1. Основы микропроцессорной техники

1.1.Общие сведения о микропроцессорных системах

1.1.1. Основные понятия и определения

Введем основные понятия и определения, принятые в микропроцессорной технике [Алексеенко А.Г., Галицин А.А., Иванников А.Д. Проектирование радиоэлектронной аппаратуры на микропроцессорах: Программирование, типовые решения, методы отладки.- М.: Радио и связь, 1984.- 272 с.].

Микропроцессор (МП) - это программно управляемое устройство, осуществляющее процесс обработки цифровой информации и управление этим процессом, реализованное в одной ли нескольких БИС.

Микропроцессорная БИС - это микросхема, реализующая часть функций или все функции МП .

Микропроцессорная ЭВМ (или микроЭВМ) - это ЭВМ, включающая микропроцессор, полупроводниковую память, средства связи с периферийными устройствами и, при необходимости, пульт управления и блок питания, объединенные одной несущей конструкцией.

В зависимости от способа конструирования микроЭВМ делят на:

- однокристальные, выполненные на одном кристалле,
- одноплатные, реализованные на одной плате,
- многоплатные, когда микропроцессор и основная память располагаются на одной плате, средства связи с периферийными устройствами - на других.

Микропроцессорная система (МПС) – информационная, измерительная, управляющая или другая специализированная цифровая система, включающая микроЭВМ и средства сопряжения с обслуживаемым объектом.

Программное обеспечение МПС (ПО МПС) - совокупность программ, которые находятся в памяти системы и реализуют алгоритм функционирования системы.

Простейшая структурная схема МПС.

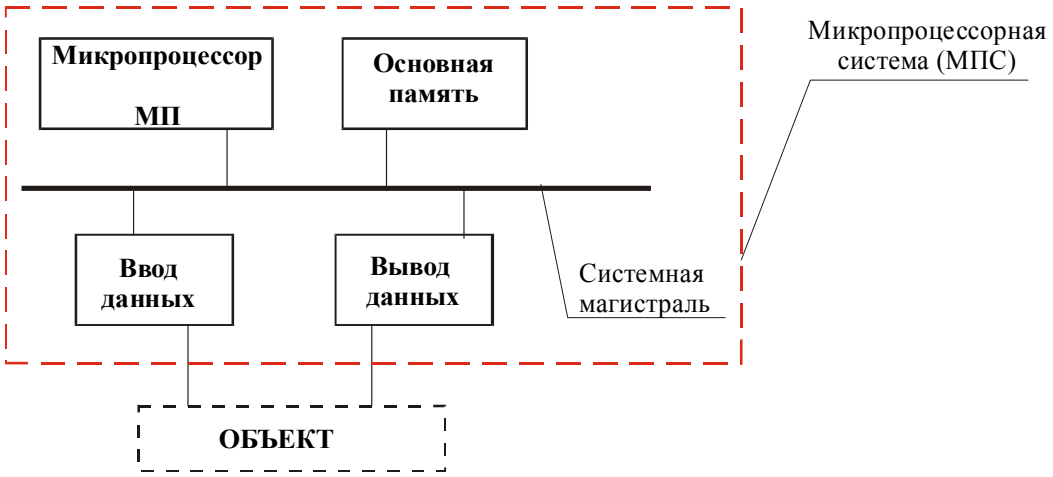


Рис.1.1 Структурная схема микропроцессорной системы.

Функциональное назначение элементов схемы:

МП - управление системой и обработка информации;
Память - хранение двух типов информации - данных и программы;

Ввод данных - ввод данных, преобразование любых информационных сигналов вида "0"-"1".

Вывод данных - вывод данных из МПС пользователю.

Системная магистраль - соединение элементов системы между собой.

Структурная схема МПС.

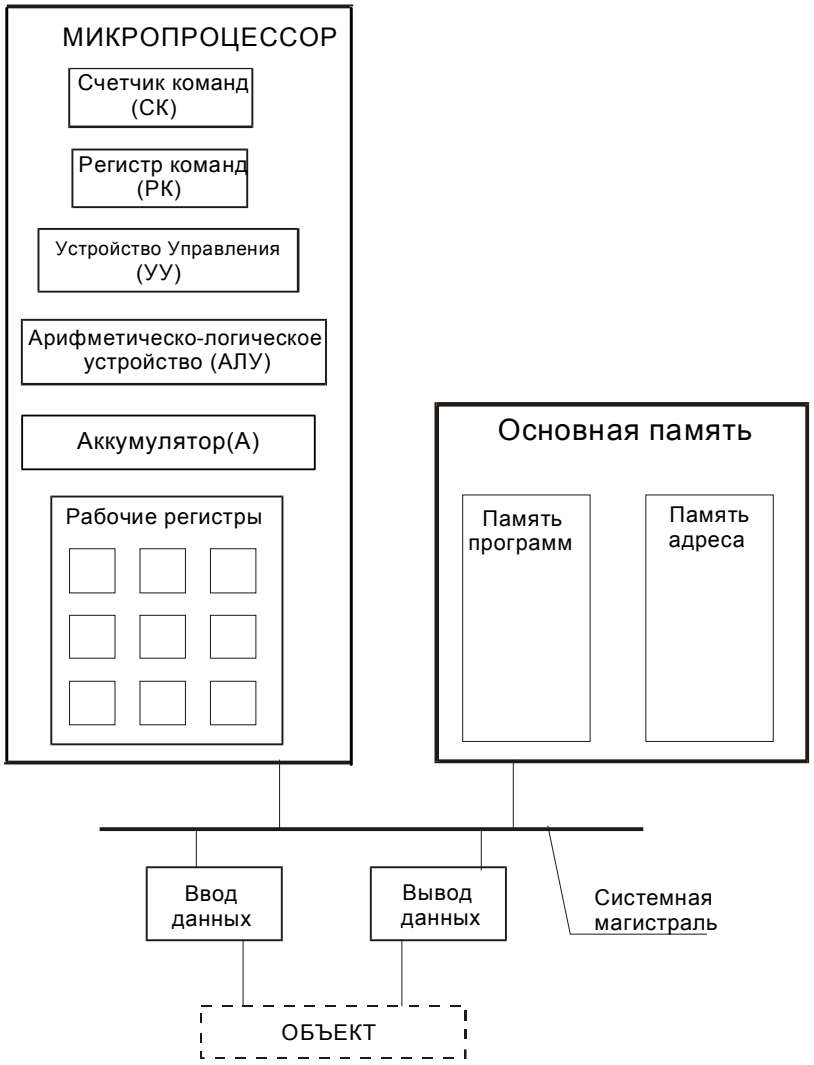


Рис.1.2 Структурная схема микропроцессорной системы.

МП содержит:

Счетчик команд (СК) - регистр, содержащий адрес следующей команды;

Регистр команд (РК) - регистр, содержащий прочитанный из памяти код команды;

Устройство управления (УУ) - управляет работой всех частей МПС. УУ получает код команды, которую надо выполнить и настраивает систему на ее выполнение.

АЛУ - арифметическое логическое устройство, где происходит обработка информации; выполняет арифметические и логические операции;

Аккумулятор (А) - регистр, где хранятся результаты работы АЛУ.

Рабочие регистры – регистры, предназначенные для хранения промежуточной информации.

МП выполняет два действия:

1) Из счетчика команд выдается адрес ячейки, из которой читается очередная команда, код которой записывается в регистр команд;
2) МП выполняет эту команду.

Примерами таких команд является: ввод данных из устройства ввода в аккумулятор, обработка введенных данных, сохранение результата предыдущей обработки в память данных, команды вывода результата обработки.

Таким образом, работу МП можно свести к 4 видам основных команд:

1) Ввод;
2) Вывод;
3) Чтение памяти (ЧтПм);
4) Запись памяти (ЗпПм);
5) Команды внутри МП.

Организация памяти.

Память является совокупностью пронумерованных ячеек. Номер ячейки - это ее адрес. Ячейка представляет собой набор элементов для хранения информации.

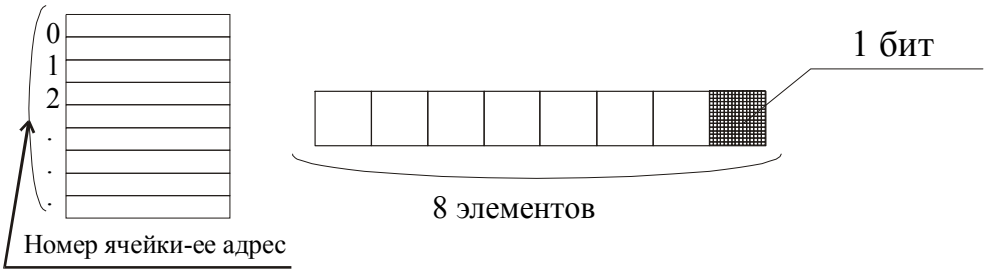


Рис.1.3. Представление памяти

Адреса и содержимое ячеек выдается в шестнадцатиричной системе счисления.

1.1.2. Структурная схема микропроцессорной системы

Подробная структурная схема МПС приведена на рис. 1.4.

Шина адреса (ША) - совокупность линий, по которым передается адрес.

Шина данных (ШД) - совокупность линий, по которым передаются данные.

Шина управления (ШУ) - предает команды всем устройствам. Шина управления передает сигналы: чтение памяти (ЧтПм), запись памяти (ЗпПм), чтение ввода (Ввод), запись вывода (Вывод).

Буфер адреса - в него поступает информация из СК и РА (адреса команд и данных);

Буфер данных - согласует между собой внешнюю и внутреннюю ЩД .

Дешифратор - предназначен для формирования индивидуальных сигналов при поступлении на вход дешифратора соответствующих адресов.

Регистры, имеющие адрес называются *портами*.

