые
```

```
        push ax          ; внутри процедуры регистры
        mov  cl,04h      ; счетчик сдвига для выделения одной hex-цифры
        shr  al,cl       ; выделяем старшую цифру hex-числа
        and  al,0fh      ; и очищаем ненужную половину байта
        call out_hex_1   ; выводим старшую цифру
        pop  ax          ; восстанавливаем исходное представление
        push ax          ; числа и опять его прячем в стек
        and  al,0fh      ; выделяем младшую цифру
        call out_hex_1   ; и выводим
        pop  ax          ; восстанавливаем
        pop  dx          ; используемые
        pop  cx          ; регистры...
        ret              ; ... и выходим
hex_out endp
```

Дополнительная литература

1. Микропроцессорный комплект К1810:Структура, программирование, применение: Справочная книга/Ю.М.Казаринов, В.М. Номоконов, Г.С.Подклетнов, Ф.В.Филиппов; Под ред. Ю.М. Казаринова. - М:Высш.шк.,1990.- 269с.

2. Intel486 ™ DX MICROPROCESSOR. INTEL CORPORATION, October, 1992.

3. Бретт Гласс. Ассорти из байтов или память ПК//Мир ПК.- 1992.- N 3.- С.38-43.

4. Левкин Г.Н.,Левкина Г.И. Введение в схемотехнику ПЭВМ IBM PC/AT. - М:Изд-во МПИ,1991. - 96с.

5. Лукач Ю.С.,Сибиряков А.Е. Архитектура ввода-вывода персональных ЭВМ IBM PC /Инженерно-техническое бюро, Сверд ловск, 1990. - 47с.

6. Абель П. Язык Ассемблера для IBM PC и программирования:Пер.с англ. Ю.В.Сальникова. - М:Высш.шк.,1992. - 447с.

7. Скэнлон Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера:Пер.с англ.- М:Радио и связь,1989. -336с.

8. Данкан Р. Профессиональная работа в MS DOS:Пер.с англ.- М:Мир,1993. - 509с.

9.Рудаков П.И., Финогенов К.Г. Программируем на языке асскмблера IBM PC –

Изд. 2-е. – Обнинск: Издательство «Принтер»,1997 г.-584 с.

8. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Данный раздел содержит задания к лабораторным работам по курсу “Организация ЭВМ” и иллюстрированные примеры их выполнения.

Тема: Логический состав процессора компьютера и назначение его компонентов. Принципы программного управления

Цель лабораторной работы:

Целью данной лабораторной работы является первоначальное знакомство с логическим процессом функционирования компьютера при выполнении программы, хранимой в оперативной памяти.

Задание к лабораторной работе:

- изучить соответствующий теоретический материал, используя конспекты и литературу;

- на примере функциональной модели ознакомиться с компонентами процессора ЭВМ, необходимыми для реализации принципа программного управления; ознакомиться с системой команд модели, принципами кодирования программы и данных и их размещением в памяти;

- заданное арифметическое выражение разложить на уровень элементарных операций; каждой элементарной операции поставить в соответствие машинную команду, т.е. составить программу для вычисления заданного выражения в машинных кодах модели;

- ввести полученную программу в моделируемую память; выполнить программу для различных вариантов исходных данных, наблюдая, как изменяется состояние компонентов процессора ЭВМ и памяти при выполнении каждой команды; разобраться, почему и как модифицируются те или иные объекты;

- составить отчет по лабораторной работе.

Пример выполнения задания

Заданное выражение: **Y:=(A+B)/(C-D)*F**

Для данного варианта задания программа в кодах модели примет вид:

адр.	Команда	комментарий
0000	0000 0010 1111	Ввести A в ячейку памяти с адресом F
0001	0000 0010 1110	Ввести B в ячейку памяти с адресом E
0010	0000 0010 1101	Ввести C в ячейку памяти с адресом D
0011	0000 0010 1100	Ввести D в ячейку памяти с адресом C
0100	0000 0010 1011	Ввести F в ячейку памяти с адресом B
0101	0111 1111 1110	Сложить содержимое ячеек F и E (A+B)
0110	1000 1101 1100	Из содержимого ячейки D вычесть содержимое ячейки C (C-D)
0111	1010 1111 1101	Содержимое ячейки с адресом F поделить на содержимое ячейки D ((A+B)/(C-D))
1000	1001 1111 1011	Содержимое ячейки с адресом F умножить на содержимое ячейки B ((A+B)/(C-D)*F)
1001	0000 0001 1111	Вывести результат в канал вывода
1010	0000 0000 0000	Останов
1011	0000 0000 0000	Ячейка для хранения F
1100	0000 0000 0000	Ячейка для хранения D
1101	0000 0000 0000	Ячейка для хранения C
1110	0000 0000 0000	Ячейка для хранения B
1111	0000 0000 0000	Ячейка для хранения A

Варианты заданий к лабораторной работе

Варианты заданий для лабораторной работы приведены в таблице 1.

Таблица 1

№ п/п	вариант задания
1	$Y:=a*b^2+c/d$
2	$Y:=a^2/(b*c)+d$
3	$Y:=(a+b)^2-c/d$
4	$Y:=(a-b)^2+c*d$
5	$Y:=a-b^2/c+d$
6	$Y:=a-b/c+d^2$
7	$Y:=a/b*c+d^2$
8	$Y:=a/b^2+c*d$
9	$Y:=a^2+b/d-c$
10	$Y:=a^2-b/c+d$
11	$Y:=a/b/c^2-d$
12	$Y:=a/b^2*c^2-d$
13	$Y:=a^3*b/c+d$
14	$Y:=(a+b)^2+c^2$

15	$Y:=(a-b)^2*(c+d)$
16	$Y:=a*b/c-d^2$

Тема: Система команд процессоров

Цель лабораторной работы:

Цель данной лабораторной работы состоит в знакомстве с группами команд процессоров и изучении особенностей функционирования команд различных групп.

Задание к лабораторной работе:

- изучить соответствующий теоретический материал, используя конспекты и литературу;

- на примере функциональной модели ознакомиться с различными типами команд процессоров ЭВМ (арифметико-логические, пересылки, ветвления и организации циклов, управления состоянием процессора) и особенностями их выполнения;

- составить программы для вычисления заданных выражений в машинных кодах модели;

- ввести полученные программы в моделируемую память; выполнить программы для различных вариантов исходных данных, наблюдая, как изменяется состояние компонентов процессора ЭВМ и памяти.

- при выполнении каждой команды; разобраться, почему и как модифицируются те или иные объекты;

- составить отчет по лабораторной работе.

Пример выполнения задания

Заданные выражения:

1
$$Y:=\begin{cases} a/b + c*a, & \text{если } b > 0 \\ c + a, & \text{если } b = 0 \\ b/a + c*a, & \text{если } b < 0 \end{cases}$$

2.

$Y:=N!$

Программа в кодах модели для вычисления первого выражения примет следующий вид:

адр.	Команда	комментарий
0000	1001 1101 1011	c*a
0001	0111 1100 1110	b + 0 (для установки регистра признаков)
0010	1011 0100 1001	переход по нулю по адресу 9
0011	1011 0010 0111	переход по плюсу по адресу 7
0100	1010 1100 1011	b/a
0101	0111 1101 1100	c*a + b/a
0110	1011 0000 1001	безусловный переход на вывод результата
0111	1010 1011 1100	a/b
1000	0111 1101 1011	c*a + a/b
1001	0000 0001 1101	вывод результата в канал вывода
1010	0000 0000 0000	останов
1011	0000 0000 0011	ячейка, хранящая значение a
1100	0000 0000 0011	ячейка, хранящая значение b
1101	0000 0000 0110	ячейка, хранящая значение c
1110	0000 0000 0000	ячейка, хранящая 0

Для второго выражения программа в кодах модели примет следующий вид:

адр.	Код команды	комментарий
0000	0001 0000 1000	загрузить в регистр 0 N
0001	0001 0001 1001	загрузить в регистр 1 1
0010	1011 0100 0101	переход по нулю по адресу 5
0011	1110 0001 0000	перемножить содержимое рег. 1 и 0
0100	1011 1111 0010	организовать цикл с началом по адр. 2
0101	0010 0001 1010	запомнить результат по адресу A
0110	0000 0001 1010	вывести результат в канал вывода
0111	0000 0000 0000	останов
1000	0000 0000 0110	ячейка памяти для хранения N
1001	0000 0000 0001	ячейка памяти для хранения 1
1010	0000 0000 0000	ячейка памяти для хранения результата (Y)

При выполнении данной работы можно использовать интегрированную оболочку, включающую редактор, ассемблер и собственно модель (см. приложение 1). В этом случае приведенные выше программы примут следующий вид.

1. Для выражения с ветвлением

```
MMM C,A     C*A
AMM B,NULL        B+0 (для установки регистра признаков)
JZ  OUR         переход по "0" на вывод результата
JP  PLUS        переход по "+" (B > 0)
DMM B,A     B/A
AMM C,B     B/A + C*A
JMP OUR     безусловный переход на вывод результата
PLUS: DMM A,B     A/B
AMM C,A     C*A + A/B
OUR:  OUT C  вывод результата в канал вывода
STOP            останов
A:    .DW 2     значения
B:    .DW 2     переменных
C:    .DW 6     выражения
NULL: .DW 0 ячейка для хранения 0
END
```

2. Для циклического выражения:

```
MOVR 0,N    загрузить в регистр 0 N
MOVR 1,ONE                загрузить в регистр 1 единицу
CYCL: JZ   OU1        при нулевом результате перейти по символическому адресу OU1
MRR  1,0                  перемножить содержимое регистров 1 и 0
LOOP CYCL           замкнуть цикл по символическому адресу CYCL
OU1:  MOVM 1,FACT         записать результат по адресу FACT
OUT  FACT           вывести результат в канал вывода
STOP            останов
N:    .DW 5     значение N
ONE:  .DW 1  константа “1”
FACT: .DS     ячейка памяти для результата
END
```

Варианты заданий к лабораторной работе

Варианты заданий к лабораторной работе приведены в таблице 2.

Таблица 2

№ п/п	выражение 1 (ветвление)	выражение 2 (цикл)
1	$Y:=\begin{cases} (A+B)*(F-E), \text{ если } F<>E \end{cases}$	$Y:= (A/B)^{(X-1)} - Z$

	⌊ (A+B), если F=E	
2	⌈ A/B - B*C, если B<>0 Y:=⌊ ⌊ A-C, если B =0	Y:= max(a(1),...,a(N)), где a(1),...,a(N) в цикле вводятся через канал ввода
3	⌈ A/B/C + F, B<>0,C<>0 Y:=⌊ ⌊ A+F, B=0 или C=0	Реализовать операцию деления целых чисел через вычитание. В результате операции деления должны получаться частное и остаток.
4	⌈ A/B + C*A, если B<>0 Y:=⌊ ⌊ A*A + C, если B =0	Y:= $X^N/2^N$
5	⌈A*B/C/D, если C<>0, D<>0 Y:=⌊ ⌊A*B, если C=0, или D=0	$Y:=\sum_{i=1}^{N}(A^i + B^i)$
6	⌈ A/B + C*A , если B<>0 Y:=⌊ ⌊ C*A , если B=0	$Y:=\sum_{i=1}^{N}(A/I + B/I)$
7	⌈ (A-C*B)/F ,если F<>0 Y:=⌊ ⌊ (A -C* B), если F=0	Y:= $A^x + B^x$
8	⌈ (A + B)*(C/F),если F<>0 Y:=⌊ ⌊ (A + B)*E, если F=0	Y:= $A^{(X+Y/2)}$
9	⌈(A*B)/(C*D),если C<>0 и D<>0 Y:=⌊ ⌊(A * B), если C=0,или D=0	$Y:=\sum_{i=1}^{N}(B+i)^I$
10	⌈ (A*B)/C ,если C<>0 Y:=⌊ ⌊ (A/B), если C=0	$Y:=\sum_{i=1}^{N}(A+i)^2$
11	⌈A/(B*C)+E, если C<>0 и B<>0 Y:=⌊ ⌊ A+E, если B=0,или C=0	$Y:=\sum_{i=0}^{N-1}(-1)^i*(a+i)$
12	⌈ (A*A)/(B*B)+F,если B<>0 Y:=⌊ ⌊ A*A + F*F, если B=0	$Y:=\prod_{i=1}^{N-1}(A+i)$
13	⌈ A/B-B*C+F, если B<>0 Y:=⌊ ⌊ C+F, если B=0	$Y:=\sum_{i=1}^{N}(I/2+i)$
14	⌈ A/B + C*A,если B<>0 Y:=⌊ ⌊ A+C*A, если B=0	$Y:=\sum_{i=1}^{N-1}(A+i^i)$
15	⌈ A*B/C/D, если C<>0, D<>0 Y:=⌊ ⌊ A+B, если C=0, или D=0	$Y:=\prod_{i=1}^{N}(A/i + B/i)$
16	⌈ A/B + C/B, если B<>0 Y:=⌊ ⌊ A-C , если B=0	$Y:=\sum_{i=1}^{N}(A^i/i)$

Тема: Система команд процессоров и методы адресации

Цель лабораторной работы:

Цель данной лабораторной работы состоит в дальнейшем изучении особенностей выполнения команд процессоров различных групп и знакомстве с методами адресации памяти при разработке программ на машинном языке.

Задание к лабораторной работе:

- изучить соответствующий теоретический материал, используя конспекты и литературу;

- на примере функциональной модели ЭВМ-2 (приложение 2) ознакомиться с системой команд процессора и особенностями выполнения команд различных групп (особое внимание уделить реализации команд умножения и деления, переходов и организации цикла со счетчиком);

- рассмотреть допустимые в системе команд формы адресации и их реализацию;

- составить программы для реализации заданных выражений в кодах модели или используя язык ассемблера модели, при этом учесть то, что программа для реализации линейного выражения должна быть реализована тремя способами - с использованием абсолютной формы адресации, с использованием относительной формы и с использованием только регистров и стэковой памяти; программа для реализации ветвящегося процесса должна обязательно использовать команды сравнения; циклическая программа должна быть реализована любыми двумя из трех способов - по схеме цикла с предусловием, цикла с постусловием, и цикла со счетчиком;

- ввести полученные программы в моделируемую память; выполнить программы для различных вариантов исходных данных, наблюдая, как изменяется состояние компонентов процессора ЭВМ и памяти при выполнении каждой команды; разобраться, почему и как модифицируются те или иные объекты, как выполняются команды и как осуществляется доступ к операндам;

- составить отчет по лабораторной работе;

Пример реализации программ по лабораторной работе

Рассмотрим реализацию программ на учебном ассемблере модели ЭВМ-2 для вычисления выражения **Y:=(A/B - C*F)*E** с использованием различных методов адресации.

а) Абсолютная адресация

```
адрес   код       мнемокод     комментарий

00      40        in 0         ; ввод a
01      18  24    stor a,A     ; запомнить a по символическому адресу A
03      41        in 1         ; ввод b
04      18  25    stor a,B     ; запомнить b по символическому адресу B
06      42        in 2         ; ввод c
07      18  26    stor a,C     ; запомнить c по символическому адресу C
09      43        in 3         ;  ввод f
0a      18  27    stor a,F     ; запомнить f по символическому адресу F
0c      44        in 4         ; ввод e
0d      18  28    stor a,E     ; Запомнить e по символическому адресу E
0f      11  24    load b,A     ; загрузить в RB A
11      90  01    mui  A,1     ; расширить знак(преобразовать слово в двойное слово)
13      98  25    dvm  a,B     ; (rA,rB)/B-> частное в rB, остаток в rA
15      19  29    stor b,tmp   ; запомнить результат деления по адресу tmp
17      11  26    load b,C     ; загрузить в rB C
19      88  27    mum  a,F     ; (rB)*(F)-> (rA,rB)
1b      10  29    load a,tmp   ; в rA загрузить TMP
1d      69        sbr  a,b     ; вычесть из (rA) (rB) (a/b - c*f)
1e      0a        mov  b,a     ; скопировать результат в rB
1f      88  28    mum  a,E     ; (rB)*(E) -> (rA,rB)
21      09        mov  a,b     ; скопировать результат в RA
22      4d        out  5       ; вывести результат в 5 порт
23      f8        stop         ; стоп
24      00        A:   .ds 1   ; ячейка памяти для A
25      00        B:   .ds 1   ; --//--        B
26      00        C:   .ds 1   ; --//--        C
27      00        F:   .ds 1   ; --//--        F
28      00        E:   .ds 1   ; --//--        E
29      00        tmp: .ds 1   ; --//--      tmp
```

В программе в 12 командах используется абсолютная адресация к операндам в памяти. Программа является неперемещаемой, т.е. при переносе программы в другие адреса памяти для восстановления ее работоспособности необходимо скорректировать коды 12 команд. Для программы, предназначенной для решения какой-либо реальной задачи, количество таких команд, естественно, будет значительно выше, со всеми вытекающими последствиями. Попробуем сократить число команд с абсолютными адресами и построить перемещаемую программу для вычисления того же выражения, используя относительную адресацию. Поскольку в модели ЭВМ-2 нет команд загрузки регистров содержимым счетчика адреса команд, то, по крайней мере одна команда (первая) будет содержать абсолютный адрес - адрес начала программы, относительно которого будут адресоваться все другие команды программы.

b) относительная адресация

```
адрес   код       мнемокод          комментарий
```

```
00      00       .base b              ; rB - базовый
00      31 00    start: mvi b,start   ; Относительный адрес - в rB: rB - базовый
02      40       in 0                 ; ввод A
03      1c 74    stor a,A             ; запомнить  по относительному адресу ((rB)+34h)
05      41       in 1                 ; ввод b
06      1c 75    stor a,B             ; запомнить b по относительному адресу ((rb)+35h)
08      42       in 2                 ; ввод C
09      1c 76    stor a,C             ; запомнить C по относительному адресу
0b      43       in 3                 ; ввод F
0c      1c 77    stor a,F             ; запомнить F по относительному адресу
0e      44       in 4                 ; ввод E
0f      1c 78    stor a,E             ; запомнить E по относительному адресу
11      21       push b               ; (rB)↓ (sp) сохранить базовый адрес в стеке
12      14 75    load a,B             ; загрузить в rA B, используя относительный адрес
14      20       push a               ; (rA)↓(sp)- сохранить rA в стеке
15      15 74    load b,A             ; загрузить в rB A, используя относительный адрес
17      90 01    mui  a,1             ; распространить знак
19      9c 81    dvm  a,1(sp)         ; разделить (rA,rB) на содержимое верхушки стека
1b      09       mov  a,b             ; переслать частное в rA
1c      29       pop  b               ; (sp)↑(rB) : вытолкнуть из стека в rB A
1d      29       pop  b               ; (sp)↑(rb) : восстановить из стека базовый адрес
1e      1c 79    stor a,tmp           ; запомнить по относительному адресу результат A/B
20      14 76    load a,C             ; загрузить в rA C
22      21       push b               ; (rB)↓(sp) : сохранить содержимое базового регистра
23      15 77    load b,F             ; загрузить в rB F ,используя относительный адрес
25      80       mur  a,a             ; (rB)*(rA)->(rA,rB),т.е. C*F
26      09       mov  a,b             ; результат C*F -> rA
27      29       pop  b               ; (sp)↑(rB) : восстановить содержимое базового рег-ра
28      21       push b               ; (rB)↓(sp) : сохранить содержимое базового регистра
29      15 79    load b,tmp           ; загрузить A/B в rB, используя относит. адрес
2b      6a       sbr  b,a             ; (rB)-(rA) -> (rB) (a/b - c*f)
2c      38       xchg                 ; обменять содержимое регистров rA и rB
2d      29       pop  b               ; (sp)↑(rB) : восстановить содержимое базового рег-ра
2e      15 78    load b,E             ; загрузить в rB E, используя относительный адрес
30      80       mur  a,a             ; (rB)*(rA)->(rA,rB) (a/b - c*f)*e
31      38       xchg                 ; обменять содержимое регистров rA и rB. рез-т в rA
32      4d       out 5                ; вывести результат в порт 5
33      f8       stop                 ; стоп
34      00       A:   .ds 1           ; ячейки
35      00       B:   .ds 1           ;    для
36      00       C:   .ds 1           ;       хранения
37      00       F:   .ds 1           ;           исходных
38      00       E:   .ds 1           ;                данных
39      00       tmp: .ds 1           ;    и промежуточного результата
```

Получили "квазиперемещаемую" программу, т.е. программу, которая может функционировать в любых адресах памяти, если при загрузке скорректировать содержимое только одной ячейки - адрес начала программы. Рассмотрим еще один

вариант данной программы. В этом варианте будем использовать только регистры и стековую память.

с) вариант программы с использованием только регистров и стековой памяти

адрес	код	мнемокод	комментарий
00	40	in 0	; ввод C в rA
01	0a	mov b,a	; (rA) -> (rB)
02	41	in 1	; ввод F в rA
03	80	mur a,a	; (rB)*(rA)->(rA,rB) (c/f, результат в паре рег. rA,rB)
04	42	in 2	; ввод E
05	20	push a	; (rA)↓(sp) : сохранить E в стеке
06	21	push b	; (rB)↓(sp) : сохранить результат c/f в стеке
07	43	in 3	; ввод B
08	20	push a	; (rA)↓(sp) : сохранить B в стеке
09	44	in 4	; ввод A
0a	0a	mov b,a	; (rA) -> (rB)
0b	90 01	mui a,1	; распространение знака
0d	9c 81	dvm a,1(sp)	; (rA,rB)/((sp)+1),частное в rB, остаток в rA : (a/b)
0f	28	pop a	; (sp)↑(rA) : B вытолкнуть из стека в rA
10	28	pop a	; (sp)↑(rA) : C*F вытолкнуть из стека в rA
11	6a	sbr b,a	; (rB)-(rA) -> (rB): (a/b - c*f)
12	28	pop a	; (sp)↑(rA) : вытолкнуть из стека E в rA
13	80	mur a,a	; (rB)*(rA) - >(rA,rB) (a/b-c*f)*e
14	09	mov a,b	; переслать результат из rB в rA
15	4d	out 5	; Вывести результат в порт 5
16	f8	stop	; останов

В результате получили полностью перемещаемую программу, в которой для хранения исходных данных и промежуточных результатов используются только регистры и ячейки стека. Т.е. под данные не задействована ни одна ячейка памяти. Это полностью перемещаемый вариант программы.

Рассмотрим применение команд сравнений и переходов на примере вычисления выражения:

$$Y := \begin{cases} 0 & \text{,при } X < 0 \\ X & \text{,при } 0 < X < 1 \\ X^4 & \text{,при } X > 1 \end{cases}$$

адрес	код	мнемокод	комментарий
00	40	in 0	;ввод X
01	b0 00	cmi a,0	;сравнить (rA) с 0 (установить регистр флагов)
03	b9 0f	jn lab1	;переход, если установлен признак N (X<0)
05	b0 01	cmi a,1 ;	;сравнить (rA) с 1 (установить рег. флагов по (rA)-1)
07	bf 11	jnp lab2	;переход, если не установлен признак P,т.е. X<=1

