Министерство образования Российской Федерации
Воронежский государственный университет

А.П.Толстобров

АРХИТЕКТУРА ЭВМ

Учебное пособие

Направление подготовки дипломированного специалиста 654700 – «Информационные системы» (специальность 07900) и направление подготовки бакалавров 511800 – «Математика. Компьютерные науки»

Воронеж 2004

УДК 681.31

Рекомендовано научно-методическим советом факультета компьютерных наук Воронежского государственного университета.

В пособии рассматриваются основные принципы построения ЭВМ с фон-неймановской архитектурой: структура центрального процессора, система команд, организация ввода-вывода, управление памятью. Оно предназначено для использования в качестве учебных материалов по дисциплине «Архитектура ЭВМ», изучаемой студентами первого курса факультета компьютерных наук, и является вводным курсом для цикла дисциплин, связанных с программированием и использованием вычислительной и микропроцессорной техники.

Составитель – А.П.Толстобров

Пособие подготовлено на кафедре Информационных систем факультета компьютерных наук Воронежского государственного университета

Рекомендуется для студентов 1 курса направления подготовки дипломированного специалиста 654700 – «Информационные системы», специальности 07900 – «Информационные системы и технологии» по дисциплине «Архитектура ЭВМ и систем» и направления подготовки бакалавров 511800 – «Математика. Компьютерные науки» по дисциплине «Архитектура ЭВМ и системное программное обеспечение»

Введение

За полувековую историю развития ЭВМ сменилось несколько поколений электронных вычислительных систем, кардинальным образом изменилась их технология и элементная база, их качественные характеристики, значительно расширилась сфера применения компьютерной техники. Эти факторы, естественно, усложняют изучение этого вида техники. Интересно, однако, что, несмотря на множество поколений, семейств, типов и конкретных реализаций ЭВМ, в основе большинства из них лежат общие принципы, сформулированые в 1946 году американским ученым Джоном фон-Нейманом. Важность изучения этих принципов при подготовке специалистов в области компьютерных технологий обусловлена не только тем, что они до сих пор лежат в основе большинства современных ЭВМ и компьютерных систем. Их знание необходимо для успешного понимания других уже «не фоннеймановских» архитектурных принципов и технических решений, используемых при построении и развитии современных компьютерных устройств и систем, понимания необходимости и условий использования этих новых принципов, достигаемого при этом эффекта и цены, которую приходится платить для его достижения.

Сложность современных вычислительных машин закономерно привела к понятию *архитектура ЭВМ,* охватывающего описание принципов организации цифровой вычислительной системы на некотором общем уровне, ориентированном в первую очередь на пользователя, интересующегося главным образом возможностями машины, а не деталями ее технического исполнения. Этот уровень не отражает такие проблемы, как управление и передача данных внутри процессора, конструктивные особенности логических схем и специфика технологии их производства. В круг рассматриваемых вопросов входят способы представление информации в ЭВМ и принципы построения устройств для выполнения арифметических и логических операций, структура центрального процессора ЭВМ, проблемы кодирования и выполнения команд ЭВМ, организация памяти ЭВМ и системы адресации, управление памятью, организация совместной работы входящих в ЭВМ устройств, операции ввода-вывода информации и т.д. Знание этих аспектов организации ЭВМ необходимо для обеспечения эффективного использования всех возможностей конкретной компьютерной системы, при программировании на машинно-ориентированном языке (например, в машинных кодах, на языке ассемблера).

1. Принципы организации ЭВМ с фон-неймановской архитектурой

1.1. Обобщенная структура ЭВМ

Типичная цифровая ЭВМ включает в себя три основных компонента: *процессор, память и внешние устройства*. Ее обобщенная блок-схема представлена на рис.1.1.

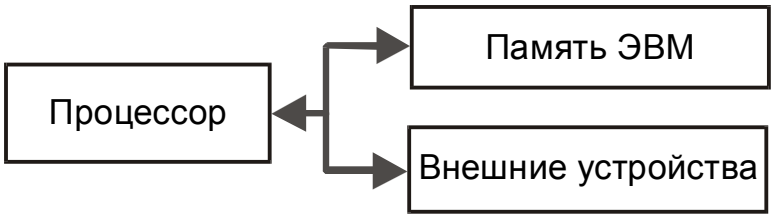

Рис.1.1

Процессор или *центральный процессор* (ЦП) – это устройство, предназначенное для выполнения основных операций по обработке данных, арифметических и логических операций над числами, управления работой других частей ЭМ.

Память или *оперативное запоминающее устройство* (ОЗУ) – предназначено для хранения кодов команд, составляющих выполняемую ЭВМ программу, и данных или операндов, т.е. двоичных чисел или кодов, над которыми процессор ЭВМ выполняет задаваемые командами операции.

Через *устройства ввода-вывода* или *внешние (периферийные) устройства* осуществляется взаимодействие ЭВМ с внешним миром.

Компоненты ЭВМ связаны друг с другом с помощью специальной *шины* или *канала* ЭВМ, представляющих собой набор линий связи, предназначенных для передачи информационных и управляющих сигналов между компонентами ЭВМ.

Приведенная схема является настолько привычной, что кажется почти очевидной. Однако, совсем не праздными являются вопросы, почему типичная ЭВМ включает в себя указанные компоненты, являются ли они обязательными, возможны ли другие способы построения ЭВМ, каково должно быть устройство основных элементов ЭВМ, наконец, что же общего между первыми ЭВМ и современными компьютерами.

В 1946 году Джон фон-Нейман вместе с группой работавших вместе с ним ученых сформулировал основные принципы, которым должно удовлетворять устройство, получившее название *электронная вычислительная машина* или *ЭВМ*. Эти принципы оказались настолько основополагающими, что и до настоящего времени, несмотря на смену большого числа поколений ЭВМ, большинство из них основано на использовании этих принципов, получивших название *фон-неймановских принципов организации ЭВМ*.

1.2. Принципы организации ЭВМ

Принципы, которым удовлетворяют ЭВМ с фон-неймановской архитектурой, заключаются в следующем:

1. *ЭВМ – это машина с **хранимой** (в памяти ЭВМ) **программой**, представленной в виде последовательности команд.*
2. *Выполняемые ЭВМ **команды и операнды**, т.е. данные, над которыми выполняется задаваемая командой операция, представлены в ЭВМ в виде **двоичного кода** с определенным количеством разрядов.*
3. ***Память ЭВМ** организована в виде **последовательности запоминающих ячеек**, в каждой из которых может храниться (запоминаться) некоторый двоичный код – число или код символа алфавита, представляющие обрабатываемые данные, код команды ЭВМ. В конкретный момент времени можно **обратиться** для записи или чтения **к любой одной из этих ячеек независимо от ее расположения в памяти**, указав адрес (порядковый номер) этой ячейки. Таким способом организованная память называется **памятью с произвольным доступом**.*
4. *В ЭВМ используется **общая память** как для хранения **данных,** так и для хранения **команд**. При этом в кодах самих данных и команд отсутствуют признаки, позволяющие явно отличать их друг от друга. Процессор различает данные и команды только по контексту выполняемой программы.*
5. ***Предназначение данных, их тип и способ использования также явно не указываются**. Они определяются и различаются по контексту выполняемой программы.*

6. *В классической фон-неймановской ЭВМ используется один центральный процессор*.

1.3. Контрольные вопросы

1. Объясните, в чем состоит принципиальный смысл формулы «ЭВМ – это машина с хранимой программой»?
2. Какая система счисления и почему выбрана в фон-неймановской ЭВМ для внутреннего представления чисел?
3. Представление в памяти фон-неймановской ЭВМ данных и команд.
4. Что такое программа ЭВМ? В каком виде и где она должна размещаться, для того чтобы процессор мог ее выполнять?
5. Для чего в ЭВМ нужна память? Особенности организации памяти фон-неймановской ЭВМ.
6. Что такое «память с произвольным доступом», возможны ли другие способы доступа к ячейкам памяти, другие способы организации памяти?
7. Что такое адрес ячейки памяти ЭВМ?
8. В ЭВМ с фон-неймановской архитектурой данные и команды хранятся:
   a) раздельно в памяти команд и памяти данных;
   b) в общей памяти;
   c) данные хранятся в памяти ЭВМ, а команды поступают от внешних устройств;
   d) команды находятся в памяти ЭВМ, а данные принимаются из портов внешних устройств;
   e) ваш вариант.

   В чем преимущество выбранного Вами решения?
9. Можно ли по содержимому ячейки памяти фон-неймановской ЭВМ определить, что в ней находится: команда, целое число без знака, число со знаком и т.д., если да, то каким образом?
10. Каким образом процессор фон-неймановской ЭВМ определяет, из каких ячеек памяти следует выбирать команды, а из каких данные?

2. Представление информации в ЭВМ. Системы счисления и арифметические операции над числами

2.1. Виды информации

Фон-неймановский компьютер представляет собой систему обработки информации, представленной в виде двоичного кода, то есть выраженной в виде последовательности нулей и единиц. Это обусловлено тем, что для представления такого кода можно использовать физические процессы и объекты, которые могут находиться в двух устойчивых состояниях. Такие процессы и объекты реализуются гораздо проще, чем имеющие большее число состояний. Кроме того, существенно проще реализуются устройства, осуществляющие обработку таким образом представленной информации, в частности, арифметических операций над числами.

Один двоичный разряд позволяет представить минимальную «порцию» информации, равную одному *биту*. Восемь двоичных разрядов образуют *байт*. Шестнадцать двоичных разрядов образуют *слово*, состоящее, в свою очередь, из *младшего* (правого) и *старшего* (левого) байтов.

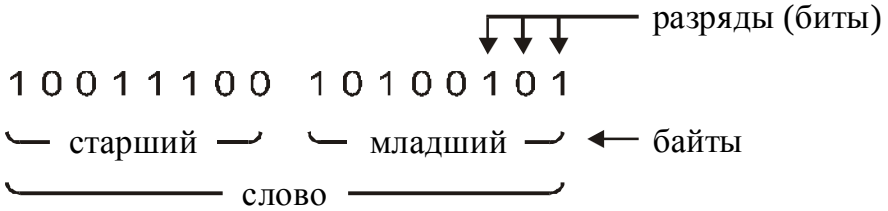

Рис.2.1

Последовательность двоичных разрядов может быть использована для кодирования различных видов информации.

- *Логическая информация*. В этом случае два состояния каждого двоичного разряда представляют собой одно из состояний логической переменной – истина (true) или ложь (false).
- *Алфавитно-символьная информация*. Для кодирования символов какого-либо алфавита используется определенное количество двоичных разрядов. Набор из n двоичных разрядов позволяет представить алфавит из 2^n символов. Обычно для кодирования алфавитно-цифровых

символов используется байт. С помощью байта можно кодировать символы алфавита, состоящего из $2^8 = 256$ различных символов. При этом возможны и практически используются различные кодировки или кодовые таблицы соответствия двоичных кодов конкретным символам: ASCII, КОИ8, DOS, Windows и другие, что создает на практике определенные трудности при интерпретации алфавитно-символьной информации в различных программных системах.

- *Числовая информация*. Двоичный код представляет собой ту или иную форму чисел (без знака и со знаком, целое, дробное с фиксированной или плавающей точкой).

2.2. Выбор системы счисления для представления чисел в ЭВМ

Общепринятой в человеческой практике формой представления чисел является использование позиционной системы счисления. В позиционной системе счисления вес, т.е. значимость каждой цифры, составляющей число, определяется его позицией внутри числа. В соответствии со своей позицией каждая цифра числа умножается на коэффициент, представляющий собой так называемое основание системы счисления, возведенное в степень, равную номеру позиции данной цифры слева направо. Например:

$5724_{10} = 5\cdot10^3 + 7\cdot10^2 + 2\cdot10^1 + 4\cdot10^0$ – целое число в десятичной системе счисления

$34{,}85_{10} = 3\cdot10^1 + 4\cdot10^0 + 8\cdot10^{-1} + 5\cdot10^{-2}$ – дробное число в десятичной системе

$2731_8 = 2\cdot8^3 + 7\cdot8^2 + 3\cdot8^1 + 1\cdot8^0$ – целое число в восьмеричной системе

$11010_2 = 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 0\cdot2^0$ – целое число в двоичной системе счисления

Для представления чисел в ЭВМ была выбрана *двоичная система счисления*. Эта система, являясь наиболее простой, использует только две цифры – 0 и 1.

В таблице 2.1 приведено взаимное соответствие чисел, представленных в десятичной и двоичной системах счисления, а также восьмеричной и шестнадцатеричной системах.

Таблица 2.1

Десятичная	Двоичная	Восьмеричная	Шестнадцатеричная	Двоично-десятичная
0	0	0	0	0
1	1	1	1	1
2	10	2	2	10
3	11	3	3	11
4	100	4	4	100
5	101	5	5	101
6	110	6	6	110
7	111	7	7	111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	1 0000
11	1011	13	B	1 0001
12	1100	14	C	1 0010
13	1101	15	D	1 0011
14	1110	16	E	1 0100
15	1111	17	F	1 0101
16	10000	20	10	1 0110
17	10001	21	11	1 0111
18	10010	22	12	1 1000
19	10011	23	13	1 1001
20	10100	24	14	10 0000
21	10101	25	15	10 0001
22	10110	26	16	10 0010
23	10111	27	17	10 0011
24	11000	30	18	10 0100
25	11001	31	19	10 0101
26	11010	32	1A	10 0110
27	11011	33	1B	10 0111
28	11100	34	1C	10 1000
29	11101	35	1D	10 1001
30	11110	36	1E	11 0000
31	11111	37	1F	11 0001
32	100000	40	20	11 0010

Причина использования восьмеричной и шестнадцатеричной систем в компьютерных приложениях лежит в простоте их перевода в двоичную систему и обратно. В таблице приведено также представление чисел в двоично-десятичной системе.

2.3. Представление в ЭВМ целых двоичных чисел без знака

Обычной моделью представления целых чисел является бесконечная числовая ось (рис.2.2), на которой при движении слева направо числа последовательно увеличиваются на единицу.

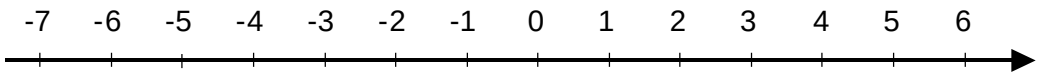

Рис.2.2

Так как в фон-неймановской ЭВМ для представления чисел используется конечное число разрядов, то, следовательно, и все множество представимых в ЭВМ целых чисел также оказывается конечным. В связи с этим представляющая эти числа числовая ось как бы замыкается сама на себя, как это показано на рис.2.3 для четырехразрядных двоичных чисел.

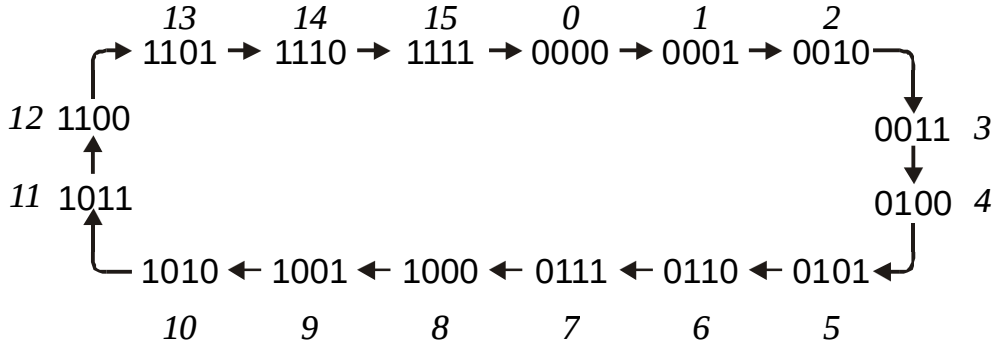

Рис.2.3

Дело в том, что в аппаратных средствах ЭВМ, оперирующих с двоичными числами с конечным числом разрядов, имеются физические элементы для представления и отображения только этих разрядов. Поэтому операция прибавления единицы к наибольшему представимому в данном примере числу 1111 приведет к переходу не к числу 10000, а к числу 0000, т.к. возникающий в результате переноса пятый разряд аппаратурой не фиксируется и для пользователя никаким образом не проявляется.

2.4. Представление в ЭВМ целых чисел со знаком

Очевидно, что при использовании для представления чисел со знаком конечного количества двоичных разрядов как отрицательные, так и положительные двоичные числа могут быть представлены только экземплярами из имеющегося конечного набора (множества) чисел. Возможная конкретная форма представления в ЭВМ чисел со знаком, т.е. взаимного соответствия двоичных кодов с фиксированным количеством разрядов конкретным положительным и отрицательным числам, может быть различной. Конкретный выбор этой формы имеет очень большое значение, т.к. от этого в большой степени определяет алгоритм выполнения основных арифметических операций над числами со знаком и, следовательно, сложность реализации устройств, осуществляющих в ЭВМ эти операции.

Обычно для представления отрицательных чисел в ЭВМ используют так называемый *двоично-дополнительный код*. Пример такой формы представления чисел со знаком для четырехразрядных чисел может быть наглядно представлен в виде замыкающейся самой на себя числовой оси, как это показано на рис.2.4.

Рис.2.4

Алгоритм получения двоично-дополнительного кода отрицательного числа из соответствующего ему положительного числа достаточно прост.

– Вначале следует ***проинвертировать*** *все разряды исходного положительного числа, т.е. заменить в нем все нули на единицы, а единицы на нули, после чего к результату* ***прибавить единицу****.*

Например:

для 4-разрядных чисел	для 8-разрядных чисел
$+3_{10} = 0011_2$	$+3_{10} = 00000011_2$
$+\ 1100$	$+\ 11111100$
$\underline{\qquad\ 1}$	$\underline{\qquad\qquad\ 1}$
$-3_{10} = 1101_2$	$-3_{10} = 11111101_2$

Каковы свойства такого представления двоичных чисел со знаком? Как видно из диаграммы на рис.2.4:

1. *Положительные числа представляются обычным образом*;
2. *У всех положительных чисел старший разряд равен нулю, а у всех отрицательных равен единице*;
3. *При переходе по часовой стрелке от одного числа к другому, как для положительных, так и для отрицательных чисел, каждое следующее число на единицу больше предыдущего, как это имеет место для обычного представления чисел на бесконечной числовой оси*;
4. *Сумма положительного числа и равного ему по абсолютной величине отрицательного числа равна нулю*.

Наиболее важным доводом в пользу такого представления чисел со знаком является то, что арифметическая операция вычитания может быть заменена в этом случае операцией алгебраического сложения чисел со знаком.

Рассмотрим примеры:

1) $3 + 4 = 7$

$+\ 3$	0011
$+\ 4$	$+0100$
$+\ 7$	0111

2) $6 - 3 = 6 + (-3) = -3$

	$1\ 1$	
$+\ 6$	0110	
$-\ 3$	$+1101$	← число –3
$+\ 3$	10011	в дв.-доп. коде
		← дв. код числа +3

возникший в результате переноса лишний разряд игнорируется

3) –2 – 5 = (–2) + (–5) = –7

```
   1 11
+  1110  ← дв.-доп. код числа –2
   1011  ← дв.-доп. код числа –5
1 1001
разряд не фиксируется →
        ↖ дв.-доп. код числа –7
```

4) 4 – 7 = 4 + (–7) = –7

```
+  0100  ← +4
   1001  ← дв.-доп. код числа –7
   1101  ← дв.-доп. код числа –3
```

Из сказанного следует, что при таком представлении чисел со знаком в ЭВМ можно обойтись без использования устройства вычитания чисел, используя устройство сложения, как для выполнения операции сложения, так и для вычитания чисел. Учитывая, что операции умножения и деления также могут быть реализованы с помощью алгоритмов, использующих операции сложения и вычитания, становится ясным, что для реализации блока ЭВМ, осуществляющего арифметические операции над двоичными числами, принципиальным является реализация именно *устройства сложения* или *сумматора*.

Следует обратить внимание, что, хотя у всех отрицательных чисел, представленных двоично-дополнительным кодом, старший разряд равен единице, этот разряд *не является знаком числа*, это такой же полноправный разряд двоичного числа, участвующий в операции сложения, как и остальные разряды.

2.5. Особенности выполнения в ЭВМ сложения двоичных чисел без знака и со знаком

Конечное число разрядов используемых в ЭВМ двоичных чисел может приводить к появлению ошибок при выполнении арифметических операций. Рассмотрим эти ситуации на примере сложения 4-разрядных двоичных чисел.

1. Примеры сложения чисел без знака.

а)

```
  3      0011
+ 2    + 0010
  5      0101  ← результат правильный
```

б)

```
  6      0110
+ 7    + 0111
 13      1101  ← результат правильный
```

Наконец,

```
  10      1010
+ 11    + 1011
  21    1 0101
```

разряд игнорируется ↗ ↖ +5 — результат неверный

Эти примеры иллюстрируют следующее правило:

– если при сложении двоичных чисел без знака происходит перенос из старшего разряда, то результат арифметической операции оказывается ***неправильным****.*

2. Примеры сложения чисел *со знаком*.

а)

```
  5      0101
+ 6    + 0110
 11      1011   ← дв.-доп. код числа –5 (результат неверный)
```

б)

```
 –3      1101
– 6    + 1010
 –9    1 0111
```

перенос из старшего разряда (игнорируется) ↗ ↖ дв. код числа +7 (результат неверный)

в)

```
  7      0111
+ 7    + 0111
 14      1110   ← дв.-доп. код числа –2 (результат неверный)
```

перенос в старший разряд ↗

г)

```
 –5      1011
– 6    + 1010
–11    1 0101
```

перенос из старшего разряда (игнорируется) ↗ ↖ дв. код числа +5 (результат неверный)

д)

```
          1   ← перенос в старший разряд
  6       0110
– 3     + 1101
  3     1 0011   ← дв.-доп. код числа 3 (результат правильный)
```

перенос из старшего разряда (игнорируется) ↗

Последние пять примеров иллюстрируют следующее правило:

– если при сложении двух двоичных чисел со знаком, представленных в двоично-дополнительном коде, происходит ***перенос в старший разряд*** *(примеры* а *и* б*) или перенос из старшего разряда (один из этих переносов) (примеры* в *и* г*), то результат операции получается* ***неверным****,*

однако,

– если ***переносов нет*** *или имеют место одновременно* ***оба*** *таких* ***переноса*** *(пример* д*), то результат операции оказывается* ***правильным****.*

Ситуацию, когда имеет место один из переносов (перенос в старший разряд или перенос из старшего разряда) обычно называют *арифметическим переполнением*, наличие которого служит признаком *неправильного* выполнения операции сложения чисел со знаком.