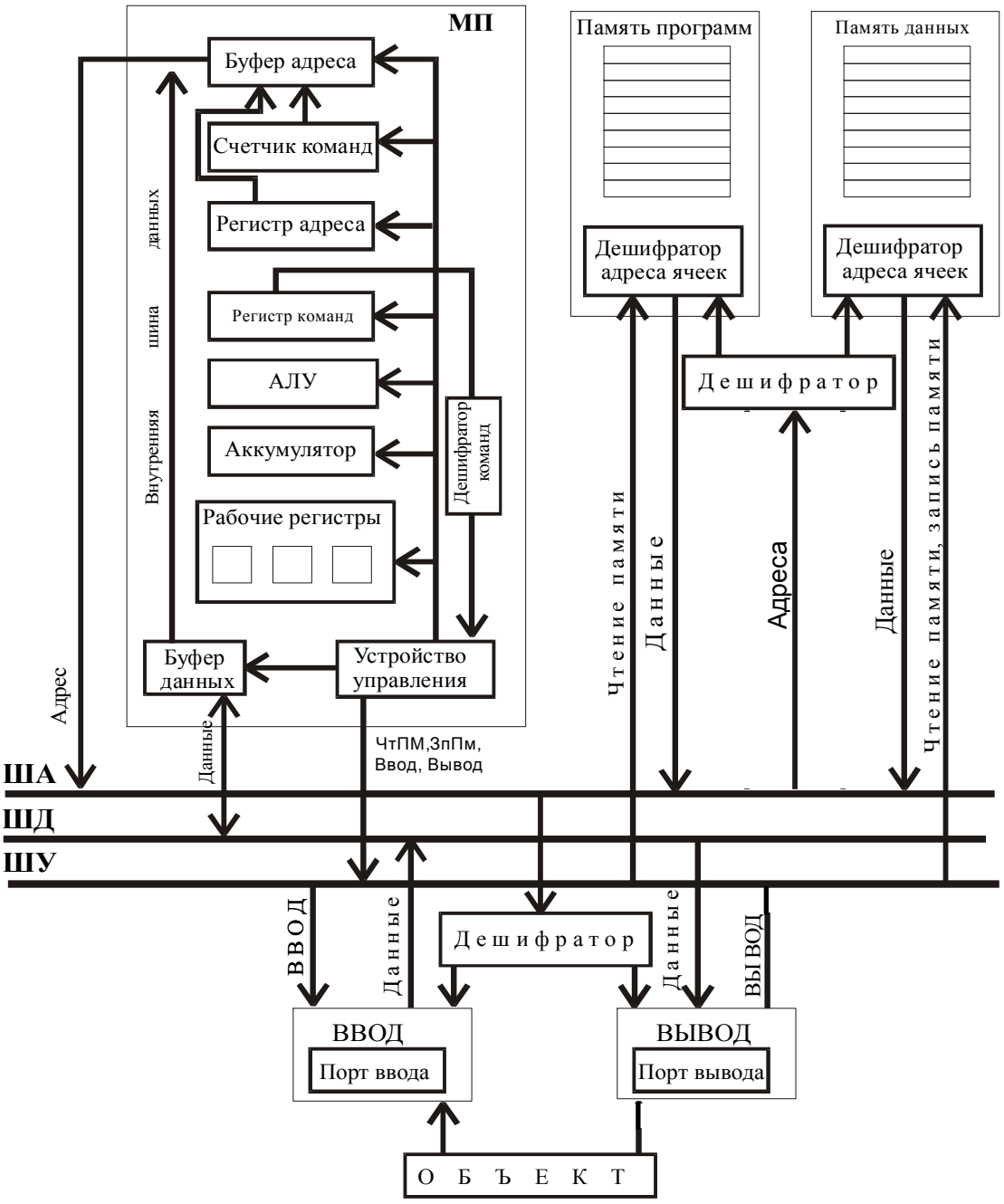


Рис.1.4 Структурная схема микропроцессорной системы.

Работа МП:

1) Из счетчика команд (или *программного счетчика*) СК через буфер адреса БА адрес поступает на шину адреса ША;
2) На ШУ на линию чтения памяти поступает активный сигнал из устройства управления УУ;
3) По адресу, установленному на ША по команде чтения памяти ШУ из ячейки памяти программ читается содержимое и выставляется на шину данных ШД;
4) С ШД код команды записывается в регистр команд РК;.
5) Из РК код команды через дешифратор поступает в УУ;
6) После выборки команды СК увеличивается на единицу, т.е. в нем формируется адрес следующей ячейки;
7) УУ, получив декодированную команду настраивает МП на выполнение команды.

Пункты с 1 по 4 описывают процесс выборки команды, пункты 5- 7 -ее выполнение.

Рассмотрим подробнее основные команды МП (см. также пункт «Временные диаграммы обмена данными в микропроцессорной системе»).

1) Ввод данных.

1) МП выставляет на ША адрес порта ввода.
2) МП выставляет на ШУ на линию ввода активный сигнал, по которому порт ввода выставляет данные на ШД.
3) Через ШД эти данные записываются в аккумулятор.

2)Вывод данных.

1) МП выставляет на ША адрес порта вывода.
2) МП на ШД выставляет данные из аккумулятора.
3) МП выставляет на ШУ активный сигнал вывода, по которому адресуемый порт вывода запоминает выводимые данные и выдает их объекту.

3)Чтение памяти.

1) МП из регистра адреса РА через БА выставляет адрес на ША.
2) МП на ШУ выставляет активный сигнал чтения памяти.
3) Память, получив адрес ячейки и сигнал, выдает на ШД содержимое адресуемой ячейки.
4) МП с ШД эти данные записывает в аккумулятор или рабочие регистры.

4)Запись памяти.

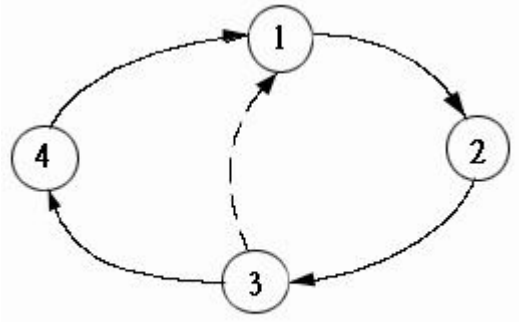
1) МП из регистра адреса РА через БА выставляет адрес на ША.
2) Из аккумулятора или рабочих регистров на ШД выставляются данные, которые надо записать в память.
3) МП на ШУ выставляет активный сигнал записи памяти и информация по этой команде записывается с ШД в адресуемую ячейку памяти.

5)Обмен данными между регистрами .

МП к системной магистрали не обращается и выполняет команду обмена данными между регистрами внутри МП.

Цикл фон Неймана.

Цикл фон Неймана - это диаграмма, которая описывает работу процессора.



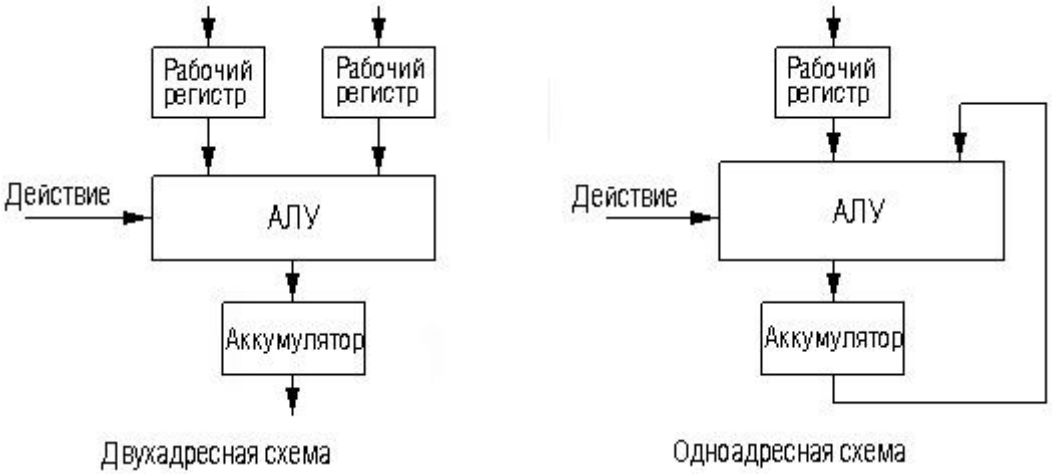
Цикл фон Неймана

"1" - выборка команды;
"2" - счетчик команд увеличивается на единицу;
"3" - дешифрирование кода команды. Если код команды содержит более одного байта (двухбайтные, трехбайтные), то есть команда хранится в нескольких ячейках памяти, то процессор может обращаться к памяти 1,2 и более раз. По первому байту МП узнает, какая это команда.
"4" - выполнение команды.

Цикл (1,2,3) выполняется столько раз, сколько байт занимает команда.

АЛУ - арифметическо-логическое устройство .

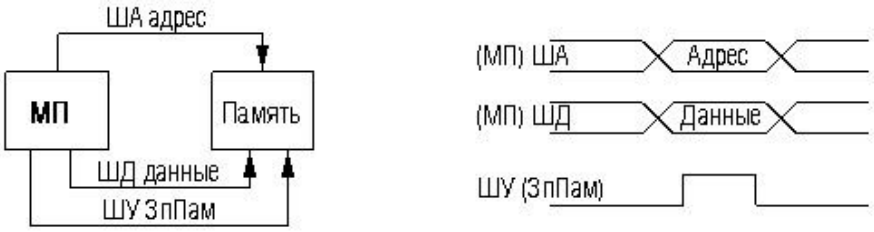
Предположим, что МП выполняет операции сложения, вычитания. Существует две схемы обработки данных: одноадресная и двухадресная.



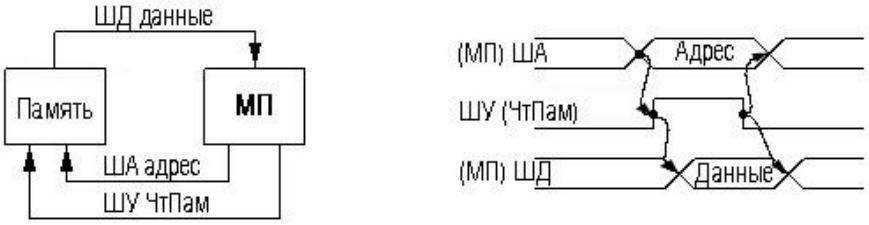
Временные диаграммы обмена данными в микропроцессорной системе.

Возможны 4 варианта обмена данными в МПС:

Запись памяти : STA addr

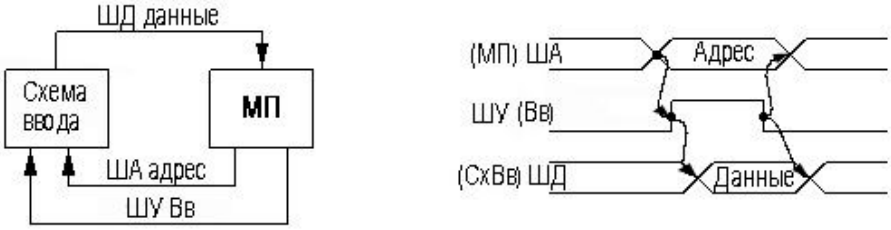


Чтение памяти: LDA addr



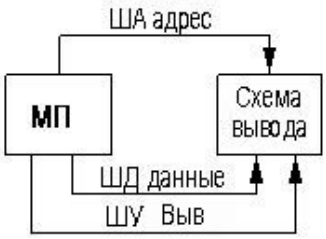
Ввод данных:

IN port

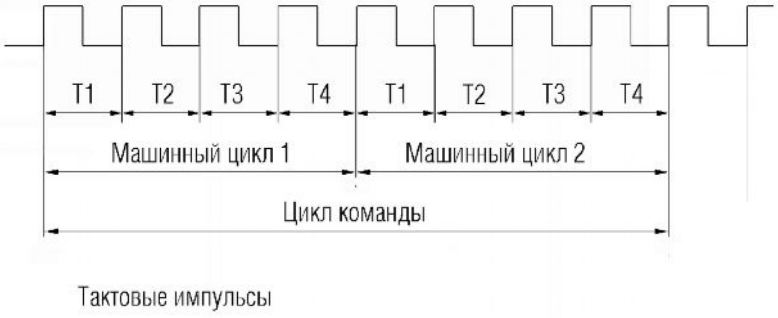


Вывод данных:

OUT port



Временная диаграмма работы МП



За время одного тактового импульса процессор выполняет какое-либо действие. Частота тактовых импульсов определяет быстродействие МП.