```
09      38       xchg            ; (rA)↔ (rB) обмен содержимого rA и rB
0a      81       mur  a,b        ; (rB)*(rB) -> (rA,rB)   : X*X
0b      81       mur  a,b        ; (rB)*(rB) -> (rA,rB)   : (X*X)*(X*X)
0c      38       xchg            ;(rA)↔(rB)    : теперь результат в (rA)
0d      b8  11   jmp  lab2       ;безусловный переход на вывод результата
0f      30  00   lab1:  mvi  a,0 ;X=0
11      49       lab2:  out  1   ; вывод результата
12      f8       stop
```

В данной программе использованы команды сравнения CMI - сравнить содержимое регистра и непосредственный операнд, а также команды переходов: JN - переход по минусу; JNP - переход по неположительному результату; JMP - безусловный переход.

Теперь рассмотрим организацию циклов в программах на машинном уровне с использованием различных способов, а именно: по схеме цикла с предусловием и по схеме цикла с постусловием. Причем организацию цикла по второй схеме рассмотрим в двух вариантах : реализация с помощью команд сравнений и переходов и с помощью команды организации цикла со счетчиком.

Первый вариант:

Вычисление выражения

$$Y:=\sum_{i=0}^{N} I!$$

по структурной схеме цикла с предусловием (while-цикл)

```
адрес   код      мнемокод        комментарий

00      31  01   mvi b,1
02      19  26   stor b,F        ; f:=1 (f- текущее значение факториала )
04      19  27   stor b,Y        ; y:=1
06      40       in 0            ; ввод N в rA
07      b0  00   cmi a,0         ; сравнить (rA) с 0 (n=0 ?)
09      ba  22   jz exit         ; перейти по символическому адресу exit при (rA)=0
0b      38       xchg            ; (rA)↔(rB) : в rA - 1(i:=1); в rB - N
0c      19  28   stor b,N        ; запомнить N по символическому адресу N
0e      11  28   cycl:  load b,N ; загрузить в rB N
10      a9       cmr  a,b        ; сравнить rA и rB  (i<=n?)
11      bb  22   jp   exit       ; переход по признаку P (i>n)
13      11  26   load b,F        ; в rB - (i-1)!
15      20       push a          ; (rA)↓ (sp) (сохранить i в стеке )
16      80       mur a,a         ; (rB)*(rA)->(rA,rB) i!:=i*(i-1)!
17      19  26   stor b,F        ; запомнить i! по символическому адресу F
19      59  27   adm  b,Y        ; f+Y
1b      19  27   stor b,Y        ; Y:=Y+F
1d      28       pop a           ; (sp)↑(rA) восстановить в rA I
1e      60  01   adi a,1         ; (rA)+1 -> (rA) : i:=i+1
20      b8  0e   jmp cycl        ; переход на проверку условия окончания цикла
22      10  27   exit:  load a,Y ; (rA)<- результат вычисления выражения
```

```
24      49      out 1           ; вывод результата в порт 1
25      f8      stop            ; стоп
26      00      F:  .ds 1       ; ячейка под текущее значение факториала
27      00      Y:  .ds 1       ; ячейка под результат
28      00      N:  .ds 1       ; ячейка для хранения N
```

Второй вариант:

Вычисление выражения

$$Y:=\sum_{i=0}^{N} I!$$

по структурной схеме цикла с постусловием (repeat...until - цикл)

```
адрес   код     мнемокод        комментарий

00      31 01   mvi b,1
02      19 24   stor b,F        ; f:=1 (f- текущее значение факториала )
04      19 25   stor b,Y        ; y:=1
06      40      in 0            ; ввод N в rA
07      b0 00   cmi a,0         ; сравнить (rA) с 0 (n=0 ?)
09      ba 20   jz exit         ; перейти по символическому адресу exit при (rA)=0
0b      38      xchg            ; (rA)↔(rB) : в rA - 1(i:=1); в rB - N
0c      19 26   stor b,N        ; запомнить N по символическому адресу N
0e      11 24   cycl:  load b,F ; в rB - (i-1)!
10      20      push a          ; (rA)↓(sp) (сохранить i в стеке )
11      80      mur a,a         ; (rB)*(rA)->(rA,rB) i!:=i*(i-1)!
12      19 24   stor b,F        ; запомнить i! по символическому адресу F
14      59 25   adm  b,Y        ; f+Y
16      19 25   stor b,Y        ; Y:=Y+F
18      28      pop a           ; (sp)↑(rA) восстановить в rA i
19      60 01   adi a,1         ; (rA)+1 -> (rA) : i:=i+1
1b      11 26   load b,N        ; (rB) <- N
1d      a9      cmr  a,b        ; сравнить (rA) с (rB) :(i>n?)
1e      bf 0e   jnp cycl        ; переход на начало цикла, если признак N (i<=n)
20      10 25   exit:  load a,Y ; (rA)<- результат вычисления выражения
22      49      out 1           ; вывод результата в порт 1
23      f8      stop            ; стоп
24      00      F:  .ds 1       ; ячейка под текущее значение факториала
25      00      Y:  .ds 1       ; ячейка под результат
26      00      N:  .ds 1       ; ячейка для хранения N
```

И, наконец, последний вариант:

Вычисление выражения

$$Y:=\sum_{i=0}^{N} I!$$

с использованием команды организации цикла со счетчиком

```
адрес   код     мнемокод        комментарий

00      31 01   mvi b,1
02      19 21   stor b,F        ; f:=1 (f- текущее значение факториала )
04      19 22    stor b,Y       ; y:=1 ,т.к. 0!=1
06      40      in 0            ; ввод N в rA
```

```
07      b0  00    cmi a,0          ; сравнить (rA) с 0 (n=0 ?)
09      ba  1d    jz exit          ; перейти по символическому адресу exit при (rA)=0
0b      38        xchg             ; (rA)↔ (rB) : в rA - 1(i:=1); в rB - N
0c      21        cycl:  push b    ; (rB)↓(sp) : сохранить rB в стеке(счетчик цикла)
0d      11  21    load b,F         ; в rB - (i-1)!
0f      20        push a           ; (rA)↓(sp) (сохранить i в стеке )
10      80        mur a,a          ; (rB)*(rA)->(rA,rB) i!:=i*(i-1)!
11      19  21    stor b,F         ; запомнить i! по символическому адресу F
13      59  22    adm  b,Y         ; f+Y
15      19  22    stor b,Y         ; Y:=Y+F
17      28        pop a            ; (sp)↑(rA) восстановить в rA i
18      60  01    adi a,1          ; (rA)+1 -> (rA) : i:=i+1
1a      29        pop b            ; (sp)↑(rB) :восстановить счетчик цикла
1b      e1  0c    loop b,cycl      ; организация цикла со счетчиком в rB
1d      10  22    exit: load a,Y   ; (rA)<- результат вычисления выражения
1f      49        out 1            ; вывод результата в порт 1
20      f8        Stop             ; стоп
21      00        F:   .ds 1       ; ячейка под текущее значение факториала
22      00        Y:   .ds 1       ; ячейка под результат
```

Варианты заданий к лабораторной работе

а) для реализации линейного выражения с использованием различных методов адресации варианты заданий выбираются из таблицы 1.

б) для реализации ветвящегося и циклического выражений варианты заданий выбираются из таблицы 2.

Тема: Индексная адресация и работа с массивами

Цель лабораторной работы:

Целью данной лабораторной работы является изучение особенностей использования индексной адресации при работе с массивами.

Задание к лабораторной работе:

- изучить соответствующий теоретический материал, используя конспекты и литературу;

- на примере функциональной модели ЭВМ-2 (приложение 2) ознакомиться с особенностями индексной адресации;

- для предложенного варианта задания реализовать программу, обеспечивающую выполнение операций над массивами;

- оформить отчет по лабораторной работе.

Пример реализации программы по лабораторной работе

Ниже приведен пример реализации программы, обеспечивающей поэлементное сложение элементов двух массивов MA и MB. Результат размещается в третьем массиве –

МС. При изучении программы особое внимание следует обратить на организацию доступа к элементам массивов с помощью индексной адресации. При выполнении лабораторной работы с использованием функциональной модели ЭВМ-2 необходимо обратить внимание на ограничение длины начального адреса (до 3fh) в командах, обеспечивающих обращение к индексируемой памяти.

```
адр   код      мнемокод          комментарий

00    b8 41    jmp  begin        ; переход на начало алгоритма
02    00       ;
02    00       ; данные          размещены в начале из за ограничений инд. адр.
02    00       ;                 (начальный адрес должен быть меньше 3fh)
02    00       ;
02    00       .org 10
10    00 00    MA:    .ds 10     ; резервирование 16 байтов под массив A
      00 00
      . . .
      00 00
20    00 00    MB:    .ds 10     ; резервирование 16 байтов под массив B
      00 00
      . . .
      00 00
30    00 00    MC:    .ds 10     ; резервирование 16 байтов под массив C
      00 00
      . . .
      00  00
40    00       N:       .ds 1    ; резервирование 1 байта под кол-во элементов
41    00 ;
41    00 ;                       собственно алгоритм программы
41    00 ;
41    40       begin: in 0       ; ввод количества элементов массивов
42    18 40    stor a,N          ;запомнить количество элементов по адресу n
44    31 00    mvi b,0           ;загрузить в pB начальный индекс эл-та масс. A
46    20       geta: push a      ;сохранить в стеке счетчик цикла
47    41       in 1              ;ввести очередной элемент массива A
48    1c 50    stor a,MA(b)      ;запомнить его по адресу MA+(b)
4a    61 01    adi b,01          ;вычислить индекс следующего элемента
4c    28       pop a             ;восстановить из стека счетчик цикла
4d    e0 46    loop a,geta       ;замкнуть цикл по метке geta
4f    31 00    mvi b,0           ;загрузить в pB индекс для массива B
51    10 40    load a,N          ;загрузить в регистр A количество элементов
53    20       getb: push a      ;сохранить в стеке счетчик цикла
54    42       in 2              ;ввести очередной элемент массива B
55    1c 60    stor a,MB(b)      ;запомнить его по адресу MB+(b)
57    61 01    adi b,01          ;вычислить индекс следующего элемента
59    28       pop a             ;восстановить из стека счетчик цикла
5a    e0 53    loop a,getb       ;замкнуть цикл по метке getb
5c    00       ;                  собственно суммирование элементов массивов
5c    10 40    load a,N          ; загрузить в pA количество элементов
5e    31 00    mvi b,0           ; загрузить инд. смещение для доступа к массивам
60    20       cycsum: push a    ;сохранить содержимое pA в стеке
```

```
61    14 50    Load a,MA(b)     ; в pA элемент массива a с учетом индекса в (b)
63    5c 60    Adm a,MB(b)      ;сложить очередные эл-ты массивов MA и MB
65    1c 70    stor a,MC(b)     ;запомнить очередной элемент массива MC
67    61 01    adi  b,01        ;вычислить индекс следующего элемента
69    28       pop a            ;восстановить из стека счетчик цикла
6a    e0 60    Loop a,cycsum    ;замкнуть цикл суммирования
6c    00       ;                Суммирование закончено - результат в памяти
6c    f8       Stop             ;останов
6d    00       .end
```

Варианты заданий для лабораторной работы

Варианты заданий для лабораторной работы выбираются из таблицы 3.

Таблица 3

1	Дана квадратная матрица NxN. Обменять местами i-ю и j-ю строки матрицы
2	Даны два одномерных массива одинаковой размерности N. Вычислить выражение вида: Y:=(A(1)+B(n))*(A(2)+B(n-1))*…*(A(n)+B(1))
3	Дан массив A размерности N. Сформировать матрицу B размерности NxN следующего вида: A(1) A(2) … A(n) 2*A(1) 2*A(2) … 2*A(n) B= ..... n*A(1) n*A(2) … n*A(n)
4	Дан массив A размерности N. Из элементов массива A сформировать массив B со следующим расположением элементов A(n/2),A(n/2-1),…,A(2),A(1),A(n),A(n-1),A(n-2),…,A(n/2+1), N – четное
5	Дан массив A, состоящий из N элементов. Получить массив B, в котором элементы располагаются следующим образом: A(1),A(n/2+1),A(2),A(n/2+2),…,A(n/2),A(n) , N – четное
6	Сформировать массив из N целых чисел. Формирование осуществляется по следующему правилу: A(1)=1; A(2)=1; A(i)=A(i/2)+A(i-2), i=1…n
7	Дан массив A, состоящий из 2*N элементов. Сформировать массив B из элементов массива A следующим образом: B(1)=A(1)*A(2)/2; B(2)=A(3)*A(4)/2;…; B(n)=A(2*n-1)*A(2*n)/2
8	Дан массив A, состоящий из N элементов. Найти отдельно суммы и количество четных, нечетных, и нулевых элементов массива. Результаты сформировать в массиве B из шести элементов.
9	Дан массив A, содержащий 2*n элементов, и массив B, содержащий N элементов. Сформировать массив C из элемнтов массивов A и B следующим образом: C(1)=(A(1)+A(2))/B(1); C(2)=(A(3)+A(4))/B(2);…; C(n)=(A(n-1)+A(2*n))/B(n)
10	Даны массивы A и B из N элементов каждый. Сформировать массив C, содержащий следующие элементы: C(1)=A(1),C(2)=B(1), C(3)=A(2),C(4)=B(2),…,C(2*n-1)=A(n),C(2*n)=B(n)
11	Дан массив A из N элементов. Сформировать массивы B и C, содержащие только отрицательные элементы из A и только положительные соответственно.
12	Даны массивы A и B из N элементов каждый. Обменять элементы массивов таким образом, чтобы A(1) встал на место B(n) и наоборот, A(2) - на место B(n-1) и наоборот, A(3) - на место B(n-2) и наоборот , и т.д.

13	Даны массивы A и B из N элементов каждый. Сформировать массив C по следующему правилу: C(1)=A(1)*B(n); C(2)=A(2)*B(n-1);…; C(n)=A(n)*B(1);
14	Дан массив A размерности N. Получить массив B размерности N, где каждый B(i) есть сумма всех ЧЕТНЫХ элементов массива A, от первого до i-го.
15	Дана квадратная матрица NxN. Обменять местами i-й и j-й столбцы матрицы.
16	Дан массив A размерности N. Сформировать матрицу B размерности NxN следующего вида: A(1) 2*A(1) … n*A(1) A(2) 2*A(2) … n*A(2) B= ..... A(n) 2*A(n) … n*A(n)

Тема: Организация подпрограмм и внутренние механизмы передачи параметров

Цель лабораторной работы:

Целью данной лабораторной работы является изучение особенностей выполнения команд передачи управления подпрограмме и возврата из подпрограммы, а так же знакомство с различными методами передачи параметров.

Задание к лабораторной работе:

- изучить соответствующий теоретический материал, используя конспекты и литературу;

- на примере функциональной модели ЭВМ-2 (приложение 2) ознакомиться с командами передачи управления подпрограмме и возврата из подпрограммы;

- для предложенного варианта задания реализовать два варианта программ, обеспечивающих передачу параметров любыми двумя способами;

- оформить отчет по лабораторной работе.

Пример реализации программ по лабораторной работе

Ниже рассматривается четыре варианта программ с различными способами передачи параметров: через регистры общего назначения, через память, через стек и с использованием списка адресов параметров

а) Передача параметров через регистры

В нижеприведенном примере проиллюстрирован прием передачи параметров через регистры общего назначения процессора. Подпрограмма **power** обеспечивает вычисление целой степени некоторого числа. Подпрограмма имеет три параметра - два входных и один выходной. Входными параметрами являются показатель степени и число, возводимое в степень. Выходным параметром является результат возведения числа в

степень. Параметры передаются подпрограмме в регистрах **A** и **B**. Результат работы подпрограммы возвращается в регистре **B**.

$$Y:=\sum_{i=1}^{N} Z^I$$

```
адрес   код      мнемокод          комментарий

00      40       in    0           ; ввод z
01      0a       mov   b,a         ;
02      41       in    1           ; ввод n
03      19 25    stor  b,z         ; запомнить z
05      31 00    mvi   b,0         ; обнулить b
07      19 26    stor  b,Y         ; Y:=0
09      31 01    mvi   b,1         ;
0b      19 27    stor  b,I         ;  i:=1
0d      20       cycl: push  a     ;  сохранить счетчик цикла
0e      10 27    load  a,I         ;  в rA – I
10      11 25    load  b,Z         ;  в rB – Z
12      c0 28    call  power       ; вызов п/п вычисления степени, рез-т возвр. ч/з rB
14      59 26    adm   b,Y         ; Y:=Y+Z^I
16      19 26    stor  b,Y         ; запомнить новое Y
18      11 27    load  b,I         ;  \
1a      61 01    adi   b,1         ;    I:=I+1;
1c      19 27    stor  b,I         ;  /
1e      28       pop   a           ;  восстановить счетчик цикла
1f      e0 0d    loop  a,cycl      ; замкнуть цикл
21      10 26    load  a,Y         ;   в rA – Y
23      4a       out   2           ; вывести результат в канал 2
24      f8       Stop
25      00       Z:    .ds 1
26      00       Y:    .ds 1
27      00       I:    .ds 1
                                   ; подпрограмма вычисления степени числа
                                   ; вход: (rA) - показатель степени; (rB) - основание ;
                                   ; выход: (rB) - z^i
28      19 33    power: stor b,zp  ; запомнить исх. число во временной переменной
2a      31 01    mvi    b,1        ; p:=1    - результат изначально равен 1
2c      20       intcyc: push a    ;  сохранить счетчик цикла
2d      88 33    Mum    a,zp       ; p:=p*z
2f      28       pop    a          ; восстановить из стека счетчик цикла
30      e0 2c    loop   a,intcyc   ; замкнуть цикл
32      c8       Ret               ; возврат в вызывающую программу
33      00       zp:    .ds 1      ; ячейка для временного хранения z
```

b) Пример передачи параметров через общую область памяти.

В следующем примере передача параметров подпрограмме осуществляется через общую область памяти. Т.е. ячейки памяти, хранящие значения, являющиеся параметрами, известны как программе, так и подпрограмме. В данном случае

подпрограмма выбирает значения I и Z непосредственно из ячеек памяти. Занесение значений параметров обеспечивает главная программа. Результат работы подпрограммы также помещается в соответствующую ячейку памяти, откуда он выбирается главной программой.

$$Y:=\sum_{i=1}^{N} Z^I$$