2.6. Контрольные вопросы

1. Сколько чисел без знака можно представить с помощью 16-ти двоичных разрядов? - положительных чисел? - отрицательных?
2. Каким образом представляются в ЭВМ числа со знаком? Обоснуйте выбор такого представления.
3. Переведите восьмеричное число 157325_8 в шестнадцатиразрядное двоичное число, укажите младший и старший байты двоичного числа, запишите такое же число (двоичное и восьмеричное), но с противоположным знаком.
4. Переведите восьмеричное число 012737_8 в шестнадцатиразрядное двоичное число, укажите младший и старший байты двоичного числа, запишите такое же число (двоичное и восьмеричное), но с противоположным знаком.
5. Переведите шестнадцатеричное число $A5C6_{16}$ в шестнадцатиразрядное двоичное число, укажите младший и старший байты двоичного числа, запишите такое же число (двоичное и шестнадцатеричное), но с противоположным знаком.
6. Переведите шестнадцатеричное число $FFAB_{16}$ в шестнадцатиразрядное двоичное число, укажите младший и старший байты двоичного числа, запишите такое же число (двоичное и шестнадцатеричное), но с противоположным знаком.
7. Что такое перенос из старшего разряда? Приведите пример сложения двоичных чисел, когда возникает такая ситуация.
8. Что такое перенос в старший разряд? Приведите пример сложения двоичных чисел, когда возникает такая ситуация.

9. Что такое арифметическое переполнение? О чем говорит возникновение арифметического переполнения при выполнении арифметической операции? Приведите пример.
10. Отметьте, в каких из приведенных примеров имеют место: переносы из старшего разряда, в старший разряд, арифметическое переполнение:
    01100110 + 01001100 =
    00100110 + 01001100 =
    11100110 + 01001100 =
    10100110 + 11001100 =
11. При сложении двух чисел со знаком произошел перенос в старший разряд. Что можно сказать о правильности или неправильности результата сложения?
12. Отметьте, в каких из приведенных примеров имеют место: переносы из старшего разряда, в старший разряд и арифметическое переполнение:
    01100110 + 01011100 =
    10110110 + 11001100 =
    11100110 + 01011100 =
    00101110 + 01001100 =
13. При сложении двух чисел со знаком произошел перенос из старшего разряда. Что можно сказать о правильности или неправильности результата сложения? Что можно сказать об этой ситуации, если складываются числа без знака?
14. При сложении двух чисел со знаком произошел перенос в старший разряд и перенос из старшего разряда. Что можно сказать о правильности или неправильности результата сложения?

3. Принципы построения устройств для осуществления арифметических и логических операций над двоичными числами

3.1. Реализация логических операций И, ИЛИ, НЕ

Для реализации операциями над двоичными числами, более простыми, чем арифметические, являются логические операции **И**, **ИЛИ**, **НЕ**.

1. Логическая операция **И** (конъюнкция, логическое умножение).

Двоичная переменная на выходе Z принимает значение ИСТИНА, обозначаемое «Т» (True), тогда и только тогда, когда ***обе*** входные переменные X и Y одновременно находятся в состоянии ИСТИНА. Если же ***хотя бы одна*** из входных переменных находится в состоянии ЛОЖЬ, обозначаемое «F» (False), двоичная переменная на выходе также будет в состоянии ЛОЖЬ.

Логическое выражение — Z = X∧Y или X·Y, или X&Y

Обозначение на схемах

Таблица истинности

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Рис.3.1

2. Логическая операция **ИЛИ** (дизъюнкция, логическое сложение). Двоичная переменная на выходе Z принимает значение ИСТИНА, когда ***хотя бы одна*** входная переменная X или Y находится в состоянии ИСТИНА.

Логическое выражение — Z = XVY или X+Y

Обозначение на схемах

Таблица истинности

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Рис.3.2

3. Логическая операция **НЕ** (отрицание, инверсия).

Двоичная переменная на выходе Z принимает значение ИСТИНА, когда входная переменная X находится в состоянии ЛОЖЬ и значение ЛОЖЬ, когда входная переменная X находится в состоянии ИСТИНА.

Логическое выражение — Z = X

Обозначение на схемах

Таблица истинности

X	Z
0	1
1	0

Рис.3.3

Логические операции И, ИЛИ, НЕ достаточно просто реализуются с помощью электронных схем, например:

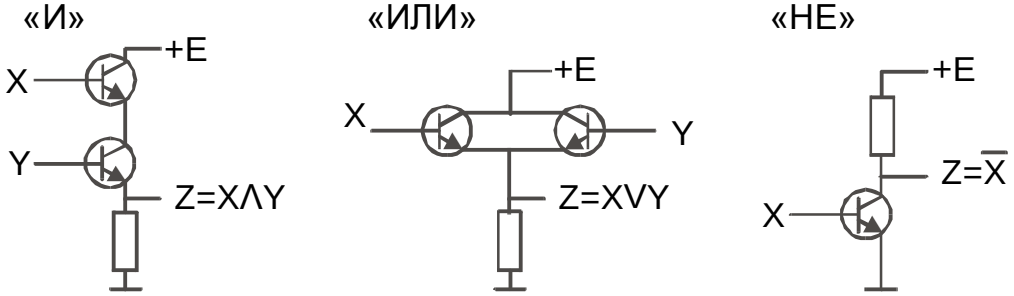

Рис.3.4

Электронные схемы, реализующие логические операции И, ИЛИ, НЕ, называются *вентильными схемами*.

3.2. Одноразрядные полусумматор и полный сумматор

Используя такие простейшие вентильные схемы, можно реализовывать схемы, выполняющие более сложные операции над двоичными сигналами. На рис.3.5 представлена цифровая схема, называемая *полусумматором*.

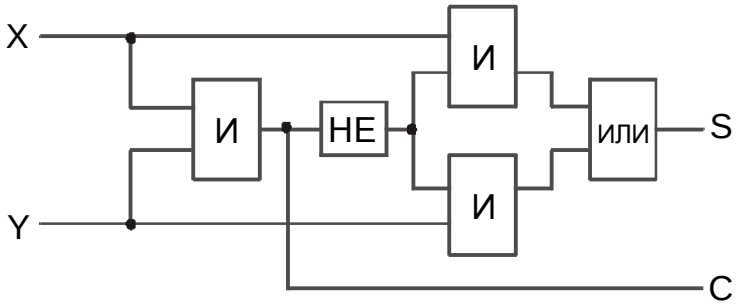

Таблица истинности

X	Y	C	Z
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Рис.3.5

Если сигналы S и C на выходе схемы представить соответственно как младший и старший разряды двухразрядного двоичного числа, то операция, осуществляемая схемой над входными сигналами, представляющими одноразрядные двоичные числа X и Y, совпадает с операцией сложения (0 + 0 = 00; 0 + 1 = 01; 1 + 0 = 01; 1 + 1 = 10). Т.е. эта схема формирует сигнал суммы S и сигнал переноса в старший разряд C, принимающий значение 1 при сложении двух единиц (1+1).

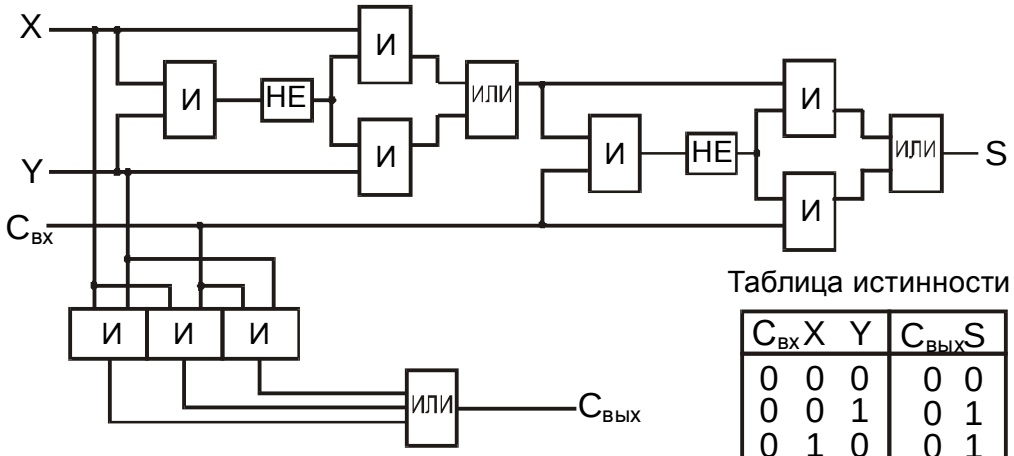

Таблица истинности

Cвх	X	Y	Cвых	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Рис.3.6

Полный сумматор, схема которого изображена на рис.3.6, в отличие от *полусумматора*, позволяет не только складывать одноразрядные двоичные числа и формировать сигнал переноса в старший разряд, но и учитывать при их сложении сигнал переноса $C_{вх}$ из предыдущего разряда.

3.3. Многоразрядный сумматор и АЛУ

Из одноразрядных полных сумматоров можно собирать устройства, уже осуществляющие сложение многоразрядных двоичных чисел. На рис.3.7 представлена возможная схема *четырехразрядного сумматора*.

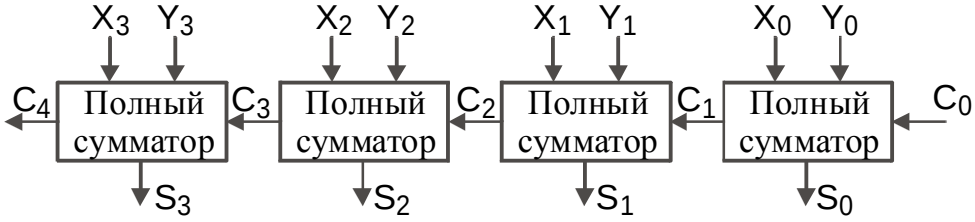

Рис.3.7

Устройство сложения является главной частью так называемого *арифметико-логического устройства (АЛУ)*, компонента центрального процессора ЭВМ, осуществляющего *арифметические* и *логические* операции над двоичными числами. Арифметико-логическое устройство обычно изображается на схемах следующим образом (рис.3.8).

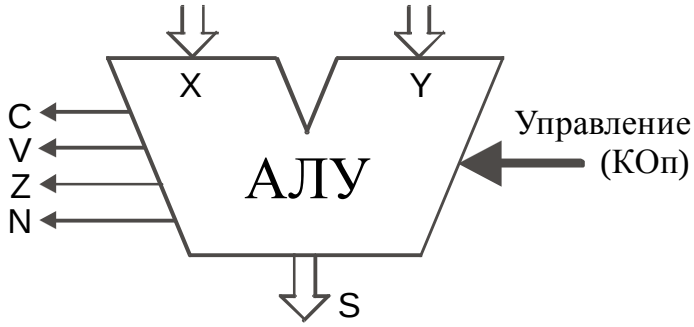

Рис.3.8.

АЛУ обычно выполняет следующие операции над входными операндами X и Y: арифметические операции – *сложение* и *вычитание* (отрицательные числа представляются в двоично-дополнительном коде) и логические операции – И, ИЛИ, НЕ.

Выбор той или иной операции, выполняемой АЛУ осуществляется подачей по специальной шине управления на его соответствующие входы *кода этой операции (КОп)*, т.е. двоичного числа с фиксированным количеством разрядов, каждому значению которого соответствует конкретная операция АЛУ. Кроме выходной шины S – результата операции, АЛУ обычно имеет

еще четыре выхода, обозначенных на рис.3.8 буквами C, V, Z и N, называемых выходами *признаков* или *флагов состояния*:

- единица на выходе C показывает наличие при выполнении операции *переноса из старшего разряда*,
- сигнал на выходе V является *признаком арифметического переполнения*,
- сигнал Z является *признаком нулевого результата*,
- сигнал N является *признаком отрицательного результата* (старший разряд результата равен единице).

3.4. Контрольные вопросы

1. Логическая операция И, ее таблица истинности и возможная схемная реализация.
2. Логическая операция ИЛИ, ее таблица истинности и возможная схемная реализация.
3. Логическая операция НЕ, ее таблица истинности и возможная схемная реализация.
4. Полусумматор, его таблица истинности и возможная схемная реализация с помощью вентильных логических элементов.
5. Одноразрядный полный сумматор, его таблица истинности и возможная схемная реализация с помощью вентильных логических элементов.
6. Многоразрядный двоичный сумматор. Как соединить однорарядные сумматоры для сложения многоразрядных двоичных чисел?
7. Арифметико-логическое устройство, его назначение, назначение его входов и выходов.

4. Элементы памяти ЭВМ

4.1. Триггеры

Элементарные вентильные логические схемы позволяют реализовать не только устройства для выполнения логических и арифметических операций,

но и устройства, обладающие свойством запоминания состояний двоичных сигналов, а следовательно, и двоичных чисел. Устройство, запоминающее состояние одной двоичной переменной, называется триггером. На рис.4.1 представлены схема и условное обозначение так называемого D-триггера.

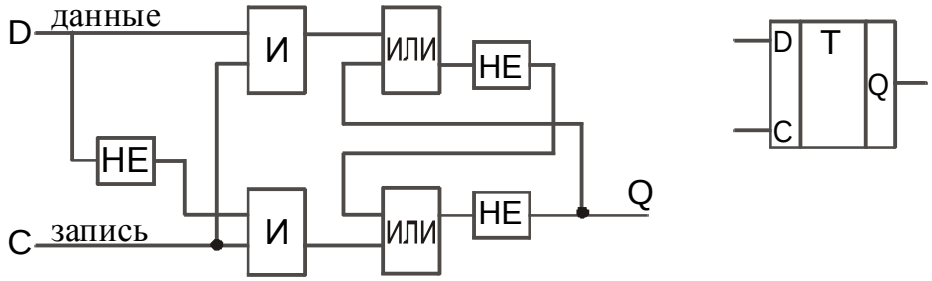

Рис.4.1

На рис.4.2 представлены временные диаграммы сигналов на входах и выходе триггера, причем низкий уровень сигнала «0» соответствует состоянию F или ЛОЖЬ, а высокий «1» соответствует состоянию T или ИСТИНА.

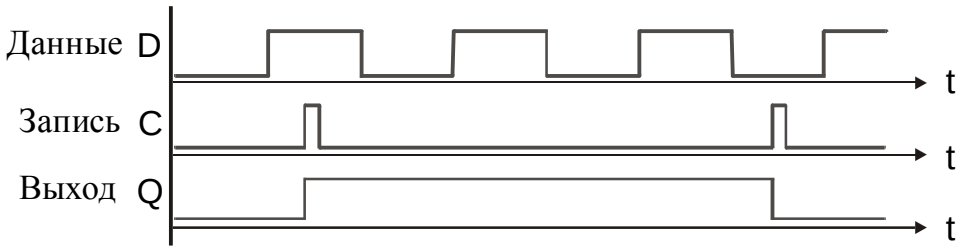

Рис.4.2

На вход D триггера поступают *данные* (запоминаемый двоичный сигнал). На вход C поступает сигнал *записи*, при переходе которого из состояния «0» в состояние «1» входной сигнал D передается на выход триггера Q, а при переходе от «1» к «0» запоминается (записывается) триггером до того момента, когда сигнал записи снова перейдет в состояние «1».

4.2. Организация запоминающего устройства с произвольной выборкой

Несколько триггеров могут объединяться в запоминающие *регистры*, в которых уже можно запоминать не один двоичный разряд, а двоичное число, количество разрядов которого совпадает с количеством триггеров в регистре.

Из таких регистров-ячеек на триггерах можно строить устройства для оперативного запоминания большого количества двоичных чисел. Возможная организация такого запоминающего устройства приведена на рис.4.3.

Одноименные разряды входных шин всех запоминающих ячеек объединяются и образуют общую *шину входных данных*, аналогично, объединенные одноименные выходы соответствующих разрядов ячеек образуют *выходную шину данных*. В конкретный момент времени запись или чтение могут производиться только в *один* из регистров ячеек памяти, выбор которой осуществляется *сигналом выбора*. Этот сигнал формируется входящим в запоминающее устройство специальным блоком, называемым *дешифратором адреса*. Дешифратор – это цифровая схема, имеющая k входов и 2^k выходов. На вход дешифратора адреса подается в двоичном коде номер выбираемой ячейки памяти, называемый *адресом ячейки*. В соответствии с этим адресом дешифратор формирует на одном из своих выходов сигнал выбора для конкретной ячейки памяти. Для фиксации адреса выбираемой ячейки памяти служит *регистр адреса памяти*, с выхода которого адресный сигнал и подается на входы дешифратора. Запись информации с входной шины данных в выбранную ячейку памяти осуществляется подачей сигнала на соответствующие входы запоминающих элементов по входной шине *запись/чтение*.

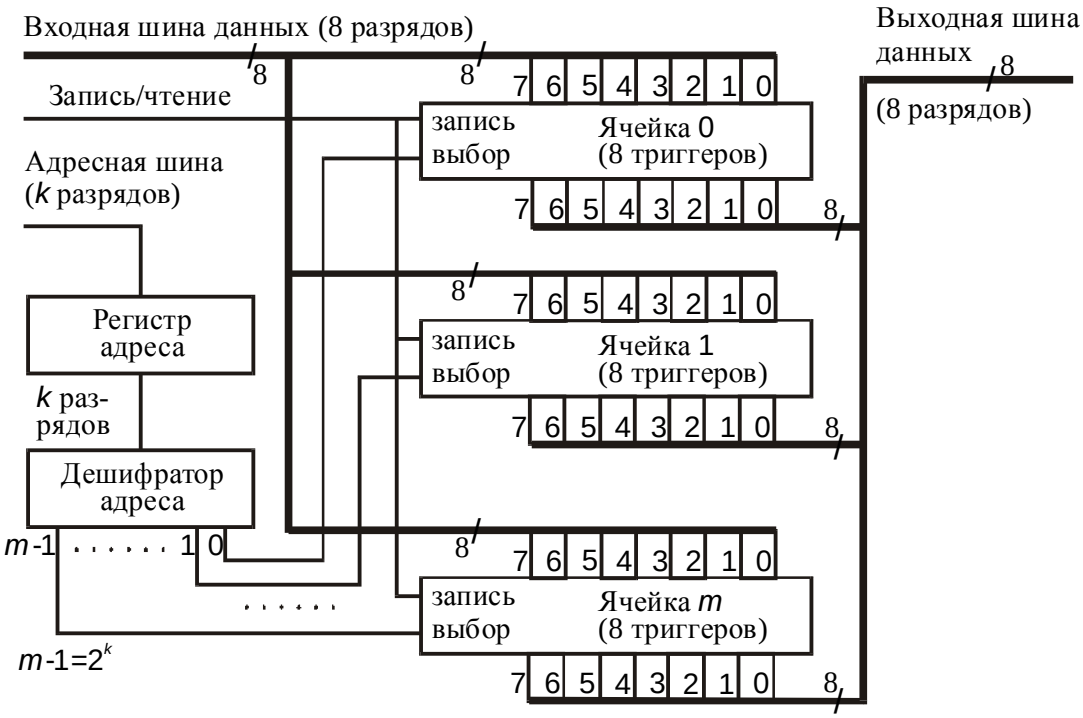

Рис.4.3

4.3. Контрольные вопросы

1. Для чего в ЭВМ нужна память?
2. Триггер, назначение и работа D-триггера.
3. Что такое запоминающий регистр, его назначение?
4. Организация оперативного запоминающего устройства с произвольной выборкой (доступом). Каким образом осуществляется обращение к ячейкам оперативного запоминающего устройства с произвольной выборкой? Возможны ли другие способы доступа к ячейкам памяти?
5. Что такое адрес ячейки памяти ЭВМ?
6. Что такое адресное пространство ЭВМ, чем определяются его размеры?

5. Базовая структура вычислительной системы

На рис.5.1 изображена базовая структура вычислительной системы, отражающая конструктивные элементы большинства ЭВМ с фон-неймановской архитектурой.

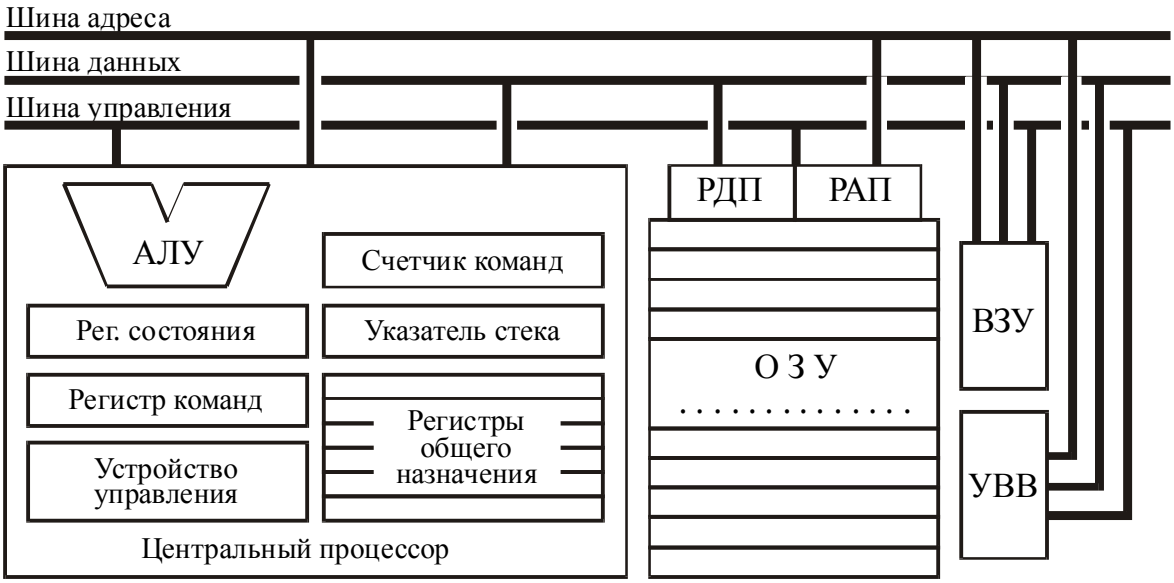

Рис.5.1

Входящие в состав ЭВМ *центральный процессор (ЦП)*, *оперативное запоминающее устройство (ОЗУ)*, *внешние запоминающие устройства (ВЗУ)* и другие *устройства ввода-вывода (УВВ)* информации подключены к трем шинам, по которым передаются данные, адреса и сигналы управления.

Рассмотрим коротко, что представляют собой эти компоненты ЭВМ.

5.1. Память ЭВМ (оперативное запоминающее устройство)

В соответствии с фон-неймановскими принципами организации ЭВМ, ее память – оперативное запоминающее устройство состоит из последовательного набора запоминающих ячеек. Каждая ячейка памяти имеет одинаковое, присущее конкретному типу ЭВМ, количество элементарных запоминающих элементов, предназначенных для запоминания одного бита информации, т.е. одного разряда двоичного числа или кода. В конкретный момент времени можно обратиться к любой, но только к одной ячейке памяти. Для этого в адресном регистре (регистр адреса памяти – РАП) устанавливается адрес (номер) этой ячейки памяти. Адрес ячейки памяти также представляет собой двоичное число с определенным количеством разрядов. Память ЭВМ с таким способом обращения к ее запоминающим ячейкам называется обычно ***памятью с произвольным доступом***.