Время, в течение которого выбирается и выполняется команда, называется циклом команд. Цикл команд состоит из одного или нескольких машинных циклов. Число машинных циклов определяется количеством обращений МП к памяти или схемам ввода – вывода при выборке и выполнении команды.

За машинный такт МП выполняет элементарное действие (к примеру, выставить адрес на шину адреса, выставить данные на шину данных).

В зависимости от типа процессора машинные циклы могут содержать либо одинаковое

Количество тактов, либо разное. Для процессора К580ВМ80 машинный цикл занимает 3-5 тактов. Первый машинный цикл должен содержать не менее 4 тактов.

Т1 – выставить адрес на шину адреса (часто выставляется управляющее слово, т.е. то что собирается делать МП - ЗП,ЧП, ВВ, ВЫВ);

Т2 – проверка готовности внешних сигналов;

Т3 – ввод кода команды (первого байта);

Т4 – декодирование первого байта команды. В зависимости от этого далее следует либо выполнение команды, либо выборка следующего байта.

1.1.3. Представление информации в микропроцессорных системах

МП-система состоит из элементов, способных находиться в двух состояниях, которым условно приписываются значения 0 и 1. Эти элементы можно считать триггерами (то есть устройствами с двумя состояниями). Таким образом, триггер может хранить информацию в один бит. Напомним, что битом называется единица информации, которая может храниться в элементе с двумя состояниями.

Линейка из n триггеров называется *n-разрядным регистром*. n-разрядный регистр может принимать 2^n различных состояний, т.е. 2^n различных комбинаций нулей и единиц. Заметим, что МП-системы включают множество регистров.

Внутри МП-системы (или ЭВМ) отсутствуют какие-либо пояснительные надписи и запятые, к которым мы привыкли, записывая информацию на бумаге. Вся информация представляется только комбинациями нулей и единиц. Смысл этих комбинаций определяет программист перед тем, как писать программу. В программе программист учитывает ранее заданный смысл комбинации 0 и 1.

Если мысленно вскрыть машину и каким-нибудь прибором (например, тестером) померить состояние регистров, и оно будет равно 10000010, то без дополнительных сведений программиста мы не сможем сказать, что это такое. Это может быть:

- набор несвязанных логических величин;
- символ;
- число или часть числа;
- команда или часть команды;
- указатель.

Логические переменные

Логическую переменную да-нет (0 или 1) можно представить в виде одного разряда, (иногда с избыточностью, например одним байтом), т.е. проще всего.

Позиционные системы счисления

Исторически сложилось так, что мы пользуемся позиционной десятичной системой счисления. В этой системе числа записываются с помощью последовательности десяти цифр 0, 1, 2, 3, 4, 5, 6, 7, 8 ,9, которые являются весовым коэффициентом при соответствующих степенях основания счисления, т.е. 10. Например, число 1985,25 на самом деле является сокращенной записью полинома

$$1*10^3+9*10^2+8*10^1+5*10^0+2*10^{-1}+5*10^{-2}$$

Цифры 1, 9, 8, 5, 2, 5 являются весовыми коэффициентами соответственно при степенях 10^3 , 10^2, 10^1, 10^0, 10^{-1}, 10^{-2}. Видим, что i-е место (позиция) цифры в записи числа определяет, что эта цифра является весовым коэффициентом при степени 10^i. В принципе, числа можно представлять в любой другой системе счисления, двоичной, троичной, пятеричной и т.д. Но люди отдали предпочтение 10-й системе счисления. Однако десятичную систему счисления в МП-системах реализовать трудно. Для них удобнее двоичная система счисления, но она тяжело воспринимается человеком. Например, попробуйте запомнить в двоичной системе счисления номер телефона, состоящий из 7 десятичных цифр. Поэтому программист для краткой записи и удобства запоминания двоичных кодов использует восьмеричную и шестнадцатеричную системы счисления.

Таблица 1.1

Системы счисления							
10-я	2-я	8-я	16-я	10-я	2-я	8-я	16-я
0	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A

3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

Примеры записи десятичного числа 130 в 2-ой, 8-ой и 16-ой системах счисления:

$1000\ 0010_2$, 202_8, 82_{16}

Алфавитно-цифровая информация

Для ЭВМ разработаны стандартные коды для представления алфавитно-цифровой информации. Для микро и мини ЭВМ практически всегда используется код ASCII – Американский стандартный код информационного обмена. Советский аналог этого кода КОИ-7. В этом коде буквы латинского алфавита имеют коды

Табл. 1.1

Буква	Код КОИ-7	16-ый код
A	0100 0001	41
B	0100 0010	42
C	0100 0011	43
. . .	. . .	. . .
. . .	. . .	. . .
X	0101 1000	58
Y	0101 1001	59
Z	0101 0010	5A
Десятичные цифры		
0	0011 0000	30
1	0011 0001	31
. . .	. . .	. . .
. . .	. . .	. . .
8	0011 1000	38
9	0011 1001	39

Представление чисел

Целые числа.

Так как целых чисел бесконечно много, а регистр имеет конечное число разрядов, в МП-системе можно представить только конечное количество целых чисел без знака. Например, 8-разрядный регистр может представлять числа от 0 до 255. Целые числа в МП-системе задаются в двоичной системе счисления.

Пример

двоичная форма представления

Целое число К

7 6 5 4 3 2 1 0 – номер разряда

0 0 0 0 0 0 0 0

0

0 0 0 0 0 0 0 1

1

0 0 0 0 0 0 1 0

2

. . . . . .

1 1 1 1 1 1 1 0

254

1 1 1 1 1 1 1 1

255

Таким образом, для целого числа К имеем:

$$K = \sum_{i=0}^{7} b_i 2^i$$

где bi – весовой коэффициент, принимающий значения 0 и 1;
i – номер позиции.

Преобразования из одной системы счисления в другую

Наиболее просто осуществляется преобразование числа из недесятеричной системы в десятеричную. Примеры

7 6 5 4 3 2 1 0 – номер позиции в двоичной системе счисления.

1 0 0 0 0 0 1 0 =

$1*2^7+0*2^6+0*2^5+0*2^4+0*2^3+0*2^2+1*2^1+0*2^0=1*128+2*1=130_{10}$

2 1 0 номер позиции в восьмеричной системе счисления.

$2\ 0\ 8_8=2*8^2+0*8^1+2*8^0=2*64+2*1=130_{10}$

1 0 номер позиции в шестнадцатеричной системе счисления.

$8\ 2_{16}=8*16^1+2*16^0=128+2=130_{10}$

Сложнее осуществлять перевод десятичного числа в эквивалентную форму другой системы счисления. Для перевода десятичного числа в двоичную, восьмеричную и шестнадцатеричную системы счисления рекомендуется сначала перевести десятеричное число в восьмеричное, а затем восьмеричное в двоичное или шестнадцатеричное. Переведем десятичное число 1986 в 8-ую, 2-ую, 16-ую системы счисления. Для этого переведем это число сначала в 8-ую форму

```
1986 | 8
-16    248 | 8
  38   -24   31 | 8
 -32     8   -24   3
   66    -8    7
  - 64   0
         2
```

$1986_{10}=3702_8$

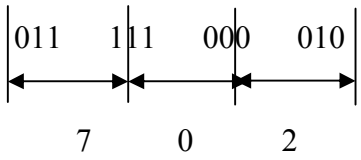
При переходе из 8-ой формы представления числа к 2-ой каждая 8-ая цифра представляется в виде трехразрядного двоичного числа (см. таблица 1.1).

3	7	0	2	8-ое число
011	111	000	010	2-ое число

Получаем

$1986_{10}= 11\ 111\ 000\ 010_2$

При переходе из 2-ой формы представления числа к 16-ой двоичная запись разбивается на четырехразрядные комбинации, начиная от запятой, затем каждая четырехразрядная комбинация переводится в 16-ричную цифру (см. таблицу 1.1)

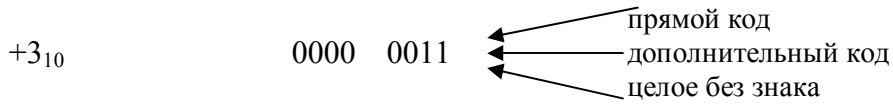


$1986_{10}=702_{16}$.

Целые числа со знаком

Для представления чисел со знаком используются значки + и -. Это соответствует прямому коду, когда отдельно представляется модуль числа и его знак. Для МП-систем такое представление почти не используется, так как реализация операции вычитания в прямом коде требует больше электрических схем. Более удобным оказалось представлять числа со знаком в *дополнительном* коде. Положительные числа в этом коде по представлению совпадают с прямым кодом и с целым числом без знака.

Пример



Чтобы выяснить, как будет выглядеть дополнительный код отрицательного числа, необходимо записать абсолютную величину этого числа и сменить ему знак по следующему правилу.

Правило смены знака числа в дополнительном коде

1. Инвертировать все разряды.
2. Добавить 1 к младшему разряду

Пример:

Записать число со знаком -3 в дополнительном коде.

1. Запишем абсолютную величину этого числа
   +3 0000 0011
2. Инвертируем разряды 1111 1100
3. Добавляем единицу +1

Получаем дополнительный код числа -3 1111 1101

При этом для положительного числа старший разряд в дополнительном коде равен 0,, а для отрицательного равен 1.

Чтобы узнать десятеричный эквивалент двоичного числа, представленного в дополнительном коде, необходимо проверить его знаковый разряд. Если он равен 0, то мы имеем положительное число, которое можно переводить в 10-ую систему счисления. Если знаковый разряд равен 1, то надо сменить знак по правилу смены знака, полученное число перевести в 10-ую систему и приписать ему знак минус.

Выпишем двоичные числа со знаком, которые можно записать в восьмиразрядном регистре

0000	0000	0
0000	0001	1
0000	0010	2
…….		
0111	1101	125
0111	1110	126

0111	1111	127	
1000	0000	-128	**Внимание! Это не нуль**.
1000	0001	-127	**Внимание! Это не единица**.
1000	0010	-126	
……			
1111	1101	-3	
1111	1110	-2	
1111	1111	-1	

Таким образом, для целого числа со знаком имеем:

$$K = -b_7 2^7 + \sum_{i=0}^{6} b_i 2^i$$

Диапазон представленных чисел со знаком в восьмиразрядном регистре

-128 … +127

В общем случае для n-разрядного регистра

$$K = -b^{n-1} 2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Примеры, поясняющие, почему в МП-системе применяется дополнительный код.