```
адрес   код      мнемокод        комментарий

00      40       in     0        ; ввод z
01      0a       mov    b,a      ;
02      41       in     1        ; ввод n
03      19 1c    stor   b,z      ; запомнить z
05      31 01    mvi    b,1      ;
07      19 1e    stor   b,I      ;  i:=1
09      31 00    mvi    b,0      ;  y:=0
0b      20       cycl:  push  a  ;  сохранить счетчик цикла
0c      c0 1f    call   power    ; вызов п/п вычисления степени
0e      59 1d    adm    b,Y      ; Y:=Y+Z**I
10      10 1e    load   a,I      ; \
12      60 01    adi    a,1      ;  > I:=I+1;
14      18 1e    stor   a,I      ; /
16      28       pop    a        ;  восстановить счетчик цикла
17      e0 0b    loop   a,cycl   ; замкнуть цикл
19      09       mov    a,b      ;   в rA - Y
1a      4a       out    2        ; вывести результат в канал 2
1b      f8       stop
1c      00       Z:     .ds 1
1d      00       Y:     .ds 1
1e      00       I:     .ds 1
                                 ; подпрограмма вычисления степени числа
                                 ; вход: i - показатель степени; z - основание.
                                 ; выход: y - z**I
                                 ; параметры выбираются по абсолютному адресу
1f      21       power:  push b  ; сохранить в стеке текущее значение Y
20      31 01    mvi    b,1      ; p:=1    - результат изначально равен 1
22      10 1e    load   a,I      ; в rA    - показатель степени- счетчик цикла
24      20       intcyc: push a  ;  сохранить счетчик цикла
25      88 1c    mum    a,z      ; p:=p*z
27      28       pop             ; восстановить из стека счетчик цикла
28      e0 24    loop   a,intcyc ; замкнуть цикл
2a      19 1d    stor   b,Y      ; запомнить y:=z**I
2c      29       pop    b        ; восстановить из стека текущее значение суммы
2d      c8       ret             ; возврат в вызывающую программу
```

c) Передача параметров через стек.

В данном случае значения, передаваемые подпрограмме, вызывающей программой последовательно записываются в стек. При этом вызывающая программа резервирует в

стеке одну ячейку для возврата результата, записывая в эту ячейку произвольное значение. В подпрограмме доступ к параметрам осуществляется с использованием адресации относительно верхушки стека. После возврата из подпрограммы вызывающая программа извлекает из стека переданные параметры (в данном случае они не изменяются и не нужны) и результат путем выполнения стандартных операций чтения верхушки стека.

$$Y:=\sum_{i=1}^{N} Z^I$$

```
адрес   код      мнемокод      комментарий

00      40       in     0      ; ввод z
01      0a       mov    b,a    ;
02      41       in     1      ; ввод n
03      19 29    stor   b,z    ; запомнить z
05      31 01    mvi    b,1    ;
07      19 2b    stor   b,I    ; i:=1
09      31 00    mvi    b,0    ;
0b      19 2a    stor   b,Y    ; Y:=0
0d      20       cycl:  push a ; сохранить счетчик цикла
0e      20       push   a      ; зарезервировать ячейку стека для z**i
0f      10 2b    load   a,I
11      20       push   a      ; записать в стек текущее значение i
12      10 29    load   a,z
14      20       push   a      ; записать в стек значение z
15      c0 2c    call   power  ; вызов п/п вычисления степени
17      28       pop    a      ; извлечь z из стека
18      28       pop    a      ; извлечь i из стека
19      29       pop    b      ; извлечь z**i из стека и записать в rB
1a      59 2a    adm    b,Y    ; Y:=Y+Z**I
1c      19 2a    stor   b,Y    ; запомнить новое значение Y
1e      60 01    adi    a,1    ; I:=I+1;
20      18 2b    stor   a,I    ; запомнить новое значение I
22      28       pop    a      ; восстановить счетчик цикла из стека
23      e0 0d    loop   a,cycl ; замкнуть цикл
25      10 2a    load   a,Y    ; в rA - Y
27      4a       out    2      ; вывести результат в канал 2
28      f8       stop
29      00       Z:    .ds 1
2a      00       Y:    .ds 1
2b      00       I:    .ds 1
; подпрограмма вычисления степени числа
; вход: I - показатель степени; Z - число, возводимое в степень.
; выход: Y - Z**i
; параметры передаются и возвращаются через стек ;
; состояние стека при входе в подпрограмму:
;          +-----------------+
;          |                 |  (sp) - вершина стека
;          +-----------------+
;          | ret address     |  (sp+1)
;          +-----------------+
;          | значение z      |  (sp+2) параметр Z
;          +-----------------+
;          | значение I      |  (sp+3) параметр I
;          +-----------------+
```

```
;          | яч. для рез-та  |  (sp+4) параметр Y
;          | счетчик цикла   |  (sp+5) - дно стека
;
2c      31  01    power:  mvi  b,1     ; p:=1    - результат изначально равен 1
2e      14  83    load a,3(sp)         ; в rA    - показатель степени- счетчик цикла
30      20        intcyc: push a       ; сохранить счетчик цикла
31      8c  83    mum  a,3(sp)         ; p:=p*z
33      28        pop  a               ; восстановить из стека счетчик цикла
34      e0  30    loop a,intcyc        ; замкнуть цикл
36      1d  84    stor b,4(sp)         ; запомнить y:=z**i
38      c8        ret                  ; возврат в вызывающую программу
```

d) Передача параметров через таблицу адресов

Доступ к параметрам обеспечивается через таблицу адресов параметров, адрес которой передается в регистре А. Перед началом работы программы производится заполнение списка адресов параметров путем засылки адреса переменной в таблицу.

$$Y:=\sum_{i=1}^{N} Z^I$$

```
адрес   код       мнемокод            комментарий

00      30  2a    mvi   a,Z           ; подготовить
02      18  2d    stor  a,TABPAR      ;
04      30  2b    mvi   a,Y           ; таблицу
06      18  2e    stor  a,TABPAR1     ;
08      30  2c    mvi   a,I           ; адресов
0a      18  2f    stor  a,TABPAR2     ; параметров
0c      40        in     0            ; ввод  z
0d      0a        mov   b,a          ;
0e      41        in     1            ; ввод n
0f      19  2a    stor  b,z           ; запомнить z
11      31  01    mvi   b,1           ;
13      19  2c    stor  b,I           ;  i:=1
15      31  00    mvi   b,0           ;  y:=0
17      20        cycl: push   a      ;  сохранить счетчик цикла
18      30  2d    mvi   a,TABPAR      ; в rA - адрес таблицы адресов параметров
1a      c0  30    call  power         ; вызов п/п вычисления степени
1c      59  2b    adm   b,Y           ; Y:=Y+Z^I
1e      10  2c    load  a,I           ;  \
20      60  01    adi   a,1           ;   > I:=I+1;
22      18  2c    stor  a,I           ;  /
24      28        pop   a             ; восстановить счетчик цикла
25      e0  17    loop  a,cycl        ; замкнуть цикл
27      09        mov   a,b           ; в rA - Y
28      4a        out   2             ; вывести результат в канал 2
29      f8        stop
2a      00        Z:    .ds 1
2b      00        Y:    .ds 1
2c      00        I:    .ds 1
2d      00        TABPAR:  .ds 1
2e      00        TABPAR1:.ds 1
```

```
2f      00          TABPAR2:.ds 1
; подпрограмма вычисления степени числа
; вход: i - показатель степени; z - число, возводимое в степень.
; выход: y - z^i
; параметры выбираются через таблицу адресов параметров, адрес которой
; передается через регистр A
30      21          power:  push b    ; сохранить в стеке текущее значение Y
31      31  01      mvi  b,1          ; p:=1 - результат изначально равен 1
33      20          push a            ; сохранить адрес таблицы адресов параметров
34      14  02      load a,2(a)       ; в rA - адрес третьего параметра
36      14  00      load a,0(a)       ; в rA - показатель степени- счетчик цикла
38      20          intcyc: push a    ; сохранить счетчик цикла
39      14  82      load a,2(sp)      ; в rA - адрес таблицы адресов
3b      14  00      load a,0(a)       ; в rA - адрес первого параметра
3d      8c  00      mum  a,0(a)       ; p:=p*z
3f      28          pop  a            ; восстановить из стека счетчик цикла
40      e0  38      loop a,intcyc     ; замкнуть цикл
42      28          pop  a            ; восстановить адрес таблицы адресов параметров
43      14  01      load a,1(a)       ; загрузить адрес второго параметра
45      1d  00      stor b,0(a)       ; запомнить y:=z^i
47      29          pop  b            ; восстановить из стека текущее значение суммы
48      c8          Ret               ; возврат в вызывающую программу
```

Варианты заданий для лабораторной работы

Варианты заданий для лабораторной работы выбираются из таблицы 4.

Таблица 4

1	$Y:= \min(U^2+V^2,A)$, где $U=\min(C,D)$, $V=\min(E^2/2,F)$;
2	$Y:= P(1) - P(T) +P^2(S-T) - P^3(1)$, где $P(X)=ax^2+bx+c$;
3	$Y:= \min(A,B,C)/\min(F,D,E) + \min(K,N,L)$;
4	$Y:= X(W^N/Z^M) - P(Q/L)$;
5	$Y:= P(X^2) - P(X-1) +P(X)$, где $P(Z)=az^n-a^n+1$;
6	$Y:= XZ^N+MQ^P+LO^{(N/2)}$;
7	$Y:=X^{P(A,W)} + Z^{P(B,U)} /V^{P(C,L)}$, где $P(S,K)=S^2/K + D - 1$;
8	$Y:=A!+(B-C)!/F!$;
9	$Y:=F(S,T) + \max(F(F(S+T,S*T),S*T^2), F^2(S-T,S/T)) + F(1,1)$, где $F(A,B)=(A^2+1)/A+(B^2+1)/B$
10	$Y:=\sum_{l=1}^{N}(\frac{X^L}{A}) \Big/ \sum_{P=1}^{M}\frac{(z+b)^P}{C/D}$
11	$Y:= F(A,B,C) + F(K,N,L)$, где $F(X,W,Z)= X^2 - W^Z$;
12	$Y:= H(S,T) + \max(H^2(S-T,S*T),H^4(S-T,S+T)) +H(1,1)$, где $H(A,B)= (B^2+1)/A +(A^2+1)/B - (A-B)^3$;
13	$Y:= \min(B(1),..., B(m))^M + \min(C(1),..., C(k))^K$
14	$Y:= (\max(A,A+B) + \max(A,B+C))/(1+ \max(A+B*C,1,15))$;
15	$Y:=P(1) - P(t) + P^2(s-t) - P^3(1)$, где $P(x)=Ax^3+Bx^2+Cx +D$;

Тема: Организация прерываний

Цель лабораторной работы:

Данная лабораторная работа посвящена знакомству с аппаратными и программными прерываниями и механизмами их обработки.

Задание к лабораторной работе:

- изучить соответствующий теоретический материал, используя конспекты и литературу;

- на примере функциональной модели ЭВМ-2 ознакомиться с командами организации и обработки аппаратных и программных прерываний;

- для заданного варианта задания реализовать программу с использованием прерываний; программа должна обеспечивать ввод исходных данных и вывод результатов через программные прерывания, а так же модификацию некоторых полей программы через аппаратные прерывания;

- оформить отчет по лабораторной работе.

Пример реализации программы

Ниже рассматривается уже известная программа вычисления суммы степеней, к которой добавлен небольшой отладчик, работающий в режиме прерываний и позволяющий в любой момент времени в ходе выполнения программы изменить содержимое регистров модели и любой ячейки памяти, после чего продолжить выполнение программы с точки прерывания.

```
адрес   код      мнемокод          комментарий

00               .org   08         ; вектор прерывания для выхода в отладчик
08       b0      .db    0b0
09               .org   10         ; начало программы с адреса  10h
10       40      in     0          ; ввод z
11       0a      mov    b,a        ;
12       41      in     1          ; ввод n
13       19 39   stor   b,z        ; запомнить z
15       31 01   mvi    b,1        ;
17       19 3b   stor   b,I        ; i:=1
19       31 00   mvi    b,0        ;
1b       19 3a   stor   b,Y        ; Y:=0
1d       20      cycl:  push  a    ; сохранить счетчик цикла
1e       20      push   a          ; зарезервировать ячейку стека для z^i
1f       10 3b   load   a,I
21       20      push   a          ; записать в стек текущее значение I
22       10 39   load   a,z
24       20      push   a          ; записать в стек значение z
```

```
25      c0 3c    call    power       ; вызов п/п вычисления степени
27      28       pop     a           ; извлечь z из стека
28      28       pop     a           ; извлечь i из стека
29      29       pop     b           ; извлечь z^i из стека и записать в rB
2a      59 3a    adm     b,Y         ; Y:=Y+Z^I
2c      19 3a    stor    b,Y         ; запомнить новое значение Y
2e      60 01    adi     a,1         ; I:=I+1;
30      18 3b    stor    a,I         ; запомнить новое значение I
32      28       pop     a           ; восстановить счетчик цикла из стека
33      e0 1d    loop    a,cycl      ; замкнуть цикл
35      10 3a    load    a,Y         ; в rA - Y
37      4a       out     2           ; вывести результат в канал 2
38      f8       stop
39      00       Z:   .ds 1
3a      00       Y:   .ds 1
3b      00       I:   .ds 1
3c      31 01    power:  mvi  b,1    ; p:=1    - результат изначально равен 1
3e      14 83    load   a,3(sp)      ; в rA    - показатель степени- счетчик цикла
40      20       intcyc: push a      ; сохранить счетчик цикла
41      8c 83    mum    a,3(sp)      ; p:=p*z
43      28       pop    a            ; восстановить из стека счетчик цикла
44      e0 40    loop a,intcyc       ; замкнуть цикл
46      1d 84    stor   b,4(sp)      ; запомнить y:=z^i
48      c8       ret                 ; возврат в вызывающую программу
; Пример очень маленького отладчика программ для модели ЭВМ 2.
; обеспечивает корректировку содержимого ячеек памяти и регистров
; функция 0 - завершение работы;
; функция 1 - корректировка регистров
; функция 2 - корректировка содержимого памяти
; отладчик получает управление через внешнее прерывание 0
; (вектор прерывания - 08)
b0              .org   0b0          ; отладчик размещается с адреса B0
b0      f0      di                  ; запретить прерывания
b1      20      push   a            ; (ra)↓(sp) сохранить rA в стеке
b2      21      push   b            ; (rB)↓(sp) сохранить rB в стеке
b3      40      fun:   in 0         ; ввести номер функции
b4      b0 00   cmi    a,00         ; номер функции 0 ?
b6      ba c2   jz     exit         ; на завершение работы отладчика
b8      b0 01   cmi    a,01         ; номер функции 1 ?
ba      ba c6   jz     reg          ; на модификацию регистров
bc      b0 02   cmi    a,02         ; номер функции 2 ?
be      ba d4   jz     mem          ; на модификацию ячеек памяти
c0      b8 b3   jmp    fun          ; иначе повтор ввода номера функции
c2      29      exit:  pop   b      ; (sp)↑(ra) Восстановить rA
c3      28      pop    a            ; (sp)↑(rB) восстановить rB
c4      e8      ei                  ; разрешить прерывания
c5      d8      rin                 ; возврат из прерывания
;  модификация содержимого регистров программы
c6      14 82   reg: load a,02(sp)  ; (rA) - содержимое rA из стека
c8      48      out    0            ; (rA) -> port 0
c9      14 81   load   a,01(sp)     ; (rA) - содержимое rB из стека
cb      49      out    1            ; (rA) -> port 1
cc      40      in     0            ; ввести новое содержимое (rA)
```

```
cd      1c  82   stor  a,02(sp)      ; запомнить новое содержимое rA
cf      41       in    1             ; ввести новое содержимое rB
d0      1c  81   stor  a,01(sp)      ; запомнить новое содержимое rB
d2      ba  b3   jz    fun           ; перейти на ввод номера функции
;  модификация содержимого ячеек памяти программы
d4      40       mem:  in 0          ; ввод адреса модифицируемой ячейки
d5      0a       mov   b,a          ;
d6      14  40   load  a,00(b)      ; (rA) <- содержимое ячейки с адр в (rB)
d8      48       out   0            ; вывод содержимого яч. памяти в канал 0
d9      40       in    0            ; ввод обновленного содержимого ячейки
da      1c  40   stor  a,00(b)      ; и запись по относительному адресу
dc      ba  b3   jz    fun          ; переход на ввод функции
```

Варианты заданий к лабораторной работе

Варианты заданий к лабораторной работе выбираются из таблицы 4.

Тема: Введение в архитектуру IBM PC

Цель лабораторной работы:

Данная лабораторная работа посвящена знакомству с архитектурой и системой команд процессоров семейства Intel80x86.

Задание к лабораторной работе:

- ознакомиться с базовой моделью программирования процессоров семейства Intel80x86, распределением адресного пространства, системой команд, методами адресации;

- ознакомиться с системой команд и использованием системного отладчика Debug (см. Приложение 4);

- для заданного варианта задания реализовать программы, используя систему команд семейства процессоров Intel80x86;

- используя отладчик, ввести программы в память компьютера и выполнить их в непрерывном и пошаговом режиме, наблюдая результаты выполнения команд по содержимому регистров процессора и оперативной памяти;

- оформить отчет по лабораторной работе.

Пример реализации программ в среде отладчика Debug

Ниже, на примере трех выражений - линейного, ветвящегося и циклического - иллюстрируются примеры реализации алгоритмов в среде отладчика Debug. Для каждого выражения приводится реализующая его программа, а так же состояние регистров и используемых ячеек памяти до и после выполнения программы.

1. Реализация выражения **Y:=(A/B - C*F)*E**

```
C:\>debug
-a100
1B6A:0100 mov ax,word ptr[200]    в регистр AX заносится содержимое слова DS:200 - A
1B6A:0103 cwd                     слово в AX преобразуется в двойное слово в DX:AX
1B6A:0104 idiv word ptr[202]      содержимое двойного слова DX:AX делится на B (A/B)
1B6A:0108 mov bx,ax               Результат из регистра AX копируется в BX
1B6A:010A mov ax,word ptr[204]    в регистр AX заносится содержимое слова DS:204 - C
1B6A:010D imul word ptr[208]      содержимое AX умножается на F (C*F)
1B6A:0111 sub bx,ax               A/B - C*F
1B6A:0113 mov ax,bx               A/B - C*F помещается в AX
1B6A:0115 imul word ptr[206]      (A/B - C*F)*E
1B6A:0119 mov word ptr[20a],ax    результат записывается по адресу DS:20a (Y:= …)
1B6A:011C int 3                   возврат в отладчик
1B6A:011D
-
```

С помощью команды отладчика E занесем в ячейки памяти 200, 202, 204,206, 208 значения A, B, C, E и F (10, 3, 2, 2 и 1)

```
-e200 0a 00 03 00 02 00 02 00 01 00
```

Просмотрим занесенные значения (при выводе на экран байты в слове выводятся в обратном порядке - т.е. 0a 00 соответствует 00 0a)

```
-d200
1B6A:0200  0A 00 03 00 02 00 02 00-01 00 89 0E DD D4 89 3E   ..............>
```

Просмотрим значения регистров до выполнения программы

```
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0100   NV UP EI PL NZ NA PO NC
1B6A:0100 A10002        MOV     AX,[0200]                   DS:0200=000A
```

Запустим программу на выполнение с помощью команды отладчика g

```
-g

AX=0002  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=011C   NV UP EI PL NZ NA PO NC
1B6A:011C CC            INT     3
```

Просмотрим содержимое ячеек памяти после выполнения программы. В ячейке памяти с адресом 1B6A:020A находится результат вычисления нашего выражения - 00 02.