Записываемые в ячейку или считываемые из нее данные (двоичное число, код) фиксируются в регистре данных памяти (РДП), в первом случае попадая туда по шине данных от центрального процессора, а в случае операции чтения – из выбранной ячейки памяти, подготавливаясь для передачи по шине данных к соответствующему приемнику этих данных. Типичной для большинства ЭВМ является *байтовая* организация памяти. При этом элементарные битовые запоминающие элементы сгруппированы в ячейки по восемь разрядов-битов (в байт) и каждой такой байтовой ячейке присвоен уникальный адрес в некотором адресном пространстве. Адресное пространство ЭВМ представляет собой множестве всех возможных адресов – от нулевого адреса до некоторого максимального адреса, определяемое размером физической памяти ЭВМ и разрядностью ее адресного слова. Например, с помощью 16-разрядного двоичного адреса можно обращаться (адресоваться) к 2^{16} (65536 или 64К) ячейкам. При такой байтовой организации памяти информационные единицы (команды, числа, коды), имеющие число разрядов больше восьми (16, 32 и т.д.), размещаются в соответствующем количестве последовательных байтовых ячеек памяти.

Таким образом, в памяти ЭВМ в виде последовательно расположенных двоичных чисел записываются команды выполняемой ЭВМ программы и данные, участвующие в процессе выполнения программы в качестве исходных операндов, промежуточных и конечных результатов выполняемых ЭВМ операций. Как уже говорилось, в фон-неймановских ЭВМ сами коды команд-инструкций и данные, находящиеся в ячейках памяти не имеют каких-либо признаков, позволяющих только по их виду отличать команды от данных и различать типы данных.

Выбор для построения ЭВМ именно такого способа организации памяти, как памяти с произвольным доступом к ячейкам по их адресу, может показаться самоочевидным. В связи с этим следует обратить внимание на то, что возможны и другие способы взаимодействия с памятью, содержащей большое количество ячеек, при которых понятие адреса ячейки не используется. В частности, ниже будет рассмотрен такой альтернативный способ организации памяти в виде *стека*. Выборка информации из памяти может также осуществляться не по месту расположения информации в памяти (адресу), а *ассоциативно* по содержанию хранящейся информации. Память, организованная таким образом, называется *ассоциативной*.

5.2. Центральный процессор

Процессор ЭВМ – это устройство, непосредственно осуществляющее процесс обработки данных и программное управление этим процессом. Процессор дешифрует и выполняет команды программы, организует обращения к оперативной памяти, в нужных случаях инициирует работу периферийных устройств, воспринимает и обрабатывает запросы, поступающие из устройств компьютера и из внешней среды («запросы прерывания»).

Процессор занимает центральное место в структуре ЭВМ, так как он осуществляет управление взаимодействием всех устройств, входящих в состав ЭВМ.

Важнейшими характеристиками процессора ЭВМ являются:

- его *разрядность*, то есть количество разрядов чисел, над которыми он может совершать операции (например, 16-разрядные процессоры PDP-11, Intel 80286, 32-разрядные VAX-11, Intel 80386 и 80486, Pentium, 64-разрядные процессоры Alpha, Pentium II;
- *система команд*, то есть набор команд, которые данный процессор может выполнять;
- *быстродействие* процессора, которое обычно характеризуют количеством команд, выполняемых процессором в секунду. Косвенным показателем быстродействия процессора является его тактовая частота: чем выше частота, тем больше быстродействие.

Как показано на рис.5.1, центральный процессор ЭВМ с фон-неймановской архитектурой обязательно содержит устройство управления, арифметико-логическое устройство и группу регистров.

Устройство управления вырабатывает последовательность управляющих сигналов, инициирующих выполнение микроопераций, обеспечивающих реализацию текущей команды, координирующих работу всех элементов центрального процессора, управляющих процессом обмена информацией с другими блоками ЭВМ.

Арифметико-логическое устройство (операционный блок процессора) предназначено для выполнения арифметических и логических операций над

двоичными числами или кодами. Характер выполняемой АЛУ операции задается командой программы.

Регистры центрального процессора в какой-то мере подобны ячейкам оперативной памяти. Некоторые из них также способны запоминать и хранить двоичные коды. Однако, размещаясь в самом центральном процессоре, его регистры выполняют свои важные специфические функции.

Необходимость наличия в составе процессора группы дополнительных регистров диктуется следующими факторами.

- Память ЭВМ, представляющая собой запоминающее устройство с произвольной выборкой, не позволяет *одновременного* считывания из различных ячеек кода команды, входных операндов для АЛУ, записи результата операции (выходного операнда АЛУ).
- Процессор должен формировать адреса команд выполняемой программы, для выборки их из памяти и последующей интерпретации и выполнения.
- Для формирования адресов ячеек памяти при выборке операндов на практике требуется использовать различные методы.
- Необходимо иметь средство контроля результата операций, выполненных АЛУ, например, наличия или отсутствия переноса из старшего разряда, наличия или отсутствия арифметического переполнения.
- Процессор должен обеспечивать возможность использования структур программ более сложных, чем простая линейная, путем применения подпрограмм, режима прерывания программ.

Для этих целей в составе фон-неймановского процессора используются следующие регистры.

- *Регистр команд (РК)* предназначен для временного хранения считанного из ячейки памяти кода текущей команды выполняемой ЭВМ программы на время ее выполнения. Эта команда интерпретируется или дешифрируется в устройстве управления процессора, обеспечивая выработку этим устройством управляющих воздействий на АЛУ, другие элементы процессора и ЭВМ для реализации действий, соответствующих выполняемой команде. В частности, при этом определяется, какая

операция должна быть выполнена, над чем, куда поместить результат операции.

- *Регистр–счетчик команд (СК)* (Programm Counter – PC) предназначен для формирования адреса выбираемой команды, т.е. адреса ячейки памяти, из которой должна быть прочитана следующая команда выполняемой программы. Этот адрес может быть принудительно помещен (записан) специальной командой выполняемой программы в регистр–счетчик команд. Однако в случаях, когда выполняемая команда никакой информации об адресе следующей команды не содержит, процессор управляет регистром–счетчиком команд таким образом, что после считывания команды из указанной им ячейки памяти и перемещения этой команды в регистр команд процессора, *содержимое регистра–счетчика команд автоматически увеличивается и становится равным адресу следующей по порядку команды*. Так, например, при байтовой организации памяти в ЭВМ, использующей для кодирования команд 16-разрядное слово (два байта), после считывания очередной команды из памяти содержимое счетчика команд автоматически увеличится на два и будет указывать на адрес следующей по порядку команды. Таким образом, *содержимое ячейки памяти, адрес которой в данный момент находится в счетчике команд, всегда интерпретируется процессором, как* ***код команды****, а не* ***данные.***
- *Регистры общего назначения (РОН)* используются процессором как временная быстродействующая память для входных и выходных данных (операндов) арифметико-логического устройства (АЛУ), а также для реализации разнообразных методов адресации операндов, т.е. способов указания местонахождения операндов, используемых в командах ЭВМ. Число разрядов в регистрах общего назначения и других перечисленных регистрах обычно совпадает с разрядностью центрального процессора ЭВМ (его АЛУ).
- *Сегментные регистры* и *регистры адреса страниц* используются для решения проблем, связанных с расширением адресуемого пространства и управлением памятью ЭВМ (см. ниже).
- *Регистр-указатель стека (УС)* (Stack Pointer – SP) – это регистр процессора, предназначенный для организации в памяти ЭВМ аппаратно управляемого стека, т.е. совокупности ячеек памяти, доступ

управляемого стека, т.е. совокупности ячеек памяти, доступ к которым осуществляется *не по адресу*, а через так называемую *«вершину стека»*. *Содержимое регистра-указателя стека интерпретируется процессором как адрес ячейки памяти, являющейся вершиной стека*. Подробнее об организации и использовании стека будет говориться ниже в соответствующих разделах. Стек, в частности, используется при реализации в ЭВМ механизмов работы с подпрограммами и обслуживания внешних и внутренних прерываний программы.

- Важную специальную роль играет в процессоре регистр, называемый *регистром состояния процессора (PC)*. Он предназначен для отражения текущего состояния процессора и для установки его режима. В определенных разрядах этого регистра указывается, в частности, состояние результата выполнения процессором текущей операции (команды).

  Это разряды-признаки (флаги состояния):
  - переноса из старшего разряда (C),
  - арифметического переполнения (V),
  - нулевого результата (Z),
  - отрицательного результата (N).

5.3. Шинная организация ЭВМ

Шина в вычислительной системе – это среда, представляющая собой соответствующим образом выполненные линии связи, через которую компоненты ЭВМ связываются друг с другом. Большинство ЭВМ имеют шинную организацию, что позволяет существенно сократить общее число линий связи между блоками ЭВМ. Как показано на рис.5.1, обычно шина ЭВМ функционально делится на три группы линий связи: *адресную шину*, *шину данных* и *шину управления*. Адресная шина переносит информацию о том, где искать инструкции (команды) или данные в памяти ЭВМ, то есть адреса соответствующих ячеек памяти; шина данных переносит эти данные или инструкции для центрального процессора; шина управления обеспечивает передачу сигналов управления между процессором и подключенными к ЭВМ устройствами.

5.4. Внешние устройства

Наиболее часто в вычислительных системах применяются следующие периферийные устройства:

- *внешние запоминающие устройства* (ВЗУ), предназначенные для *долговременного* хранения *больших* объемов информации, это – накопители на магнитных дисках, на магнитных лентах, CD ROM и др.;
- *клавиатура* – устройство, предназначенное для ввода информации в ЭВМ, осуществляет генерацию двоичного кода в соответствии с нажатой клавишей или комбинацией клавиш;
- манипуляторы *мышь* и *джойстик*;
- *видеомонитор* – устройство для отображения на экране выводимой пользователю информации;
- *печатающие устройства* и *графопостроители*, предназначенные для вывода текстовой или графической информации путем ее печати на бумаге;
- *сканеры* – устройства для ввода изображения;
- *модемы* – для обмена информацией между ЭВМ по телефонным линиям и многие другие устройства.

5.5. Контрольные вопросы

1. Общая структура вычислительной системы, назначение ее элементов.
2. Процессор ЭВМ, его компоненты и их назначение.
3. Какой элемент процессора предназначен для указания адреса команды, которую следует выбирать из памяти для выполнения?
4. Назначение регистра-счетчика команд, почему он так называется?
5. Для чего в процессоре нужно устройство управления?
6. Для чего нужен регистр команд процессора?
7. Для чего нужен регистр состояния процессора?
8. Для чего нужен в процессоре регистр-указатель стека?
9. Регистры общего назначения, для чего они используются?
10. Что такое разрядность ЭВМ, параметры каких элементов ЭВМ она определяет?
11. Регистр адреса памяти, регистр данных памяти, где они расположены, их назначение?
12. Чем определяется размер физической памяти ЭВМ?

13. Можно ли обращаться к ячейкам памяти, не указывая их адреса?
14. Что такое канал (шинная организация) ЭВМ?

6. Упрощенный цикл выполнения команд в ЭВМ

6.1. Цикл выполнения команд

Исходя из изложенной выше информации и базовой структуры ЭВМ, рассмотрим последовательность операций, осуществляемую процессором при выполнении команд выполняемой программы. Программа, которую выполняет ЭВМ, обычно записана в последовательных ячейках ее памяти в виде последовательности закодированных команд (инструкций). Для того, чтобы процессор начал выполнение программы, необходимо поместить адрес ячейки памяти, в которой находится первая команда программы, в регистр-счетчик команд (СК) процессора. Работа процессора при этом представляет собой последовательность следующих операций.

- *Процессор помещает содержимое своего регистра-счетчика команд в регистр адреса памяти (РАП) запоминающего устройства.*
- *Затем из ячейки памяти, адрес которой помещен в регистр адреса памяти, извлекается код команды и через шину данных передается в регистр команд процессора.*
- *После считывания из памяти содержимого ячейки, адрес которой указывается регистром–счетчиком команд, содержимое этого регистра автоматически увеличивается и становится равным адресу следующего по порядку слова. (Например, если команды занимают в памяти ЭВМ два байта, то увеличение счетчика команд производится на два).*
- *Вслед за этим устройство управления процессора начинает интерпретацию команды, находящейся в его регистре команд. Вначале определяется операция, которая должна быть выполнена процессором по данной команде, и в соответствии с ней вырабатываются сигналы для управления элементами процессора, в частности, сигналы, осуществляющие переключение АЛУ в режим выполнения нужной операции.*

Для определенности предположим, что команда, которую в данный момент выполняет процессор, является командой сложения двух чисел–операндов, находящихся в некоторых ячейках памяти.

- *Декодируя код операции, задаваемый командой, устройство управления процессора определяет, в частности, необходимость выборки операндов для выполнения конкретной команды. (В рассматриваемом примере для команды сложения необходимы два операнда).*
- *После этого устройство управления в результате дальнейшей интерпретации команды определяет адрес первого операнда и помещает его в регистр адреса памяти (РАП).*
- *Содержимое этой ячейки, т.е. первый операнд, считывается и по шине данных передается в один из регистров процессора, подключенного к входу первого операнда АЛУ.*
- *Затем аналогичным образом определяется адрес второго операнда, производится его считывание в регистр и установка на другом входе АЛУ.*
- *После выборки и подготовки необходимых команде операндов соответствующая ей операция выполняется (в данном примере происходит сложение выбранных операндов).*
- *Результат операции пересылается в память ЭВМ по адресу, также определяемому устройством управления в результате интерпретации кода команды.*

На этом цикл выполнения команды заканчивается, процессор снова помещает содержимое своего регистра–счетчика команд (уже увеличенное) в регистр адреса памяти, производит считывание следующей по порядку команды в свой регистр команд и повторяет цикл декодирования и выполнения команды.

6.2. Контрольные вопросы

1. Цикл выполнения команды ЭВМ (на примере выполнения команды сложения двух чисел, находящихся в ячейках памяти ЭВМ).
2. Цикл выполнения команды ЭВМ (на примере выполнения команды условного перехода).

7. Система команд ЭВМ и адресация операндов

7.1. Типы команд

Важной характеристикой ЭВМ, в большой степени определяющей разделение ЭВМ на различные типы, является *система команд ЭВМ.*

Команда представляет собой код, определяющий операцию вычислительной машины и данные, участвующие в операции. Команда содержит также в явной или неявной форме информацию об адресе, по которому размещается результат операции, и об адресе следующей команды. Под системой команд ЭВМ обычно понимают набор инструкций, которые может выполнять ее центральный процессор, способы кодирования этих инструкций и методы указания местонахождения операндов. Мощность и гибкость системы команд во многом определяют качественные показатели ЭВМ в целом.

Набор команд ЭВМ обычно включает в себя большое количество (от нескольких десятков до сотен) команд различного назначения. Назначение команд существенным образом влияет на способы их кодирования и длину.

По характеру выполняемых операций различают следующие основные группы команд: команды арифметических операций, команды логических (поразрядных) операций (И, ИЛИ, НЕ и др.), команды пересылки кодов между компонентами ЭВМ, команды управления порядком выборки и исполнения команд (команды передачи управления), команды задания режима работы процессора и др.

В команде, как правило, содержатся не сами операнды, а информация об адресах ячеек памяти или регистрах, в которых они находятся. Код команды можно представить состоящим из нескольких частей или полей, имеющих определенное функциональное назначение при кодировании командной информации. Как показано на рис. 7.1, командный код в общем случае состоит из *операционной* части и *адресной* части.

Рис. 7.1

Каждая из этих частей, в свою очередь, может состоять из нескольких полей. Это особенно характерно для адресной части.

Операционная часть содержит *код операции* (КОп), который задает вид операции (сложение, пересылка, переход и т.д.). *Адресная часть* команды содержит информацию о местонахождении (адресах) операндов и результата операции, а в некоторых случаях и об адресе следующей команды. *Форматом команды* называют разметку номеров разрядов (бит) кода команды, определяющих расположение и границы отдельных полей команды.

Важной и сложной проблемой при проектировании ЭВМ является выбор структуры и формата команды, назначения и длины отдельных ее полей. Естественно стремление разместить в команде в возможно более полной форме информацию о предписываемой командой операции и местонахождению необходимых операндов и результата операции. Однако в условиях современных ЭВМ, когда значительно возросло количество выполняемых ЭВМ команд и емкость адресуемой памяти (длина адресного слова), это приводит к недопустимо большой длине кода команды и усложнению ее формата. Решение выбора формата команд особенно усложняется в малых ЭВМ и микропроцессорах, работающих с коротким машинным словом.

Простейшим типом команд являются так называемые *безадресные команды*, т.е. команды, для выполнения которых операнды не требуются. Типичные команды такого типа это: ОСТАНОВ (HALT), ОЖИДАНИЕ ВНЕШНИХ СОБЫТИЙ (WAIT), НЕТ ОПЕРАЦИИ (NOP) и др. У этих команд всё отводимое под команду машинное слово представляет собой код операции.

Более сложными типами команд являются команды, для выполнения которых требуется один или несколько операндов. В этом случае при кодировании команды, помимо кода, описывающего выполняемую командой функцию, необходимо тем или иным образом указывать участвующие в операции входные операнды (операнды – источники данных) или их местонахождение (в регистрах или памяти), а также место, куда должен быть помещен результат операции (обычно называемый приемником данных).

В связи с этим, в зависимости от количества операндов, в ЭВМ можно представить следующие типы разновидности команд:

- *одноадресные команды*, оперирующие только одним операндом (например, команды ОЧИСТИТЬ ЯЧЕЙКУ ПАМЯТИ, УВЕЛИЧИТЬ ЗНАЧЕНИЕ ЯЧЕЙКИ НА ЕДИНИЦУ, ИЗМЕНЕНИТЬ ЗНАК ЧИСЛА и другие;
- *двухадресные команды*, для работы которых необходимо указание двух адресов (например, команда КОПИРОВАТЬ СОДЕРЖИМОЕ ОДНОЙ ЯЧЕЙКИ В ДРУГУЮ);
- *трехадресные команды*, например команда – СЛОЖИТЬ СОДЕРЖИМОЕ ДВУХ ЯЧЕЕК И РЕЗУЛЬТАТ ПОМЕСТИТЬ ПО ТРЕТЬЕМУ АДРЕСУ;
- *четырехадресные*, см. пример команды приведенной выше, но в которой добавлено указание адреса команды, которая должна выполняться после данной.

Для адресации каждого операнда в коде команды выделяется специальное поле (несколько разрядов). Очевидно, что количество указываемых в команде операндов и способ указания их адресов существенным образом влияют на длину команды (количество разрядов необходимых для кодирования команды). В связи с этим во многих ЭВМ ограничиваются одноадресными и двухадресными командами. В этом случае, как, например, в команде сложения, требующей указания адресов двух входных операндов и третьего адреса для помещения результата операции, ограничиваются указанием только двух адресов, помещая результат операции, т.е. операнд–приемник, по адресу одного из двух входных операндов–источников. Естественно, что при этом после выполнения операции значение одного из операндов будет утеряно.

Учитывая, что адрес операнда занимает одно машинное слово, при непосредственном указании в команде адресов операндов, можно размещать команду в памяти ЭВМ в нескольких следующих подряд ячейках памяти. Однако, проблема адресации операндов, т.е. способов указания их местонахождения, на самом деле является более сложной, чем просто проблема увеличения длины команды. Обычно в ЭВМ используется не один, а несколько способов формирования исполнительного адреса операнда по информации, указываемой в соответствующем поле команды. Это связано, в частности, с необходимостью обеспечения эффективного преобразования в машинные коды программ, написанных на языках высокого уровня, с использованием при программировании разнообразных способов структурной организации

данных (массивов, стеков, очередей и др.). Например, в качестве исполнительного адреса операнда может указываться один из регистров процессора, адрес ячейки памяти, в команде может указываться адрес ячейки памяти, содержащей адрес операнда или смещение этого адреса, относительно адреса самой команды и др. Обычно, чем мощнее ЭВМ, тем разнообразнее методы адресации операндов. Рассмотрим типичные для большинства ЭВМ методы адресации операндов.

7.2. Способы адресации операндов

Следует различать понятия *адресный код* в команде и *исполнительный адрес*. Адресный код – это *информация об адресе* операнда, содержащаяся в команде. Исполнительный адрес – это сам *адрес* (номер) ячейки памяти, к которой производится фактическое обращение. Как правило, адресный код не совпадает с исполнительным адресом.

Выбор способов адресации, формирования исполнительного адреса и преобразования адресов является одним из важнейших вопросов разработки ЭВМ ее системы команд.

Рассмотрим методы адресации, используемые в современных ЭВМ.

Подразумеваемый операнд. В команде не содержится явных указаний об адресе операнда; операнд подразумевается и фактически задается кодом операции команды. Например, в команде УВЕЛИЧИТЬ ЧИСЛО НА ЕДИНИЦУ первый операнд (увеличиваемое число должен адресоваться явным образом), а второй (приращение на единицу) – не адресуется, в памяти ЭВМ не содержится и является подразумеваемым.

Подразумеваемый адрес. В команде не содержится явных указаний об адресе участвующего в операции операнда или адреса, по которому помещается результат операции, но этот адрес подразумевается. Например, команда может содержать адреса обоих операндов, участвующих в операции, при этом подразумевается, что результат операции помещается по адресу одного из операндов, или команда указывает только адрес одного операнда, а адрес

второго, которым является содержимое специального регистра (называемого регистром результата или аккумулятором), подразумевается. Аналогично в самой команде можно обойтись без указания адреса следующей команды в случае, когда последовательно выполняемые команды программы располагаются друг за другом в следующих подряд ячейках памяти. В этом случае при выполнении команды *подразумевается*, что адрес следующей команды может быть получен из адреса текущей путем его простого наращивания (инкрементирования содержимого счетчика команд).

Непосредственная адресация. В команде содержится не адрес операнда, а *непосредственно* сам операнд. При непосредственной адресации не требуется обращения к памяти для выборки операнда и ячейки для его хранения. Это способствует уменьшению времени выполнения программы и занимаемого программой объема памяти. Ниже будет рассмотрена реализация этого способа адресации, использующая при определении адреса операнда регистр-счетчик команд процессора.

Относительная адресация или *базирование*. Исполнительный адрес определяется суммой адресного кода и некоторого числа, называемого *базовым адресом*. Адресный код в этом случае играет роль *смещения* фактического исполнительного адреса *относительно базового* адреса.

Регистровая адресация. При адресации операнда используется содержимое указанного в команде какого-либо регистра процессора.

Косвенная адресация. Адресный код команды указывает адрес ячейки памяти, в которой находится адрес операнда или команды. Другими словами, косвенная адресация может быть определена как «адресация адреса». В некоторых ЭВМ используется многоступенчатая косвенная адресация.

Ниже рассматривается использование для реализации различных методов адресации регистров процессора.

7.3. Режимы адресации с помощью регистров общего назначения

1. Регистровый метод адресации.

При использовании этого метода адресации в команде указывается регистр общего назначения (например, его номер) и содержимое этого регистра интерпретируется процессором как операнд.