0000 0001

+1

1+2=3 +

+

0000 0010

+2

0000 0011

+3

0000 0001

+1

1+(-2)= -1 .+

+

1111 1110

-2

1111 1111

-1

1111 1111

-1

(-1)+2=+1 +

+

0000 0010

+2

отбрасывается единица 1 | 0000 0001 +1

1111 1111

-1

(-1)+(-2)=+1 +

+

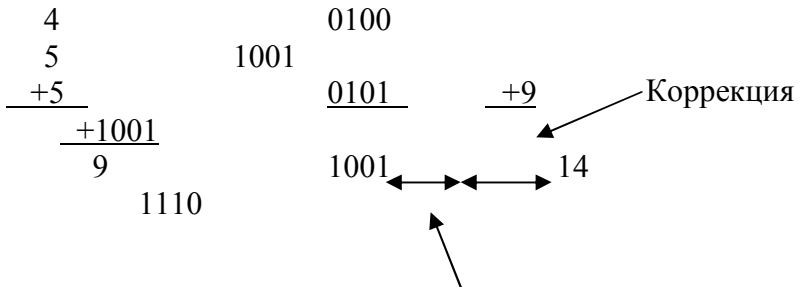
				1111	1110
	-2				
отбрасывается единица	1		1111	1101	-1

Из примеров видно, что при использовании дополнительного кода в МП-системе реализуется только арифметическая операция сложения. Для представления чисел, которые не вписываются в диапазоны представления чисел в одном регистре, используются два, три и т.д. регистров.

Двоично-десятичная система представления чисел

В двоично-десятичной системе десятичные цифры от 0 до 9 представляются четырехразрядными двоичными комбинациями от 0000 до 1001 (см. таблицу 1.1). Комбинации от 1010 до 1111 не используются. Преобразования из двоично-десятичной системы в десятеричную систему и обратные преобразования выполняются путем прямой замены 4-х двоичных цифр одной десятичной цифрой (и обратной замены). Две двоично-десятичные составляют восемь разрядов, т.е. один байт. Таким образом, с помощью одного байта можно представлять значения от 0 до 99. Используя один байт для представления каждых двух десятичных цифр, можно формировать двоично-десятичные числа с любым требуемым числом десятичных разрядов.

Сложение двоично-десятичных чисел, имеющих один десятичный разряд, выполняется так же, как и сложение четырехразрядных двоичных чисел без знака. Если результат превышает 1001, необходимо производить коррекцию путем прибавления двоичного кода числа 6. Примеры:



Хотя при двоично-десятичном представлении чисел память используется менее эффективно, чем при двоичном представлении, оно нашло весьма широкое представление. Это объясняется тем, что двоично-десятичная система позволяет избежать преобразований в двоичную систему и обратно при каждой операции ввода-вывода. Некоторые ЭВМ имеют полный набор команд для обработки данных в двоично-десятичной системе. Ряд ЭВМ имеет команду «Десятичная коррекция», которая позволяет приспособить двоичные операции для обработки десятичных данных. Например, такая команда имеется в МП К580ВМ80.

Действительные числа

Действительные числа имеют две бесконечности: по значению числа ($-\infty$ и $+\infty$) и по точности представления числа (например, числа $\boldsymbol{\pi}$). В связи с этим, в отличие от целых чисел, представление действительного числа ограничено по значению и точности.

Представление действительных чисел с фиксированной запятой

Программист может считать, что в регистре запятая находится не правее младшего разряда, как в целых числах, а где-то внутри регистра, хотя фактически этой запятой в регистре нет.

Пример:

$0011,\ 1100_2 = 3{,}75_{10}$

Диапазон представления чисел с фиксированной запятой легко получается из диапазона представления целых чисел с помощью масштабного коэффициента 2^m, где m-число разрядов дробной части числа.

$$\frac{-2^{n-1}}{2^m} \le x \le \frac{+2^{n-1}-1}{2^m}$$

$$-2^{n-1-m} \le x \le +2^{n-1-m} - \frac{1}{2^m}$$

Например, для n=8 и m=4 имеем

$$-2^3 \le x \le 2^3 - \frac{1}{2^4}$$

$$-8 \le x \le 8 - \frac{1}{16}$$

Погрешность представления чисел с фиксированной запятой

$$\Delta = x_{точн.} - x_{предст.} = \frac{1}{2^m}$$

Наиболее широко применяется представление с фиксированной запятой, когда запятая стоит после старшего разряда, т.е. при представлении чисел, меньших 1. Недостатком представления чисел с фиксированной запятой является малый диапазон и то, что погрешность представления задается абсолютной погрешностью Δ. На практике чаще всего интересует относительная погрешность, т.е. например, количество значащих десятичных цифр в представлении числа.

Представление действительных чисел с плавающей запятой

В этом представлении погрешность является относительной величиной. Действительное число с плавающей запятой имеет вид:

$$x = M * 2^P,$$

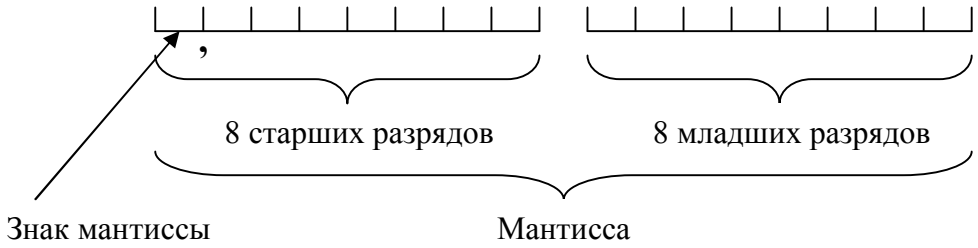
где P – порядок числа.

M – мантисса числа, причем

$0 \le |M| \le 1$ - не нормализованная мантисса

$0{,}5 \le |M| \le 1$ - нормализованная мантисса

Таким образом, действительное число с плавающей запятой представляется в МП-системе двумя числами M и P, где M – число с фиксированной запятой, у которого запятая стоит перед старшим значащим разрядом регистра; P – целое число со знаком. В МП-системах используется разное представление мантиссы. Например,



Мантисса хранится в памяти в нормализованном виде, т.е. после запятой идет значащая цифра. Для положительных чисел первая значащая цифра 1, а для отрицательных 0. Диапазон порядка для восьмиразрядных регистров от -128 до +127. Диапазон действительных чисел от -2^{127} до $+2^{127}$, что соответствует десятичному диапазону от -10^{38} до $+10^{38}$. Относительная погрешность δ задается выражением

$\delta = 2^{-m}$, где m – количество разрядов после запятой в представлении мантиссы.

Если мантисса занимает два 8-ричных регистра, то m=15. Следовательно,

$\delta = 2^{-15} \approx 10^{-4}$,

т.е. четыре десятичных знака.

1.1.4. Основная память

Классификация и место памяти в МП представлены на рисунках ниже.

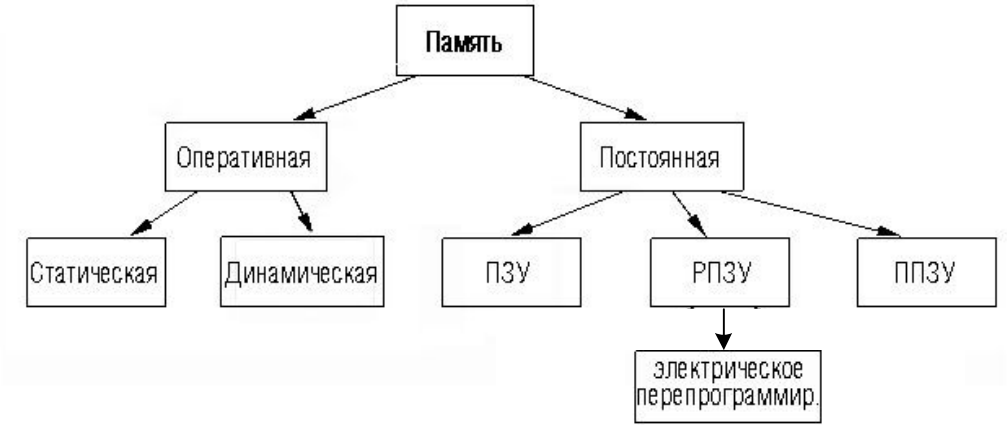


Рис. Классификация основной памяти

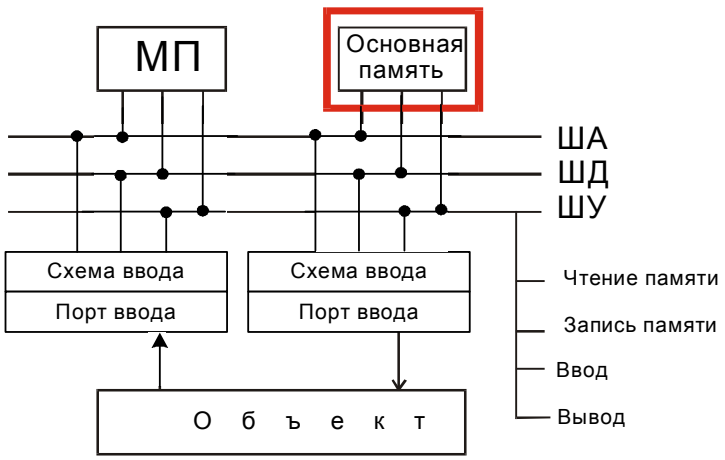


Рис. Место основной памяти в структурной схеме МП

Последовательность команд (т.е. программа) хранится в области основной памяти, называемой памятью программы. Данные, обрабатываемые по этим командам, хранится в памяти данных. Такое разделение памяти иногда достаточно условно, так как память

программы и память данных не всегда расположены отдельно. В этом случае считается, что если ячейка памяти содержит команду, то она является частью программы, а если ячейка памяти содержит данные, то она является частью памяти данных вне зависимости от того, где она физически находится в МП-системе.

Различают *энергозависимую* и *энергонезависимую* память. В энергозависимой памяти информация теряется при выключении питания, например, так происходит в оперативной памяти (ОЗУ). В энергонезависимой памяти информация не теряется при выключении питания, как, например, в постоянной памяти (ПЗУ).

Статическая память- информация сохраняется вплоть до выключения питания. *Динамическая память* – через некоторое время информация разрушается, поэтому ее необходимо возобновлять.

ПЗУ -Постоянное Запоминающее Устройство - информация записывается при изготовлении МПС.

ППЗУ - Программируемое Постоянное Запоминающее Устройство - ее можно также запрограммировать один раз, но это может быть выполнено пользователем.

РПЗУ - перепрограммируем Запоминающее Устройство - возможно перепрограммирование бесконечное количество раз.

РПЗУ с ультрафиолетовым стиранием- информация стирается при помощи ультрафиолетового излучения. В настоящее время практически не используется.

РПЗУ перепрограммируемая электрически- информация стирается электрическим током. Сейчас наиболее часто используется.

Число различных команд, которые может выполнить МП-система, частично определяется разрядностью машинных слов. В МП-системах с восьмиразрядными машинными словами можно представить максимум 2^8=256 команд. Такие МП-системы называются восьмиразрядными МП-системами. Если МП-система работает с 16-разрядными словами, то они являются 16-разрядными МП-системами.