```
-d200
1B6A:0200  0A 00 03 00 02 00 02 00-01 00 02 00 DD D4 89 3E   ..............>
```

Изменим значение IP на 0100

```
-rip
IP 011C
:100
```

А теперь выполним программу в режиме пошаговой трассировки

```
-t
AX=000A  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0103   NV UP EI PL NZ NA PO NC
1B6A:0103 99            CWD
-t

AX=000A  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0104   NV UP EI PL NZ NA PO NC
1B6A:0104 F73E0202      IDIV    WORD PTR [0202]                 DS:0202=0003
-t

AX=0003  BX=0001  CX=0000  DX=0001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0108   NV UP EI PL NZ NA PO NC
1B6A:0108 89C3          MOV     BX,AX
-t

AX=0003  BX=0003  CX=0000  DX=0001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=010A   NV UP EI PL NZ NA PO NC
1B6A:010A A10402        MOV     AX,[0204]                   DS:0204=0002
-t

AX=0002  BX=0003  CX=0000  DX=0001  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=010D   NV UP EI PL NZ NA PO NC
1B6A:010D F72E0802      IMUL    WORD PTR [0208]                 DS:0208=0001
-t
```

```
AX=0002  BX=0003  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0111   NV UP EI PL NZ NA PO NC
1B6A:0111 29C3          SUB     BX,AX
-t

AX=0002  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0113   NV UP EI PL NZ NA PO NC
1B6A:0113 89D8          MOV     AX,BX
-t

AX=0001  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0115   NV UP EI PL NZ NA PO NC
1B6A:0115 F72E0602      IMUL    WORD PTR [0206]                 DS:0206=0002
-t

AX=0002  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0119   NV UP EI PL NZ NA PO NC
1B6A:0119 A30A02        MOV     [020A],AX                       DS:020A=0002
-t

AX=0002  BX=0001  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=011C   NV UP EI PL NZ NA PO NC
1B6A:011C CC            INT     3
-
```

2. Реализация выражения **Y:=** $\begin{cases} 0, X < 0 \\ X, 0 < X \le 1 \\ X^4, X > 1 \end{cases}$

Выполним команду дизассемблирования для просмотра текста программы, введенного ранее с помощью команды A.

```
-u100 121
1B6A:0100 A10002        MOV     AX,[0200]               занести в AX значение X из
DS:0200
1B6A:0103 3D0000        CMP     AX,0000         сравнить AX(X) с нулем
1B6A:0106 7C0A          JL      0112            Если X<0, то перейти к команде 112
1B6A:0108 3D0100        CMP     AX,0001         сравнить AX(X) с единицей
```

```
1B6A:010B 7F0D        JG      011A                если X>1, то перейти к команде 11A
1B6A:010D A30202      MOV     [0202],AX           занести результат в ячейку DS:202
1B6A:0110 EB0F        JMP     0121                перейти к команде 121
1B6A:0112 C70602020000  MOV   WORD PTR [0202],0000   занести результат (X<0)
1B6A:0118 EB07        JMP     0121                перейти к команде 121
1B6A:011A F7E8        IMUL    AX                  X:=X*X
1B6A:011C F7E8        IMUL    AX                  X:=X*X
1B6A:011E A30202      MOV     [0202],AX           занести результат по адресу DS:202
1B6A:0121 CC          INT     3                   выйти в отладчик
-
```

Проверка работы алгоритма для значения X=-1

```
-e200 ff ff 00 00
-d200 20f
1B6A:0200  FF FF 00 00 02 00 02 00-01 00 02 00 DD D4 89 3E   ................>
-g
AX=FFFF  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0121   NV UP EI NG NZ NA PE NC
1B6A:0121 CC          INT     3
-d200 20f
1B6A:0200  FF FF 00 00 02 00 02 00-01 00 02 00 DD D4 89 3E   ................>
-
```

Проверка работы алгоритма для значения X=1

```
-rip
IP 0121
:100
-e200 01 00 00 00
-g

AX=0001  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0121   NV UP EI PL ZR NA PE NC
1B6A:0121 CC          INT     3
-d200 20f
1B6A:0200  01 00 01 00 02 00 02 00-01 00 02 00 DD D4 89 3E   ................>
-
```

Проверка работы алгоритма для значения X=2

```
-rip
```

```
IP 0121
:100
-e200 02 00 00 00
-g

AX=0010  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0121   NV UP EI PL NZ NA PO NC
1B6A:0121 CC            INT     3
-d200 20f
1B6A:0200  02 00 10 00 02 00 02 00-01 00 02 00 DD D4 89 3E   ..............>
-
```

3. Реализация выражения **Y:=** $\sum_{I=1}^{N} I!$

```
C:\>debug
-a100
1B6A:0100 mov cx,word ptr[200]    занести значение N в CX (счетчик цикла)
1B6A:0104 mov bx,0                обнулить регистр BX для накопления суммы
1B6A:0107 push cx                 сохранить счетчик цикла в стеке
1B6A:0108 mov ax,1                занести в регистр AX 1 для вычисления I!
1B6A:010B imul cx                 выполнить очередное умножение при вычислении I!
1B6A:010D loop 10b                замкнуть цикл вычисления I!
1B6A:010F add bx,ax               добавить к сумме значение I!
1B6A:0111 pop cx                  восстановить CX из стека
1B6A:0112 loop 107                замкнуть внешний цикл вычисления суммы
1B6A:0114 mov word ptr[202],bx    занести результат в ячейку с адресом DS:202
1B6A:0118 int 3                   вернуться в отладчик
1B6A:0119
```

Для проверки работы алгоритма заносим в ячейки 200 и 202 значения N и Y (3 и 0)

```
-e200
1B6A:0200  03 00 00 00
-d200 203
1B6A:0200  03 00 00 00                          ....
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
```

```
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0100   NV UP EI PL NZ NA PO NC
1B6A:0100 8B0E0002      MOV     CX,[0200]                   DS:0200=0003
```

Запуск программы

```
-g

AX=0000  BX=0009  CX=0000  DX=0000  SP=001C  BP=0000  SI=0000  DI=0000
DS=1B6A  ES=1B6A  SS=1B6A  CS=1B6A  IP=0118   NV UP EI PL NZ NA PO NC
1B6A:0118 CC            INT     3
-d200 203
1B6A:0200  03 00 09 00                                ....
-
```

В результате выполнения в ячейке DS:202 получили результат 9

Варианты заданий к лабораторной работе

а) для реализации линейного выражения варианты заданий выбираются из табл. 1.

б) для реализации ветвящегося и циклического выражений варианты заданий выбираются из таблицы 2.

Тема: Программирование на языке ассемблера IBM PC и использование системы прерываний

Цель лабораторной работы:

Закрепление знакомства с системой команд Intel80x86. Ознакомление с общей структурой программы, директивами ассемблера IBM PC, использованием системы программных прерываний.

Задание к лабораторной работе:

Реализовать программу, используя систему команд и язык ассемблера IBM PC.

Указания к работе:

Для ввода данных в виде строки символов использовать прерывание DOS INT 21h функцию 0Ah, при этом:

на входе dx должен содержать адрес буфера, в который осуществляется ввод, регистр AH должен содержать код функции - 0Ah; первый байт буфера должен содержать максимальное число вводимых символов; после выхода из прерывания второй байт буфера содержит фактическое число введенных символов, исключая символ <ENTER>; начиная с третьего байта начинается введенная строка символов.

Пример:

Описание буфера в области описания данных

Разрешается ввести 10 символов

```
                |
buffer   db     0ah,0,11 dup(' ')
                          |__________  Резервирование места под строку
```

…

Работа с буфером в теле программы

…

```
  lea dx,buffer    ; загрузили в dx адрес буфера
  mov ah,0ah       ;  код функции - в регистр AH
  int 21h          ; обратились к 21 прерыванию для ввода строки
```

…

Для преобразования введенного в символьном виде числа во внутреннее представление можно воспользоваться подпрограммой STR2BIN. На вход подпрограмме необходимо передать в bx адрес буфера, содержащего число в символьном представлении, причем первый байт этого буфера - количество символов в числе (если для ввода строки использовалась функция 0Ah прерывания int 21h, то на вход надо подать адрес буфера+1).

При выходе из подпрограммы в регистре AX имеем число во внутреннем представлении, которое можно использовать в вычислениях.

Пример:

```
buffer   db    0ah,0,11 dup(' ')
         …
;        ввод строки через 21 прерывание
         lea dx,buffer   ; загрузка адреса буфера
         mov ah,0ah      ; загрузка в ah  кода функции ввода строки
         int 21h         ; обращение к 21 прерыванию
;        преобразование введенной строки символов в число
         mov bx,dx       ; перегрузить в bx адрес буфера
         inc bx          ; увеличить адрес на единицу
         call str2bin    ; обратиться к подпрограмме преобразования
;        после этого в ax - число во внутреннем представлении
         …
```

Для вывода результатов используются подпрограмма bin2str (для преобразования числа из внутреннего представления в строку символов) и функция 9 прерывания int 21h.

Пример :

```
n               dw      ?
number                  db      7 dup(“ “),”$”
        …
        mov     ax,n    ; занесли в ax число, подлежащее перекодировке
        lea     bx,number   ; занесли в bx адрес буфера для символьного представления
        call    bin2str     ; обратились к процедуре преобразования числа
; после выполнения процедуры bin2str в number имеем символьное представление числа
; теперь выполняем вывод числа на экран с использованием функции 09 прерывания DOS
; int 21h
        lea     dx,number+1 ; в dx - адрес буфера вывода
        mov     ah,09       ; в ah - номер функции
        int     21h         ; вывод на экран
        …
```

Трансляция может быть осуществлена с помощью турбоассемблера TASM, при этом командная строка выглядит следующим образом:

>tasm /l <имя_файла_типа .asm> <enter>

После отработки ассемблера при отсутствии ошибок получаем файл типа .obj, который подается на вход компоновщика (линкера). Компоновка осуществляется компоновщиком TLINK, при этом командная строка выглядит следующим образом:

>tlink /t <имя .obj файла><enter>

при этом, при отсутствии ошибок создается выполняемый com-файл.

Пример реализации программы

Ниже приведен пример реализации программы для выражения $Y:=\sum_{i=1}^{N} Z^I$

```
prg     segment para public 'code'
        assume cs:prg,ds:prg,es:prg,ss:prg
        org 100h
beg:    jmp start
messin  db  0ah,0dh,'Введите значение N$'
messout db  0dh,0ah,'Результат вычисления выражения :$'
N       dw  ?
Y       dw  ?
input_Buf db  06,00,5 dup(?)
answer  db  7 dup (?),'$'
include bin2str.asm
include str2bin.asm
; процедура вычисления факториала числа
; исходное число передается через регистр CX, результат возвращается через AX
fact    proc
```

```
        push cx
        mov ax,1
cyc:    imul cx
        loop cyc
        pop cx
        ret
fact    endp
; начало основной программы
start:  lea dx,messin  ; в DX - адрес сообщения messin
        mov ah,09h   ; в AH - номер функции вывода строки
        int 21h      ; вызов прерывания DOS для вывода строки на экран
        lea dx,input_buf       ; в dx - адрес буфера ввода
        mov ah,0ah             ; в AH - номер функции ввода числа с клавиатуры
        int 21h                ; вызов прерывания ввода числа с клавиатуры
; преобразование введенной строки символов в число
        mov bx,dx       ; перегрузить в bx адрес буфера
        inc bx          ; увеличить адрес на единицу
        call str2bin    ; обратиться к подпрограмме преобразования
; после этого в ax - число во внутреннем представлении
        mov N,ax      ; запомнить число в переменной N
        mov cx,N      ; поместить N в cx - счетчик повторений цикла
        mov bx,0      ; установить начальное значение суммы равным 0
cycl:   call fact     ; вычислить факториал очередного значения
        add bx,ax     ; добавить результат к значению суммы
        loop cycl     ; замкнуть цикл
; на выходе из цикла в bx получаем результат
; преобразуем его в текстовую форму и выведем на экран
        mov Y,bx      ; поместить результат в Y
        mov ax,bx     ; поместить результат в ax
        lea bx,answer ;  поместить в BX адрес буфера для символьного представления
        call bin2str    ; обратиться к подпрограмме преобразования
; собственно вывод на экран
        lea dx,messout         ; в dx - адрес буфера вывода
        mov ah,09h             ; в ah - номер функции вывода на экран
        int 21h                ; обратиться к функции вывода через 21 прерывание
        lea dx,answer+1
        mov ah,09h
        int 21h
        int 20h         ; а затем завершаем работу
prg     ends
        end beg
```

Далее приводятся тексты используемых процедур преобразования чисел

```
; процедура преобразования строки символов в двоичное число.
; вход - ds:bx   - адрес строки
; выход - ax - число со знаком
;
str2bin  proc
        push bx
        push cx
        sub ax,ax       ; обнулить ax
        sub ch,ch       ; очистить ch
        mov cl,byte ptr [bx]  ; счетчик символов - в CX
blanks: cmp byte ptr[bx+1],' '   ;цикл пропуска пробелов
```

```
        jne chk_neg
        inc bx
        loop blanks
        jmp  good       ; не обнаружено чисел. результат - 0
;
; Следующие команды выполняются в случае знака  "-"
;
chk_neg: cmp byte ptr [bx+1],'-'
        jne chk_pos
        inc bx              ; продвинуть указатель
        dec cx              ; уменьшить счетчик
        call conv_ab        ; вызвать программу преобразования
        jc thru             ;
        cmp ax,32768        ; число < -32768
        ja  no_good         ; да, ошибка
        neg ax              ; изменить знак числа
        js  good            ; o'key , выходим
;
; Следующие команды выполняются в случае, если знак не минус
;
chk_pos: cmp byte ptr [bx+1],'+'   ; sign '+' ?
        jne go_conv         ; нет , идти на преобразование
        inc bx              ; сместить указатель в строке
        dec cx              ; уменьшить счетчик символов
go_conv: call conv_ab       ; преобразовать строку в число
        jc thru             ; в случае ошибки - на выход
        cmp ax,32767        ; число не слишком велико ?
        ja no_good          ; слишком велико, ошибка
good:   clc                 ; очистить флаг CY
        jnc thru
no_good: stc                ; set carry flag
thru:   pop cx              ; восстановить значения регистров
        pop bx
        jmp skipit
;
; Процедура собственно преобразования числа
;
conv_ab proc
        push bp
        push bx
        push si
        mov  bp,bx      ; поместить указатель на строку в bp
        sub  bx,bx      ; очистить bx
range:  cmp byte ptr ds:[bp+1],'0'  ; символ - цифра ?
        jb  non_dig             ; нет не цифра
        cmp byte ptr ds:[bp+1],'9'  ; символ > '9'
        jbe digit               ; символ - цифра
non_dig: stc                    ; установить флаг ошибки
        jc end_conv             ; и выйти из процедуры
digit:  mov si,10
        push dx                 ; сохранить dx
        mul si                  ; (ax)*(si)->(dx;ax)
        pop dx                  ; восстановить dx
        mov bl,ds:[bp+1]        ; взять очередной символ из строки
```

```
        and bx,0fh              ; преобразовать символ в двоичное
        add ax,bx               ; добавить цифру к старому результату
        jc  end_conv             ; если результат слишком велик, то выйти
        inc bp                  ; адрес след цифры в строке
        loop range              ; замкнуть цикл по количеству символов
        clc                     ; сбросить флаг переноса(нет ошибки)
end_conv:pop si                 ; восстановить регистры и выйти
        pop bx
        pop bp
        ret
conv_ab  endp
skipit:  ret
str2bin  endp