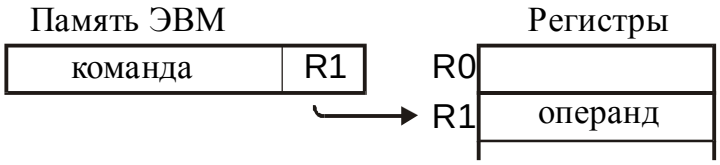

Рис.7.2

2. Косвенно-регистровый метод адресации.

Содержимое указанного в команде регистра интерпретируется процессором как адрес ячейки памяти, в которой находится операнд.

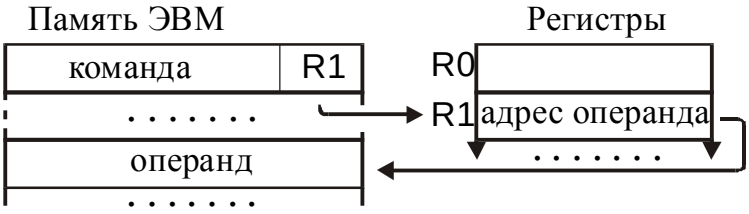

Рис.7.3

3. Автоинкрементный метод адресации *(адресация с автоувеличением).*

Содержимое указанного в команде регистра интерпретируется процессором как адрес ячейки памяти, в которой находится операнд (как в косвенно-регистровом методе), но после выборки операнда содержимое регистра увеличивается (инкрементируется) обычно на 1 или 2, таким образом, чтобы указывать на адрес следующей по порядку ячейки памяти.

4. Косвенно-автоинкрементный метод адресации.

Содержимое указанного в команде регистра интерпретируется процессором как адрес ячейки памяти, в которой находится *адрес* операнда, и после выборки операнда содержимое регистра (адрес адреса) увеличивается (инкрементируется), таким образом, чтобы указывать на адрес следующей по порядку ячейки.

***5. Автодекременный метод адресации** (адресация с автоуменьшением).*

При интерпретации команды содержимое указанного в команде регистра вначале уменьшается (инкрементируется) обычно на 1 или 2, после чего уменьшенное содержимое регистра интерпретируется процессором как адрес ячейки памяти, в которой находится операнд.

6. Косвенно-автодекременный метод адресации.

При интерпретации команды содержимое указанного в команде регистра вначале уменьшается (инкрементируется), после чего уменьшенное содержимое регистра интерпретируется процессором как адрес ячейки памяти, в которой находится *адрес* операнда.

Наличие в ЭВМ *автоинкрементного* и *автодекрементного* методов адресации позволяет, например, удобно организовывать работу с одномерными массивами данных, размещенных в последовательных ячейках памяти.

7. Косвенная адресация со смещением (индексный метод адресации).

При использовании этого метода адресации инструкция (команда) занимает в памяти на одно слово (ячейку) больше. В дополнительном слове команды указывается так называемое смещение (индексное слово). *Исполнительный адрес операнда определяется в этом случае как сумма содержимого указанного в команде регистра и смещения (индексного слова).*

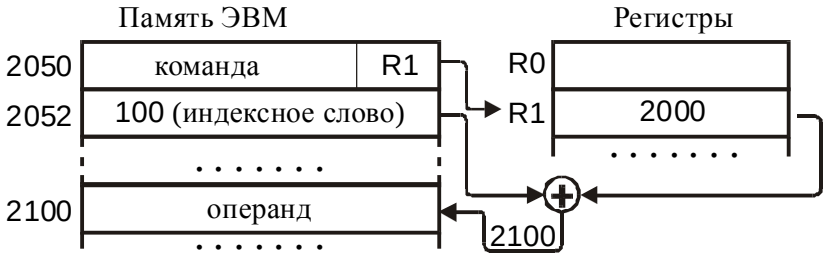

Рис.7.8

Т.е. число, находящееся во втором слове команды, указывает смещение фактического адреса операнда относительно адреса, содержащегося в указанном в команде регистре общего назначения. Это смещение может быть как положительным числом, так и отрицательным, т.е. сдвигать указанный в регистре адрес, как в сторону старших адресов, так и в сторону младших ад-

ресов. Ссылку на содержимое данного регистра как на базовый адрес могут осуществлять и другие команды, используя при этом свои значения фактического смещения адреса.

7.4. Режимы адресации со ссылкой на регистр-счетчик команд

В этих режимах адресации при формировании исполнительного адреса операнда используется текущее содержимое *регистра-счетчика команд*.

1. Непосредственный метод адресации.

При использовании этого режима адресации *операнд указывается непосредственно в команде*, после кода операции. Процессор получает в этом случае адрес операнда непосредственно из своего регистра–счетчика команд.

Для реализации этого метода адресации фактически используется автоинкрементный метод адресации через регистр–счетчик команд. Действительно, как уже говорилось выше, при считывании процессором из памяти команды, адрес которой находится в его счетчике команд, содержимое этого регистра автоматически увеличивается и становится равным адресу следующей за считанным командным словом ячейки памяти, т.е. адресу указанного во втором слове команды операнда. После считывания операнда из этого адреса процессор снова автоматически увеличивает содержимое регистра-счетчика команд, после чего его содержимое будет указывать на следующую за операндом ячейку памяти, в которой может размещаться следующая команда программы.

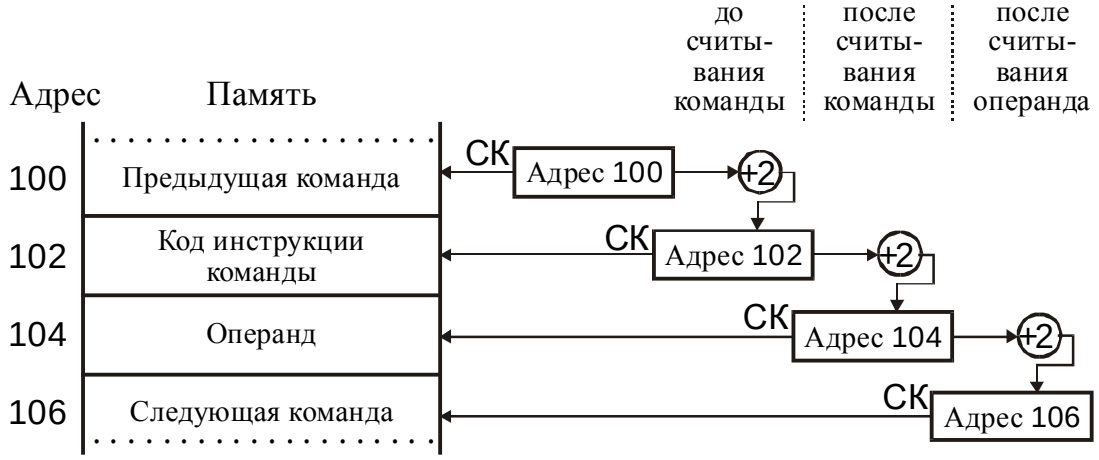

Рис.7.9

2. *Абсолютный метод адресации.*

При использовании данного метода адресации *во втором слове команды, т.е. в следующей за кодом команды ячейке памяти, указывается адрес операнда*. Фактически это косвенно–автоинкрементный метод адресации через регистр–счетчик команд.

Рис.7.10

Как обычно, после считывания процессором из ячейки памяти кода команды его регистр-счетчик команд указывает на адрес следующей за кодом команды ячейки памяти, и при указании в команде этого метода адресации процессор интерпретирует содержимое регистра–счетчика команд уже как адрес ячейки памяти, содержащей адрес операнда. После считывания операнда содержимое счетчика команд снова автоматически наращивается (благодаря использованию косвенно–автоинкрементного метода) и становится равным адресу следующей ячейки памяти.

3. *Относительный метод адресации.*

Во втором слове команды указывается относительный адрес операнда, т.е. величина смещения адреса операнда относительно адреса самой команды (текущего содержимого регистра-счетчика команд процессора).

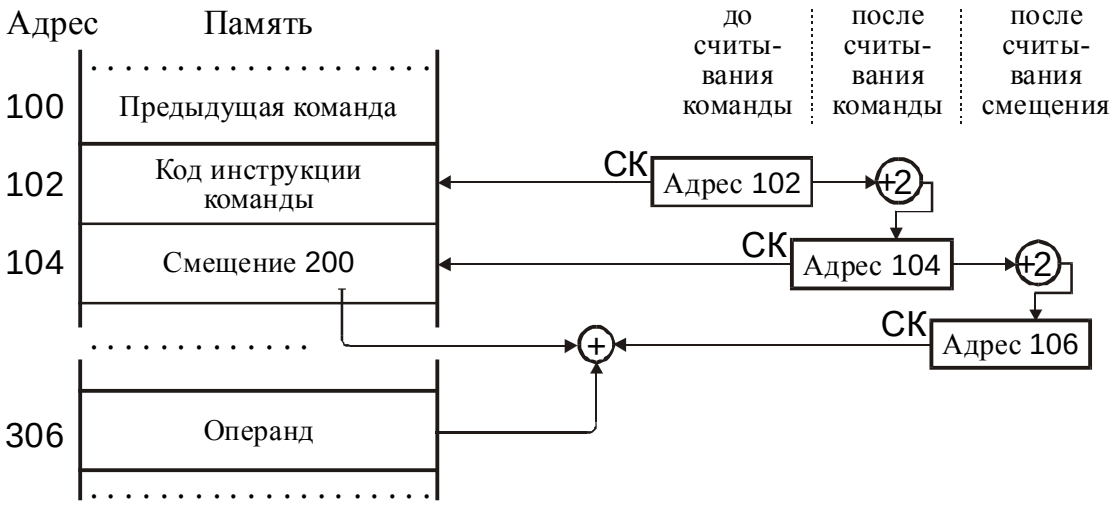

Рис.7.10

7.5. Стек. Организация стека в памяти ЭВМ

Стек – это один из способов организации памяти, отличающийся от уже рассмотренной выше организации памяти с *произвольным доступом*. Как уже говорилось, в память с произвольным доступом представляет собой набор одинаковых ячеек памяти, доступ к которым для чтения или записи осуществляется *независимо от их расположения* в памяти путем *указания адреса* требуемой ячейки памяти. Память ЭВМ, организованная в виде стека, также включает в себя определенное количество одинаковых запоминающих ячеек, однако доступ к этим ячейкам для записи или чтения не может осуществляться произвольным образом, т.е. в любое время к любой ячейке.

Доступ к ячейкам стека осуществляется *не по адресам* ячеек, а только через так называемую «*вершину стека*» – ячейку памяти стека, доступную в данный момент времени.

При последовательной записи данных (кодов, чисел) в стек загружаемые данные последовательно помещаются на «вершину стека», при этом ранее записанные значения как бы ***проталкиваются*** *вглубь стека. В каждый момент времени для чтения в стеке доступно только слово, находящееся на его вершине. При этом прочитать слово, находящееся на вершине стека, можно только один раз, так как после чтения это слово «выталкивается» из стека и на его вершине оказывается слово, записанное перед вытолкну-*

тым. Таким образом, записанные в стек данные могут быть последовательно прочитаны только в порядке, обратном порядку их записи в стек путем последовательного «выталкивания» этих данных из глубины стека на его вершину. При этом информация о вытолкнутых данных теряется.

Обычно такой способ доступа к данным коротко формулируют в виде: «*последним пришел – первым вышел*».

Аналогично еще один используемый в ЭВМ способ организации данных, называемый очередью, формулируют в виде: «*первым пришел – первым вышел*» (или «*последним пришел – последним вышел*»).

Оставляя пока в стороне вопросы целесообразности использования стека процессором ЭВМ, рассмотрим реализацию этого, как будет видно ниже, важного вида памяти. Обычно в ЭВМ стек реализуется в адресном пространстве ее оперативной памяти выделением под него определенного количества последовательных ячеек. Специфический способ доступа к ячейкам памяти, выделенным под стек, осуществляется путем специального режима адресации с использованием одного из регистров процессора, называемого ***регистром – указателем стека*** (УС или SP).

При работе со стеком *содержимое регистра – указателя стека интерпретируется процессором как адрес «вершины стека»*. Более точно – как адрес выделенной под стек ячейки памяти, в которой находится число, записанное в стек последним. В связи с этим процедура начальной установки состояния стека использующей его программой (определение выделяемой под стек области памяти) состоит в начальной установке регистра – указателя стека, а именно, записи в него адреса ячейки памяти, являющейся «вершиной» пустого стека. Общепринятым является заполнение стека данными в сторону уменьшения адресов его ячеек.

Запись числа в стек, организованный в памяти ЭВМ, иллюстрируется на рис.7.11.

Вначале процессор корректирует содержимое регистра-указателя стека, смещая его на одну ячейку в сторону уменьшения адресов (при четных адресах ячеек памяти уменьшая его на два), после чего в указываемую этим регистром ячейку памяти пересылается записываемое число.

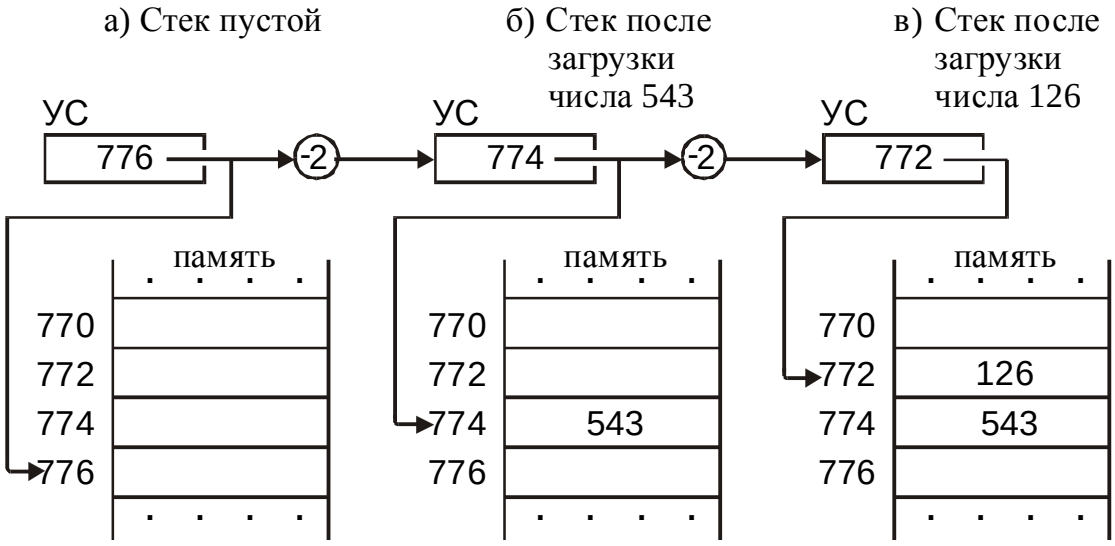

Рис.7.11

При засылке в стек следующего числа опять происходит коррекция, т.е. уменьшение содержимого указателя стека, и запись следующего числа уже в новую ячейку памяти. Таким образом, *регистр – указатель стека всегда содержит адрес последней записанной ячейки стека, т.е. указывает на вершину стека*.

Метод адресации ячеек памяти, который используется при засылке чисел в стек, носит название *косвенная адресация через регистр (в данном случае указатель стека) с предварительным автоуменьшением его содержимого* (рассмотренный выше автодекрементный метод адресации). В некоторых ЭВМ такая адресация может использоваться с любым из регистров общего назначения процессора.

Чтение информации из стека или, как еще говорят, выталкивание числа из стека происходит следующим образом (см. рис.7.12).

Число прочитывается из ячейки памяти, адрес которой находится в регистре–указателе стека, после чего содержимое этого регистра автоматически корректируется в сторону увеличения адресов. При прочтении следующего числа производится аналогичная операция.

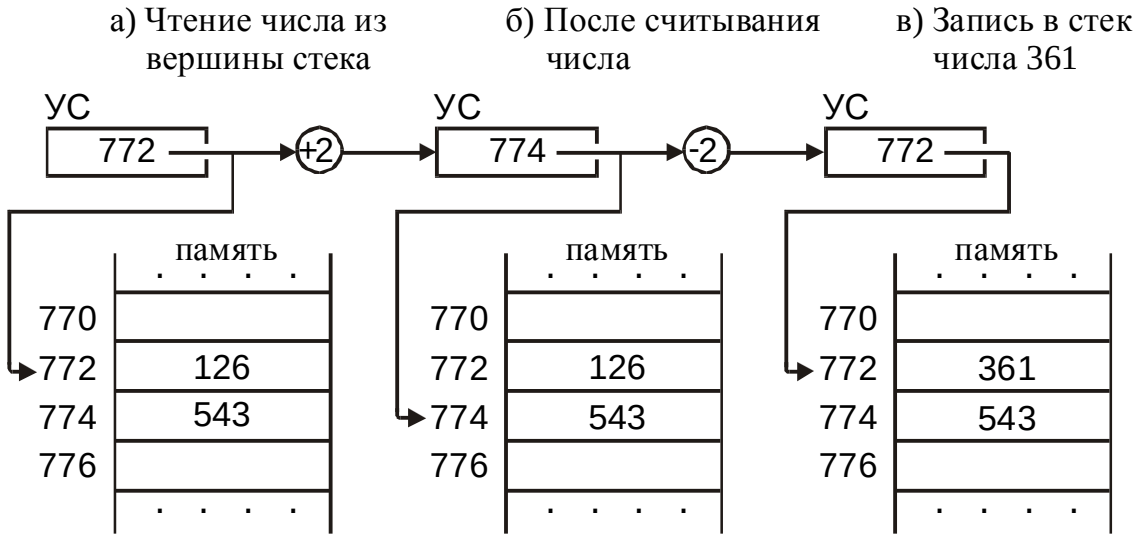

Рис.7.12

Следует обратить внимание на то, что хотя при считывании числа из вершины стека используется термин «*выталкивание*», само число из ячейки стека никуда не девается. Просто коррекция регистра–указателя стека делает содержимое этой ячейки не доступным с помощью механизма работы со стеком. При записи в стек очередного числа, как показано на рис.7.12в, оно будет записано по адресу числа, вытолкнутого (считанного) из стека непосредственно перед этим.

Метод адресации ячеек памяти, используемый при *чтении* чисел из стека, называется *косвенной адресацией через регистр (в данном случае регистр-указатель стека) с предварительным автоувеличением* (рассмотренный выше автоинкрементный метод адресации).

Приведенные выше методы адресации операндов являются типичными для большинства ЭВМ различных типов, хотя используемый для их реализации формат команд может существенно различаться, при этом в каждом конкретном типе ЭВМ они могут дополняться и другими более изощренными методами адресации, увеличивающими гибкость организации различных структур данных при программировании.

7.6. Контрольные вопросы

1. Типы команд ЭВМ.
2. Какая информация кодируется в команде ЭВМ?
3. Какие проблемы приходится решать при кодировании команд ЭВМ?
4. Что такое адресация операндов, методы адресации?
5. Что такое адресный код, исполнительный адрес, чем отличаются эти понятия?
6. Для чего в ЭВМ необходимо наличие разнообразных методов адресации операндов?
7. Что такое подразумеваемый операнд, подразумеваемый адрес?
8. Непосредственная адресация.
9. Абсолютная адресация.
10. Относительная адресация.
11. Регистровая адресация.
12. Косвенная адресация.
13. Регистровый и косвенно-регистровый методы адресации.
14. Косвенная адресация со смещением.
15. Методы адресации, использующие счетчик команд при формировании исполнительного адреса операнда.
16. Использование регистра-счетчика команд при реализации непосредственного, абсолютного и относительного методов адресации.
17. Какой метод адресации следует использовать в перемещаемой программе для адресации данных, расположенных в теле программы?
18. Какой метод адресации следует использовать в перемещаемой программе для адресации регистров внешних устройств?
19. Что такое стек?
20. Организация стека в оперативной памяти ЭВМ.
21. Для чего используется стековая память?
22. Проиллюстрируйте изменение состояния элементов процессора при последовательной записи в стек трех чисел.
23. Какие методы адресации можно использовать для записи числа в стек, для чтения числа из стека?

8. Команды управления выполнением программы

8.1. Команды безусловной и условной передачи управления

Команды управления выполнением программы имеют большое значение, поскольку они позволяют вмешиваться в процесс автоматического формирования процессором адреса следующей выполняемой команды.

Как уже говорилось выше, *процессор всегда выбирает очередную команду из той ячейки памяти, адрес которой в данный момент находится в регистре-счетчике команд*. Осуществляя соответствующее изменение содержимого регистра–счетчика команд процессора, команды передачи управления позволяют вместо тривиальной последовательной выборки команд из памяти реализовать переходы в нужные точки программы. Переходы на адрес, задаваемый командами передачи управления, могут быть *безусловными*, т.е. выполняемыми в любом случае, и *условными*. Выполнение или не выполнение последних зависит от результата выполнения процессором команды, предшествующей команде условного перехода, например, команда – ВЕТВЛЕНИЕ, ЕСЛИ РЕЗУЛЬТАТ ОПЕРАЦИИ РАВЕН НУЛЮ. Команды безусловных и условных переходов позволяют обеспечивать, в частности, возможность реализации таких важных конструкций программирования, как ***циклы***.

Как видно из вышесказанного, действием, совершаемым этими командами, является *принудительное изменение содержимого регистра-счетчика команд* процессора в соответствии со значением задаваемого в команде тем или иным образом адреса перехода к другой команде программы. При указании места перехода обычно используют два способа: в первом указывают *абсолютный* адрес, на который следует осуществить переход, во втором *величину смещения* адреса перехода относительно текущего значения счетчика команд, т.е. адреса самой команды перехода. Первый способ обеспечивает возможность передачи управления в любую точку адресного пространства, но требует больше места для размещения команды и больше времени на ее выполнение. Второй способ обеспечивает более компактное кодирование команд перехода, так как обычно для указания смещения используют всего один байт. Указываемое в команде смещение может быть как положительным, так и отрицательным числом, благодаря чему безусловное или услов-

ное ветвление в программе может осуществляться как в сторону старших, так и в сторону младших адресов относительно текущего содержимого счетчика команд. При этом, если для указания смещения используется один байт, максимальная величина смещения оказывается ограниченной диапазоном ±128 ячеек памяти.

В системе команд ЭВМ существует целый набор разнообразных команд условных ветвлений. Это команды, осуществляющие переход в случае, например, когда результат предыдущей операции равен нулю, больше нуля, больше или равен нулю, меньше нуля и т.д. Каким же образом в процессоре осуществляется анализ результата предыдущей операции при выполнении команд условных переходов?

Как уже говорилось выше, после выполнения процессором каждой команды в зависимости от ее результата устанавливаются разряды–признаки (флаги) Z, N, V и C регистра состояния процессора. Например, после выполнения команды CMP (сравнение двух операндов) по состоянию флажков регистра состояния процессора можно определить соотношение величин сравниваемых операндов: *равны, не равны, больше, меньше* и т.д. При выполнении команд условных ветвлений процессор *анализирует соответствие флажков регистра состояния процессора задаваемому командой условию* и при выполнении этого условия осуществляет модификацию своего регистра-счетчика команд на величину задаваемого в команде смещения (положительного или отрицательного). При невыполнении заданного условия содержимое счетчика команд не корректируется и происходит обычная выборка следующей по порядку команды.