Часто вне зависимости от разрядности машинного слова МП-системы основная память имеет побайтовую организацию, т.е. одна ячейка памяти имеет восемь разрядов. Поэтому емкость памяти, т.е. максимальное число восьмиразрядных ячеек (слов) памяти, выражается в байтах или килобайтах (1 Кбайт=1024 байт), или мегабайтах (1 Мбайт=1024 Кбайт). Иногда указывается число разрядов в ячейке памяти и число ячеек памяти, например, 1Кх4 или1Кх8. Это обозначает, что емкость памяти составляет 1024 четырехразрядных и 1024 восьмиразрядных ячеек (слов) соответственно.

Кроме длины машинного слова важной характеристикой МП-системы является разрядность адреса (номера) ячейки памяти, содержащей слово. Большинство восьмиразрядных и многие их 16-разрядных МП-систем имеют двухбайтовые адреса, с помощью которых можно адресовать 2^{16}=65536 восьмиразрядных слов. В этом случае самый младший разряд будет

$$\underbrace{0000\ \ldots\ 0000}_{16\text{ разрядов}}{}_{(2)} = 0000_{16}$$

а самый старший адрес будет

$$\underbrace{1111\ \ldots\ 1111}_{16\text{ разрядов}}{}_{(2)} = FFFF_{16}$$

Часто разряды одного машинного слова физически расположены в разных интегральных схемах (кристаллах) памяти. Кристаллы памяти, которые обеспечивают

хранение машинного слова, называют *модулем памяти*. Варианты модулей памяти для хранения восьмиразрядных слов показаны на рисунках ниже.

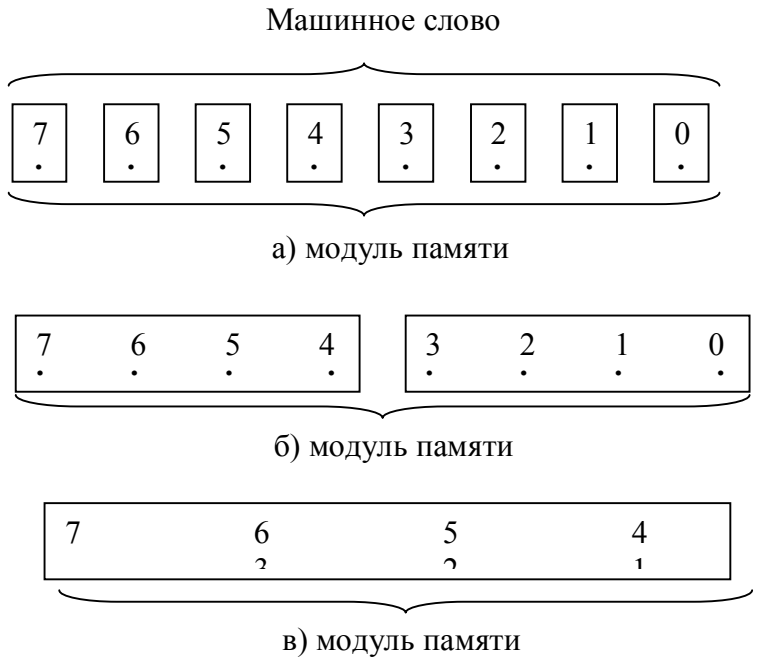


Рис. Варианты модулей памяти для хранения восьмиразрядных слов.

На приведенных выше рисунках дан пример модуля памяти, состоящего из восьми кристаллов (а). Один разряд каждого слова содержится в одном кристалле. На (б) показан модуль памяти, состоящий из двух кристаллов, в каждом из которых хранится четыре разряда всех слов. На (в) дан пример размещения 8-разрядного слова в одном кристалле.

Адресация

Число разрядов в адресе памяти зависит от емкости памяти. Если емкость памяти составляет 1Кбайт=1024 слов, то необходимо использовать десятиразрядный адрес ($1024=2^{10}$). Если основная память МП-системы содержит более одного модуля памяти, часть кода должна указывать, в каком модуле памяти расположено данное слово. Эта часть называется *кодом выбора модуля* или *кодом выбора кристалла*. Часть адреса, которая выбирает слово памяти внутри модуля, называется *адресом слова*

Пример адреса

0011 011100101101

Выбор модуля Адрес слова в модуле

Декодирование адреса слова осуществляется в самих кристаллах памяти. Для декодирования адреса модуля используется отдельная интегральная схема (дешифратор). От особенностей организации основной памяти зависит способ адресации, т.е. какие разряды адреса используются для выбора модуля, и какие для адресации слова в модуле.

Примечание.

МП-система может содержать различные типы памяти, например, модуль ОЗУ) и модуль ПЗУ.

При адресации в соответствии системой выбора модуля и адресом слов необходимо быть последовательным в ее использовании и в этом случае модули ПЗУ и ОЗУ и т.д. имеют свои коды выбора модуля.

Рассмотрим пример

Дано. Основная память МП-системы состоит из 4-х модулей: одного модуля ПЗУ и 3-х модулей ОЗУ. Емкость модуля ПЗУ 1Кбайт=2^{10} слов, емкость каждого из модулей ОЗУ

4Кбайта=2^{12} слов. Адресация памяти в данной МП-системе осуществляется по 16-разрядной шине.

Вопрос. Как адресовать слово?

Возможное решение. Чтобы выбрать слово в модуле ПЗУ требуется 10 разрядов для адреса слова (2^{10} =1024=1К) и остается 6 разрядов для выбора модуля. Если выберем код 000000 в качестве кода выбора этого модуля ПЗУ, то адреса слов будут лежать в пределах

от $\underbrace{0000\ldots0000}_{16\text{ разрядов}}{}_{(2)} = 0000_{16}$ $\underbrace{0000001\ldots1}_{10\text{ разрядов}}{}_{(2)} = 03FF_{16}$

Чтобы выбрать слово в одном из модулей ОЗУ, требуется 12 разрядов для адреса слова (2^{12}=4096=4К) и остается 4 разряда для выбора модуля ОЗУ.

Если назначим код 0001 для первого модуля ОЗУ, адреса слов в этом модуле будут находиться в пределах

от $0001\underbrace{0\ldots0}_{12\text{ разрядов}}{}_{(2)} = 1000_{16}$ $0001\underbrace{1\ldots1}_{12\text{ разрядов}}{}_{(2)} = 1FFF_{16}$

Во втором модуле ОЗУ, для выбора которого используется код 0010, адреса слов будут находиться в пределах

от $0010\underbrace{0\ldots0}_{12\text{ разрядов}}{}_{(2)} = 2000_{16}$ до $0010\underbrace{1\ldots1}_{12\text{ разрядов}}{}_{(2)} = 2FFF_{16}$

В третьем модуле ОЗУ, который выбирается кодом 0011, адреса слов будут лежать в пределах

от $0011\underbrace{0\ldots0}_{12\text{ разрядов}}{}_{(2)} = 3000_{16}$ до $0011\underbrace{1\ldots1}_{12\text{ разрядов}}{}_{(2)} = 3FFF_{16}$

На рисунке ниже схематически представлены адреса всех машинных слов. Такая схема называется *картой памяти*.

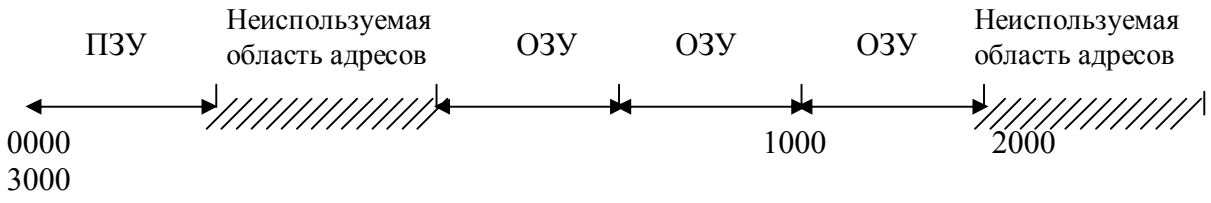


Рис. Карта памяти МП-системы

Пример организации памяти приведен на рисунке ниже

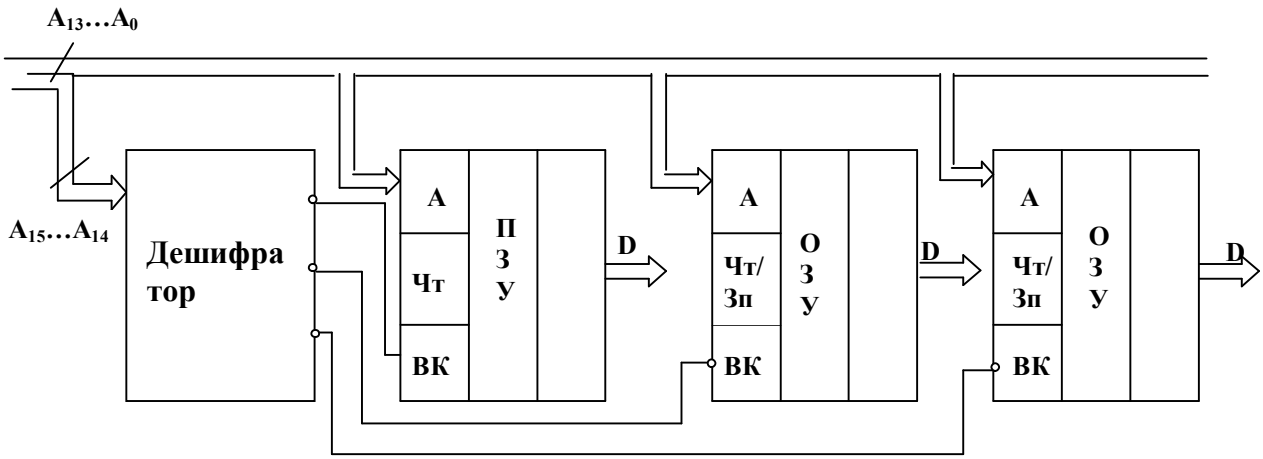
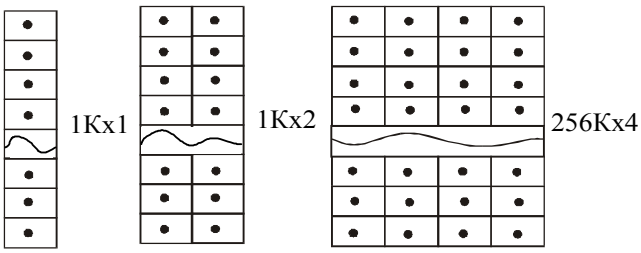


Рис. Пример организации памяти

Примеры структур кристаллов памяти приведены на рисунке ниже.