; процедура преобразования двоичного числа в строку символов
; вход - ax - число;
; выход - строка по адресу ds:bx
bin2str proc
      push dx
        push cx
        push bx
        push si
        push ax
        mov cx,6
fill_buff:
        mov byte ptr [bx+1],' '    ; заполнить строку пробелами
        inc bx
        loop fill_buff
        mov  si,10              ; подготовиться к делению на 10
        or  ax,ax               ; если в (ax) отрицательное число
        jns clr_div
        neg ax                  ; то изменить его знак
clr_div:
        sub dx,dx               ; очистить старшую половину делимого
        div si                  ;(dx;ax)/10 -> (рез -> ax, ост->dx)
        add dx,'0'              ; преобразовать остаток в симв. вид
        dec bx
        mov [bx+1],dl           ; запомнить очередн. цифру в строке
        or  ax,ax               ; в регистре ax - 0?
        jnz clr_div             ; нет, повторить процедуру
        pop ax
        or ax,ax                ; исх.число отрицательно ?
        jns no_more             ; нет, тогда все
        dec bx
        mov byte ptr [bx+1],'-'    ; поместить в строку знак '-'
no_more:
        pop si
        pop bx
        mov byte ptr [bx],6             ; поместить счетчик символов
        pop cx
        pop dx
        ret
bin2str endp
```

Варианты заданий к лабораторной работе

Варианты заданий выбираются из таблицы 4.

ПРИЛОЖЕНИЕ 1

Функциональная модель ЭВМ-1

Функциональная модель ЭВМ представляет собой программу, моделирующую поведение некоторой гипотетической ЭВМ при ее функционировании. Функциональная модель имеет: оперативную память, состоящую из шестнадцати двенадцатиразрядных ячеек, блок регистров общего назначения, состоящий из двух регистров - 0 и 1, регистр команд, содержащий код текущей выполняемой команды, четырехразрядный регистр счетчика адреса команд, четырехразрядный регистр признаков (признак отрицательного результата, нулевого результата, положительного результата и признак переполнения), арифметико-логическое устройство с сумматором, канал ввода информации и канал вывода. В ходе функционирования модели ее состояние отображается на экране дисплея в виде специальной таблицы, внешний вид которой показан на рисунке, представленном ниже.

<table>
<tr><th>адрес</th><th>память</th><th colspan="5">регистры</th><th>ввод</th></tr>
<tr><td>0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111</td><td>000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000
000000000000</td><td colspan="3">0</td><td colspan="2">1</td><td></td></tr>
<tr><td colspan="3">000000000000</td><td colspan="2">000000000000</td><td>вывод</td></tr>
<tr><td colspan="3">рег. команд</td><td colspan="2">000000000000</td><td></td></tr>
<tr><td colspan="4">счетчик команд</td><td>0000</td><td rowspan="5">e -редактор
s -пошаговое выполнение
r -непрерывное выполнение
q - конец работы</td></tr>
<tr><td colspan="5">регистр признаков</td></tr>
<tr><td>n</td><td>z</td><td>p</td><td>v</td><td></td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td></td></tr>
<tr><td colspan="5">сумматор</td></tr>
<tr><td colspan="5">000000000000</td><td></td></tr>
</table>

Для управления работой модели используются четыре основных команды, мнемоническое обозначение и функциональное назначение которых показаны в нижнем правом углу таблицы. Первая команда - редактор - обеспечивает занесение и исправление информации в моделируемой памяти ЭВМ (состояние памяти отображается в двоичном виде). При входе в режим редактирования маркер устанавливается в старший бит нулевого слова моделируемой памяти. Перемещение маркера осуществляется с помощью клавиш управления курсором - HOME, UP, PGUP, LEFT, RIGHT, END, DOWN,

PGDOWN; ввод нуля и единицы - клавишами <0> и <1> соответственно. Нажатие клавиши <E> обеспечивает выход из режима редактирования содержимого памяти. Ввод каких-либо других символов в режиме редактирования блокируется.

В режиме пошагового выполнения модель осуществляет выполнение очередной команды, указываемой содержимым счетчика адреса команд, модифицирует состояние изменившихся полей и приостанавливает цикл функционирования до тех пор, пока на клавиатуре не будет нажата любая клавиша.

В режиме непрерывного выполнения каждая команда, записанная в памяти модели, выполняется с задержкой около трех секунд, так, чтобы можно было проследить, каким образом модифицируются поля в таблице, отображающей состояние модели.

Работа модели завершается при вводе команды Q, при этом запрашивается подтверждение на окончание работы.

Система команд модели

1. Модель оперирует с целыми числами, представленными в формате с фиксированной точкой в двенадцатиразрядной сетке. Разряд 11 является знаковым, разряды 0 ... 10 - информационые. Положительные числа представляются в прямом двоичном коде, отрицательные числа - в дополнительном двоичном коде. Диапазон представления целых чисел составляет -2048 ... 2047.

2. Любая команда модели ЭВМ занимает одно слово памяти, т.е., в данном случае, двенадцать разрядов, и имеет следующий формат:

код операции(4)	адрес 1-го операнда(4)	адрес 2-го операнда(4)

Здесь в скобках указана длина каждого поля команды в битах, таким образом, каждая команда состоит из трех четырехбитовых полей.

3. Ниже приводятся коды операций и правила кодирования операндов в командах. Все приводимые коды - шестнадцатеричные. Команды в приводимом ниже описании упорядочены в порядке возрастания кода операции.

3.1. "Останов"

Код операции - 0, в поле первого операнда всегда указывается 0, в поле второго операнда - всегда 0.

3.2. "Вывод в канал"

Код операции - 0, в поле первого операнда указывается 1 - адрес канала вывода, в поле второго операнда указывается адрес ячейки памяти, содержимое которой выводится в канал вывода.

3.3. "Ввод из канала ввода"

Код операции - 0, в поле первого операнда указывается 2 -адрес канала ввода, в поле второго операнда указывается адрес ячейки памяти, в которую помещается информация, вводимая через канал ввода.

3.4. "Загрузить регистр содержимым ячейки памяти"

Код операции - 1, первый операнд - адрес одного из двух имеющихся в модели регистров общего назначения (0 или 1), второй операнд - адрес ячейки памяти, из которой осуществляется загрузка содержимого. По данной команде содержимое указанной ячейки памяти копируется в указанный регистр.

3.5. "Запомнить содержимое регистра в памяти"

Код операции - 2, первый операнд - адрес одного из регистров общего назначения, второй операнд - адрес ячейки памяти. При выполнении команды содержимое указанного регистра копируется в ячейку памяти с указанным адресом.

3.6. Группа команд арифметики. Метод адресации - регистр-память.

Коды операций: "сложить" - 3, "вычесть" - 4, "умножить" - 5, "делить" - 6. В поле первого операнда указывается адрес одного из регистров общего назначения, содержащего первый операнд, в поле второго операнда указывается адрес ячейки памяти, содержащей второй операнд. Результат операции всегда располагается по месту первого операнда.

3.7. Группа команд арифметики. Метод адресации - память-память.

Коды операций: "сложить" - 7; "вычесть" - 8; "умножить" - 9; "делить" - A. В поле первого операнда указывается адрес ячейки памяти, содержащей первый операнд, в поле второго операнда адрес ячейки памяти, содержащей второй операнд. Результат операции располагается по месту первого операнда.

3.8. Группа команд переходов и организации цикла.

Код операции - B. Тип перехода определяется первым операндом, вторым операндом указывается адрес перехода. В зависимости от условий перехода первый операнд может кодироваться одним из следующих способов:

0 - безусловный переход, вне зависимости от содержимого регистра признаков;

1 - переход по переполнению (признак V =1);

2 - переход по положительному результату (признак P=1);

3 - переход по переполнению или положительному результату (V=1 или P=1);

4 - переход по нулевому результату (признак Z=1);

5 - переход по неотрицательному и неположительному результату (N <> 1 и P<>1);

6 - переход по положительному или нулевому результату (P=1 или Z=1);

7 - переход по неотрицательному результату (N <> 1);

8 - переход по отрицательному результату (признак N=1);

9 - переход по отрицательному результату или переполнению (V=1 или N=1);

A - переход по отрицательному или положительному результату (N=1 или P=1);

B - переход по ненулевому результату (признак Z не равен 1);

C - переход по отрицательному или нулевому результату (признак N=1 или Z=1);

D - переход по неположительному результату (признак P <> 1);

E - переход по отсутствию переполнения (признак V <> 1);

F - организация цикла со счетчиком в нулевом регистре.

Команда выполняется следующим образом:

- из содержимого нулевого регистра вычитается 1 и результат записывается в нулевой регистр, кроме того, в зависимости от результата вычитания устанавливается содержимое регистра признаков;

- проверяется состояние регистра признаков; если результат операции неотрицательный (т.е. признак N не равен 1), то в программный счетчик записывается адрес, заданный вторым операндом команды, иными словами, осуществляется переход по адресу, заданному вторым операндом команды, в противном случае выполняется команда, следующая за командой организации цикла.

3.9. Группа команд арифметики. Метод адресации - регистр-регистр.

Коды операций: "сложить" - C; "вычесть" - D; "умножить" - E; "делить" - F. Оба операнда располагаются в регистрах общего назначения. Результат располагается на месте первого операнда. В поле первого и второго операндов команды содержатся адреса регистров общего назначения, содержащих первый и второй операнды соответственно.

4. На содержимое сумматора и регистра признаков влияют все команды арифметики и пересылки из регистров в память и обратно.

5. При возникновении переполнения наряду с признаком переполнения V устанавливается признак отрицательного или положительного результата, причем результат операции усекается до 12 разрядов.

6. При необходимости можно принудительно прервать выполнение программы в модели (например, если программа "зациклилась"), для этого на клавиатуре необходимо нажать клавишу <I>.

Пример программирования модели

Y:= (A+B)/(C-D)*F

Для данного варианта задания программа в кодах модели примет вид:

адр.	Команда	комментарий
0000	0000 0010 1111	Ввести A в ячейку памяти с адресом F
0001	0000 0010 1110	Ввести B в ячейку памяти с адресом E
0010	0000 0010 1101	Ввести C в ячейку памяти с адресом D
0011	0000 0010 1100	Ввести D в ячейку памяти с адресом C
0100	0000 0010 1011	Ввести F в ячейку памяти с адресом B
0101	0111 1111 1110	Сложить содержимое ячеек F и E (A+B)
0110	1000 1101 1100	Из содержимого ячейки D вычесть содержимое ячейки C (C-D)
0111	1010 1111 1101	Содержимое ячейки с адресом F поделить на содержимое ячейки D ((A+B)/(C-D))
1000	1001 1111 1011	Содержимое ячейки с адресом F умножить на содержимое ячейки B ((A+B)/(C-D)*F)
1001	0000 0001 1111	Вывести результат в канал вывода
1010	0000 0000 0000	Останов
1011	0000 0000 0000	Ячейка для хранения F
1100	0000 0000 0000	Ячейка для хранения D
1101	0000 0000 0000	Ячейка для хранения C
1110	0000 0000 0000	Ячейка для хранения B
1111	0000 0000 0000	Ячейка для хранения A

Ассемблер функциональной модели ЭВМ-1

Для изучения общих принципов программирования на языке ассемблера, групп команд ассемблеров, и упрощения программирования функциональной модели ЭВМ-1, для нее разработана интегрированная среда, включающая упрощенный редактор текста, ассемблер и собственно моделирующую программу.

Ассемблер функциональной модели следует общим принципам построения лингвистических процессоров данного типа, и обеспечивает использование мнемонических обозначений команд, символических адресов, средств резервирования и описания полей памяти. Программа на ассемблере модели состоит из последовательности строк, в каждой из которых записывается одна команда ассемблера. Каждая команда ассемблера состоит из мнемокода операции и операндов. Каждая строка может иметь метку - идентификатор, состоящий не более чем из пяти символов, и отделяющийся от мнемокода операции символом ":" ("двоеточие"). На этапе ассемблирования любой метке соответствует некоторое значение программного счетчика, таким образом, метка может

рассматриваться как символический адрес, на который могут ссылаться другие команды программы. Кодирование операндов в командах осуществляется следующим образом:

- если операнд является регистром, то в поле операндов он кодируется номером регистра(0 или 1);
- если операндом является содержимое ячейки памяти, то в поле операндов записывается символический или абсолютный шестнадцатеричный адрес соответствующей ячейки памяти;
- если операнд является адресом ячейки памяти, то в поле операндов указывается либо символический, либо абсолютный шестнадцатеричный адрес памяти.

Ниже приведен список команд ассемблера функциональной модели с указанием мнемокодов и типов операндов, используемых в командах.

1. INP A - ввод слова из канала ввода и запись его по адресу A.

2. OUT A - вывод содержимого ячейки памяти с адресом A в канал вывода.

3. MOVR R,A - загрузка регистра R содержимым ячейки памяти с адресом A.

4. MOVM R,A - запись содержимого регистра R в ячейку памяти с адресом A.

5. ARM R,A - сложить содержимое регистра R с содержимым ячейки памяти с адресом A.

6. SRM R,A - из содержимого регистра R вычесть содержимое ячейки памяти с адресом A.

7. MRM R,A - перемножить содержимое регистра R и ячейки памяти с адресом A.

8. DRM R,A - поделить содержимое регистра R на содержимое ячейки памяти с адресом A.

9. ARR R1,R2 - сложить содержимое регистров R1 и R2.

10. SRR R1,R2 - из содержимого R1 вычесть содержимое R2.

11. MRR R1,R2 - содержимое регистра R1 умножить на содержимое регистра R2.

12. DRR R1,R2 - содержимое регистра R1 поделить на содержимое регистра R2.

13. AMM A1,A2 - сложить содержимое ячеек памяти с адресами A1 и A2.

14. SMM A1,A2 - из содержимого ячейки с адресом A1 вычесть содержимое ячейки памяти с адресом A2.

15. MMM A1,A2 - перемножить содержимое ячеек памяти с адресами A1 и A2.

16. DMM A1,A2 - содержимое ячейки памяти с адресом A1 поделить на содержимое ячейки памяти с адресом A2.

17. JMP A - безусловный переход по адресу A.

18. JV A - переход по переполнению по адресу A.

19. JP A - переход по положительному результату по адресу A.

20. JPV A - переход по переполнению или положительному результату по адресу A.

21. JZ A - переход по нулевому результату.

22. JZV A - переход по неотрицательному и неположительному результату.

23. JZP A - переход по положительному или нулевому результату.

24. JZPV A - переход по неотрицательному результату.

25. JN A - переход по отрицательному результату.

26. JNV A - переход по отрицательному результату или переполнению.

27. JNP A - переход по отрицательному или нулевому результату.

28. JNPV A - переход по ненулевому результату.

29. JNZ A - переход по отрицательному или нулевому результату.

30. JNZV A - переход по неположительному результату.

31. JNZP A - переход по непереполнению.

32. LOOP A - организация цикла со счетчиком в регистре 0.

33. STOP - останов.

Кроме того, в состав команд входит команда END, сообщающая ассемблеру о конце программы (строго говоря, это не команда, а псевдокоманда, поскольку в машинный код она не переводится, а служит для управления процессом трансляции), а также две директивы : директива определения содержимого ячейки памяти и директива резервирования ячейки памяти.

Директива определения ячейки памяти присваивает ячейке памяти символический адрес и обеспечивает занесение в эту ячейку значения, указанного в качестве параметра директивы. Формат директивы следующий:

[метка:] .DW параметр

Здесь параметр - шестнадцатеричное число не более чем из трех значащих цифр. Если число начинается с шестнадцатеричных цифр A, B,C, D, E или F, то перед первой значащей цифрой числа должен стоять 0.

Директива резервирования ячейки памяти только присваивает ячейке памяти символический адрес. Формат директивы следующий:

[метка:] .DS

При записи программ на ассемблере функциональной модели

следует придерживаться следующих правил:

- для записи меток, мнемокодов операций и символических адресов используются прописные и строчные буквы латинского алфавита и цифры от 0 до 9;
- присутствие двоеточия вслед за идентификатором метки является обязательным;
- первый операнд отделяется от мнемокода операции хотя бы одним пробелом, если операндов два, то второй операнд отделяется от первого запятой.

При обнаружении ошибок в тексте программы ассемблер при распечатке выдает в строке, в которой обнаружена ошибка, код ошибки, состоящий из одного символа. Коды ошибок и соответствующие ситуации приведены ниже:

T - слишком длинное имя, усекается до 5 символов;

O - слишком длинное шестнадцатеричное число, усекается до трех символов;

D - дублирующая метка;

R - неверное выражение для регистра;

U - неопознан мнемокод операции;

A - ошибка адресации;

I - ошибка в директиве;

S - синтаксическая ошибка в команде;

Q - ошибка в параметре директивы;

W - предупреждающий код, говорит о том, что в данной команде или директиве возможна ошибка (сомнительный синтаксис).

Работа в интегрированной среде модели достаточно проста и не требует знания каких либо специфических особенностей. При запуске программы на экран дисплея выдается главное меню, состоящее из четырех пунктов:

```
укажите режим работы :
e - редактирование
a - ассемблирование
m - выполнение
q - закончить работу
```

При выборе первого режима (редактирование) происходит выход в текстовый редактор, при этом экран дисплея принимает следующий вид:

```
Назначение клавиш:
F2-очистка экрана и буфера текста;
F3-вставка строки перед текущей;
F4-удаление текущей строки;
F5-вставка символа в позиции курсора
F6-удаление символа над курсором;
F10- выход из режима редактирования;
Home   -в начало строки;
Up     -на строку вверх;
PgUp   -на первую строку;
```

	Left -на позицию влево; Right -на позицию вправо; End -в конец строки; Down -на строку вниз; PgDown-на последнюю строку; <Backspace>- удалить символ перед курсором; <Cr> - в начало следующей строки;

Программа, набираемая на ассемблере функциональной модели, отображается в левой половине экрана, а в правой половине экрана постоянно присутствует текст подсказки, напоминающей назначение функциональных клавиш и клавиш управления курсором. При выходе из редактора текст программы сохраняется во внутреннем буфере, и при последующих входах в редактор восстанавливается на экране. При выборе в главном меню режима ассемблирования вызывается однопроходоый ассемблер, обеспечивающий анализ исходного текста программы и преобразование его в коды команд функциональной модели ЭВМ. Протокол трансляции выводится на экран дисплея в следующем виде:

адр.	Код	ошибка	исходный текст
0000	000100000110		MOVR 0,A
0001	000100010111		MOVR 1,B
0010	110000000001		ARR 0,1
0011	001000001000		MOVM 0,C
0100	000000011000		OUT C
0101	000000000000		STOP
0110	000000000101		A: .DW 5
0111	000000000110		B: .DW 6
1000	000000000000		C: .DS
1001	000000000000		END

не обнаружено ошибок
нет предупреждений
нажмите любую клавишу

Из представленного примера видно, что листинг протокола трансляции содержит четыре колонки: колонку адреса команды, колонку сгенерированного кода команды, колонку ошибок (которая, естественно, не является пустой только при наличии ошибок), и колонку исходного текста. Кроме того, в нижней части протокола трансляции выводятся сообщения о количестве ошибок и предупреждений, либо об их отсутствии, а также сообщения о фатальных ошибках ассемблера, если таковые имеют место.

При выборе в главном меню режима моделирования происходит вызов собственно моделирующей программы, при этом в моделируемую память загружаются коды ассемблированной программы.

Пример построения программы на ассемблере функциональной модели ЭВМ-1

Выражение **Y:=N!**

Программа, реализующая данное выражение, будет иметь вид:

```
        MOVR 0,N
        MOVR 1,ONE
CYCL:JZ      OU1
        MRR  1,0
        LOOP CYCL
OU1:  MOVM 1,FACT
OUT   FACT
        STOP
N:      .DW 5
ONE:  .DW 1
FACT:.DS
END
```

ПРИЛОЖЕНИЕ 2

Функциональная модель микроЭВМ-2

Так же, как и функциональная модель ЭВМ-1, функциональная модель микроЭВМ-2 представляет собой программу, моделирующую поведение некоторой гипотетической ЭВМ при выполнении программы, загруженной в оперативную память.

В отличие от функциональной модели ЭВМ-1, настоящая модель значительно полнее как по функциональному составу, так и с точки зрения системы команд. Настоящая функциональная модель имеет: оперативную память, состоящую из 256 восьмиразрядных ячеек, центральный процессор и восемь каналов ввода/вывода. В составе процессора имеются:

- Восьмиразрядный программный счетчик (PC);
- восьмиразрядный регистр команд;
- восьмиразрядный указатель стека (SP);
- регистр признаков (PS), содержащий признак отрицательного результата N, признак нулевого результата Z, признак положительного результата P, и признак переполнения V;
- арифметико-логическое устройство;
- блок регистров общего назначения, состоящий из двух восьмиразрядных регистров - регистра A и регистра B.

В ходе функционирования модели ее состояние отображается на экране дисплея в виде таблицы, внешний вид которой показан на рисунке, представленном ниже.

каналы ввода/вывода							
#0 : 00	#1 : 00	#2 : 00	#3 : 00	#4 : 00	#5 : 00	#6 : 00	#7 : 00

регистры		адр.	память	
A: 00	B: 00	00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	команды:
		10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	e -редак-
ук. стека: ff		20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	тор;
		30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	r -выпол-
PC : 00		40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	нение;
		50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	s-пошаго-
рег.команд: 00		60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	вое выпо-
		70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	лнение;
рег.признаков		80	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	l-загруз-
		90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ка файла;
N:0 Z:0 P:0 V:0		a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	u-выгруз-
		b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ка файла;
рег. алу :0000		c0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	c-очистка
		d0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	памяти;
		e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	q -конец;
		f0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Для управления работой модели используется семь команд: EDIT, RUN, STEP, LOAD, UNLOAD, CLEAR и QUIT.

Команда EDIT обеспечивает занесение информации в моделируемую память и ее корректировку (состояние памяти и регистров модели отображается и вводится в шестнадцатеричном виде). При входе в режим редактирования маркер устанавливается в старшую тетраду нулевого байта памяти. Перемещение маркера осуществляется с помощью управляющих клавиш, ввод шестнадцатеричных цифр - нажатием клавиш 0 ...9, A ... F. Ввод других символов блокируется. Выход из режима редактирования осуществляется нажатием клавиши ESC.

Команда RUN обеспечивает запуск программы, размещенной в моделируемой памяти, на выполнение, при этом в ходе дополнительного диалога запрашивается стартовый адрес программы и задержка при выполнении каждой команды (0 ... 3 сек). Выполнение программы можно прервать в любой момент времени, нажав на клавиатуре клавишу <E>.

Команда STEP обеспечивает запуск программы на выполнение в пошаговом режиме (после выполнения очередной команды процесс приостанавливается до нажатия любой клавиши на клавиатуре). Так же, как и при выполнении команды RUN, задается стартовый адрес программы. Аналогичным же образом в любой момент времени выполнение программы может быть прекращено путем нажатия клавиши <E>.

Команда LOAD обеспечивает загрузку моделируемой памяти содержимым файла, имя которого запрашивается в ходе дополнительного диалога. Файл должен содержать абсолютный двоичный код программы, состоящий из команд модели ЭВМ-2, или, иначе говоря, двоичный дамп памяти модели. Файл может быть подготовлен либо путем использования команды UNLOAD, либо посредством ассемблера функциональной модели, входящего в комплекс программ.

Команда UNLOAD обеспечивает запись содержимого памяти модели в файл, имя которого запрашивается в ходе дополнительного диалога. В дальнейшем содержимое файла может быть загружено в моделируемую память посредством команды LOAD, либо дизассемблировано с помощью входящего в состав комплекса программ функциональной модели дизассемблера.

Команда CLEAR дает возможность очистить содержимое памяти модели, и, наконец, команда QUIT завершает сеанс работы с моделью.

Следует заметить, что функциональная модель имеет моделируемую систему аппаратных и программных прерываний (восемь программных и восемь аппаратных), причем моделирование аппаратных прерываний в ходе выполнения моделируемой программы (т.е. после ввода команды RUN или STEP) может быть вызвано нажатием

клавиш <0> ... <7>. Соответствующие вектора моделируемых аппаратных прерываний расположены по адресам 08 ... 0F. По адресам 00 ... 07 могут быть расположены вектора программных прерываний.

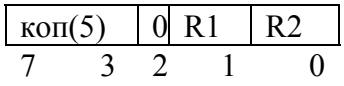
Система команд модели

1. Модель оперирует с целыми числами, представленными в формате с фиксированной точкой в восьмиразрядной сетке. Разряд 7 является знаковым, разряды 0 ... 6 - информационные. Положительные числа представляются в прямом двоичном коде, отрицательные - в дополнительном коде. Диапазон представления чисел составляет 0 ... 255 в беззнаковой форме и -128 ... 127 для чисел со знаком.

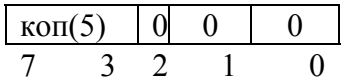
2. Система команд модели включает команды длиной 1 и 2 байта. В зависимости от функционального назначения и метода адресации команды могут иметь один из следующих форматов:

команды длиной в один байт

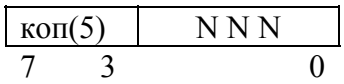
а)



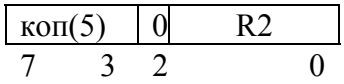
б)



в)



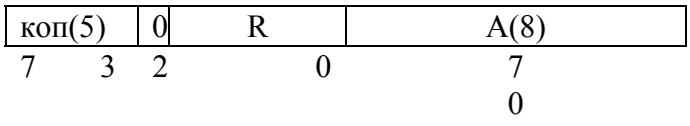
г)



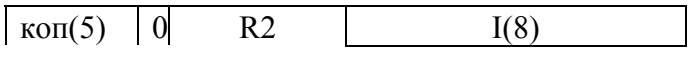
команды длиной 2 байта

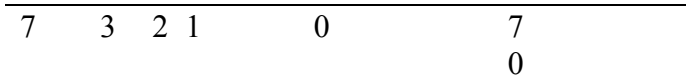
младший байт старший байт

д)

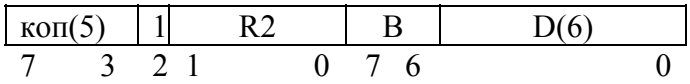


е)

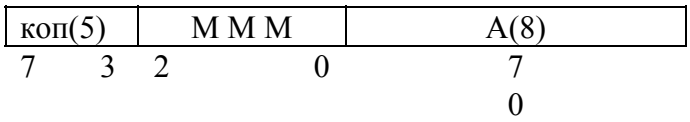




ж)



з)



Использованы следующие обозначения:

- КОП - код операции; цифра в скобках обозначает количество разрядов, занимаемых полем;
- R1, R2 - одноразрядный код регистра общего назначения; регистр A имеет код 0, регистр B имеет код 1;
- R - двухразрядный код регистра общего назначения или управляющего регистра; регистр A имеет код 00, регистр B имеет код 01, регистр указателя стека SP имеет код 10, регистр признаков имеет код 11;
- N N N - трехразрядное двоичное число; в командах обмена с портом ввода/вывода - адрес порта ввода/вывода; в командах организации программных прерываний - номер прерывания;
- A - абсолютный восьмиразрядный адрес памяти;
- I - восьмиразрядный непосредственный операнд;
- B - двухразрядный код базового или индексного регистра; в качестве базового или индексного регистра могут быть использованы регистры общего назначения A и B и регистр указателя стека SP;
- D - шестиразрядное смещение или базовый адрес (может принимать значение в диапазоне 0 ... 63);
- M M M - трехразрядный код условия в командах перехода.

Цифры под схемой формата обозначают номера битов в байте, причем под младшим понимается байт, размещенный по младшему адресу.

Система команд модели обеспечивает использование таких методов адресации как абсолютная, относительная или индексная (совпадают с точностью до обозначений), непосредственная, регистровая, косвенно-регистровая, стековая (частный случай косвенно-регистровой автоинкрементной/декрементной), косвенная.

3. Ниже описывается набор команд функциональной модели микроЭВМ-2. Для каждой команды указывается мнемокод, формат, шестнадцатеричный код операции и дается краткое описание выполняемой функции. При описании набора команд использованы те же обозначения, что и при описании форматов.

3.1 Пересылка регистров: MOV R1,R2. Формат а. Код операции - 1. Содержимое регистра R2 копируется в R1.

3.2. Загрузка регистра : LOAD R,A или LOAD R,D(B). Формат д или ж. Код операции -2. Содержимое ячейки памяти с абсолютным адресом A или относительным адресом (B)+D копируется в регистр R.

3.3. Запись регистра: STOR R,A или STOR R,D(B). Формат д или ж. Код операции - 3. Содержимое регистра R копируется в ячейку памяти по абсолютному адресу A или относительному адресу (B)+D.

3.4. Запись в стек: PUSH R. Формат г. Код операции - 4. Содержимое регистра R записывается в вершину стека, адресуемую регистром SP.

3.5. Восстановление из стека: POP R. Формат г. Код операции - 5. Содержимое вершины стека переписывается в регистр R.

3.6. Непосредственная загрузка: MVI R,I. Формат е. Код операции - 6. В регистр R загружается непосредственный операнд I.

3.7. Обмен регистров: XCHG. Формат б. Код операции - 7. Регистры A и B обмениваются содержимым.

3.8. Ввод из порта: IN N. Формат в. Код операции - 8. Содержимое порта с адресом N заносится в регистр A.

3.9. Вывод в порт: OUT N. Формат в. Код операции - 9. Содержимое регистра A выводится в порт с адресом N.

3.10. Сложение регистров: ADR R1,R2. Формат а. Код операции - A. Складывается содержимое регистров R1 и R2, результат помещается в R1.

3.11. Сложение регистра с памятью: ADM R,A или ADM R,D(B). Формат д или ж. Код операции - B. К содержимому регистра R добавляется содержимое ячейки памяти с адресом A или (B)+D, результат помещается в R.

3.12. Сложение непосредственное: ADI R,I. Формат е. Код операции - C. Содержимое регистра R складывается с непосредственным операндом, результат записывается в регистр R.

3.13. Вычитание регистров: SBR R1,R2. Формат а. Код операции - D. Из содержимого регистра R1 вычитается содержимое регистра R2, результат помещается в регистр R1.

3.14. Вычитание: SBM R,A или SBM R,D(B). Формат д или ж. Код операции - E. Из содержимого регистра R вычитается содержимое ячейки памяти с адресом A или (B)+D, результат помещается в регистр R.

3.15. Вычитание непосредственное: SBI R,I. Формат е. Код операции - F. Из содержимого регистра R вычитается непосредственный операнд I, результат помещается в регистр R.

3.16. Умножение регистров: MUR R1,R2. Формат а. Код операции - 10. Содержимое регистра R1+1 умножается на содержимое регистра R2, результат помещается в паре регистров - старшая часть - в R1, младшая - в R1+1. Поскольку в модели ЭВМ имеется только два регистра, то в качестве R1 всегда должен указываться регистр A, при этом первый сомножитель будет размещаться в регистре B, а результат - в паре регистров A и B.

3.17. Умножение: MUM R,A или MUM R,D(B). Формат д или ж. Код операции - 11. Содержимое регистра R+1 умножается на содержимое ячейки памяти с адресом A или (B)+D, результат располагается в паре регистров.

3.18. Умножение непосредственное: MUI R,I. Формат е. Код операции - 12. Содержимое регистра R+1 умножается на непосредственный операнд I, результат размещается в паре регистров.

3.19. Деление: DVM R,A или DVM R,D(B). Формат д или ж. Код операции - 13. Содержимое пары регистров - R и следующего за ним - делится на содержимое ячейки памяти с адресом A или (B)+D, результат размещается в паре регистров: в R - остаток, в R+1 - частное. Поскольку модель имеет всего два регистра, то в команде в качестве R всегда должен указываться регистр A, при этом делимое будет в паре регистров A и B, частное - в регистре B, остаток - в регистре A.

3.20. Деление непосредственное: DVI R,I. Формат е. Код операции - 14. Содержимое пары регистров делится на непосредственный операнд I.

3.21. Сравнение регистров: CMR R1,R2. Формат а. Код операции -15. Признак результата устанавливается в соответствии с операцией (R1)-(R2). Содержимое регистров не изменяется.

3.22. Сравнение непосредственное: CMI R,I. Формат е. Код операции - 16. Признак результата устанавливается в соответствии с операцией (R)-I. Содержимое регистра не изменяется.

3.23. Группа команд переходов. Формат з. Код операции - 17. В зависимости от трехразрядного кода условия возможны следующие переходы:

* безусловный переход JMP; код условия 000;
* переход по минусу JN; код условия 001;

* переход по нулю JZ; код условия 010;
* переход по плюсу JP; код условия 011;
* переход по переполнению JV; код условия 100;
* переход по не минусу JNN; код условия 101;
* переход по не нулю JNZ; код условия 110;
* переход по не плюсу JNP; код условия 111;

Проверяется состояние регистра признаков. Если бит, соответствующий условию перехода, установлен, в программный счетчик заносится адрес перехода.

3.24. Переход к подпрограмме CALL A или CALL D(B). Формат д или ж. Код операции - 18. Адрес следующей за CALL команды заносится в вершину стека, в программный счетчик заносится адрес подпрограммы.

3.25. Возврат из подпрограммы: RET. Формат б. Код операции - 19. Из вершины стека извлекается адрес возврата и записывается в программный счетчик.

3.26. Переход к программному прерыванию: INT N. Формат в. Код операции - 1A. В стек заносится содержимое регистра признаков и адрес следующей за INT команды. Содержимое ячейки памяти с адресом N (N лежит в диапазоне 0 ... 7) записывается в программный счетчик.

3.27. Возврат из прерывания: RIN. Формат б. Код операции 1B. Из вершины стека восстанавливается содержимое регистра признаков и программного счетчика.

3.28. Организация цикла: LOOP R,A или LOOP R,D(B). Формат д или ж. Код операции 1C. Из содержимого регистра R вычитается 1. Если результат операции не равен нулю, осуществляется переход по адресу A или (B)+D. Если результат операции равен нулю, выполняется команда, следующая за командой организации цикла.

3.29. Разрешение прерываний: EI. Формат б. Код операции - 1D. Разрешается обработка всех прерываний.

3.30. Запрещение прерываний: DI. Формат б. Код операции - 1E. Обработка аппаратных и программных прерываний запрещается. Программные прерывания игнорируются, аппаратные ставятся в очередь (длина очереди- 1 элемент) до тех пор, пока не будет разрешена обработка.

3.31. Стоп: STOP. Формат б. Код операции - 1F. Выполнение моделируемой программы прекращается.

3.32. Пустая операция: NOP. Формат б. Код операции - 0. Не выполняется никаких действий. Счетчик команд увеличивается на 1.

4. На содержимое регистра признаков влияют команды арифметики, сравнения и восстановления из стека регистра признаков. Остальные команды не влияют на признаки результата.

5. При возникновении переполнения наряду с признаком переполнения V устанавливается признак результата, причем при выполнении операций сложения, вычитания или умножения результат усекается до восьми разрядов только при пересылке его в регистр или память. Возникновение переполнения при делении приводит к усечению результата непосредственно в регистрах сумматора.

Ассемблер функциональной модели

Общая характеристика

Ассемблер функциональной модели обеспечивает преобразование программы, написанной в мнемокодах, в двоичный файл в кодах функциональной модели "интеллоподобной архитектуры". Созданный ассемблером двоичный файл может непосредственно загружаться в моделируемую память функциональной модели с целью последующего выполнения.

Дополнительными возможностями ассемблера являются следующие:

- синтаксический и частичный семантический контроль исходной программы;
- выдача совмещенного листинга исходного и объектного кода на экран дисплея в ходе ассемблирования;
- выдача совмещенного листинга в файл на диск с целью последующего анализа;
- построение таблицы перекрестных ссылок;
- генерирование развернутых сообщений об ошибках и сомнительных ситуациях в совмещенном листинге.

Ассемблер функциональной модели построен по однопроходовой схеме с динамическим формированием таблицы перекрестных ссылок и листинга и фиксацией в памяти для разрешения ссылок. Для работы ассемблера необходима следующая минимальная конфигурация аппаратных средств:

- ПЭВМ IBM PC XT/AT;
- минимум 128 K оперативной памяти;
- один накопитель на гибком магнитном диске;
- дисплейный адаптер CGA и выше.

Минимальный объем свободной оперативной памяти зависит от объема ассемблируемой программы и для типовой учебной задачи средней сложности составляет около 50 K.

Общий синтаксис программы

Программа на ассемблере представляет собой последовательность строк, каждая из которых имеет следующий вид:

[МЕТКА:]ОПЕРАЦИЯ [ОПЕРАНДЫ][;КОММЕНТАРИЙ]

или

; КОММЕНТАРИЙ

МЕТКА является идентификатором, состоящим не более чем из восьми символов. В качестве символов могут быть использованы буквы латинского алфавита и цифры 0...9, причем первым символом метки обязательно должна быть буква.

ОПЕРАЦИЯ является мнемокодом машинной инструкции или директивы ассемблера. Операция отделяется от поля метки (если таковая существует) символом ":", который является обязательным.

Мнемокоды инструкций и директивы ассемблера описаны ниже.

ОПЕРАНДЫ являются символическими операндами инструкций или директив ассемблера. В качестве операндов, в зависимости от типа инструкции или директивы ассемблера, могут быть использованы имена (символические адреса), десятичные и шестнадцатеричные числа, имена регистров процессора модели (**A,B** для регистров общего назначения **A** и **B, SP** для регистра указателя стека, **PS** для регистра признаков процессора), индексированные имена (т.е. конструкция вида **ИМЯ(ИМЯ РЕГИСТРА)**), индексированные выражения (т.е. конструкция вида **ЧИСЛО (ИМЯ РЕГИСТРА)**). Операнды отделяются от поля операции хотя бы одним пробелом, а между собой - символом "," (запятой).

КОММЕНТАРИЙ представляет собой любую последовательность символов, предваряемую символом "; " отделяющимся от поля операндов или операции хотя бы одним пробелом.

Описание мнемокодов машинных инструкций

Ниже описывается набор мнемонических инструкций ассемблера функциональной модели.

1.Пересылка регистров: **MOV R1,R2**

Содержимое регистра R2 копируется в R1.

2.Загрузка регистра : **LOAD R,A** или **LOAD R,D(B)**

Содержимое ячейки памяти с абсолютным адресом A или относительным адресом (B)+D копируется в регистр R.

3.Запись регистра: **STOR R,A** или **STOR R,D(B).**

Содержимое регистра R копируется в ячейку памяти по абсолютному адресу A или относительному адресу (B)+D.

4.Запись в стек: **PUSH R**

Содержимое регистра R записывается в вершину стека, адресуемую регистром SP.

5.Восстановление из стека: **POP R**

Содержимое вершины стека переписывается в регистр R.

6.Непосредственная загрузка: **MVI R,I**

В регистр R загружается непосредственный операнд I.

7.Обмен регистров: **XCHG**

Регистры A и B обмениваются содержимым.

8.Ввод из порта: **IN N**

Содержимое порта с адресом N заносится в регистр A.

9.Вывод в порт: **OUT N**

Содержимое регистра A выводится в порт с адресом N.

10.Сложение регистров: **ADR R1,R2**

Складывается содержимое регистров R1 и R2, результат помещается в R1.

11.Сложение регистра с памятью: **ADM R,A** или **ADM R,D(B)**

К содержимому регистра R добавляется содержимое ячейки памяти с адресом A или (B)+D, результат помещается в R.

12.Сложение непосредственное: **ADI R,I**

Содержимое регистра R складывается с непосредственным операндом, результат записывается в регистр R.

13.Вычитание регистров: **SBR R1,R2**

Из содержимого регистра R1 вычитается содержимое регистра R2, результат помещается в регистр R1.

14.Вычитание: **SBM R,A** или **SBM R,D(B)**

Из содержимого регистра R вычитается содержимое ячейки памяти с адресом A или (B)+D, результат помещается в регистр R.

15.Вычитание непосредственное: **SBI R,I**

Из содержимого регистра R вычитается непосредственный операнд I, результат помещается в регистр R.

16.Умножение регистров: **MUR R1,R2**

Содержимое регистра R1+1 умножается на содержимое регистра R2, результат помещается в паре регистров - старшая часть - в R1, младшая - в R1+1. Поскольку в модели ЭВМ имеется только два регистра, то в качестве R1 всегда должен указываться регистр A, при этом первый сомножитель будет размещаться в регистре B, а результат - в паре регистров A и B.

17.Умножение: **MUM R,A** или **MUM R,D(B)**

Содержимое регистра R+1 умножается на содержимое ячейки памяти с адресом A или (B)+D, результат располагается в паре регистров A и B.

18.Умножение непосредственное: **MUI R,I**

Содержимое регистра R+1 умножается на непосредственный операнд I, результат размещается в паре регистров.

19.Деление: **DVM R,A** или **DVM R,D(B)**

Содержимое пары регистров - R и следующего за ним - делится на содержимое ячейки памяти с адресом A или (B)+D, результат размещается в паре регистров: в R - остаток, в R+1 - частное. Поскольку модель имеет всего два регистра, то в команде в качестве R всегда должен указываться регистр A, при этом делимое будет в паре регистров A и B, частное - в регистре B, остаток - в регистре A.

20.Деление непосредственное: **DVI R,I**

Содержимое пары регистров делится на непосредственный операнд I.

21.Сравнение регистров: **CMR R1,R2**

Признак результата устанавливается в соответствии с операцией (R1)-(R2). Содержимое регистров не изменяется.

22.Сравнение непосредственное: **CMI R,I**

Признак результата устанавливается в соответствии с операцией (R)-I. Содержимое регистра не изменяется.

23.Группа команд переходов: **Jxx A**

Возможны следующие переходы:

* безусловный переход **JMP**;
* переход по минусу **JN**;
* переход по нулю **JZ**;
* переход по плюсу **JP**;
* переход по переполнению **JV**;
* переход по не минусу **JNN**;
* переход по не нулю **JNZ**;
* переход по не плюсу **JNP**;

Проверяется состояние регистра признаков. Если бит, соответствующий условию перехода, установлен, в программный счетчик заносится адрес перехода. A

24.Переход к подпрограмме: **CALL A** или **CALL D(B)**

Адрес следующей за CALL команды заносится в вершину стека, в программный счетчик заносится адрес подпрограммы.

25.Возврат из подпрограммы: **RET**

Из вершины стека извлекается адрес возврата и записывается в программный счетчик.

26.Переход к программному прерыванию: **INT N**

В стек заносится содержимое регистра признаков и адрес следующей за INT команды. Содержимое ячейки памяти с адресом N (N лежит в диапазоне 0 ... 7) записывается в программный счетчик.

27.Возврат из прерывания: **RIN**

Из вершины стека восстанавливается содержимое регистра признаков и программного счетчика.

28.Организация цикла: **LOOP R,A** или **LOOP R,D(B)**

Из содержимого регистра R вычитается 1. Если результат операции не равен нулю, осуществляется переход по адресу A или (B)+D. Если результат операции равен нулю, выполняется команда, следующая за командой организации цикла.

29.Разрешение прерываний: **EI**

Разрешается обработка всех прерываний.

30.Запрещение прерываний: **DI**

Обработка аппаратных и программных прерываний запрещается. Программные прерывания игнорируются, аппаратные ставятся в очередь (длина очереди - 1 элемент) до тех пор, пока не будет разрешена обработка.

31.Стоп: **STOP**

Выполнение моделируемой программы прекращается.

32.Пустая операция: **NOP**

Не выполняется никаких действий. Счетчик команд увеличивается на 1.

Описание директив ассемблера

Наряду с мнемоническими инструкциями процессора функциональной модели строка команды в программе может содержать директивы ассемблера. Общий формат директив следующий:

.ИМЯ ДИРЕКТИВЫ [ОПЕРАНДЫ]

Знак "**.**" является обязательным и должен непосредственно предшествовать имени директивы.

Ассемблер функциональной модели распознает семь директив, описанных ниже.

1. Установить адрес размещения в памяти:

.ORG A

Директива предписывает ассемблеру установить значение счетчика адреса команд равным абсолютному адресу A. Все следующие за директивой команды будут транслироваться начиная с указанного адреса.

2. Режим десятичных чисел:

.DEC

Все числа, используемые ниже в операндах инструкций и директив, трактуются как десятичные.

3. Режим шестнадцатеричных чисел:

.HEX

Все числа, используемые ниже в операндах инструкций и директив, трактуются как шестнадцатеричные.

Режим включен по умолчанию.

4. Назначить базовый регистр:

.BASE R

Включает режим относительной адресации начиная с текущей точки программы и назначает указанный регистр базовым. При включенном режиме относительной адресации для каждого операнда памяти ассемблер строит ссылку в виде D(B), где D - смещение от базовой точки до операнда, B - базовый регистр. В качестве базового могут быть использованы регистры A, B и указатель стека SP. Ассемблер не обеспечивает загрузку содержимого базового регистра - позаботиться об этом должен сам программист. Отменить назначение базового регистра и выключить режим относительной адресации можно с помощью директивы .BASE с операндом NONE. По умолчанию действует режим .BASE NONE.

5. Резервировать память:

.DS N

Резервирует N байтов памяти и записывает в них 0.

6. Резервировать память и задать значения:

.DB НАБОР ЗНАЧЕНИЙ

где набор значений - последовательность чисел, разделенных запятыми.

Резервирует последовательность байтов памяти и записывает в них значения, указанные в наборе значений.

7. Завершить ассемблирование:

.END

Предписывает ассемблеру завершить трансляцию программы.

Сообщения об ошибках

Ниже приведены сообщения ассемблера об ошибках и возможные причины их возникновения.

1.Группа сообщений о серьезных ошибках, при которых код программы не генерируется.

"Дублирующее имя "

Повторное использование имени в поле метки.

"Не определено поле операции"

Неверная запись мнемокода операции или директивы.

"Нет мнемокода операции"

Отсутствует или неверен мнемокод операции или директивы.

"Регистровый операнд не найден"

Отсутствует или неверно задан операнд типа РЕГИСТР.

"Конец строки обнаружен до конца команды"

Отсутствует обязательный операнд команды.

"Ошибка во втором операнде"

Неверно задан второй операнд команды для инструкций типа РЕГИСТР-РЕГИСТР, РЕГИСТР-ПАМЯТЬ, РЕГИСТР-ИНДЕКСИРУЕМАЯ ПАМЯТЬ.

"Числовой операнд не найден"

Отсутствует числовой операнд в командах обмена с портами ввода вывода или в командах организации программных прерываний.

"Слишком большое число "

Значение числового операнда превышает допустимый предел (в командах обмена с портами и организации прерываний число должно быть в диапазоне 0...7, непосредственный операнд и адрес не могут превышать 255).

"Тип операнда не согласуется с командой"

Попытка использования операнда типа АДРЕС или НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД в команде типа РЕГИСТР-РЕГИСТР, использование операнда типа РЕГИСТР в команде типа РЕГИСТР-ПАМЯТЬ или РЕГИСТР-НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД и др.

"Операнд-адрес не найден"

Отсутствует операнд типа АДРЕС в команде типа РЕГИСТР-ПАМЯТЬ.

"Адрес перехода не найден"

Отсутствует или неверно задан операнд типа АДРЕС ПЕРЕХОДА в командах перехода и организации цикла.

"Числовой операнд директивы не найден"

Отсутствует обязательный числовой операнд в директивах .ORG, .DS или .DB .

"Тип операнда не согласуется с директивой"

В директиве указан неподходящий по типу операнд.

"Операнд не является регистром"

В команде типа РЕГИСТР-РЕГИСТР неверно указано имя регистра.

"Неподходящий операнд в директиве"

В директиве указан неподходящий по типу операнд.

"Операнд директивы не найден"

В директиве отсутствует обязательный операнд.

"Неизвестная директива"

Неверно указано имя директивы.

"Регистр не согласуется с командой"

В команде указан регистр, недопустимый для данной команды (обычно это регистр SP или PS).

"Слишком длинное смещение"

Смещение, явно или неявно заданное в команде типа РЕГИСТР-ИНДЕКСИРУЕМАЯ ПАМЯТЬ превышает по абсолютной величине 63.

"Индекс не является регистровым выражением"

Индекс, указанный в скобках в команде типа РЕГИСТР-ИНДЕКСИРУЕМАЯ ПАМЯТЬ, не является именем регистра.

2. Группа предупреждающих сообщений, при которых код программы генерируется, но, возможно, с ошибками.

"Операнд директивы не найден. Полагается NONE "

Отсутствует операнд в директиве .BASE .

"Длинное имя. Усечено."

Длина имени превышает 8 символов. Имя усекается до 8 символов.

"Слишком длинное десятичное число. Усечено"

Величина десятичного числа превышает 255. Усекается во внутреннем представлении до 8 двоичных разрядов.

"Слишком длинное шестнадцатеричное число. Усечено"

Длина шестнадцатеричного числа превышает два разряда. Число усекается до двух разрядов.

3. Группа сообщений о фатальных ошибках, прекращающих нормальную работу ассемблера.

"Нет доступной памяти. Выполнение прекращено"

При очередном запросе памяти под таблицу символов или листинг запрос не удовлетворен. Работа ассемблера аварийно завершается.

"Нет доступной памяти. Полный листинг не сгенерирован"

При очередном запросе памяти под листинг запрос операционной системой не удовлетворен. В данном случае, если полностью выполнен первый проход и нет ошибок, код может быть сгенерирован, но совмещенный листинг выведен не будет.

"Ошибка открытия файла листинга"

Ошибка открытия файла листинга. Возможно, нет места на диске.

"Ошибка закрытия файла листинга"

Возможно, произошел сбой при вводе/выводе.

“Ошибка при выводе кода в файл”

Возможно, нет места на диске.

Пример реализации программы на ассемблере модели

Данная программа обеспечивает поэлементное суммирование двух массивов - A и B и размещение результатов в третьем массиве - C.

Совмещенный листинг ассемблера выглядит следующим образом:

```
 1 00 40        IN 0        ; ввод количества элементов массивов
 2 01 18 2e     STOR A,N    ; запомнить количество элементов по адресу n
 3 03 31 2f     MVI B,A     ; загрузить в pB адрес массива A
 4 05 20   geta:PUSH A      ; сохранить в стеке счетчик цикла
 5 06 41        IN 1        ; ввести очередной элемент массива A
 6 07 1c 40     STOR A,00(B) ; запомнить его по адресу (b)+0
 7 09 61 01     ADI B,01    ; вычислить адрес следующего элемента
 8 0b 28        POP A       ; восстановить из стека счетчик цикла
 9 0c e0 05     LOOP A,geta  ; замкнуть цикл по метке geta
10 0e 31 39     MVI B,B     ; загрузить в pB адрес массива B
11 10 10 2e     LOAD A,N    ; загрузить в регистр A количество элементов
```

```
12 12 20    getb:PUSH A       ; сохранить в стеке счетчик цикла
13 13 42         IN 2         ; ввести очередной элемент массива B
14 14 1c 40      STOR A,00(B) ; запомнить его по адресу (b)+0
15 16 61 01      ADI B,01     ; вычислить адрес следующего элемента
16 18 28         POP A        ; восстановить из стека счетчик цикла
17 19 e0 12      LOOP A,getb  ; замкнуть цикл по метке getb
18 1b 10 2e      LOAD A,N     ; загрузить в pA количество элементов
19 1d 20         PUSH A       ; сохранить содержимое pA в стеке
20 1e 30 2f      MVI  A,A     ; загрузить в pA адрес массива a
21 20 20         PUSH A       ; сохранить адрес в стеке
22 21 30 39      MVI  A,B     ; загрузить в pA адрес массива b
23 23 20         PUSH A       ; сохранить адрес в стеке
24 24 30 43      MVI A,C      ; загрузить в pA адрес массива c
25 26 20         PUSH A       ; сохранить адрес в стеке
26 27 c0 60      CALL sum     ; вызвать подпрограмму суммирования
27 29 28         POP A        ; очистить  стек
28 2a 28         POP A
29 2b 28         POP A
30 2c 28         POP A
31 2d f8         STOP         ; останов
32 2e 00    N:   .DS 1    ; резервирование 1 байта под количество элементов
33 2f 00 00 A:   .ds 0a   ; резервирование 10 байтов под массив A
    00 00
    00 00
    00 00
    00 00
34 39 00 00 B:   .ds 0a   ; резервирование 10 байтов под массив B
    00 00
    00 00
    00 00
    00 00
35 43 00 00 C:   .ds 0a   ; резервирование 10 байтов под массив C
    00 00
    00 00
    00 00
    00 00
```

```
36 4d 00    ;
37 4d 00        .org 60      ; установить абсолютный адрес 60h
38 60 00    ;    подпрограмма   суммирования двух массивов
39 60 15 82 sum: LOAD B,02(SP) ; загрузить в pB адрес массива C
40 62 19 8f      STOR B,ADC    ; запомнить адрес по адресу ADC
41 64 15 83      LOAD B,03(SP) ; загрузить в pB адрес массива B
42 66 19 8e      STOR B,ADB    ; запомнить адрес по адресу ADB
43 68 15 84      LOAD B,04(SP) ; загрузить в pB адрес массива A
44 6a 19 8d      STOR B,ADA    ; запомнить адрес по адресу ADA
45 6c 14 85      LOAD A,05(SP) ; загрузить в pA количество элементов
46 6e 20    cycl:PUSH A        ; сохранить счетчик цикла в стеке
47 6f 10 8d      LOAD A,ADA    ; загрузить в pA адрес элемента массива A
48 71 20         PUSH A        ; сохранить адрес в стеке
49 72 11 8e      LOAD B,ADB    ; загрузить в pB адрес элемента массива B
50 74 14 00      LOAD A,00(A)  ; загрузить в pA очередной элемент массива A
51 76 5c 40      ADM A,00(B)   ; сложить элементы массивов A и B
52 78 61 01      ADI B,01  ; вычислить адрес следующего элемента массива B
53 7a 19 8e      STOR B,ADB    ; запомнить его
54 7c 11 8f      LOAD B,ADC    ; загрузить в pB адрес элемента массива C
55 7e 1c 40      STOR A,00(B)  ; запомнить сумму A(I)+B(I) в C(I)
56 80 61 01      ADI B,01  ; вычислить адрес следующего элемента массива C
57 82 19 8f      STOR B,ADC    ; запомнить его
58 84 28         POP A      ; извлечь из стека адрес очередного элемента A
59 85 60 01      ADI A,01  ; вычислить адрес следующего элемента массива A
60 87 18 8d      STOR A,ADA    ; запомнить его
61 89 28         POP A         ; извлечь из стека счетчик цикла
62 8a e0 6e      LOOP A,cycl   ; замкнуть цикл по количеству элементов
63 8c c8         RET           ; вернуться в вызывающую программу
64 8d 00    ADA:      .ds 1    ; резервирование
65 8e 00    ADB:      .ds 1    ; ячеек
66 8f 00    ADC:      .ds 1    ; под адреса массивов
```

***** Таблица перекрестных ссылок *****

имя	определение	ссылки
SUM	60	28
N	2e	1c 11 2

```
GETB    12              1a
GETA    5               d
CYCL    6e              8b
C       43              25
B       39              22  f
ADC     8f              83  7d  63
ADB     8e              7b  73  67
ADA     8d              88  70  6b
A       2f              1f  4
```

**** Не обнаружено ошибок. Успешное завершение ****

Дизассемблер функциональной модели

Дизассемблер функциональной модели предназначен для восстановления ассемблерного текста программы из программы в кодах функциональной модели ЭВМ. Реассемблированный текст по формату совместим с ассемблером функциональной модели ЭВМ, т.е. дизассемблированный текст может быть напрямую оттранслирован в код без дополнительного редактирования.

Например, исходный текст программы на ассемблере функциональной модели выглядит следующим образом:

```
;**** y:=(a/b - c*f)* e

        in 0            ; ввод a
        stor a,A        ; запомнить a по символическому адресу A
        in 1            ; ввод b
        stor a,B        ; запомнить b по символическому адресу B
        in 2            ; ввод c
        stor a,C        ; запомнить c по символическому адресу C
        in 3            ;  ввод f
        stor a,F        ; запомнить f по символическому адресу F
        in 4            ; ввод e
        stor a,E        ; Запомнить e по симв. адресу E
        load b,A        ; загрузить в RB A
        mui  A,1        ; расширить знак
        dvm  a,B        ;(rA,rB)/B-> частное в rB, остаток в rA
```

```
        stor b,tmp      ; запомнить результат деления по симв.адр.TMP
        load b,C        ; загрузить в rB C
        mum  a,F        ; (rB)*(F)-> (rA,rB)
        load a,tmp      ; в rA загрузить TMP
        sbr  a,b        ; вычесть из (rA) (rB) (a/b - c*f)
        mov  b,a        ; скопировать результат в rB
        mum  a,E        ; (rB)*(E) -> (rA,rB)
        mov  a,b        ; скопировать результат в a
        out  5          ; вывести рез-т в 5 порт
        stop            ; стоп
A:      .ds 1           ; ячейка памяти для A
B:      .ds 1           ; --//--    B
C:      .ds 1           ; --//-     C
F:      .ds 1           ; --//      F
E:      .ds 1           ; --//--    E
TMP:    .ds 1
        .end
```

Код программы (в шестнадцатеричном представлении) , будет иметь вид:

```
адрес   данные
00      40 18 24 41 18 25 42 18 26 43 18 27 44 18 28 11
10      24 90 01 98 25 19 29 11 26 88 27 10 29 69 0a 88
20      28 09 4d f8
```

Дизассемблированный текст программы в данном случае будет выглядеть следующим образом:

```
00   40        IN 0
01   18 24     STOR A,24
03   41        IN 1
04   18 25     STOR A,25
06   42        IN 2
07   18 26     STOR A,26
09   43        IN 3
0a   18 27     STOR A,27
0c   44        IN 4
0d   18 28     STOR A,28
0f   11 24     LOAD B,24
```

```
11   90 01    MUI A,01
13   98 25    DVM A,25
15   19 29    STOR B,29
17   11 26    LOAD B,26
19   88 27    MUM A,27
1b   10 29    LOAD A,29
1d   69       SBR A,B
1e   0a       MOV B,A
1f   88 28    MUM A,28
21   09       MOV A,B
22   4d       OUT 5
23   f8       STOP
24   00       NOP
25   00       NOP
26   00       NOP
27   00       NOP
28   00       NOP
```

ПРИЛОЖЕНИЕ 3

Функции BIOS

В табл.П3.1 приведен перечень прерываний, используемых для активизации некоторых функций BIOS.

Таблица П3.1

Шестнадцатеричный код	Назначение
10	операции ввода/вывода на экран видеомонитора
11	определение конфигурации оборудования
12	определение объема оперативной памяти
13	обмен с дисками
14	организация обмена по последовательным портам
15	ввод/вывод для накопителя на магнитной ленте кассетного типа (устаревшая), работа с XMS памятью
16	ввод с клавиатуры
17	обслуживание принтера
18	инициализация системы кассетного БЕЙСИКа
19	загрузка операционной системы
1A	функции даты и времени
1B	обработчик прерывания Ctrl-Break
1C	"заглушка" для модификации прерывания по таймеру
1D	адрес таблицы видеопараметров
1E	адрес таблица параметров дискеты
1F	адрес второй половины таблицы знакогенератора для графического режима

Код функции передается в регистре AH. Для передачи параметров в обоих направлениях используются регистры микропроцессора, причем обеспечивается сохранение всех регистров, не используемых для передачи параметров. Ниже, в качестве примера рассмотрен ряд прерываний BIOS и условия их выполнения.

1. Ввод с клавиатуры (INT 16H)

Это прерывание обеспечивает ввод с клавиатуры. В регистр AX считывается код сканирования клавиши и код символа. Режимы обработки задаются в AH.

Вход:

(AH)=0 считать следующий символ с клавиатуры. Вернуть код символа в AL, код сканирования - в AH

(AH)=1 установить ZF, если символ введен:

(ZF)=1 -символа нет

(ZF)=0 -символ введен (AX)

(AH)=2 - возврат текущего состояния клавиатуры

Выход: AL, AH, ZF

Все регистры сохраняются, кроме AX и флагов.

2. Видеосервис (INT 10H)

Это прерывание обеспечивает интерфейс видеорежимов. В данном случае, в качестве примеров описаны простейшие функции, поддерживаемые самым простым типом цветного графического видеоадаптера - CGA.

(AH)=0 установка режима. (AL) содержит значение режима.

Символьные режимы:

(AL)=0 40*25 Черно/белый,

(AL)=1 40*25 Цветной,

(AL)=2 80*25 Черно/белый,

(AL)=3 80*25 Цветной.

Графические режимы:

(AL)=4 320*200 Цветной,

(AL)=5 320*200 Черно/белый,

(AL)=6 640*200 Черно/белый,

(AH)=1 установка размера курсора.

Вход:

(CH) - начальная строка растра;

(CL) - конечная строка растра;

Выход: нет

(AH)=2 установка позиции курсора.

Вход:

(DH,DL)= Строка, колонка. (0,0) - левый верхний угол,

(BH)= Номер страницы (0 - для графического режима).

Выход: нет

(AH)=3 чтение позиции курсора.

Вход:

(BH)= номер страницы (0 - для графического режима).

Выход:

(DH,DL)= строка, колонка текущей позиции курсора.

(AH)=4 считать позицию светового пера.

(AH)=5 выбор активной страницы (только для текстовых режимов).

Вход:

(AL)=значение новой страницы (0-7 для режимов 0 и 1, 0-3 для режимов 2 и 3).

(AH)=6 прокрутка активной страницы вверх.

Вход:

(AL)= Число строк прокрутки (освобождающиеся строки внизу окна заполняются пробелами), AL=0 очистка всего окна;

(CH,CL)= строка, колонка верхнего левого угла окна;

(DH,DL)= Строка, колонка правого нижнего угла окна;

(BH)= атрибут для строк заполнения окна.

Выход: нет

(AH)=7 Прокрутка активной страницы вниз.

Вход:

(AL) = Число строк прокрутки (освобождающиеся строки вверху окна заполняются пробелами), AL = 0 Очистка всего окна (пробелами);

(CH,CL)= Строка, колонка верхнего левого угла окна;

(DH,DL)= Строка, колонка нижнего правого угла окна;

(BH)= атрибут для строк заполнения.

Выход: нет

Операции обработки символов:

(AH)=8 чтение атрибута/символа в текущей позиции курсора.

Вход:

(BH)= номер страницы (только для текстового режима).

Выход:

(AL)= прочитанный символ, (AH)= атрибут прочитанного символа (только для текстового режима).

(AH)=9 запись атрибута/символа в текущую позицию курсора.

Вход:

(BH)= номер страницы (только для текстовых режимов), (CX)= счетчик символов (повторений) для записи, (AL)= символ для записи;

(BL)= атрибут символа (текст) или цвет (графика).

Выход: нет

(AH)=0A запись только символа в текущую позицию курсора.

Вход:

(BH)= номер страницы (только для текстовых режимов),

(CX)= счетчик символов (повторений) для записи;

(AL)= символ для записи.

Для записи/чтения символов в графическом режиме знаки формируются в знакогенераторе, который находится в ПЗУ адаптера дисплея и содержит коды для младшей половины таблицы.

Для считывания/записи дополнительной таблицы знаков пользователь может через прерывание 1FH (ячейка 0007C) инициализировать указатель на 1Кбайт таблицу, содержащую коды 128 знаков второй половины таблицы ASCII, и поместить требуемую таблицу в ОЗУ.

Операции обработки графики:

(AH)=0B установка цветовой палитры.

Вход:

(BH)= цветовая палитра (0-127);

(BL)= управление цветом (для текущей цветовой палитры):

BH=0 выбор цвета фона, BL (0-15),

BH=1 выбор палитры:

BL=0 - зеленый (1), красный (2), желтый (3),

BL=1 - голубой (1), сиреневый (2), белый (3).

(AH)=0C запись точки.

Вход:

(DX)= номер строки;

(CX)= номер колонки;

(AL)= значение цвета.

Выход: нет

(AH)=0D чтение точки.

Вход:

(DX)= номер строки;

(CX)= номер колонки.

Выход:

(AL)= возвращается прочитанная точка.

Вывод в режиме телетайпа кода символов:

(AH)=0E запись в режиме телетайпа.

Вход:

(AL)= символ для записи;

(BL)= цвет;

(BH)= номер страницы в текстовом режиме.

Выход: нет

(AH)=0F текущий видеорежим. Возвращает текущие состояние видеосистемы.

Выход:

(AL)= текущий установленный режим;

(AH)= число столбцов на экране;

(BH)= текущая активная страница.

3. Определение размера памяти (INT 12H)

Это прерывание возвращает размер непрерывной области оперативной памяти в системе.

Размер памяти устанавливается во время диагностики по включению питания.

Выход: (AX) - размер памяти в Кбайтах.

4. Определение конфигурации (INT 11H)

Прерывание возвращает слово состояния, соответствующее штатным устройствам максимально возможной конфигурации компьютера, устанавливаемой при инициализации по включению питания.

Выход: (AX) - указывающие конфигурацию оборудования: биты 15, 14 - число печатающих устройств; бит 13 - не используется; бит 12 - игровой ввод/вывод подключен; биты 11, 10, 9 - количество подключенных последовательных портов; бит 8 - не используется; биты 7, 6 - число накопителей на ГМД, (00-1, 01-2, 10-3, 11-4 при бите 0=1); биты 5, 4 - начальный видеорежим: (00 - не используется, 01 - 40*25 цветной, 10 - 80*25 цветной, 11 - 80*25 черно/белый); биты 3, 2 - размер памяти, предоставляемой под операционную систему (в блоках по 16 или 64 K), бит 1 - сопроцессор 80x87 установлен, бит 0 - загрузка с дискеты (бит указывает, что в системе есть устройство ГМД). Другие регистры не используются.

5. Функции даты и времени (INT 1AH)

Прерывание выполняет установку/считывание часов.

Вход:

(AH)=0 - прочитать текущее значение часов.

Выход:

(CX) - старшая часть счетчика;

(DX) - младшая часть счетчика;

AL=0 - если таймер не прошел 24 часа с момента последнего считывания;

AL<>0 - если на следующий день.

(AH)=1 - установить текущее состояние часов

CX - старшая часть счетчика;

DX - младшая часть счетчика.

Счет выполняется 1193180/65536 раз в секунду (или 18.2 раза в сек).

6. Функции печати (INT 17H)

Это прерывание обеспечивает вывод на печатающее устройство.

Выполняются следующие операции:

(AH)=0 - печать знака из (AL).

Выход:

(AH)=1, если знак не мог быть напечатан (тайм-аут).

(AH)=1 - инициализация порта печати.

При возврате в (AH) устанавливается состояние печати.

(AH)=2 - считывание состояния печати в (AH).

Биты AH:

0 - тайм-аут;

1 - не используется;

2 - не используется;

3 - ошибка ввода/вывода;

4 - выбрано (готово);

5 - нет бумаги;

6 - подтверждение;

7 - занято.

(DX) - номер устройства печати (0, 1, 2), если их несколько.

Регистр AH модифицируется, другие регистры не изменяются.

7. Функция начальной загрузки операционной системы

Если накопитель на гибком магнитном диске готов, то загрузка возможна. Дорожка 0, сектор 1 считываются по адресу 0:7C00h, и на него передается управление (начальный загрузчик системы). В случае, если накопитель на гибком магнитном диске не готов, и в системе есть накопитель на жестком диске, делается попытка загрузки с жесткого диска, если и она неуспешна, делается попытка загрузить кассетный Бэйсик.

8. Операции дискового ввода/вывода (INT 13H)

Реализуются следующие операции:

На входе в регистре AH указывается режим работы:

(AH)=0 - сброс;

(AH)=1 - считывание состояния дисковой системы в регистр AL.

Состояние соответствует последней выполняемой операции.

Для операций чтения/записи, верификации, форматирования распределение регистров следующее:

(DL) - номер устройства (0-3);

(DH) - номер головки (0-1);

(CH) - номер дорожки (0-79);

(CL) - номер сектора (1-9);

(AL) - количество секторов;

(ES:BX) - адрес буфера.

(AH)=2 - считывание с заданного сектора;

(AH)=3 - запись;

(AH)=4 - верификация;

(AH)=5 - форматирование.

При форматировании (ES:BX) указывает на список параметров форматирования.

На выходе возвращается состояние операции:

CF=0 - успешное завершение (AH=0);

CF=1 - неверная операция (AH - код ошибки).

ПРИЛОЖЕНИЕ 4

Команды отладчика

Далее в настоящем разделе приведены команды отладчика DEBUG (MS DOS). Для каждой команды дается ее формат и приводятся примеры использования.

АССЕМБЛИРОВАТЬ **A[адрес]**

Команда обеспечивает занесение последовательности машинных команд в память с использованием мнемоник ассемблера. Адрес указывается либо в формате СЕГМЕНТ:СМЕЩЕНИЕ либо просто СМЕЩЕНИЕ, тогда в качестве адреса сегмента выбирается содержимое регистра CS.

ПРИМЕР: занести последовательность команд со смещением 100 относительно текущего состояния CS