Большое количество разнообразных команд условных ветвлений позволяет обрабатывать переходы по всем возможным условиям, касающимся результата предыдущей операции: *равно/не равно (нулю)*, *больше/меньше*, *не больше*, *не меньше* и т.д., причем при интерпретации сравниваемых операндов как чисел без знака, так и чисел со знаком, а также в зависимости от наличия или отсутствия переноса из старшего разряда и арифметического переполнения.

8.2. Контрольные вопросы

1. Команды безусловной и условной передачи управления (безусловного и условных переходов), их назначение, какие элементы процессора используются при их выполнении и каким образом?
2. Какую роль при выполнении команд условных ветвлений играет регистр состояния процессора?

9. Подпрограммы. Использование стека при вызове подпрограмм и возврате из них

9.1. Проблемы реализации подпрограмм

Подпрограммой обычно называют часть программы, к которой можно многократно обращаться из других, причем разных, мест основной программы с последующим возвратом в место, откуда произошел вызов (см. рис.9.1).

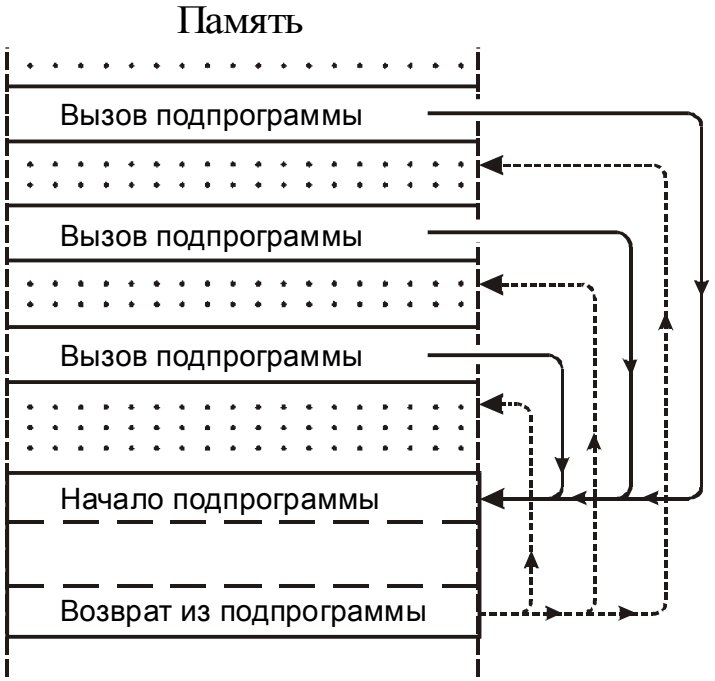

Рис.9.1

Использование механизма подпрограмм является эффективным средством, позволяющим существенно экономить занимаемый программой объем памяти, устраняя дублирование в программе многократно выполняемых

одинаковых действий путем обращения к одному и тому же программному модулю, оформленному в виде подпрограммы. Использование подпрограмм также значительно облегчает написание больших программ, поскольку из подпрограмм можно составлять библиотеки для общего использования. На языках высокого уровня эквивалентным подпрограмме конструкциями являются процедуры и функции.

Очевидно, что при реализации механизма работы с подпрограммами существенными являются два момента: *вызов подпрограммы*, т.е. передача управления на адрес первой команды подпрограммы, и *возврат из подпрограммы,* т.е. переход к команде, адрес которой является следующим за командой вызвавшей подпрограмму. При этом следует заметить, что более сложной задачей является решение проблемы *возврата* из подпрограммы. Дело в том, что *адрес возврата из подпрограммы заранее не известен* и жестко связан с адресом команды вызова подпрограммы. Это делает *невозможным указание этого адреса в команде, осуществляющей возврат из подпрограммы*. Положение осложняется тем, что на практике вызовы подпрограмм могут оказаться *вложенными* друг в друга, т.е. в процессе выполнения подпрограммы может быть осуществлен переход на другую, еще одну подпрограмму и т.д., как это показано на рис.9.2. Степень вложенности подпрограмм друг в друга может быть произвольной и, что важно, может быть вообще не известна программисту, использующему вызов готовой подпрограммы, написанной кем-то другим.

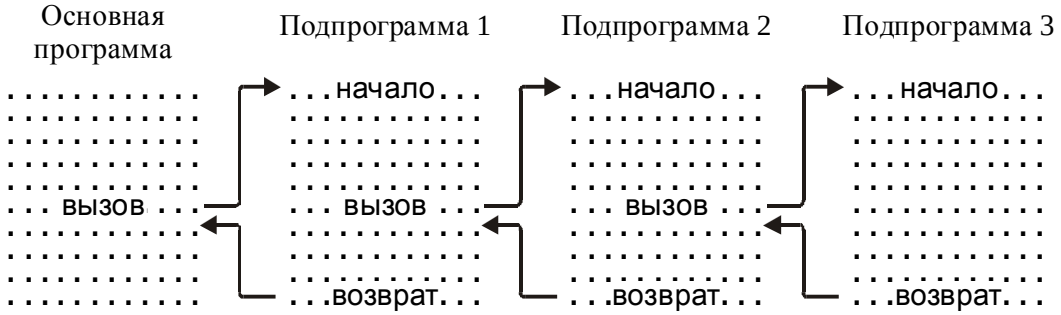

Рис.9.2

Рассмотрим, каким образом могут быть решены указанные выше проблемы *вызова* подпрограммы и *возврата* из нее.

9.2. Реализация вызова подпрограмм и возврата в основную программу

Адрес ячейки памяти, на которую следует возвратиться из после выполнения подпрограммы, становится известным в момент обработки процессором команды ВЫЗОВ ПОДПРОГРАММЫ. Фактически это адрес команды, следующей за командой вызова подпрограммы. Мы знаем, что в момент после выборки этой команды из памяти в регистр команд процессора адрес, находящийся в это время в регистре–счетчике команд процессора, является адресом следующей по порядку команды. Другими словами, в момент, когда процессор приступает к интерпретации команды вызова подпрограммы, нужный нам адрес возврата из подпрограммы находится *в регистре-счетчике команд процессора*. Поэтому вполне естественным для обеспечения в процессе выполнения команды ВЫЗОВ ПОДПРОГРАММЫ возможности правильного возврата из этой подпрограммы является предварительное сохранение в определенном месте текущего содержимого счетчика команд (адреса возврата) перед тем, как оно будет замещено адресом начала подпрограммы. Теперь для возврата из подпрограммы можно использовать этот предварительно сохраненный адрес возврата.

Следующим важным вопросом, который должен быть решен, является определение места, в котором должен быть сохранен адрес возврата из подпрограммы. При этом следует учесть, что если мы хотим реализовать возможность использования вложенных подпрограмм, то, во-первых, мы должны запоминать не *один* адрес возврата, а адреса возврата из *каждой* вызываемой подпрограммы, и, во-вторых, чтобы не нарушать логику программы, сохраненные адреса возврата должны извлекаться для использования в порядке, *противоположном* порядку их запоминания при последовательном входе во вложенные подпрограммы.

Таким местом, в котором удобно осуществлять запоминание адресов возврата, является ***стек***. Собственно необходимость реализации механизма работы с подпрограммами и является причиной (хотя и не единственной) введения в архитектуру ЭВМ памяти, организованной в виде стека.

Представляя собой запоминающую структуру с одной точкой входа-выхода – вершиной стека, стек позволяет естественным образом решить проблему запоминания адреса возврата (засылка в стек) при вызове подпро-

граммы и возврата из подпрограммы по этому адресу, извлекаемому (выталкиваемому) из стека. Принцип организации стека: *последним пришел – первым вышел,* как нельзя лучше подходит для реализации механизма вложенных подпрограмм: *при последовательном входе в подпрограммы адреса возврата последовательно засылаются в стек, а при выходе из подпрограмм выталкиваются из стека в порядке, противоположном их записи в стек.*

В ЭВМ автоматизация этих действий при обращении к подпрограммам осуществляется с помощью имеющейся в системе команд любой ЭВМ пары команд – это команда ВЫЗОВ ПОДПРОГРАММЫ, которая в разных ЭВМ может иметь мнемонику CALL, JSR, JSB, и команда ВОЗВРАТ ИЗ ПОДПРОГРАММЫ (RETURN, RET, RTS, RSB).

Аргументом команды ВЫЗОВ ПОДПРОГРАММЫ служит адрес входа в подпрограмму (адрес первой команды подпрограммы), задаваемый любым приемлемым методом адресации.

На рис.9.3 показано последовательное изменение состояния регистра-счетчика команд (СК), регистра-указателя стека (УС) и ячеек памяти стека при вложенных вызовах подпрограмм и возврате из них.

При выполнении команды ВЫЗОВ ПОДПРОГРАММЫ процессор осуществляет следующие действия:

- *текущее содержимое регистра-счетчика команд процессора (адрес, на который необходимо вернуться из вызываемой подпрограммы) пересылается в стек;*
- *задаваемый в команде вызова подпрограммы адрес входа в подпрограмму помещается в регистр-счетчик команд.*
- *после этого, как обычно, из памяти считывается команда, адрес которой находится в счетчике команд, т.е. первая команда подпрограммы, и начинается выполнение подпрограммы.*

Командой ВОЗВРАТ ИЗ ПОДПРОГРАММЫ должна завершаться любая подпрограмма.

- *Получив эту команду, процессор извлекает (выталкивает) из стека записанный туда ранее командой* ВЫЗОВ ПОДПРОГРАММЫ *адрес возврата и помещает его в свой регистр-счетчик команд.*
- *Тем самым управление передается команде, следующей за командой вызова подпрограммы.*

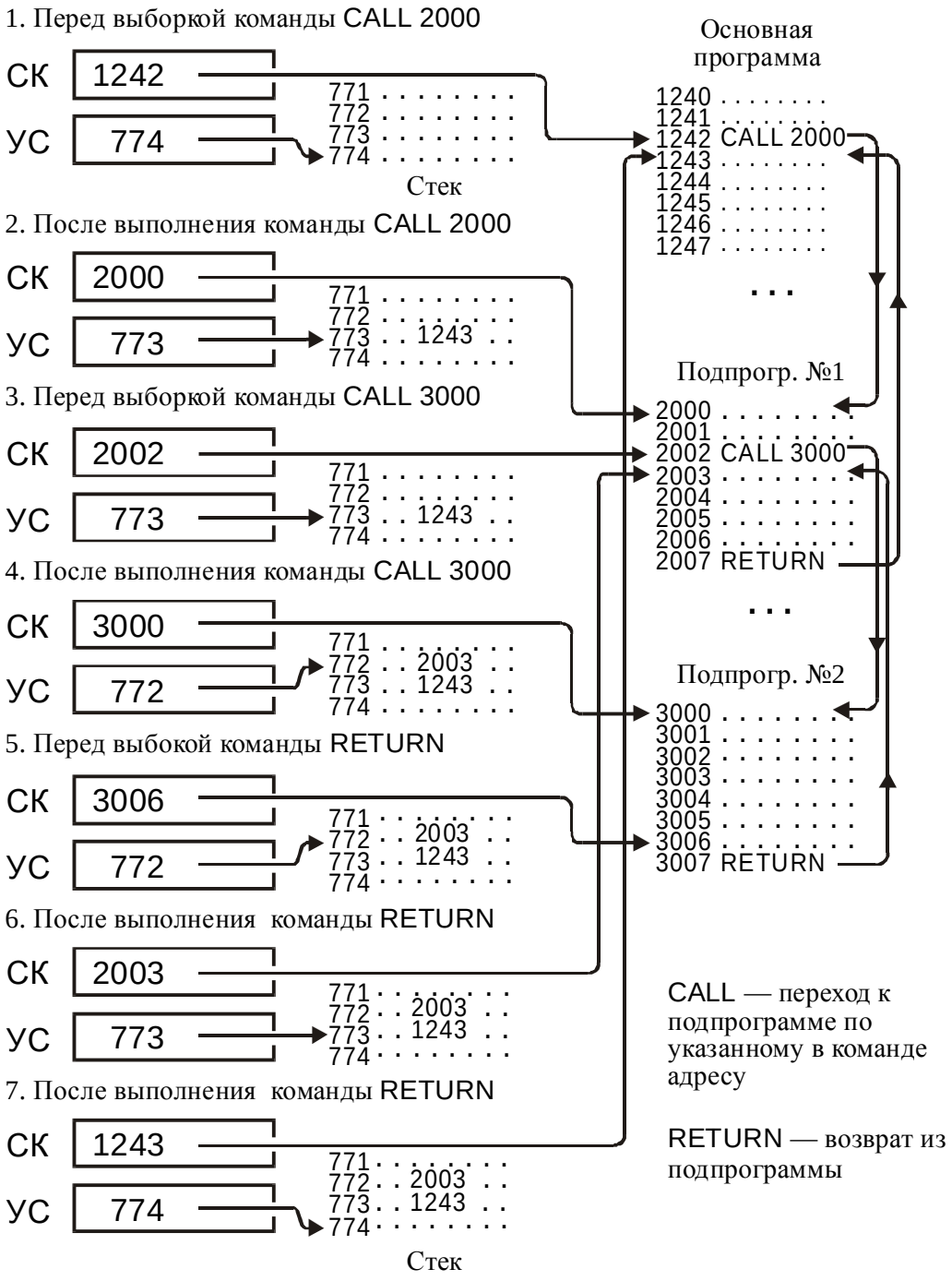
1. Перед выборкой команды CALL 2000
СК 1242
УС 774
771 . . . . . .
772 . . . . . .
773 . . . . . .
774 . . . . . .
Стек
2. После выполнения команды CALL 2000
СК 2000
УС 773
771 . . . . . .
772 . . . . . .
773 . . 1243 . .
774 . . . . . .
3. Перед выборкой команды CALL 3000
СК 2002
УС 773
771 . . . . . .
772 . . . . . .
773 . . 1243 . .
774 . . . . . .
4. После выполнения команды CALL 3000
СК 3000
УС 772
771 . . . . . .
772 . . 2003 . .
773 . . 1243 . .
774 . . . . . .
5. Перед выбокой команды RETURN
СК 3006
УС 772
771 . . . . . .
772 . . 2003 . .
773 . . 1243 . .
774 . . . . . .
6. После выполнения команды RETURN
СК 2003
УС 773
771 . . . . . .
772 . . 2003 . .
773 . . 1243 . .
774 . . . . . .
7. После выполнения команды RETURN
СК 1243
УС 774
771 . . . . . .
772 . . 2003 . .
773 . . 1243 . .
774 . . . . . .
Стек
Основная программа
1240 . . . . . .
1241 . . . . . .
1242 CALL 2000
1243 . . . . . .
1244 . . . . . .
1245 . . . . . .
1246 . . . . . .
1247 . . . . . .
. . .
Подпрогр. №1
2000 . . . . . .
2001 . . . . . .
2002 CALL 3000
2003 . . . . . .
2004 . . . . . .
2005 . . . . . .
2006 . . . . . .
2007 RETURN
. . .
Подпрогр. №2
3000 . . . . . .
3001 . . . . . .
3002 . . . . . .
3003 . . . . . .
3004 . . . . . .
3005 . . . . . .
3006 . . . . . .
3007 RETURN
CALL — переход к подпрограмме по указанному в команде адресу
RETURN — возврат из подпрограммы

Рис. 9.3

9.3. Контрольные вопросы

1. Что такое подпрограммы, для чего они нужны?
2. Какие проблемы необходимо решить для обеспечения возможности работы с подпрограммами?
3. Что такое вложенные подпрограммы?
4. Какую роль при реализации механизма подпрограмм играет стек?
5. Почему для перехода к подпрограмме не используют команды безусловного или условных переходов?
6. Что происходит при выполнении процессором команды ВЫЗОВ ПОДПРОГРАММЫ?
7. Что происходит при выполнении процессором команды ВОЗВРАТ ИЗ ПОДПРОГРАММЫ?

10. Особенности RISC-архитектуры

10.1. ЭВМ с расширенным и сокращенным набором команд.

Одним из направлений развития архитектуры ЭВМ, направленным на повышение их производительности, является усложнение структуры процессоров путем *расширения* системы (набора) выполняемых им команд, введения более сложных команд, выполняющих операции, все более приближающиеся к примитивам языков высокого уровня, увеличения числа методов адресации и т. д.

Однако расширение и усложнение набора команд порождает и ряд нежелательных побочных эффектов, в ряде случаев сводящих на нет выигрыш от введения более сложных команд, в том числе и с точки зрения уменьшения времени выполнения программ.

Дело в том, что расширение набора команд, увеличение числа способов адресации, введение сложных команд сопровождается увеличением длины кода команды (в первую очередь, кода операции), приводит к увеличению числа форматов команд. Это, в свою очередь, вызывает усложнение и замедление процесса дешифрации кода операции и других процедур обработки команд. Причем это касается не только добавляемых усложненных команд, но, к сожалению, и всего набора команд, в том числе и самых простых. Возрастающая сложность процедур обработки команд заставляет прибегать к

микропрограммной реализации управляющего устройства процессора вместо более быстродействующих устройств управления с «жесткой» («схемной») логикой. Усложнение процессора приводит к увеличению длин межсоединений на кристалле процессора, что является фактором, затрудняющим повышение тактовой частоты процессора.

С другой стороны, анализ статистики практического использования в программах конкретных команд показал, что на самом деле наиболее часто подавляющее большинство составляющих программу команд составляет сравнительно небольшой набор достаточно простых команд, а команды, выполняющие сложные операции используются в программах существенно реже. Таким образом, в очень большой степени общее время выполнения программы определяется эффективностью реализации команд реализующих достаточно простые, но часто используемые операции, а не сложные, но используемые гораздо реже.

Это привело к появлению в восьмидесятые годы альтернативного подхода к развитию системы команд процессоров. А именно к появлению процессоров с так называемой RISC-архитектурой. Сокращение RISC (по-английски это – Reduced Instruction Set Computers) означает – *ЭВМ с* ***сокращенным*** *набором команд*.

RISC-архитектура предполагает реализацию в ЭВМ *сокращенного* набора простейших, но часто употребляемых команд. Это позволяет существенно упростить аппаратные средства процессора и благодаря этому получить возможность существенно повысить эффективность выполнения этих команд. При использовании выбор набора команд и структуры процессора направлены на то, чтобы команды набора выполнялись за один машинный цикл процессора. Выполнение более сложных, но редко встречающихся операций обеспечивают подпрограммы. В ЭВМ с RISC-архитектурой машинным циклом называют время, в течение которого производится выборка двух операндов из регистров, выполнение операции в АЛУ и запоминание результата в регистре. Большинство команд в RISC-процессоре являются быстрыми командами типа «регистр-регистр» и выполняются без обращения к оперативной памяти. К оперативной памяти обращается лишь в командах *загрузки регистров* и *запоминание в оперативной памяти*. Чтобы это стало возможным, RISC-процессор обычно содержит достаточно большое число регистров общего назначения.

Благодаря характерным для RISC-архитектуры особенностям – сокращенному набору команд (обычно не более 50–100), небольшому числу (обычно 2-3) простых способов адресации (в основном регистровой), небольшому числу простых форматов команд с фиксированными размерами и функциональным назначением их полей – упрощается управляющее устройство процессора. При его реализации обычно обходятся без микропрограммного уровня управления и управляющей памяти, используя более быстродействующую «схемную логику». Упрощение структуры процессора существенно уменьшает размеры процессора, облегчая его реализацию на одном кристалле даже с учетом увеличения до нескольких десятков числа регистров общего назначения.

Большое число регистров, особенно при наличии обеспечивающего их эффективное использование «оптимизирующего компилятора», позволяет до предела сократить обращение к оперативной памяти, путем сохранения в регистрах промежуточных результатов, передачи через регистры операндов из одних программ в другие программы или подпрограммы, отказа для передач на сохранение в оперативную память содержимого регистров при прерываниях.

В настоящее время, несмотря на продолжающиеся дискуссии по поводу преимуществ и недостатков процессоров с расширенным и сокращенным наборами команд, на рынке компьютеров представлены оба этих направления. Можно сказать, что постепенно каждое из них находит свою «нишу». В частности большое число компьютеров, предназначенных для использования в качестве серверов, используют процессоры с сокращенным набором команд, т. е. RISC-архитектуру.

10.2. Контрольные вопросы

1. В чем разница между процессорами с расширенным и сокращенным набором команд?
2. Какие проблемы с точки зрения времени выполнения команд возникают при расширении набора команд процессора путем добавления команд, выполняющих сложные операции?
3. За счет чего возникает выигрыш в производительности процессора с сокращенным набором команд?

11. Организация ввода-вывода. Обмен информацией ЭВМ с внешними устройствами

11.1. Внешние устройства

Ни один компьютер не может быть представлен без определенного набора подключенных к нему внешних или периферийных устройств. Назначение этих устройств – обеспечить *ввод* в ЭВМ и *вывод* из нее информации в виде наиболее подходящем для использующего компьютер человека. Внешние или периферийные устройства выполняют определенные функции по приему и обработке информации, передаваемой в них из ЭВМ, и по формированию информации, передаваемой из них в ЭВМ. Существует множество внешних устройств ЭВМ различного назначения, среди которых отметим следующие.

Клавиатура ЭВМ – это устройство, предназначенное для формирования и ввода в ЭВМ двоичных кодов в соответствии с нажатой клавишей или комбинацией клавиш.

Монитор – это устройство, предназначенное для отображения на экране выводимой из ЭВМ информации в виде символьного текста или графического изображения.

Печатающие устройства и *графопостроители* предназначены для вывода текстовой и графической информации на бумагу.

С*канеры* – устройства для ввода в ЭВМ изображений и текстов в графическом виде, а также манипуляторы типа *мышь или джойстик*, *модемы, сетевые адаптеры*, *цифро-аналоговые* и *аналого-цифровые устройства ввода/вывода информации* и многие другие.

Особую роль среди внешних устройств ЭВМ занимают ***внешние запоминающие устройства*** **(ВЗУ)**. Это устройства, предназначенные для *долговременного* хранения *больших* объемов информации. Большая часть внешних запоминающих устройств использует магнитный способ записи информации. Это так называемые *накопители на магнитных дисках* (НМД) и *магнитных лентах* (НМЛ). Отметим сразу существенные особенности этих двух типов устройств, связанные с доступом к записанной на них информации.

Накопители на лентах относятся к устройствам **с** *последовательным доступом*, так как запись информации на ленту и считывание ее может осуществляться только последовательно путем перемотки ленты мимо головки записи/чтения запоминающего устройства.

Накопители на магнитных дисках относятся к внешним запоминающим устройствам **с** *произвольным* или *прямым доступом*. Доступ магнитных головок к концентрическим магнитным дорожкам вращающихся дисков устройства осуществляется быстрым радиальным перемещением его блока головок. Это обеспечивает практически одинаково малое время доступа, как к наружным, так и к внутренним дорожкам дисков.

11.2. Общие принципы организации ввода-вывода

Производительность и эффективность использования ЭВМ в большой степени определяется тем, как осуществляются в ней операции ввода и вывода информации.