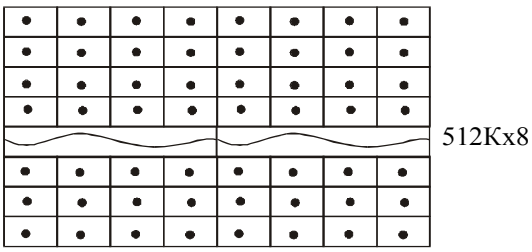
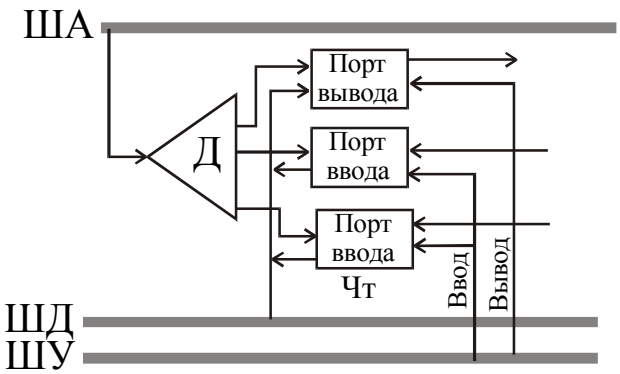


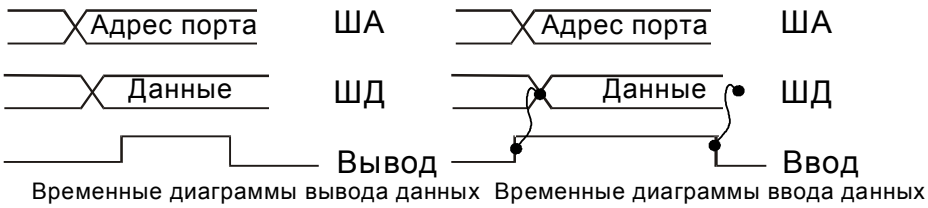
Рис. Примеры структур кристаллов памяти

1.1.5. Модуль ввода-вывода

Модуль ввода-вывода может содержать одновременно порты ввода и порты вывода. Структурная схема модуля ввода-вывода показана на рисунке ниже.



Временные диаграммы ввода и вывода данных показаны на рис. ниже.



Временные диаграммы вывода данных Временные диаграммы ввода данных

При выводе данных из МП на модуль вывода по ША подается адрес порта, по ШД подаются данные, по ШУ подается управляющий сигнал Вывод. Данные защелкиваются в буфере порта вывода и выводятся пользователю. Они сохраняются до тех пор, пока в порт вывода не запишется другая информация.

При вводе данных из МП на модуль ввода по ША подается адрес порта, по ШУ подается управляющий сигнал Ввод. Управляющий сигнал Ввод открывает порт ввода и данные из внешнего мира поступают на ШД, с которой данные фиксируются, например, в аккумуляторе микропроцессора.

1.1.6. Введение в программирование на языке ассемблера

Редактор связей (компоновщик) позволяет получить выполняемую программу.
Отладчик позволяет по шагам выполнять программу и следить за состоянием программы.
Транслятор переводит коды команд в машинный код. Средства разработки - все сервисные программы, рассмотренные выше. Текст программы набирается в любом текстовом редакторе (расширение файла .asm). При транслировании получается файл с расширением .obj. Компоновщик создает из объектного файла исполняемый файл с расширением .exe,.com.
Формат программы на языке ассемблера состоит из четырех полей:

Метка	Код операции	Операнд	Комментарии
			;

Пример. Разработать программу для МП-системы, структурная схема которой приведена ниже.

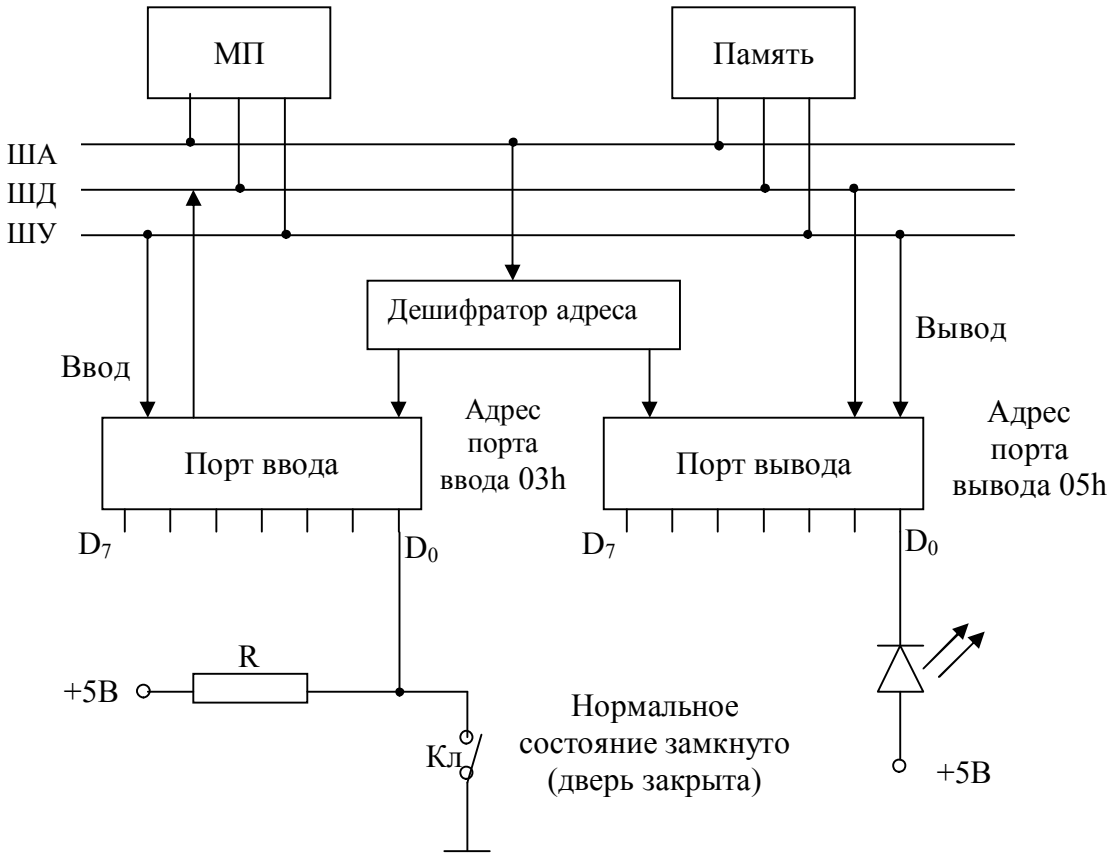


Рис. Структурная схема микропроцессорной системы сигнализации

Метка	Код операции	Операнд	;	Комментарии
METO:	MVI	A,1	;	1→A
	OUT	05H	;	A→порт вывода
MET1:	IN	03H	;	порт ввода→A
	ANI	00000001B	;	A&00000001B→A
	JZ	MET1		Ключ разомкнут?
	MVI	A,0	;	Да. 00000001→A
	OUT	05H	;	A→порт вывода
MET2:	IN	03H	;	порт ввода→A
	ANI	00000001B	;	A&00000001B
	JNZ	MET2		Ключ замкнут?
	JMP	METO		Да.

1.2. Архитектура однокристальных процессоров и программирование на языке ассемблера

Под архитектурой МП понимается его структурная схема и система команд

1.2.1. Архитектура простейших гипотетических микропроцессоров

Рассмотрим гипотетический процессор с аккумулятором (МПА), структура которого приведена на рисунке ниже.

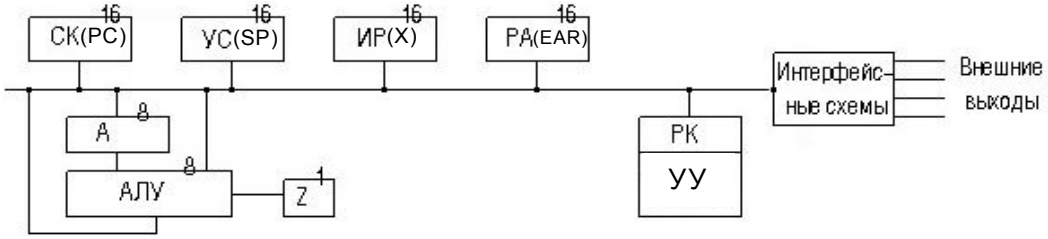


Рис. Структура гипотетического процессора с аккумулятором (МПА)

Z - *признак нуля*, признак результата завершения команды в АЛУ. Z=1, если результат равен 0 и Z=0, если результат не равен 0.

SP - *указатель стека*. Стек- это такой способ организации хранения данных, при котором существует доступ только к последней ячейке памяти, в которую была записана информация. В указателе стека хранится адрес верхушки стека - ячейки, в которую последний раз была записана информация.

X - *индексный регистр*. В него записывается адрес ячейки, из которой надо взять данные или куда надо поместить данные. Вторая функция индексного регистра - хранение промежуточных данных.

EAR - *регистр адреса,* предназначен для формирования и хранения адреса ячейки с данными. Если необходимо обратиться к данным, то адрес выставляется на ША из этого регистра.

Система команд для МПА.

Команды работы с аккумулятором.

NOP [no operation] - выбирая эту команду, МП ничего не делает, а переходит к следующей команде. Команда, используется для организации задержки.

CLRA [A = 0] - обнуление содержимого аккумулятора.

CMA - инвертирование содержимого аккумулятора.

NEGA [A = -A] - меняет знак у числа в аккумуляторе.

Команды загрузки.

LDA data [data -> A] - загрузить в аккумулятор константу.

LDA addr [MEM[addr] ->A] - загрузить в аккумулятор данные из ячейки памяти с адресом addr.

LDA @X - загрузить в аккумулятор данные из ячейки памяти, адрес которой хранится в регистре X.

Команды сохранения

STA [STORE] - сохранить;

STA addr [A -> MEM[addr]] - сохранить из аккумулятора данные в ячейку памяти с адресом addr.

STA @X [A -> X[MEM]] - сохранить из аккумулятора данные в ячейку памяти, адрес которой хранится в регистре X.

Арифметические и логические операции.

ADDA addr [A + addr -> A] - добавить в аккумулятор константу.

ADDA @X [A + X[MEM] -> A] - добавить в аккумулятор данные из ячейки, адрес которой хранится в регистре X.

ANDA addr [A&MEM[addr] -> A] - логическое умножение содержимого аккумулятора и данных из ячейки с адресом addr; результат - в аккумуляторе.

ANDA @X [A&X[MEM] -> A] - логическое умножение содержимого аккумулятора и данных из ячейки, адрес которой хранится в регистре X; результат умножения - в аккумуляторе.

CMPA addr [A - MEM[addr]] - вычитает из содержимого аккумулятора содержимое ячейки с адресом addr, при этом содержимое аккумулятора не меняется, но формируется признак нуля.

Команды работы со стеком.

LDSP addr [адрес -> SP] - инициализация стека.

Команды условной передачи управления.

BEQ offset - если признак нуля Z=1, то происходит переход. offset – восьмиразрядное число со знаком. Смещение происходит относительно счетчика команд, т.е. содержимое счетчика команд суммируется со смещением и получается адрес следующей команды.

BNE offset – если Z=0, то переход на offset;

BR offset – всегда переход на offset.