```
-a100
1094:0100 mov ax,10
1094:0103 mov bx,20
1094:0106 add ax,bx
1094:0108 mov ax,4c00
1094:010B int 21
1094:010D
-
```

Ввод заканчивается при нажатии клавиши <enter> в ответ на приглашение отладчика на ввод очередной команды. (Приглашением отладчика в случае данной команды является адрес, по которому будет размещена очередная команда, адрес выводится в формате СЕГМЕНТ:СМЕЩЕНИЕ.)

СРАВНИТЬ **C диапазон адрес**

Команда обеспечивает сравнение двух областей памяти на совпадение содержимого. Первая область задается адресом начала и адресом конца, т.е. диапазоном адресов; вторая область задается адресом начала. В ходе работы команды на экран дисплея выводятся адреса и содержимое ячеек памяти, в случае, если это содержимое различно.

ПРИМЕР:

Сравнить содержимое областей памяти. Первая область расположена по адресу FE00:0000 и занимает 22 байта памяти, вторая область расположена по адресу 1094:100

-cfe00:0000 0015 1094:100

ответ команды сравнения:

FE00:0000 EA B8 1094:0100
FE00:0001 05 10 1094:0101
FE00:0002 E0 00 1094:0102
FE00:0003 00 BB 1094:0103
FE00:0004 F0 20 1094:0104
FE00:0005 E9 00 1094:0105
FE00:0006 34 01 1094:0106
FE00:0007 17 D8 1094:0107
FE00:0008 00 B8 1094:0108
FE00:000A 00 4C 1094:010A
FE00:000B 00 CD 1094:010B
FE00:000C 00 21 1094:010C
FE00:000D 00 8A 1094:010D
FE00:000E 49 7E 1094:010E
FE00:000F 42 04 1094:010F
FE00:0010 4D 80 1094:0110
FE00:0011 20 E7 1094:0111
FE00:0012 43 06 1094:0112
FE00:0013 4F 80 1094:0113
FE00:0014 4D FF 1094:0114
FE00:0015 50 06 1094:0115

-

ДАМП ПАМЯТИ **D [диапазон] или D [адрес]**

Выводит содержимое указанной области памяти в шестнадцатеричном и символьном виде.

ПРИМЕР 1:

-d100

1094:0100 B8 10 00 BB 20 00 01 D8-B8 00 4C CD 21 8A 7E 04 ....L.!.~.
1094:0110 80 E7 06 80 FF 06 75 18-8B 76 02 B3 34 00 83 10 ......u..v..4...
1094:0120 75 06 C6 46 00 02 EB 05-C6 46 00 01 4E E9 83 00 u..F.....F..N...

1094:0130 80 FF 02 75 05 C6 46 00-00 C3 E8 BC EB B4 3B CD   ...u..F.....;.

1094:0140 21 72 39 8B FA 33 C0 8B-C8 49 26 8A 05 47 0A C0   !r9..3...I&..G..

1094:0150 74 0C 32 E4 E8 43 E2 74-F1 47 FE C4 EB EC 4F A0   t.2..C.t.G....O.

1094:0160 7C 8D C6 46 00 02 0A E4-75 05 3A 45 FF 74 05 AA |..F....u.:E.t..

1094:0170 C6 46 00 01 80 4E 04 06-E8 74 00 C3 E8 21 DB 3D   .F...N...t...!.=

-

ПРИМЕР 2:

-df000:0000

F000:0000 30 31 32 33 41 41 41 41-4D 4D 4D 4D 49 49 49 49 0123AAAAMMMMIIII

F000:0010 42 42 42 42 49 49 49 49-4F 4F 4F 4F 53 53 53 53   BBBBIIIIOOOOSSSS

F000:0020 28 28 28 28 43 43 43 43-29 29 29 29 41 41 41 41   ((((CCCC))))AAAA

F000:0030 4D 4D 4D 4D 49 49 49 49-31 31 31 31 32 32 32 32   MMMMIIII11112222

F000:0040 2F 2F 2F 2F 31 31 31 31-32 32 32 32 2F 2F 2F 2F   ////11112222////

F000:0050 31 31 31 31 39 39 39 39-39 39 39 39 31 31 31 31   1111999999991111

F000:0060 20 44 61 74 65 3A 2D 31-32 2F 31 32 2F 39 31 20   Date:-12/12/91

F000:0070 28 43 29 31 39 38 35 2D-31 39 39 31 2C 41 4D 49   (C)1985-1991,AMI

-

ПРИМЕР 3:

-df000:fff0 ffff

F000:FFF0 EA D4 04 A1 02 31 32 2F-31 32 2F 39 31 00 FC 00   .....12/12/91...

МОДИФИКАЦИЯ СОДЕРЖИМОГО ПАМЯТИ E адрес [список значений]

Обеспечивает модификацию последовательности байтов или слов памяти начиная с указанного адреса.

ПРИМЕР:

-e1094:10d 10 20 30 40 50 60 70 80 90 a0 b0 c0 d0 e0 f0

ЗАПОЛНИТЬ ПАМЯТЬ УКАЗАННЫМ ЗНАЧЕНИЕМ F диапазон список

Память в указанном диапазоне заполняется указанными в списке значениями. В качестве значений могут быть указаны шестнадцатеричные числа и символьные строки в апострофах.

ПРИМЕР 1:

-f1094:10d 11f 01 22 33

Память в диапазоне 1094:10d - 1094:11f заполняется последовательностью байтов 01 22 33.

ПРИМЕР 2:

-f1094:10d 11f 'alfa'

Память в диапазоне 1094:10d - 1094:11f заполняется последовательностью строк 'alfa'

ВЫПОЛНИТЬ **G[=адрес] [адрес(а)]**

Начинает выполняться последовательность команд с указанного после знака = адреса. При достижении адреса (адресов), указанных далее в списке, происходит останов программы.

Дальнейшее выполнение возможно повторным вводом команды G без параметров.

HEX АРИФМЕТИКА **H число1 число2**

Вычисляет сумму и разность указанных шестнадцатеричных чисел.

ПРИМЕР:

-h0ae2 9f3

14D5 00EF

ВВОД ИЗ ПОРТА **Iадрес порта**

Из порта с указанным адресом вводится значение и выводится на экран

ПРИМЕРЫ:

-i 61

20

-i278

FF

-i60

1C

ЗАГРУЗКА **L[адрес] [накопитель] [первый сектор] [количество]**

Содержимое диска, указанного параметром НАКОПИТЕЛЬ, начиная с сектора, указанного параметром ПЕРВЫЙ СЕКТОР (количество секторов указывается параметром КОЛИЧЕСТВО), загружается в область памяти, начинающуюся с адреса АДРЕС. Допускаются сокращенные формы команды: при использовании команды в формате L [адрес] имя загружаемого файла должно быть задано с помощью команды N. В этом случае при отсутствии параметра АДРЕС exe-файлы загружаются по адресу CS:0, com-файлы - по адресу CS:100.

ПРИМЕР 1:

Загрузить в память с адреса CS:0 с накопителя A четыре сектора с сектора 1

-l100 0 1 4

Примечание: диск A имеет номер 0, B - 1, C - 2 и т.д.

ПРИМЕР 2:

Загрузить в память файл, имя которого ранее было определено с помощью команды N

-l

СКОПИРОВАТЬ ОБЛАСТЬ ПАМЯТИ **M диапазон адрес**

Указанная диапазоном область памяти копируется по указанному адресу.

ПРИМЕР:

-m100 10c 10d

область памяти CS:100-CS:10C копируется в область памяти начиная с адреса CS:10D

ИМЯ ФАЙЛА **N[полная спецификация файла] [список аргументов]**

Задает имя файла для последующих операций чтения с диска и записи на диск с помощью команд L и W

ПРИМЕР 1:

-nproba.com

при последующем выполнении команд L и W информация будет считываться из файла proba.com , расположенного в текущем каталоге.

ПРИМЕР 2:

-nc:\temp\com\myprg.com

при последующем выполнении команд L и W информация будет считываться из файла myprg.com или записываться в этот же файл, расположенный на диске C: в подкаталоге temp\com.

ВЫВЕСТИ В ПОРТ **O адрес_порта байт**

Вывести в порт, указанный параметром АДРЕС ПОРТА байт, указанный параметром БАЙТ.

ПРИМЕР:

-o61 4b

вывести в порт 61h значение 4Bh

-o61 48

вывести в порт 61h значение 48h

ВЫСОКОУРОВНЕВАЯ ТРАССИРОВКА P[=адрес] [количество_команд]

Выполнить количество команд, указанное параметром КОЛИЧЕСТВО КОМАНД, с адреса, указанного параметром АДРЕС. После выполнения каждой команды на экран выводится состояние всех регистров процессора и мнемоническое обозначение следующей выполняемой команды. В случае отсутствия параметра КОЛИЧЕСТВО КОМАНД будет выполнена одна команда. Команда P не обеспечивает, в отличие от команды T, вход в подпрограммы, прерывания и циклы, т.е. при трассировке, например, команды CALL подпрограмма выполняется в обычном, не трассируемом режиме.

ПРИМЕР:

```
-p5
AX=0010  BX=0000  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0103   NV UP DI PL NZ NA PE NC
1094:0103 BB2000        MOV     BX,0020

AX=0010  BX=0020  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0106   NV UP DI PL NZ NA PE NC
1094:0106 01D8          ADD     AX,BX

AX=0030  BX=0020  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0108   NV UP DI PL NZ NA PE NC
1094:0108 B8004C        MOV     AX,4C00

AX=4C00  BX=0020  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=010B   NV UP DI PL NZ NA PE NC
1094:010B CD21          INT     21
-
```

ВЫХОД Q

Выход из отладчика.

ПРОСМОТР И ИЗМЕНЕНИЕ РЕГИСТРОВ R[имя_регистра]

Команда обеспечивает просмотр и изменение регистров процессора. Может использоваться в трех форматах:

1. R - обеспечивает вывод на экран состояние регистров процессора.

ПРИМЕР:

-r

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=1094 ES=1094 SS=1094 CS=1094 IP=0100 NV UP EI PL NZ NA PO NC

1094:0100 B81000 MOV AX,0010

-

2. R имя регистра - позволяет изменить значение регистра заданного параметром ИМЯ РЕГИСТРА.

ПРИМЕР:

-rax

AX 0000

:1000

-

3. RF - позволяет просмотреть и изменить состояние отдельных флагов в регистре признаков процессора.

ПРИМЕР:

-rf

NV UP EI PL NZ NA PO NC -cy

-

Для установленного и сброшенного состояния каждый флаг имеет свое мнемоническое обозначение:

ПЕРЕПОЛНЕНИЕ	OV	NV
НАПРАВЛЕНИЕ	DN	UP
ЗАПРЕШЕНИЕ ПРЕРЫВАНИЙ	DI	EI
ЗНАК	NG	PL
НУЛЬ	ZR	NZ
ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС	AC	NA
ЧЕТНОСТЬ	PE	PO
ПЕРЕНОС	CY	NC

ПОИСК **S диапазон список_значений**

В указанном диапазоне отыскивается указанный список значений. При нахождении указанных значений на экран выводится адрес, по которому эти значения расположены в памяти.

ПРИМЕР:

в области памяти с адреса 1094:100 по адрес 1094:200 отыскать байты, в которых записано CD.

```
-s100 200 cd
1094:010B
1094:013F
1094:01E0
```

ТРАССИРОВКА **T[=адрес] [количество_команд]**

Выполнить количество команд, указанное параметром КОЛИЧЕСТВО КОМАНД, с адреса, указанного параметром АДРЕС. После выполнения каждой команды на экран выводится состояние всех регистров процессора и мнемоническое обозначение следующей выполняемой команды. В случае отсутствия параметра КОЛИЧЕСТВО КОМАНД будет выполнена одна команда. В случае отсутствия параметра АДРЕС будет выполняться команда, расположенная по адресу CS:IP. В отличие от команды P данная команда обеспечивает вход в подпрограммы, циклы, прерывания.

ПРИМЕР 1:

```
AX=0010  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0103   NV UP EI PL NZ NA PO CY
1094:0103 BB2000        MOV     BX,0020
-t

AX=0010  BX=0020  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0106   NV UP EI PL NZ NA PO CY
1094:0106 01D8          ADD     AX,BX
-t

AX=0030  BX=0020  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0108   NV UP EI PL NZ NA PE NC
1094:0108 B8004C        MOV     AX,4C00
-t

AX=4C00  BX=0020  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=010B   NV UP EI PL NZ NA PE NC
1094:010B CD21          INT     21
-t
```

```
AX=4C00  BX=0020  CX=0000  DX=0000  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=0968  IP=063C   NV UP DI PL NZ NA PE NC
0968:063C 9C            PUSHF
```

ПРИМЕР 2:

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0100   NV UP DI PL NZ NA PE NC
1094:0100 B81000        MOV     AX,0010
-t5

AX=0010  BX=0000  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0103   NV UP DI PL NZ NA PE NC
1094:0103 BB2000        MOV     BX,0020

AX=0010  BX=0020  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0106   NV UP DI PL NZ NA PE NC
1094:0106 01D8          ADD     AX,BX

AX=0030  BX=0020  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=0108   NV UP DI PL NZ NA PE NC
1094:0108 B8004C        MOV     AX,4C00

AX=4C00  BX=0020  CX=0000  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=1094  IP=010B   NV UP DI PL NZ NA PE NC
1094:010B CD21          INT     21

AX=4C00  BX=0020  CX=0000  DX=0000  SP=FFF8  BP=0000  SI=0000  DI=0000
DS=1094  ES=1094  SS=1094  CS=0968  IP=063C   NV UP DI PL NZ NA PE NC
0968:063C 9C            PUSHF
```

ДИЗАССЕМБЛЕР **U [диапазон] или U[адрес]**

Обеспечивает дисассемблирование кода из указанного диапазона или начиная с указанного адреса.

ПРИМЕР:

```
-u100 10c
1094:0100 B81000        MOV     AX,0010
```

```
1094:0103 BB2000       MOV     BX,0020
1094:0106 01D8         ADD     AX,BX
1094:0108 B8004C       MOV     AX,4C00
1094:010B CD21         INT     21
-
```

Последовательность кодов с адреса 1094:100 по 1094:10c переводится в последовательность команд ассемблера.

ЗАПИСЬ W[адрес] [накопитель] [первый_сектор] [количество]

Обеспечивает запись информации из оперативной памяти на диск. Возможны два варианта команды:

1. W адрес накопитель сектор количество

Содержимое памяти с указанного адреса помещается в указанное количество секторов на указанном накопителе.

ПРИМЕР :

-w2000:100 0 1 1

содержимое памяти с адреса 2000:100 будет помещено в первый сектор нулевого (A) накопителя. На диск будет записано 512 байт, поскольку длина сектора - 512, и в команде записи в качестве параметра КОЛИЧЕСТВО указана 1.

2. W или W адрес

В этом формате команда используется для записи данных на диск в файл, имя которого было определено с помощью команды N. Команда W без параметров может использоваться для записи в файл, ранее считанный командой L. Команда W адрес позволяет "сбросить" в файл содержимое произвольной области памяти. В этом случае пара регистров BX:CX должна содержать записываемое на диск количество байтов.

ПРИЛОЖЕНИЕ 5

СРОКИ СДАЧИ ОТЧЕТОВ ПО ЛАБОРАТОРНЫМ РАБОТАМ

№ п.п	Тема работы	Срок сдачи (неделя семестра)	Теоретический материал, приложения
1.	Логический состав процессора компьютера и назначение его компонентов. Принципы программного управления	2	Разделы 1,2,8 Приложение 1
2.	Система команд процессоров	4	Раздел 2,3,8 Приложение 1
3.	Система команд процессоров и методы адресации	6	Раздел 3,4,8 Приложение 2
4.	Индексная адресация, работа с массивами	7	Раздел 4,8 Приложение 2
5.	Организация подпрограмм и внутренние механизмы передачи параметров	10	Раздел 5,8 Приложение 2
6.	Организация прерываний	12	Раздел 5,8 Приложение 2
7.	Введение в архитектуру IBM PC	14	Раздел 7,8 Приложение 4
8.	Программирование на языке ассемблера IBM PC и использование системы прерываний	17	Раздел 7,8 Приложение 3

Содержание