При реализации в ЭВМ системы ввода-вывода информации приходится решать сложные технические проблемы. Эти проблемы в основном обусловлены следующими факторами.

Необходимо обеспечение возможности совместного использования различных ЭВМ с переменным составом различного внешнего оборудования, отвечающего решениям конкретных задач пользователя. При этом назначение, типы, формы исполнения, производители и технические характеристики, как ЭВМ, так и внешних устройств чрезвычайно разнообразны. При этом конструкция внешних устройств, как правило, в большой степени независима от конструкции конкретных ЭВМ, с которыми они могут быть использованы.

Должна быть обеспечена унификация программирования операций ввода-вывода для максимальной независимости разрабатываемого программного обеспечения от конкретной аппаратуры, на которой оно может быть использовано.

Выполнение этих требований усложняется еще следующими факторами. Скорость, с которой различные внешние устройства могут выдавать или принимать информацию, может лежать в очень широких пределах и во мно-

гих случаях оказывается гораздо ниже скорости работы процессора и памяти ЭВМ. Кроме того, сам момент времени, в который должен осуществиться акт передачи информации, во многих случаях определяется не ЭВМ, не выполняемой процессором программой, а внешним устройством, т.е. при выполнении программы заранее *не может быть известен*. Очевидный пример – ввод информации пользователем путем нажатия клавиш клавиатуры.

Для реализации обмена информацией между ЭВМ и внешними (периферийными) устройствами предназначена ***подсистема ввода-вывода ЭВМ***.

На рис.10.1 показано подключение внешних устройств к внутренним информационным шинам ЭВМ, его центральному процессору и памяти.

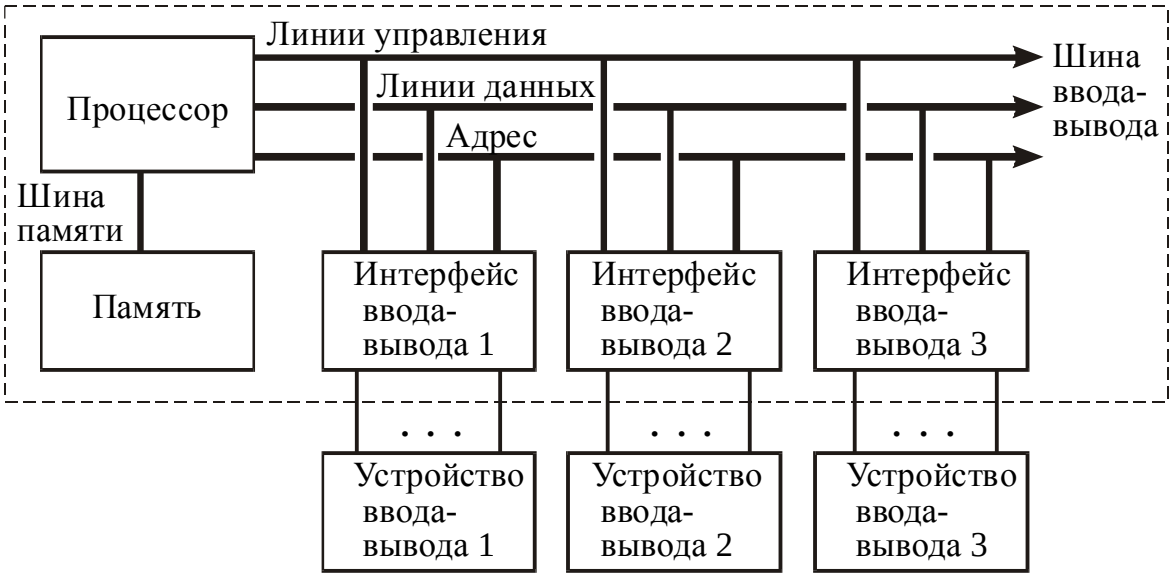

Рис.10.1

Шина ввода-вывода, через которую внешние устройства подключаются к процессору, включает в себя *адресные линии*, *линии ввода-вывода данных* и *линии управления*. Обычно шина ввода-вывода и шина памяти логически независимы, хотя в некоторых ЭВМ адресные линии и линии данных этих шин могут быть совмещены.

К ЭВМ *внешние устройства* подключаются через так называемые *порты ввода-вывода*. Для сопряжения конкретного внешнего устройства с конкретной ЭВМ служат так называемые *интерфейсные устройства* (или просто интерфейсы). *Интерфейс* представляет собой совокупность линий и шин,

сигналов, электронных схем и алгоритмов (протоколов передачи и обработки данных), обеспечивающих унификацию и стандартизацию передачи данных между ЭВМ и внешними устройствами. Интерфейс ввода-вывода управляет действиями определенного вида внешнего устройства в соответствии с командами процессора ЭВМ: преобразует данные из внутреннего представления ЭВМ в форматы, используемые конкретным внешним устройством, и выполняет обратное преобразование в формат, используемый процессором конкретной ЭВМ. Конструктивно внешние устройства обычно выполняются как самостоятельные устройства, размещаемые отдельно от ЭВМ, интерфейсы же с портами ввода-вывода почти всегда являются частью ЭВМ.

Для выполняемой процессором программы, точнее для программиста, программирующего операцию ввода-вывода, *порт ввода-вывода* представляет собой регистр или набор регистров, содержимое которых программно доступно для чтения и/или записи во время операций ввода-вывода. *Передача (вывод) числа или кода из ЭВМ во внешнее устройство фактически реализуется их передачей из регистра процессора или ячейки памяти ЭВМ в регистр данных интерфейса внешнего устройства*. Аналогично, *прием (ввод) информации из внешнего устройства заключается в чтении процессором содержимого (числа или кода) регистра данных интерфейса*.

Передача или прием информации через порты ввода-вывода во многих ЭВМ осуществляется по предназначенным для этого информационным шинам с помощью специальных команд ввода-вывода. В некоторых ЭВМ, в которых для адресации портов используется тоже адресное пространство, что и для адресации памяти, работа с ячейками памяти и регистрами внешних устройств осуществляется одним и тем же набором команд.

Обычно в ЭВМ используется три основных режима обмена информацией с внешними устройствами.

1. *Программный обмен данными по флагу готовности устройства* – это передача или прием данных по инициативе и под управлением программы, выполняемой центральным процессором ЭВМ.
2. *Программный обмен данными в режиме прерывания программы*. Инициатива обмена информацией в этом случае принадлежит внешнему устройству, для этого им осуществляется прерывание выполняемой

процессором программы и принудительный перевод процессора на выполнение программы обслуживания внешнего устройства, затребовавшего прерывание, по окончании которой процессор возвращается к выполнению прерванной программы.

3. Обмен данными *в режиме прямого доступа к памяти* (ПДП). В этом режиме управление обменом берет на себя внешнее устройство. Это позволяет осуществлять быстрый обмен данными непосредственно между основной памятью ЭВМ и интерфейсом ввода-вывода внешнего устройства без участия процессора.

Рассмотрим особенности этих режимов обмена информацией более подробно.

11.3. Программный режим ввода-вывода по опросу готовности внешнего устройства.

В этом режиме обмен информацией между ЭВМ и внешними устройствами осуществляется *по инициативе и под управлением выполняемой программы* пользователя. Основная проблема, которую при этом требуется решать, связана с тем, что скорость обработки информации внешним устройством, как правило, существенно ниже быстродействия процессора и поэтому внешнее устройство оказывается готовым к приему или передаче информации далеко не во все моменты времени. Вследствие этого для обеспечения корректной передачи информации необходим механизм, позволяющий программным путем определять состояние готовности или неготовности внешнего устройства к приему или передаче данных. Это необходимо, для того, чтобы собственно передача данных в регистр или чтение их из регистра данных внешнего устройства осуществлялись в те в моменты времени, когда внешнее устройство закончило обработку или формирование очередного слова или байта данных и *готово* к приему или передаче следующего.

Для обеспечения возможности определения состояния готовности или не готовности внешнего устройства в интерфейс внешнего устройства вводится так называемый *регистр состояния внешнего устройства* (РС ВУ). Один из разрядов регистра состояния устройства устанавливается *самим внешним устройством* в единицу или сбрасывается в ноль в зависимости от его со-

стояния готовности или не готовности. Этот разряд часто называют *флагом готовности устройства*. Процессор, обращаясь с помощью соответствующих команд к регистру состояния внешнего устройства и анализируя состояние его разрядов, может определить, таким образом, готово внешнее устройство к приему/передаче данных или нет, и в зависимости от этого осуществлять пересылку данных или ожидать, момента готовности устройства. На рис.11.1 представлен алгоритм (блок-схема) программы реализующей такую процедуру. Такой способ организации ввода-вывода получил название *программного ввода-вывода данных по опросу флага готовности внешнего устройства*.

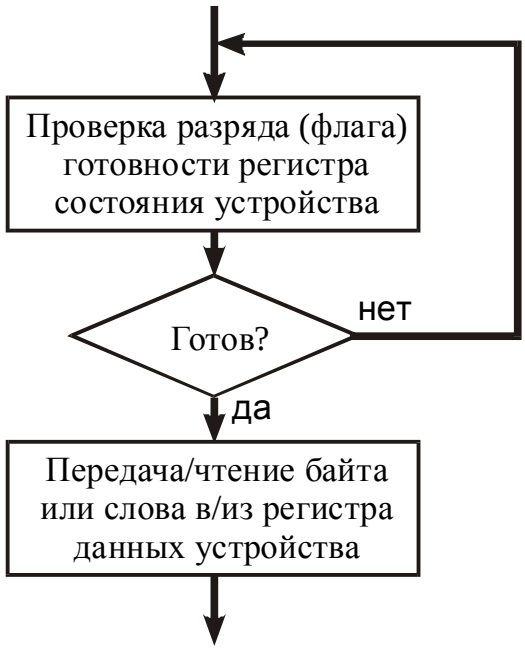

Рис.11.1

Как, исходя из такого алгоритма, будет выглядеть, например, процедура *вывода* на принтер текстового сообщения, состоящего из некоторого числа символов, хранящихся в памяти ЭВМ в виде массива кодов этих символов? Выполняющая такую операцию программа должна *осуществлять последовательную передачу этих кодов из ячеек памяти* ***в регистр данных принтера*** *с предварительной циклической проверкой* ***флага готовности принтера*** *в его* ***регистре состояния*** *перед выводом* ***каждого*** *символа*, как это показано на рис.11.1.

Для ввода в ЭВМ символов текста с клавиатуры аналогичным образом должен проверяться разряд готовности в регистре состояния клавиатуры, который устанавливается в единицу интерфейсом клавиатуры при нажатии на одну из ее клавиш и помещении соответствующего этой клавише кода в регистр данных клавиатуры. Из регистра данных клавиатуры код нажатой клавиши может быть прочитан выполняемой процессором программой, после чего происходит автоматический сброс флага готовности клавиатуры (установка его в состояние «*не готов*»). Таким образом, код нажатой клавиши нельзя прочитать больше одного раза.

Как видно из вышеизложенного, время, затрачиваемое на реализацию процедуры ввода-вывода массива данных при использовании программно-управляемого режима с опросом флага готовности устройства, определяется быстродействием внешнего устройства. Все это время процессор ЭВМ должен быть занят выполнением процедуры ввода-вывода, причем, как видно, фактически большую часть этого времени составляет выполнение циклической проверки флага готовности устройства. Если таким образом необходимо организовать обмен информацией с несколькими внешними устройствами, то необходимо обеспечивать поочередную проверку готовности каждого устройства.

Таким образом, особенностью режима программно управляемого ввода-вывода с опросом готовности внешнего устройства является нерациональное использование времени работы центрального процессора.

11.4. Программно управляемый обмен информацией в режиме прерывания программы

Концепция прерывания – одно из важнейших изобретений в области архитектуры ЭВМ, относящихся к повышению эффективности операций ввода-вывода. До признания этой концепции в вычислительных системах обычно использовался метод программного опроса, рассмотренный выше, когда процессор «по кругу» проверял каждый порт ввода-вывода (регистры состояния), чтобы определить, не нуждается ли в обслуживании какое-либо из подключенных к нему внешних устройств. При обнаружении такого запроса (по флагу готовности устройства) процессор переходил на программу обслуживания данного устройства.

Существенным недостатком такого метода является явно неэффективное использование самого дорогого ресурса ЭВМ – процессорного времени, поскольку время, затрачиваемое на выполнение опроса готовности внешних устройств, не может быть использовано для решения других задач.

Концепция прерывания программы позволяет устранить этот недостаток. Она предусматривает обработку процессором событий, момент возникновения которых обусловлен процессами, протекающими во внешних устройствах, то есть вне ЭВМ, без выполнения рутинных операций циклического опроса внешних устройств. Механизм, реализующий прерывания, включает в себя как аппаратные средства (особенности архитектуры ЭВМ), так и программные.

В самых общих чертах концепция работы ЭВМ с внешними устройствами *в режиме прерывания программы* состоит в следующем.

- *Пусть процессор выполняет некоторую программу (назовем ее основной).*
- *В некоторый момент времени в каком-либо внешнем устройстве происходит событие, связанное с необходимостью выполнения ЭВМ определенных действий, например, по вводу-выводу информации.*
- *Это внешнее устройство посылает процессору сигнал запроса (требования) прерывания выполняемой им программы.*
- *Получив этот сигнал, процессор приостанавливает (прерывает) выполнение основной программы и переходит на выполнение программы обслуживания внешнего устройства, затребовавшего прерывание.*
- *По окончании программы обслуживания внешнего устройства (обслуживания прерывания) процессор возвращается к выполнению прерванной основной программы.*

Таким образом, процессор в данном режиме уже не тратит время на опрос готовности внешних устройств к обмену информацией, реализуя сеансы обмена только по запросам самих внешних устройств.

Какие проблемы должны быть решены для реализации режима прерывания?

1. Как видно из вышеизложенного, работа основной (прерываемой) программы должна приостанавливаться *без какого-либо ущерба для* обеспечения возможности продолжения *ее последующего выполнения* с точки прерывания.
2. Учитывая, что к ЭВМ может быть подключено большое число разнообразных внешних устройств, должен быть реализован механизм, обеспечивающий переход процессора на *выполнение программы обслуживания именно того устройства, которое затребовало прерывание*. При этом нежелательна жесткая, раз и навсегда заданная привязка программы обслуживания к определенному месту в памяти ЭВМ, так как даже для одного и того же внешнего устройства алгоритмы обслуживания, используемые программами различных пользователей, могут существенно отличаться.
3. Необходим механизм, обеспечивающий *разрешение конфликтных ситуаций*, возникающих, когда запросы на прерывание поступают одновременно от нескольких внешних устройств, а также механизм, позволяющий программным путем *запрещать или разрешать прерывания* всем или некоторым внешним устройствам, когда это необходимо.

Существуют различные способы организации работы ЭВМ в режиме прерывания программы, в той или иной степени удовлетворяющие перечисленным требованиям. Наиболее полно они удовлетворяются в широко используемой современными ЭВМ так называемой системе с *прерыванием по вектору*.

При *векторной организации системы прерываний* в памяти ЭВМ помимо основной программы (программ), размещаются *программы обработки прерываний*, т.е. программы обслуживания внешних устройств, вызывающих прерывания. Кроме того, часть памяти отводится под так называемые *векторы прерывания*.

Обычно вектор прерывания – это две ячейки памяти, адреса которых жестко закреплены за конкретными внешними устройствами. В первой ячейке вектора должен быть помещен адрес программы обслуживания прерывания от соответствующего внешнего устройства, а во второй ячейке вектора раз-

мещается содержимое слова состояния процессора, которое должно быть установлено в регистре состояния процессора во время выполнения программы обслуживания прерывания от этого устройства.

Векторы прерывания внешних устройств обычно располагаются в начальной области памяти, занимая несколько сот ячеек памяти. Очевидно, что в этой области не следует размещать какие-либо программы.

Алгоритм обработки прерывания выглядит при этом следующим образом.

1. Пусть в некоторый момент времени какое-либо внешнее устройство, которому необходим обмен информацией с ЭВМ, посылает процессору по специальной линии (общей для всех внешних устройств) сигнал запроса (требования) прерывания. Процессор в этот момент может быть занят выполнением какой-либо команды основной программы.
2. Получив от внешнего устройства сигнал запроса прерывания, процессор выполняет комплекс действий, обеспечивающих возможность возобновления выполнения прерываемой программы после обслуживания внешнего устройства без нарушения логики программы. Эти действия заключаются в следующем:
   - процессор *завершает выполнение команды основной программы*, во время которой его застал запрос на прерывание;
   - *запоминает в стеке содержимое регистра состояния процессора*;
   - вслед за этим *запоминает в стеке содержимое регистра-счетчика команд*, в котором в этот момент находится адрес команды, следующей за только что выполненной командой основной программы, т.е. *адрес, на который процессор должен вернуться по окончании обслуживания прерывания*.
3. Обеспечив сохранение текущего состояния выполняемой программы, процессор посылает внешнему устройству (опять же по общей для всех внешних устройств линии) сигнал предоставления прерывания, сообщая внешнему устройству о своей готовности к обслуживанию прерывания.

После этого выполняется этап идентификации внешнего устройства, затребовавшего прерывание, и определение места расположения в памяти ЭВМ программы его обслуживания.

4. После получения от процессора сигнала предоставления прерывания, внешнее устройство передает процессору адрес своего вектора прерывания. Как уже говорилось, адреса векторов прерывания, которые уникальны для каждого внешнего устройства, являются, по сути, их идентификаторами.

5. Процессор, получив от внешнего устройства адрес его вектора прерывания, пересылает в свой регистр-счетчик команд содержимое первой ячейки вектора, т.е. адрес программы обслуживания данного внешнего устройства. После этого он пересылает в свой регистр состояния содержимое второй ячейки вектора, т.е. приводит слово состояния процессора в соответствие с тем, которое требуется во время обслуживания устройства (подробнее о том, зачем это может использоваться, будет сказано ниже).

Перечисленные операции завершают процесс перехода процессора к выполнению программы обслуживания внешнего устройства, затребовавшего прерывание. Выбирая из памяти и выполняя команду, адрес которой находится в его регистре-счетчике команд, т.е. первую команду программы обслуживания прерывания, процессор обычным образом начинает выполнение этой программы.

Выполнение программы обслуживания внешнего устройства завершается специальной командой ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ (в различных ЭВМ эта команда имеет мнемонику RTI, REI, IRET).

6. Получив команду ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ, завершающую программу обслуживания, процессор выталкивает из стека сохраненный там адрес ячейки основной программы, на которую ему надо вернуться из прерывания, и пересылает его в свой регистр-счетчик команд, затем выталкивает из стека и записывает в свой регистр состояния прежнее слово состояния процессора, полностью восстанавливая тем самым свое состояние, имевшее место до перехода к обслуживанию прерывания.

7. Процессор возобновляет выполнение прерванной программы.

Обратим внимание на некоторые ключевые моменты изложенного механизма процесса прерывания программы.

Использование организации прерываний по векторам, адреса которых для каждого внешнего устройства жестко фиксированы в памяти ЭВМ, обеспечивает пользователю ЭВМ полную свободу в размещении в ее памяти программ обслуживания прерываний. Адреса этих программ заносятся в соответствующие ячейки векторов основной программой.

Использование стека для сохранения адреса возврата из прерывания и текущего слова состояния процессора позволяет естественным образом (как и при организации подпрограмм) реализовать процесс обслуживания *вложенных прерываний*, т.е. ситуаций, когда до завершения обслуживания прерывания от какого-либо внешнего устройства процессор по соответствующему запросу переходит на обслуживание прерывания от другого внешнего устройства, после окончания которого возвращается в прерванную программу обслуживания первого устройства. Механизм работы стековой памяти автоматически обеспечивает правильный порядок запоминания содержимого счетчика команд и слова состояния процессора при последовательном входе во вложенные программы обслуживания прерываний и извлечения их из стека при последовательном возврате из этих программ.

Важной, еще не рассмотренной, стороной механизма прерываний является *обслуживание прерываний в соответствии с различными приоритетами внешних устройств*, которые в общем случае могут зависеть от конкретного контекста выполняемой программы. Действительно, в зависимости от выполняемых в конкретные моменты времени функций, участвующие в совместной работе устройства, включая и сам процессор, могут обладать различными приоритетами. Возможны ситуации, когда особая важность выполняемого процессором фрагмента программы делает вообще нежелательным какие-либо прерывания этого фрагмента программы. Разные назначение и функции внешних устройств делает необходимым, например, разрешение конфликта при одновременном требовании прерывания несколькими устройствами в пользу устройства с более высоким в соответствии с решаемой им задачей приоритетом. Возможны случаи, когда вообще нежелательны прерывания от конкретных внешних устройств, например, не используемых в конкретной программе пользователя.

Для решения перечисленных вопросов существуют соответствующие архитектурные средства, в частности, средства организации приоритетов. Рассмотрим их.

Существует возможность *программного управления приоритетом процессора* путем соответствующей установки определенного разряда (разрядов) его регистра состояния. Такая установка приоритета процессора может осуществляться специальной командой программным путем. Эта возможность может быть использована также при обслуживании прерываний от устройств, когда по определенным причинам пользователю нежелательно, чтобы программа обслуживания конкретного устройства прерывалась какими-либо другими устройствами. Для этого достаточно в слове состояния процессора, хранящемся во второй ячейке вектора прерывания данного устройства, установить более высокий приоритет процессора, что обеспечит запрет прерываний до конца обслуживания данного устройства, если до того этот разряд не будет очищен программным путем.

Существует возможность *запрета прерывания конкретным внешним устройствам*. Для этого используется определенный разряд в регистрах состояния внешних устройств. Тем самым прерывания от конкретных устройств «*маскируются*» и процессором не замечаются. Это в частности удобно в тех случаях, когда программа пользователя работает только с частью устройств, подключенных к ЭВМ, т.к. позволяет не включать в нее программные модули обслуживания неиспользуемых устройств.

Во многих ЭВМ используется так называемая цепочечно-групповая схема установления приоритета устройства при прерывании (см. рис.11.2).