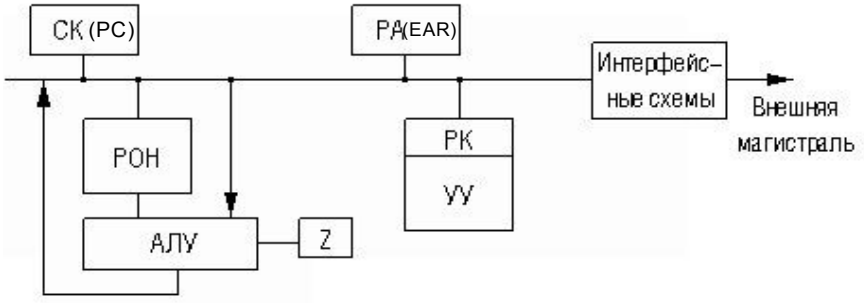
JMP addr – преход на команду с адресом addr. Такой переход называется безусловным.

Команды работы с подпрограммами.

CALL addr – обращение к подпрограмме (СК->стек, addr->СК);

RETURN – возвращение из подпрограммы (Стек->СК).

Микропроцессор с регистрами общего назначения (МПРОН).



РОН – регистры, в которых могут храниться как адреса, так и данные. Любой из регистров общего назначения может исполнять роль аккумулятора, индексного регистра, указателя стека.

Команды могут быть для восьми- (CLRB), так и шестнадцатиразрядных данных (CLR).

NOP [no operation] - выбирая эту команду, МП ничего не делает, а переходит к следующей команде. Команда, используется для организации задержки.

CLR(B) op - обнуление содержимого операнда op.

COM (B) op - инвертирование операнда op.

NEG(B) op - меняет знак у операнда op.

Команды загрузки/сохранения .

LD(B) reg, op - загрузить операнд op в регистр reg .

ST(B) reg, op - сохранить данные операнда в регистре reg.

ADD(B) op,reg - op+reg-> reg

AND(B reg, op - op&reg →reg

CMP(B) reg, op – reg-op формируется признак нуля. Результат никуда не записывается. Выполняется тогда, когда надо сравнить два числа, если они равны, то формируется признак нуля.

Команды условной передачи управления.

BEQ offset - если признак нуля Z=1, то происходит переход на offset.

BNE offset – если Z=0, то переход на offset;

BR offset – всегда переход на offset.

JMP op – переход по адресу в операнде (безусловный переход.)

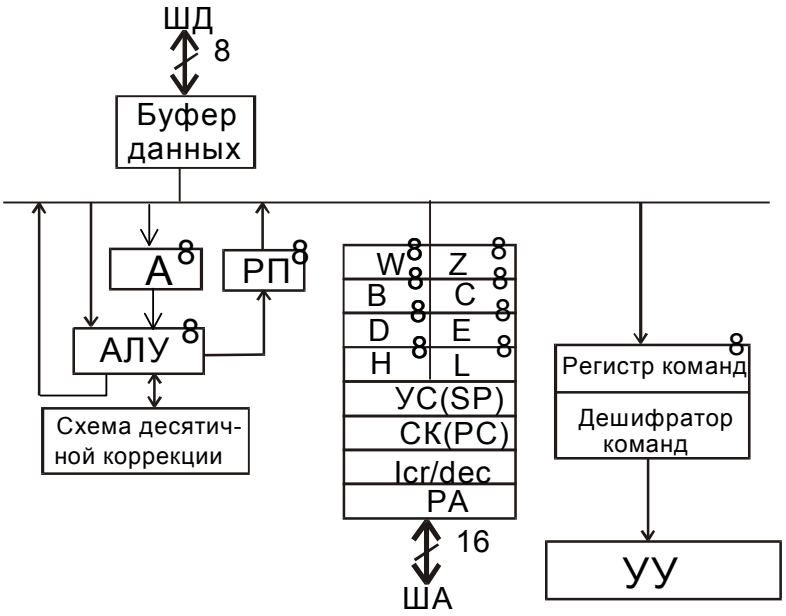
Команды работы с подпрограммами.

CALL op – обращение к подпрограмме;

RET – возвращение из подпрограммы.

1.2.2. Структурная схема и система команд микропроцессора I8080

Структурная схема МП I8080 (КР580ВМ80) представлена на рисунке ниже



Структура МП К580ВМ80

Условные обозначения:

СК- счетчик команд;

А- аккумулятор;

РП- регистр признаков;

АЛУ - арифметико-логическое устройство;

УУ - устройство управления.

Рассмотрим основные функции составляющих МП.

Арифметико-логическое устройство (АЛУ). В МП имеется АЛУ, предназначенное для выполнения арифметических и логических операций. В АЛУ рассматриваемого МП выполняются арифметические операции сложения и вычитания и логические операции И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ.

Регистры общего назначения можно использовать во всех арифметических и логических командах. Эти регистры можно объединять в пары, тогда получится 16-тиразрядный регистр:

(B,C) – B-пара;

(D,E) – D-пара;

(H,L) – H-пара.

В то же время каждый их них имеет определенную специализацию. Так например, H-пара часто используется для адресации к памяти (в нее помещается адрес ячейки и производится косвенная регистровая адресация).

Управляющие сигналы.

Управляющие сигналы устройства управления показаны на рисунке ниже

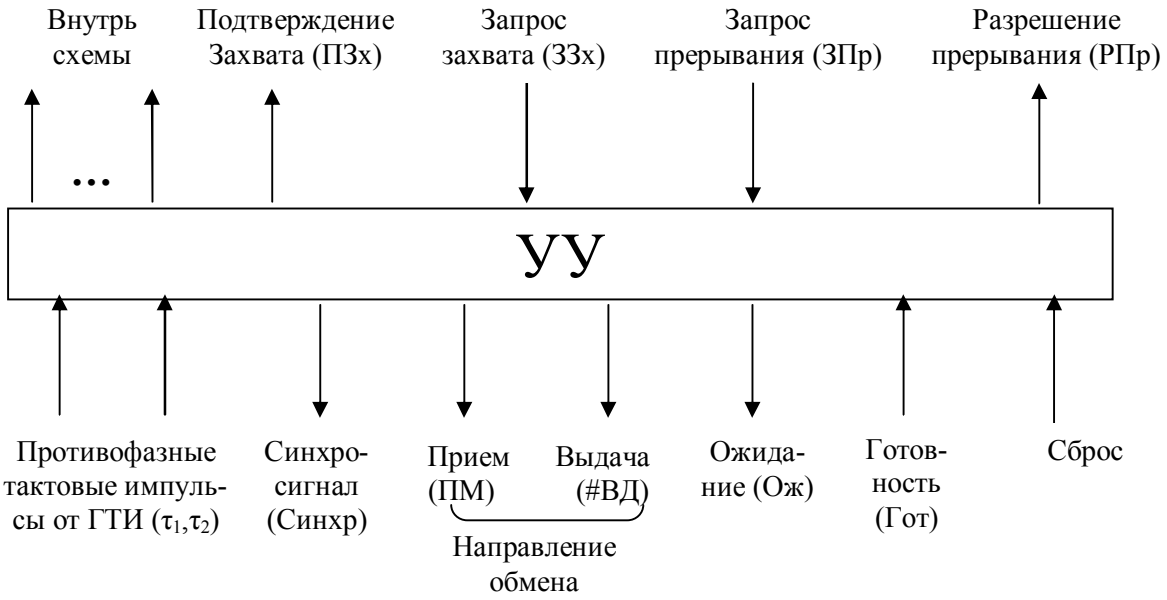


Рис. Управляющие сигналы устройства управления

Синхросигналом *Синхр* можно записать в регистр управляющее слово (УС).

Сброс - по этому сигналу счетчик команд СК обнуляется.

Прием (ПМ) - сигнал приема МП данных с ШД.

Выдача (ВД) - сигнал выдачи МП данных на ШД.

Работу МП можно приостановить, если снять сигнал готовности (Гот)

Получив сигнал ЗПр, микропроцессор перестает выполнять текущую программу и переходит на подпрограмму обслуживания устройства, которое выдало этот запрос. Этот сигнал проверяется на последнем такте каждого машинного цикла команды.

ЗЗх и ПЗх - обеспечивают режим прямого доступа к памяти. Получив сигнал ЗЗх, МП отключается от магистрали и с этого момента КПДП (контроллер прямого доступа к памяти) начинает управлять магистралью.

Управляющее слово (УС).

D7	D6	D5	D4	D3	D2	D1	D0

D7 – ЧТ (Чтение) выставляется 1, если предполагается, что на данном машинном цикле будут читаться данные из памяти

D6 – ВВД (Ввод). выставляется 1, если предполагается, что на данном машинном цикле будут вводиться данные, т.е. адресная шина содержит адрес устройства ввода, а шина данных-вводимую информацию.

D5 – Ц1 (М1) (Первый цикл команды). Признак машинного цикла выборки первого байта команды.

D4 – ВЫВ (Вывод). Выставляется 1, если предполагается, что на первом машинном цикле будут выводиться данные.

D3 – ПОС (Подтверждение останова). Признак останова по команде HALT.

D2 – СТК (Операции со стеком). Выставляется 1, если предполагается обращение к стеку (не получила развития).

D1 - ЗП/ВЫВ (Запись или вывод). Совмещается с командой вывода, следовательно, необходимо учитывать разряд D4.

D0 – ППр (Подтверждение прерывания).

Регистр признака.

Выполнение какой-либо операции может ставиться в зависимость от значения результата выполнения предыдущей операции, например, в том случае, когда при

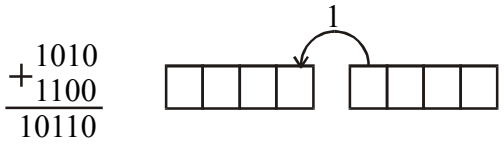
сложении появляется единица переноса. Чтобы можно было обратиться к информации о результатах вычислений, АЛУ соединяют со специальным набором триггеров, которые устанавливаются в 1 или сбрасываются в 0 в зависимости от результата произведенных вычислений. Каждый из триггеров хранит какой-нибудь признак. В совокупности эти триггеры образуют регистр признаков (регистр флагов), приведенный на рисунке ниже.

S	Z	0	AC	0	P	1	CY
D7	D6	D5	D4	D3	D2	D1	D0
Регистр признаков МП							

Признак знака S. Отрицательные числа представляются в МП в дополнительном коде. Следовательно, старший разряд отображает знак числа:1- для отрицательных чисел, 0 - для положительных. Поэтому старший разряд результата выполнения операций в АЛУ запоминается в виде признака знака для дальнейшего использования.

Признак нуля Z. Признак нуля отмечает случай появления в АЛУ результата 0000 0000_2.Признак нуля позволяет организовывать циклы, в течение которых МП-система находится в ожидании какого-то события.

Признак переноса CY. Является одним из важнейших признаков. При сложении двух восьмиразрядных чисел этот признак показывает, нужно ли переносить 1 в младший разряд следующего байта. Он также показывает, нужно ли занимать единицу из старшего байта при вычитании двух восьмиразрядных чисел.