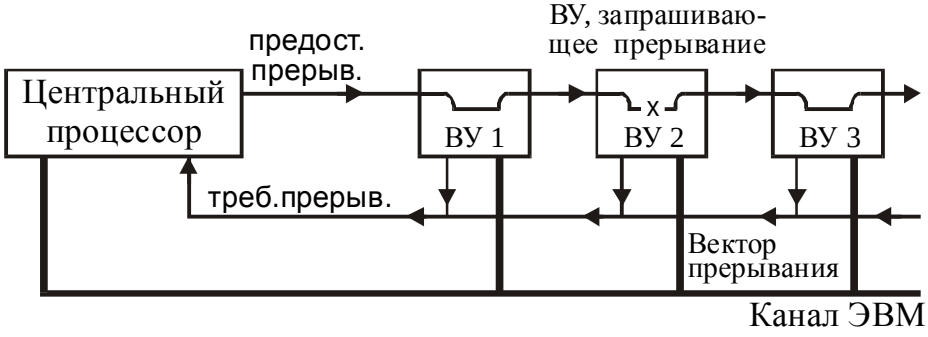

Рис.11.2

Как видно из схемы, сигнал предоставления (подтверждения) прерывания, проходя последовательно через интерфейсы всех внешних устройств, доходит до устройства, затребовавшего прерывание. Само же это устройство запрещает дальнейшее распространение этого сигнала к другим, более удаленным от процессора устройствам. Таким образом, если два устройства одновременно затребуют прерывание, то сигнал предоставления прерывания поступит только на то из них, которое расположено ближе по цепочке к процессору. *Фактический приоритет устройства в обслуживании прерывания будет определяться «близостью» расположения его интерфейса к центральному процессору.*

В ЭВМ могут быть реализованы, кроме того, так называемые *немаскируемые прерывания*. Обычно это прерывания, сигнализирующие процессору о внешних событиях особой важности (например, носящих катастрофический характер), таких, как отключение питания, сбой памяти и т.п. Немаскируемые прерывания признаются и обрабатываются процессором всегда *независимо от установки его приоритета*, т.е. они обладают наивысшим приоритетом. Обслуживание такого рода прерываний позволяет в ряде случаев спасти данные и программы пользователей от необратимых нарушений.

Кроме прерываний, вызываемых внешними по отношению к ЭВМ событиями, в ЭВМ обычно реализованы и так называемые внутренние прерывания. Такие прерывания возникают автоматически по сигналам в самом процессоре, например в случае получения процессором не существующего кода команды, обращения по не существующему адресу, ошибки деления и др. Как и внешние прерывания, внутренние прерывания обеспечивают переход процессора на программы обслуживания этих нештатных ситуаций, задаваемый соответствующими векторами прерываний.

В системе команд многих ЭВМ существуют также специальные команды, вызывающие *командные прерывания*, или *команды-ловушки*. При обработке этих команд также осуществляется переход на программу, адрес которой задается векторами прерывания, связанными с этими командами. Использование командных прерываний в какой-то мере напоминает подпрограммы. Однако их использование оказывается более удобным при решении задач взаимодействия программ операционной системы и пользовательских программ. С помощью команд программных прерываний обычно организуются библиотеки системных макрокоманд, операций ввода-вывода и других опера-

ций, при которых необходимо взаимодействие пользовательских и системных программ.

11.5. Прямой доступ к памяти

До сих пор описываемые процессы управления вводом-выводом информации подразумевали активное участие в них центрального процессора ЭВМ. Именно процессор осуществлял в них управление компонентами ЭВМ, формирование адресов данных и портов ввода-вывода, контроль величины передаваемых массивов данных, управление шинами адреса и данных.

На практике часто возникает необходимость передачи без какой-либо обработки больших массивов информации из внешнего устройства в оперативную память или наоборот. Обычно в таком режиме проводится обмен с внешними запоминающими устройствами, например, с накопителями на магнитных дисках. При использовании с этой целью рассмотренных выше способов обмена информацией с внешними устройствами, процессор должен для пересылки *каждого элемента данных* (слова или байта) выполнить целую группу команд и обращений к памяти (чтение команд, определение и считывание адресов, пересылка данных, подсчет количества переданных единиц данных, выполнение проверки на окончание передаваемого массива данных и др.). Очевидно, что вовлечение процессора в операции такого типа приводит к нерациональному использованию его «способностей» и «времени», поскольку никаких манипуляций данными, кроме их передачи не требуется. Более того, учитывая высокую скорость, с которой современные запоминающие устройства могут записывать и считывать данные, при использовании изложенных методов обмена информацией уже процессор, а не внешнее устройство может ограничивать скорость передачи больших массивов данных. Причина этого в том, что в рассмотренных режимах ввода-вывода пересылку каждого байта информации процессору требуется выполнить целую серию дополнительных операций по опросу готовности устройства или обработке его прерываний.

Для решения этого вопроса, а именно, для уменьшения «накладных расходов» при передаче данных большими блоками, непосредственно в память ЭВМ и из нее во внешнее устройство, используется метод *прямого доступа к памяти* (ПДП), по-английски – Direct Memory Access (DMA).

В режиме *прямого доступа к памяти* обмен данными между внешними устройствами и памятью ЭВМ происходит *без участия процессора*. Обменом в режиме прямого доступа к памяти управляет уже не процессор, выполняющий команды какой-либо программы, а электронные схемы, внешние по отношению к процессору. Обычно схемы, управляющие обменом в режиме ПДП, размещаются в специальном устройстве, называемом *контроллером прямого доступа к памяти*.

При необходимости осуществления сеанса прямого доступа к памяти контроллер ПДП посылает процессору сигнал *требование прямого доступа*. Процессор, получив этот сигнал, завершает цикл обращения к каналу ЭВМ, приостанавливая (и, в отличие от прерываний, не обязательно завершая) выполнение очередной команды. После этого выдает в контроллер ПДП сигнал *предоставление прямого доступа* и отключается от шин системного интерфейса. С этого момента все шины канала ЭВМ управляются контроллером ПДП. Контроллер ПДП, используя адресные шины и шины данных ЭВМ, самостоятельно формирует адрес соответствующей ячейки памяти и осуществляет пересылку данных в память или из ячейки памяти, после чего возвращает управление процессору ЭВМ. Контроллер ПДП самостоятельно осуществляет подсчет числа переданных байтов, осуществляя только что описанным способом захват управления шинами ЭВМ.

Передача информации в режиме ПДП может осуществляться как сразу большими блоками информации, так и малыми порциями, вписывая их в промежутки между циклами обращения процессора к каналу. Захват контроллером ПДП циклов управления шинами ЭВМ для работы с ее памятью, конечно, замедляет выполнение процессором основной программы. Однако, при правильной организации работы процессора, во время цикла ПДП он может не приостанавливать полностью свою работу, а выполнять операции, не связанные с обращением к памяти по шинам «данных-адреса». Учитывая сказанное, режим прямого доступа к памяти позволяет гораздо более эффективно, чем другие режимы ввода-вывода, использовать время процессора при вводе-выводе больших блоков информации.

11.6. Буферы данных в системах ввода-вывода

В организации систем ввода-вывода важное место занимают *буферы данных* (буферные запоминающие устройства или регистры). Буферы данных выполняют следующие функции:

- согласование *форматов данных*, с которыми работают передающее и принимающее информацию устройства;
- согласование *скоростей* работы передающего и принимающего устройств;
- виртуальное (кажущее) *изменение* количественных и качественных *характеристик* периферийного устройства.

К характеристикам периферийного устройства, существенным для обмена информацией, можно отнести следующие:

- формат единиц информации (бит, байт, слово и т.д.), которые передает или принимает устройство;
- интервал времени между последовательно передаваемыми единицами информации, и интервал времени между запросами внешнего устройства на обслуживание;
- максимально допустимое время ожидания внешним устройством обслуживания со стороны ЭВМ.

Благодаря наличию буфера, увеличивается интервал времени между запросами, или, другими словами, во столько же раз уменьшается частота запросов на передачу или прием информации. Наличие буфера может приводить к виртуальному (кажущемуся) качественному изменению характеристик внешнего устройства. Например, во внешнем запоминающем устройстве с магнитной лентой передача при записи и считывании информации производится с постоянной скоростью. Другими словами, ЗУ с магнитной лентой является устройством с синхронной передачей, навязывающей ЭВМ свой темп приема (выдачи) информации. Однако, если есть буфер, емкость которого достаточна для промежуточного хранения большого блока данных, считываемых (или записываемых) с ленты, то по отношению к процессору ЗУ с магнитной лентой становится устройством с асинхронной поблочной передачей информации, работающем в произвольном темпе.

11.7. Контрольные вопросы

1. В чем состоят принципиальные проблемы организации обмена информацией с внешними устройствами (организации ввода/вывода)?
2. Что такое интерфейс внешнего устройства?
3. Основные виды внешних устройств, их особенности.
4. Что такое порт ввода/вывода?
5. Режимы обмена информацией ЭВМ с внешними устройствами. Кто в каждом режиме является инициатором обмена, и кто осуществляет управление обменом?
6. Каким образом определяется готовность внешнего устройства к обмену информацией?
7. Что такое регистр данных и регистр состояния внешнего устройства?
8. Алгоритм программного ввода/вывода по опросу флага готовности внешнего устройства?
9. Алгоритм вывода символа на экран терминала в режиме программно-управляемого ввода/вывода с опросом готовности устройства.
10. В чем состоит основной недостаток программно-управляемого ввода/вывода с опросом готовности устройства?
11. В чем состоит концепция ввода/вывода в режиме прерывания программы?
12. Какие принципиальные проблемы должны решаться при реализации ввода/вывода в режиме прерывания программы?
13. Что процессор делает при получении от внешнего устройства сигнала требования прерывания?
    a) прекращает выполнение программы;
    b) прекращает выполнение программы и подает внешнему устройству сигнал предоставления прерывания;
    c) заканчивает выполнение текущей команды;
    d) прекращает выполнение программы и переходит на программу обслуживания внешнего устройства;
    e) продолжает выполнение текущей команды;
    f) ваш вариант.

14. Каким образом процессор узнает адрес, по которому в памяти ЭВМ находится программа обслуживания внешнего устройства, затребовавшего прерывание?

15. Что такое вектор прерывания?

16. Роль стека при реализации механизма прерывания.

17. Откуда появляется информация (и какая) в векторе прерывания?
    a) вводится пользователем с клавиатуры ЭВМ;
    b) передается из внешнего устройства;
    c) устанавливается программным путем;
    d) ваш вариант.

18. После выполнения каких действий процессор посылает внешнему устройству сигнал предоставления прерывания?

19. Что делает внешнее устройство при получении от процессора сигнала предоставления прерывания?
    a) передает процессору адрес своей программы обслуживания;
    b) передает процессору свой вектор прерывания;
    c) передает процессору адрес своего вектора прерывания;
    d) начинает прием/передачу данных;
    e) ждет от процессора специальной команды ввода/вывода;
    f) ваш вариант.

20. Что такое приоритет процессора, каким образом его можно устанавливать?

21. Что такое приоритет внешнего устройства, каким образом его можно устанавливать?

22. Каким образом можно запретить прерывание всем внешним устройствам, когда может возникнуть такая необходимость?

23. Каким образом можно запретить/разрешить прерывание конкретному внешнему устройству, когда может возникнуть такая необходимость?

24. Что произойдет, если два внешних устройства одновременно затребуют прерывание?

25. Что такое вложенные прерывания?

26. При переходе в режим прерывания процессор запоминает в стеке:
    a) адрес возврата в прерванную программу;
    b) вектор прерывания;

c) слово состояния процессора;
d) адрес внешнего устройства, вызвавшего прерывание;
e) адрес подпрограммы обслуживания прерывания;
f) ваш вариант.

27. Каким образом при окончании обслуживания внешнего устройства, вызвавшего прерывание, осуществляется возврат в прерванную программу?

28. Какие операции необходимо осуществить в программе, если вы хотите работать с внешними устройствами в режиме прерывания?

29. Что такое немаскируемые прерывания?

30. Что такое программные прерывания?

31. Что такое прямой доступ к памяти?

32. В чем отличие прямого доступа к памяти от других (каких?) режимов ввода/вывода информации?

33. В режиме прямого доступа к памяти управление вводом/выводом осуществляется:

a) программой пользователя;
b) внешним устройством;
c) процессором;
d) ваш вариант.

34. Какие внешние устройства обычно используют ввод/вывод информации в режиме прямого доступа к памяти и почему?

35. Что должно «уметь» внешнее устройство, осуществляющее ввод/вывод информации в режиме прямого доступа к памяти?

12. Управление памятью ЭВМ, расширение адресного пространства, динамическое распределение памяти

12.1. Проблемы управления памятью и расширения адресного пространства

Оперативная память ЭВМ, как и ее центральный процессор, являются наиболее важными ресурсами ЭВМ, существенным образом определяющими эффективность работы ЭВМ в целом. Важность памяти определяется тем, что *никакая программа не может быть выполнена ЭВМ, если она не находится в ее оперативной памяти*. В связи с этим сложность и время выполнения решаемых ЭВМ задач в большой степени определяются объемом ее памяти.

Одним из широко используемых методов существенного повышения эффективности использования ЭВМ является организация ее работы *в многозадачном* (мультипрограммном) и *многопользовательском* режимах. В мультипрограммных системах размещение всех исполняемых программ полностью в оперативной памяти в большинстве случаев невыполнимо, так как программы обычно имеют большую длину, а емкость используемой оперативной памяти всегда ограничена. Однако нет принципиальной необходимости в том, чтобы вся программа находилась в оперативной памяти, так как в любой момент времени работа программы концентрируется на определенных сравнительно небольших участках. Таким образом, в оперативной памяти можно хранить только используемые в данный период времени части программ, а неиспользуемые могут располагаться во внешнем запоминающем устройстве. При этом, программируя свою программу, пользователь не знает, в комбинации с какими программами будет выполняться его программа, какое место в памяти отведет ей операционная система, которая должна обеспечивать возможность *независимой* работы программистов над своими программами, подлежащими мультипрограммной обработке.

В связи с этим, помимо стремления к *увеличению* доступного объема памяти ЭВМ, большое значение приобретает проблема эффективного *управления* памятью, динамического распределения адресного пространства памяти между различными задачами в процессе их выполнения.

Как уже говорилось выше, одним из наиболее важных факторов, определяющих объем адресного пространства ЭВМ, является разрядность ЭВМ, т.е. количество разрядов операндов-данных и адресов, с которыми оперирует ее центральный процессор. Так в 16-разрядных ЭВМ пользователь может непосредственно использовать в программе адреса, изменяющиеся от нуля до $2^{16}=65536$, выражаемые 16-разрядными двоичными словами. *Для того, чтобы ЭВМ могла обеспечить обращение к большему адресному пространству, в ней обязательно должны быть аппаратные средства, позволяющие формировать адресное слово с большим количеством разрядов*. И такие средства в большинстве современных 16-разрядных ЭВМ имеются.

Процессоры, оперирующие словами с большим количеством разрядов, например 32-х, 64-х разрядные, позволяют непосредственно адресоваться к значительно большему адресному пространству, во много раз превышающему достижимый на современном этапе объем физической памяти ЭВМ. Однако и для них, может быть даже в большей степени, остается важной проблема эффективного динамического распределения имеющейся памяти между большим числом задач и пользователей.

Ниже будут рассмотрены основные архитектурные решения, используемые для увеличения доступного адресного пространства 16-разрядных ЭВМ и управления расширенной памятью. При этом, однако, надо иметь в виду, что проблема управления памятью ЭВМ решается комплексом аппаратно-программных средств, т.е. не только определенными архитектурными особенностями построения процессора ЭВМ, но и тесно связанными с ними программными средствами операционной системы ЭВМ.

12.2. Физическое и виртуальное адресные пространства

При решении проблемы адресного пространства и управления памятью ЭВМ очень важной является концепция *разделения физического и виртуального адресных пространств*. Под *физической памятью* ЭВМ понимают множество имеющихся в аппаратуре ЭВМ ячеек оперативной памяти. *Виртуальной памятью* называют адресное пространство, с которым оперирует пользователь в своих программах. Причем концепция выделения таких двух видов памяти – физической и виртуальной подразумевает, что *в общем случае используемые программой физические и виртуальные адреса*

чае используемые программой физические и виртуальные адреса не совпадают.

Принцип виртуальной памяти предполагает, что пользователь при подготовке своей программы имеет дело не с физической оперативной памятью, действительно работающей в составе конкретной вычислительной системы, а с виртуальной (т. е. кажущейся) одноуровневой памятью. Емкость виртуальной памяти равна всему адресному пространству, определяемому размером адресных полей в форматах команд и базовых регистров. Пользователь имеет в своем распоряжении все адресное пространство системы независимо от объема ее фактической физической памяти и объемов и расположения областей памяти, необходимых для других программ, участвующих в мультипрограммной обработке. На всех этапах подготовки программ, включая загрузку в оперативную память, программа представляется в *виртуальных адресах*, начинающимися от нулевого адреса. Пользователь заранее не знает, где в физической памяти будет располагаться его программа. Лишь при самом исполнении машинной команды с помощью *специальных аппаратных и системных программных средств* производится преобразование виртуальных адресов в реальные адреса действующей памяти. Другими словами, эти средства *осуществляют отображение виртуальных адресов одной или нескольких программ на конкретные адреса физической памяти*, отображение, более или менее эффективное с точки зрения общих критериев решения ЭВМ пользовательских задач.

Преобразование виртуальных адресов в физические упрощается, если физическую и виртуальную память разбить на блоки, содержащие одинаковое число байт. Такие блоки обычно называются *страницами* или *сегментами*. Страницам виртуальной и физической памяти присваивают номера, называемые номерами соответствующих виртуальных и физических страниц. Каждая физическая страница способна хранить одну из виртуальных страниц. Порядок расположения (нумерация байт) в виртуальной и физической страницах сохраняется одним и тем же.

В мультипрограммной системе страничная организация памяти дает определенные преимущества. Когда новая программа загружается в оперативную память, она может быть направлена в любые свободные в данный момент физические страницы независимо от того, расположены они подряд или нет. Не требуется перемещения информации в остальной части памяти.

Страничная организация позволяет сократить объем передачи информации между внешним запоминающим устройством и оперативной памятью. Программа может загружаться в оперативную память не целиком. Вначале в оперативную память загружается начальная страница программы, которой передается управление. Другие страницы программы не должны загружаться до тех пор, пока она действительно не понадобится. Если по ходу программы делается попытка выборки слов из другой страницы, то производится автоматическое обращение к операционной системе, которая осуществляет загрузку требуемой страницы.

12.3. Расширение адресуемого пространства в 16-ти разрядных ЭВМ

Рассмотрим основные принципы архитектурных решений, используемых для управления памятью в 16-разрядных процессорах ЭВМ. В этих ЭВМ физическое адресное пространство больше виртуального, ограниченного разрядностью адреса и составляющего 2^{16} байт.

Модификация, то есть преобразование используемого в программе виртуального адреса в физический адрес расширенной памяти, осуществляется в ЭВМ путем прибавления к виртуальному адресу так называемой *константы перемещения*, смещающей виртуальный адрес в соответствующее место физической памяти, как это показано на рис.12.1.

Рис.12.1

Для формирования константы перемещения, определяющей фактический адрес в физической памяти, в состав процессора вводят один или несколько

дополнительных регистров. Понятно, что, вообще говоря, число разрядов в константе перемещения должно быть такое же, как и в адресе физической памяти, однако, для того, чтобы обеспечить возможность формирования констант перемещения программным путем, регистры для хранения констант перемещения делают 16-разрядными.

При формировании физического адреса в процессоре к исходному виртуальному адресу прибавляется содержимое регистра, содержащего константу перемещения, но не непосредственно, а со сдвигом влево на соответствующее количество разрядов с дополнением недостающих младших разрядов нулями, как это показано на рис.12.1. Вследствие того, что эти дополняемые младшие разряды константы перемещения не могут изменяться, точность, с которой она может позиционировать виртуальный адрес в физической памяти, определяется числом таких не модифицируемых младших разрядов (на рис.12.1 это шесть разрядов). Другими словами, страница виртуальной памяти отображается не в любое место физической памяти, а дискретно по адресам кратным числу, определяемому количеством добавляемых к содержимому регистра с константой перемещения не модифицируемых младших разрядов. Сама записываемая в такой регистр *константа перемещения задает фактически смещение начала физической страницы (сегмента) от начала физической памяти, а виртуальный адрес задает относительный адрес внутри физической страницы (сегмента)*. В связи с этим, регистры процессора, используемые для указания констант перемещения, в некоторых ЭВМ называются регистрами адреса страниц (РАС). В процессорах Intel они называются сегментными регистрами.

В процессорах Intel 8086, 80286 имеется четыре 16-разрядных сегментных регистра (CS, DS, SS и ES), которые используются для различных целей. Регистр CS указывает, из какого сегмента физической памяти осуществляется выборка команд; регистр DS указывает, из какого сегмента выбираются данные (операнды); регистр SS идентифицирует текущий сегмент стека и, наконец, регистр ES – текущий дополнительный сегмент данных. Организация выполнения в ЭВМ нескольких пользовательских программ, размещенных в разных областях физической памяти, состоит в программировании сегментных регистров таким образом, чтобы виртуальные адреса каждой пользовательской программы переадресовывались в соответствующие области физической памяти.

12.4. Страничная организация памяти

Использование вместо одного сегментного регистра набора регистров адреса страниц позволяет гораздо более гибко организовать отображение виртуального адресного пространства на физическое. Использование набора регистров адреса страниц иллюстрируется на рис.12.2.

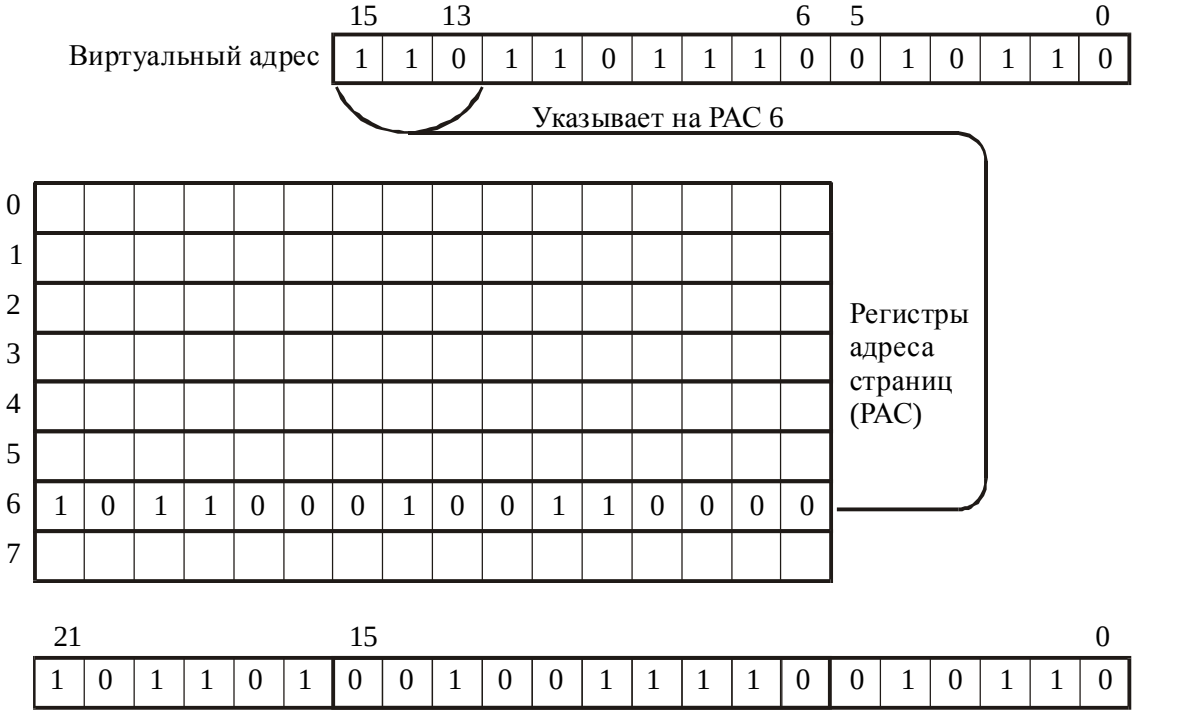

Рис.12.2

В данном случае, как видно из рисунка, для указания констант перемещения используется восемь так называемых *регистров адреса страниц*. Причем номер регистра адреса страницы, который используется для переадресации конкретного виртуального адреса, определяется тремя старшими разрядами конкретного виртуального адреса, используемого в команде.

Такой способ формирования физических адресов позволяет отображать непрерывное виртуальное адресное пространство в восемь физических страниц, произвольным образом, независимо друг от друга, размещенных в физической памяти (рис.12.3). Содержащиеся в регистрах адреса страниц восемь констант перемещения, определяющих конкретное размещение фраг-

ментов пользовательской программы в физической памяти ЭВМ, иногда называются *картой памяти* или *таблицей страниц* программы.

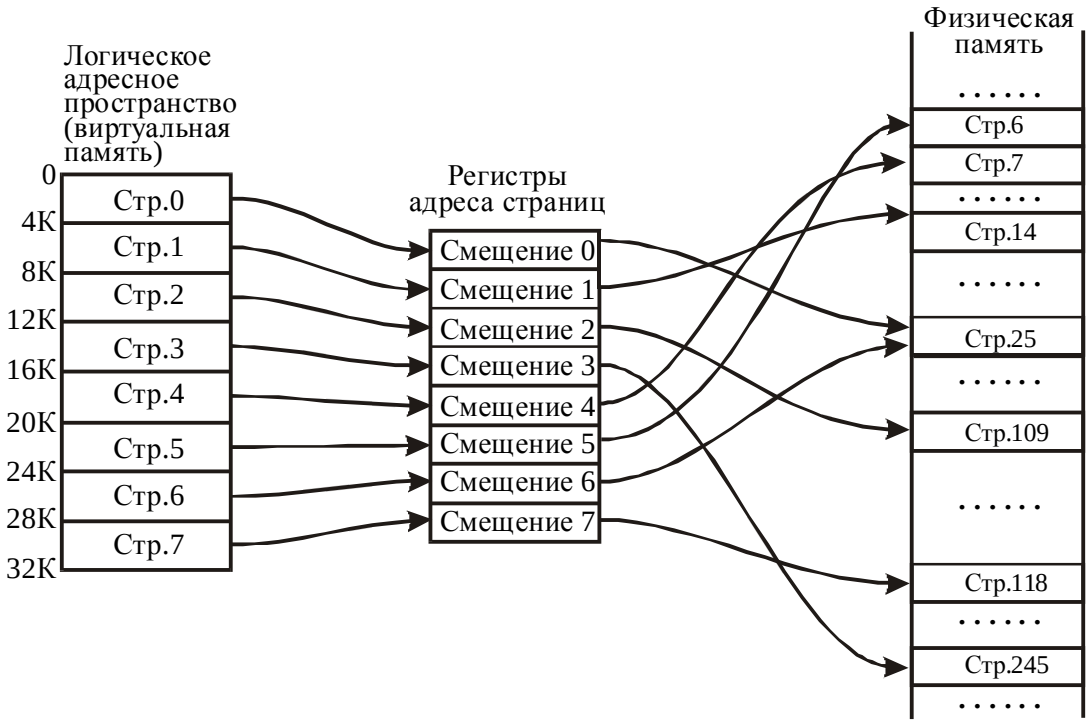

Рис.12.3

Для каждой пользовательской программы используется своя карта памяти, обеспечивающая отображение адресов именно этой программы в свою область физической памяти. При необходимости отдельные страницы разных программ могут перекрываться. Установка начальных значений регистров адреса страниц (карты памяти) и их изменение в процессе работы осуществляется программным путем и обычно выполняется программными средствами операционной системы, обеспечивающими распределение памяти между программами пользователей.

Кроме регистров адреса страниц, в процессоре могут использоваться еще восемь так называемых *регистров описания страниц*. Эти регистры используются для записи информации, связанной с характером работы с конкретной страницей, например, установка режима доступа к данной странице: запись и чтение, только чтение, запрет доступа, контроль того, осуществлялась ли запись информации в данную страницу или нет.

12.5. Управление памятью в многозадачном режиме

Как уже говорилось выше, в 32-х и 64-х разрядных ЭВМ также должно обеспечиваться управление памятью с целью ее эффективного распределения между различными процессами. Для этого также должно осуществляться отображение используемого программами виртуального адресного пространства на физическую память ЭВМ, обеспечивая каждому пользователю возможность использования в программах непрерывного пространства адресов, начиная с нулевого адреса. В отличие от 16-разрядных ЭВМ 32-х и 64-х разрядный адрес позволяет оперировать виртуальным адресным пространством объемом соответственно 2^{32} и 2^{64} байта, что во много раз превышает реализуемый объем физической памяти. Поэтому средства управления памятью отображают в физической памяти только часть виртуального адресного пространства выполняемого процесса. Каждый процесс, соответствующий определенным пользовательским или системным программам, имеет свое собственное адресное пространство, однако, несколько процессов при необходимости могут обращаться к общему пространству (страницам) памяти.

Виртуальное адресное пространство 32-х разрядной ЭВМ разделяется на страницы, например по 512 байт (9 разрядов в адресе). Программа строится в виде последовательности набора страниц, пронумерованных от нуля до некоторого значения. Старшие разряды (с 9-го по 31-й) виртуального адреса определяют номер виртуальной страницы. Физическая память также разделяется на страницы по 512 байт. Адресация физической памяти выполняется с помощью 24-разрядного адреса. Младшие 9 разрядов физического адреса указывают номер байта в странице, а старшие 15 разрядов – номер страницы в физической памяти.

Отображение виртуального адресного пространства в физическую память осуществляется аппаратными средствами процессора, называемыми *диспетчером памяти*, которые, кроме *преобразования виртуальных адресов в физические*, обеспечивают *распределение памяти между исполняемыми процессами* и *управление доступом к памяти*, то есть ее защиту. Общая структура отображения виртуальных адресов в физическую память представлена на рис.12.4.

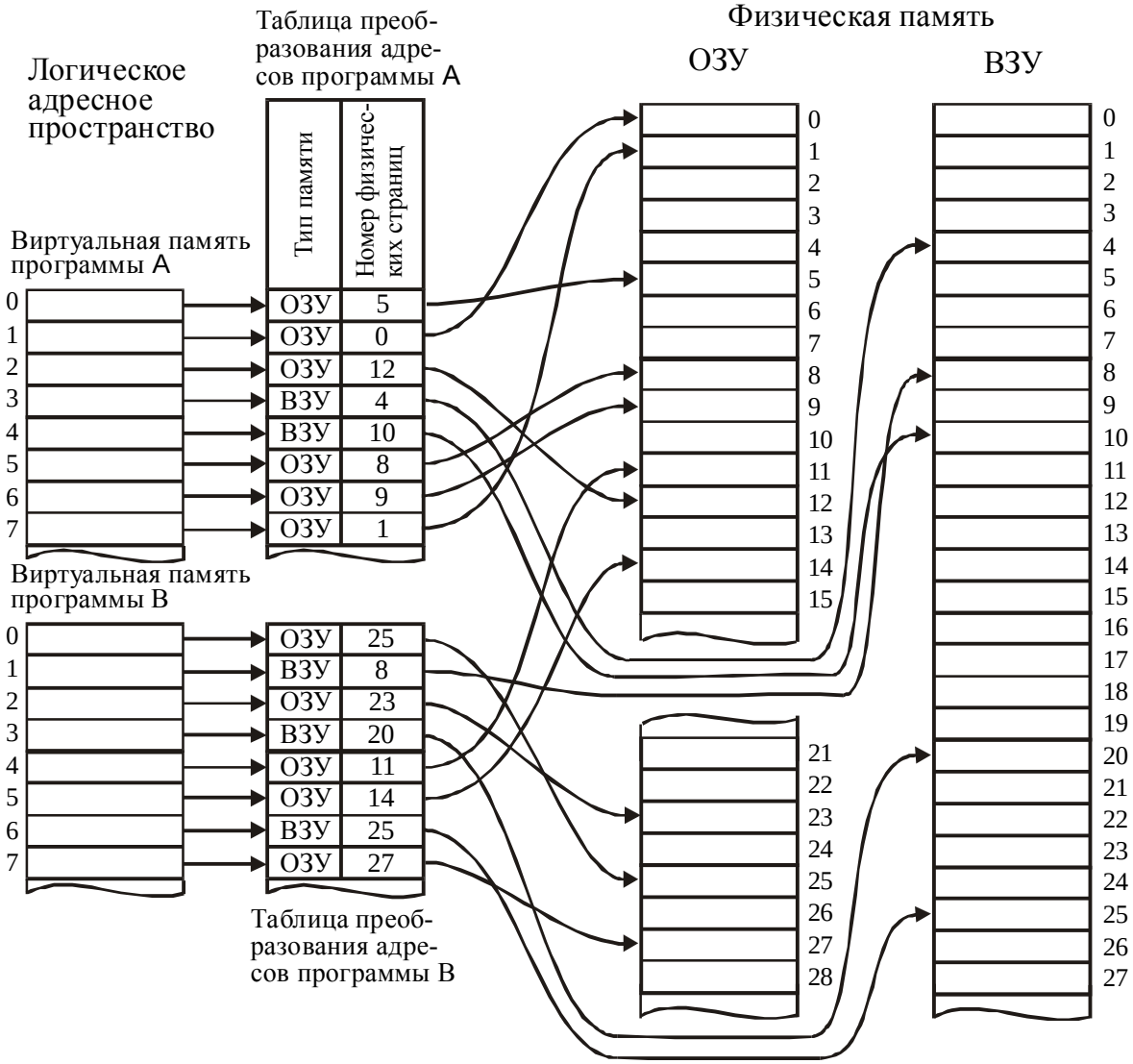

Рис.12.4

На рис. 12.4 показано соответствие между виртуальными и физической памятями, устанавливаемое страничными таблицами двух программ. Видно, что физические страницы могут содержаться в текущий момент времени как в оперативной, так и во внешней памяти.

Виртуальная страничная организация памяти позволяет хранить в физической памяти только часть страниц процесса, а другую часть страниц, которые в данный момент не используются, – на диске. При обращении к страницам они загружаются с диска в физическую память, по мере необходимости, замещая в ней те страницы, которые в данный момент не используются.

Такая процедура загрузки страниц с диска называется *обменом страниц* или *свопингом* (от английского *swap*). При выполнении программы диспетчер памяти должен каждый виртуальный адрес, генерируемый процессом, преобразовать (транслировать) в соответствующий физический адрес. Для трансляции виртуальных адресов в физические используется структура данных, называемая *таблицей страниц*. Эта таблица содержит информацию, необходимую для трансляции адресов. Таблица страниц представляет собой набор записей, по одной на каждую страницу виртуальной памяти. Каждая запись указывает на *режим доступа* процессора (чтение, запись, недоступна) к соответствующей странице физической памяти, *расположение страницы* в физической памяти или на диске, *номер страницы* в физической памяти, если она находится в памяти.

Страничная таблица для каждой программы формируется операционной системой в процессе распределения памяти и перерабатывается ею каждый раз, когда в распределении памяти происходят изменения. Процедура обращения к памяти состоит в том, что номер виртуальной страницы извлекается из адреса (рис. 12.5) и используется для входа в страничную таблицу, которая указывает номер соответствующей физической страницы.

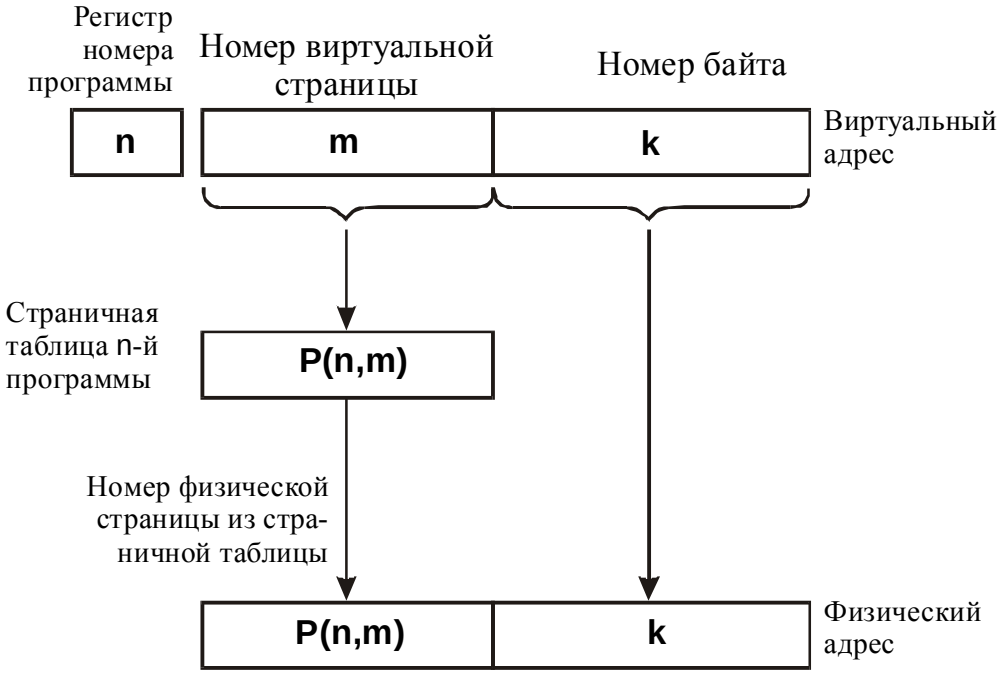

Рис.12.5

Этот номер вместе с номером байта, взятым непосредственно из виртуального адреса, представляет собой физический адрес, по которому происходит обращение к оперативной памяти. Если страничная таблица указывает на размещение требуемой информации во внешней памяти, то обращение к оперативной памяти не может состояться немедленно. Операционная система должна организовать передачу нужной страницы из внешней памяти в оперативную.

Для каждой из программ, обрабатываемых в мультипрограммном режиме, организуется своя виртуальная память и создается своя страничная таблица, при этом все программы делят между собой одну общую физическую память.

Страничные таблицы программ хранятся в оперативной памяти. Обращение к нужной строке активной страничной таблицы в оперативной памяти происходит *по адресу*, который определяется номером активной программы и номером виртуальной страницы (рис. 12.4).

Для ускорения преобразования адресов может использоваться небольшая сверхоперативная память, в которую передается из оперативной памяти страничная таблица активной программы.

12.6. Контрольные вопросы

1. В чем состоит проблема управления памятью ЭВМ, в чем отличие этой проблемы для 16-ти и 32-х (64-х) разрядных ЭВМ?
2. В чем состоит концепция разделения понятий физического адресного пространства (физической памяти) и логического (логической, виртуальной памяти)?
3. Какой максимальный объем памяти можно адресовать с помощью 16-ти разрядного адресного слова?
4. Каким образом в 16-ти разрядных ЭВМ организована возможность формирования адреса для работы с оперативным запоминающим устройством объемом 1 Мбайт (сколько разрядов в адресе для этого требуется)?
5. Каким образом осуществляется отображение страниц логического адресного пространства программы пользователя на страницы физической памяти?

6. Для чего используются регистры адреса страниц (сегментные регистры) процессора?
7. Что такое и для чего используются страничные таблицы?
8. Что такое свопинг?

13. Согласование пропускных способностей процессора и памяти ЭВМ

13.1. Расслоение памяти. Кэш-память

Непрерывный рост производительности (скорости работы) ЭВМ проявляется в первую очередь в повышении скорости процессов внутри ЭВМ. Этот рост достигается использованием новых, более быстродействующих электронных схем, а также специальных архитектурных решений, например – конвейерной и векторной обработки данных. Быстродействие оперативной памяти также растет. Однако оно все время отстает от быстродействия аппаратурных средств процессора, в значительной степени потому, что одновременно происходит опережающий рост ее емкости, что делает более трудным уменьшение времени цикла работы памяти.

Без согласования пропускных способностей процессора и памяти невозможно в машине организовать производительность, соответствующую быстродействию процессора.

Преодолеть указанное противоречие и согласовать пропускные способности памяти и процессора помогают специальные структурные решения.

Конвейеризация процедур цикла выполнения команды (рабочего цикла машины). В простейшем случае это решение выражается в *параллельном* во времени выполнении операций в АЛУ и выборки из памяти следующей команды.

«Расслоение» оперативной памяти путем многомодульного построения с *«веерной»* («чередующейся») адресацией. Смежные адреса информационных единиц, соответствующие ширине выборки (слово, двойное слово и т. п.), размещаются по разным модулям. В этом случае повышается пропускная

способность оперативной памяти за счет перекрытия во времени обращений к разным модулям памяти.

Буферизация памяти. Это решение состоит в использовании *буферных памятей* небольшой емкости, существенно более быстродействующих, чем оперативная память, включенных между процессором и основной оперативной памятью. Буферные памяти скрыты от программиста в том смысле, что он не может их адресовать и может даже не знать об их существовании. Такого рода запоминающие устройства получили название ***кэш-память*** (от английского cache – тайник).

При обращении процессора к оперативной памяти для считывания в кэш-память передается блок информации, содержащий нужное слово. При этом происходит опережающая выборка, так как высока вероятность, что ближайшие обращения будут происходить к словам этого же блока, уже размещенном в кэш-памяти. Это приводит к значительному уменьшению среднего времени, затрачиваемого на выборку данных.

Эффективность кэш-памяти зависит от ее емкости, размера блока, соотношения времен считывания слова из кэш и блока из оперативной памяти. Следует отметить, что *расслоение памяти* существенно уменьшает время считывания из оперативной памяти блока данных, позволяя при том с задержкой на один такт считывать группу слов из ячеек оперативной памяти с последовательными адресами.

Обычно используют два типа кэш-памяти.

- Кэш с запоминанием новой информации одновременно в кэш и в оперативную память («сквозное запоминание» или по-английски store-through),
- Кэш с запоминанием новой информации только в кэш-память и копированием ее в оперативную память только при передаче в другие устройства или при вытеснении из кэш (store-in-cache).

Кэш-память является довольно сложным устройством. Это связано с тем, что она должна содержать средства, определяющие, находится ли в кэш-памяти блок данных со словом, которое запрашивает процессор или нет.

13.2. Контрольные вопросы

1. В чем состоит проблема согласования пропускной способности процессора и памяти?
2. Что такое и для чего используется расслоение памяти?
3. Что такое кэш-память?

Литература

1. Столлингс Вильям. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. – М. : Изд. дом «Вильямс», 2002. – 896 с.
2. Танненбаум Э. Архитектура компьютера. – СПб.: Питер, 2002. – 704 с.
3. Архитектура ЭВМ: Учебные материалы для студентов 1-го курса физического факультета и факультета компьютерных наук; Сост.: А.П. Толстобров. – Воронеж: ВГУ, 2000. – № 496. – 94 с.
4. Архитектура ЭВМ: Учебные материалы к практическим занятиям для студ. 1 курса д/о физического факультета и факультета компьютерных наук / ВГУ, кафедра информационных систем; Сост.: А. С. Коваль, А. В. Сычев. – Воронеж: ВГУ, 2000. – № 112. – 84 с.

Дополнительная литература

5. Каган Б.М. Электронные вычислительные машины и системы. – М.: Энергоатомиздат, 1991. – 590 с.
6. Френк Т.С. PDP-11: архитектура и программирование: Пер. с англ. – М.: Радио и связь, 1986. – 371 с.

Содержание

1. Принципы организации ЭВМ с фон-неймановской архитектурой 4
1.1. Обобщенная структура ЭВМ 4
1.2. Принципы организации ЭВМ 5
1.3. Контрольные вопросы 6
2. Представление информации в ЭВМ. Системы счисления и арифметические операции над числами 7
2.1. Виды информации 7
2.2. Выбор системы счисления для представления чисел в ЭВМ 8
2.3. Представление в ЭВМ целых двоичных чисел без знака 10
2.4. Представление в ЭВМ целых чисел со знаком 11
2.5. Особенности выполнения в ЭВМ сложения двоичных чисел без знака и со знаком 13
2.6. Контрольные вопросы 15
3. Принципы построения устройств для осуществления арифметических и логических операций над двоичными числами 17
3.1. Контрольные вопросы 21
4. Элементы памяти ЭВМ 21
4.1. Триггеры 21
4.2. Организация запоминающего устройства с произвольной выборкой 23
4.3. Контрольные вопросы 24
5. Базовая структура вычислительной системы 24
5.1. Память ЭВМ (оперативное запоминающее устройство) 25
5.2. Центральный процессор 27
5.3. Шинная организация ЭВМ 30
5.4. Внешние устройства 31
5.5. Контрольные вопросы 31
6. Упрощенный цикл выполнения команд в ЭВМ 32
6.1. Цикл выполнения команд 32
6.2. Контрольные вопросы 33
7. Система команд ЭВМ и адресация операндов 34
7.1. Типы команд 34
7.2. Способы адресации операндов 37
7.3. Режимы адресации с помощью регистров общего назначения 39
7.4. Режимы адресации со ссылкой на регистр-счетчик команд 41
7.5. Стек. Организация стека в памяти ЭВМ 43
7.6. Контрольные вопросы 47
8. Команды безусловной и условной передачи управления **Ошибка! Закладка не опре**
8.1. Контрольные вопросы 50

9. Подпрограммы. Использование стека при вызове подпрограмм и возврате из них 50
9.1.Проблемы реализации подпрограмм 50
9.2.Реализация вызова подпрограмм и возврата в основную программу....52
9.3.Контрольные вопросы 55
10. Особенности RISC-архитектуры....55
10.1. ЭВМ с расширенным и сокращенным набором команд. 55
10.2. Контрольные вопросы 57
11. Организация ввода-вывода. Обмен информацией ЭВМ с внешними устройствами....58
11.1. Внешние устройства 58
11.2. Общие принципы организации ввода-вывода....59
11.3. Программный режим ввода-вывода по опросу готовности внешнего устройства. 62
11.4. Программно управляемый обмен информацией в режиме прерывания программы....64
11.5. Прямой доступ к памяти 72
11.6. Буферы данных в системах ввода-вывода 74
11.7. Контрольные вопросы 75
12. Управление памятью ЭВМ, расширение адресного пространства, динамическое распределение памяти 78
12.1. Проблемы управления памятью и расширения адресного пространства 78
12.2. Физическое и виртуальное адресные пространства....79
12.3. Расширение адресуемого пространства в 16-ти разрядных ЭВМ....81
12.4. Страничная организация памяти 83
12.5. Управление памятью в многозадачном режиме 85
12.6. Контрольные вопросы 88
13. Согласование пропускных способностей процессора и памяти ЭВМ 89
13.1. Расслоение памяти. Кэш-память 89
13.2. Контрольные вопросы 91

Составитель кандидат технических наук, доцент
Толстобров Александр Павлович

Редактор *Тихомирова О.А.*