Действие признака переноса

Признак вспомогательного переноса AC. Один из вариантов признака переноса является признак вспомогательного переноса АС, который устанавливается в единицу, если происходит перенос разряда из младшей тетрады байта в старшую. Используется при сложении чисел, записанных в двоично-десятичном коде.

Признак паритета (четности) P. Признак устанавливается равным 1. если в результате операции общее число единиц является четным. Используется, как правило, при контроле четности данных при передаче, так как позволяет выявить возможные при передаче ошибки.

Система команд МП.

Всю систему команд МП можно разбить на следующие группы:

1) команды пересылок;
2) арифметические команды;
3) логические команды и команды сдвига;
4) команды управления;
5) специальные команды.

Рассмотрим каждую группу команд в отдельности.

1) Команды пересылок.

Команды пересылок предписывают микропроцессору передачу данных из одного блока в другой, например, из одного регистра в другой. Эти команды всегда определяют источник и приемник данных, причем в команде сначала следует адрес приемника, а потом адрес источника.

Содержимое источника при этом не изменяется.

Команда MOV

MOV A,B; B→A

MOV A,M; M→A, M адрес ячейки памяти , который предварительно надо поместить в H-пару.

Команда MVI

Команда MVI отличается от команды MOV тем, что в качестве источника данных используется восьмиразрядная константа, которая следует непосредственно за кодом операции. Приемником данных является регистр r или ячейка памяти M.

MV1 C,$data_{8;}$ $data_8 \rightarrow C$

Команда LXI

Команда LXI может быть использована для загрузки регистровых пар (B-пары, D-пары, H-пары) и указателя стека SP шестнадцатиразрядным числом, которое непосредственно следует за кодом операции.

LXI D,$data_{16}$; $data_{16} \rightarrow$(D,E)

Команды LDA, STA

По команде LDA в аккумулятор загружается содержимое ячейки памяти, адрес которой следует за кодом операции. По команде STA содержимое аккумулятора записывается в ячейку памяти, адрес которой следует за кодом операции.

LDA $addr_{16}$; $addr_{16} \rightarrow A$

STA $addr_{16}$; $A \rightarrow addr_{16}$

Команды LDA и STA являются трехбайтными. В первом байте содержится код операции, во втором байте $< B_2 >$ - младший байт адреса ($addr_{МЛ}$), а в третьем байте $< B_3 >$ - старший байт адреса ($addr_{СТ}$).

Команда LDAX.

По команде LDAX в аккумулятор загружается содержимое ячейки памяти, адресуемой регистровой парой (B,C) или (D,E). Команда STAX осуществляет передачу содержимого аккумулятора в ячейку памяти, адресуемой регистровой парой (B,C) или (D,E).

LDAX B**;** косвенная адресация. Предварительно в B-пару загружается адрес ячейки, содержимое которой загружается в аккумулятор.

Операции с портом.

Команды IN и OUT управляют обменом информации между аккумулятором A и портами ввода-вывода. В команде IN (ввод) источником является порт ввода port, а приемником - аккумулятор A. В команде OUT (вывод) источником является аккумулятор A, а приемником порт вывода port. Адрес порта непосредственно следует за кодом операции IN или OUT.

IN $addr_{port}$;

OUT $addr_{port}$;

Работа со стеком.

Команда LXI

LXI SP,$data_{16}$; $data_{16} \rightarrow$ SP. С этого момента известно, где в памяти находится стек.

Команды пересылок PUSH (Поместить в стек) и POP (Вытолкнуть из стека) всегда оперирует с регистровой парой (B,C), (D,E), (H,L) или парой регистров (A,РП), образующей слово состояния программы PSW. Напомним, что указатель стека SP содержит адрес той ячейки в стеке, в которую в последний раз была записана информация, т.е адрес верха стека.

По команде PUSH (Поместить в стек) выполняются действия:

а) содержимое SP сначала уменьшается на 1 (декрементируется);

б) старший байт загружается в стек;

в) содержимое SP вновь декрементируется;

г) младший байт загружается в стек.

PUSH B; Записать в стек B-пары

По команде POP (Вытолкнуть из стека) выполняются обратные действия. Сначала младший байт выталкивается из стека в МП и содержимое SP увеличивается на 1 (инкрементируется). Затем старший байт выталкивается из стека в МП и содержимое SP инкрементируется.

POP B; вытолкнуть из стека в B-пару

2) Арифметические команды.

Под управлением команд этой группы микропроцессор может выполнять в АЛУ арифметические операции. Напомним, что МП I8080 выполняет только сложение и вычитание. Поскольку этот МП является одноадресным, то один из операндов в арифметических операциях всегда помещается в аккумулятор, неявно адресуемый самим кодом операции. Вслед за кодом операции необходимо указывать местонахождение второго операнда. Результат (сумма или разность) помещается в аккумулятор. Операнд, который перед арифметической операцией находился в аккумуляторе, после выполнения операции уничтожается.

Команда сложения ADD, ADC

ADD r**;** A+B→A

ADC r**;** A+B+CY→A

Команда ADC является разновидностью команды ADD. По команде ADC происходит не только сложение двух операндов, но и сложение с признаком переноса CY, оставшимся от предыдущей операции. Результат сохраняется в аккумуляторе.

Команда сложения ADI, ACI

Команда *ADI* осуществляет сложение операнда, который непосредственно следует за кодом операции, с содержимым аккумулятора. По команде *ACI* непосредственный операнд суммируется с содержимым аккумулятора и с признаком переноса CY

ADI $data_8$; A+ $data_8$ →A

ACI $data_8$; A+ $data_8$ +CY→A

Команда вычитания SUB, SBB

Команда *SUB* позволяет микропроцессору непосредственно вычесть содержимое одного из регистров общего назначения или ячейки памяти M из содержимого аккумулятора. Команда *SBB* является разновидностью команды *SUB*. По этой команде осуществляется вычитание с заемом.

SUB r A – r →A

SUB M A – M →A

SBB r A – r - CY →A

SBB M A – M - CY →A

Команда вычитания SUI, SBI

По команде *SUI* из содержимого аккумулятора вычитается операнд, который непосредственно следует за кодом операции. По команде *SBI* из содержимого аккумулятора вычитается и непосредственный операнд, и признак заёма CY.

SUI $data_8$; A+ $data_8$ →A

SBI $data_8$; A+ $data_8$ +CY→A

Внимание!

Все описанные арифметические команды сложения и вычитания изменяют (модифицируют) содержимое всех признаков регистра признаков.

Команды INR, DCR

Команда INR является разновидностью команды ADD. По этой команде МП увеличивает на 1 содержимое одного из регистров РОН, аккумулятора или ячейки памяти

М. Команда DCR является разновидностью команды вычитания SUB. По этой команде МП уменьшает на 1 содержимое одного из регистров РОН, аккумулятора или ячейки памяти М.

INR	r	r + 1 → r
INR	M	M + 1 → M
DCR	r	r – 1 → r
DCR	M	M – 1 → M

Внимание!

Эти команды модифицируют все признаки за исключением признака переноса CY.

Команды INX, DCX

Команда инкремента INX и декремента DCX позволяют соответственно увеличить и уменьшить на 1 содержимое регистровых пар (B-, D- , H- пары) и указателя стека SP.

INX	rp;	rp + 1 → r
DCX	rp;	rp – 1 → r

Внимание!

Эти команды не модифицируют регистр признаков.

Команды DAD, DAA

Команда двойного сложения DAD суммирует содержимое регистровой пары (H,L) и адресуемой регистровой пары rp (rp это B-,D- H- пары, SP)

DAD	rp;	(H,L) + rp → (H,L)

Внимание!

Эта команда модифицирует только признак переноса.

Команда десятичной коррекции аккумулятора DAA осуществляет перевод 8-разрядного двоичного числа в аккумуляторе в две цифры двоично-десятичного кода с правильной установкой признака переноса CY. При этом производятся следующие действия:

1. Если младшая тетрада содержит число, больше 9_{10}, или установлен признак вспомогательного переноса AC=1, то содержимое аккумулятора увеличивается на 6_{10}.
2. Если после этого старшая тетрада аккумулятора содержит число, большее 9, или установлен признак вспомогательного переноса CY=1, то в старшую тетраду прибавляется 6_{10}

Внимание!

При десятичной коррекции модифицируются все признаки регистра признаков.

3) Логические команды и команды сдвига.

С помощью команд логических операций и сдвигов могут быть выполнены следующие действия:

- логические операции И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ (сложение по модулю 2) с использованием двух операндов. Один из операндов всегда размещается в аккумуляторе. Результат выполнения команды фиксируется в аккумуляторе.

Команда ANA

Команда ANA осуществляет логическую операцию И над содержимым аккумулятора и содержимым одного из регистров РОН, аккумулятора и ячейки М

ANA	r;	A & r→ A
ANA	M;	A & M→ A

Команда ORA

Команда ORA реализует логическую ИЛИ над содержимым аккумулятора и содержимым одного из регистров РОН, аккумулятора и ячейки М

ORA	r;	A ∨ r→A

ORA M A ∨ M→A

Команда XRA

Команда XRA осуществляет логическую операцию ИСКЛЮЧАЮЩЕЕ ИЛИ над содержимым аккумулятора и содержимым одного из регистров РОН, аккумулятора и ячейки М

XRA r; A ⊕ r→ A

XRA M; A ⊕ M→ A

Команды ANI, ORI, XRI

Эти команды отличаются от команд ANA, ORA, XRA тем, что второй операнд следует непосредственно за кодом операции.

ANI $data_8$; A & $data_8$→ A

ORI $data_8$; A ∨ $data_8$→ A

XRI $data_8$; A ⊕ $data_8$→ A

Команды CMP, CPI

Команда CMP используется для сравнения двух чисел, одно из которых находится в аккумуляторе, а другое в одном из регистров РОН, аккумуляторе или ячейке М. При сравнении одно из чисел вычитается из другого числа. В соответствии с результатом формируются признаки регистра признаков. Содержимое аккумулятора при этом не изменяется.

CMP r; A – r

CMP M; A – M

CPI $data_8$ A – $data_8$

Разница между командами CMP и SUB в том, что при выполнении команды CMP результат операции не фиксируется в аккумуляторе.

Внимание!

Команды логических операций и сравнения модифицируют регистр признаков.

Команда CMA

Команда CMA используется для инвертирования содержимого аккумулятора.

CMA; $\overline{A}$ → A

Внимание!

Команда CMA не модифицирует регистр признаков.

Команды STC, CMC

STC; 1 → CY

CMC; $\overline{C}\,\overline{Y}$ → CY

Команда STC устанавливает признак переноса CY. Команда CMC инвертирует признак переноса.

Команды сдвига RLC, RRC, RAL, RAR

Если надо произвести операции сдвига над данными, то их необходимо предварительно поместить в аккумулятор. Операндом однобайтных команд сдвига является содержимое аккумулятора, в котором формируется результат. Сдвиги выполняются влево (RLC, RAL) и вправо (RRC, RAR) только на один разряд. Выполнение команд сдвига поясняется на рисунке ниже.

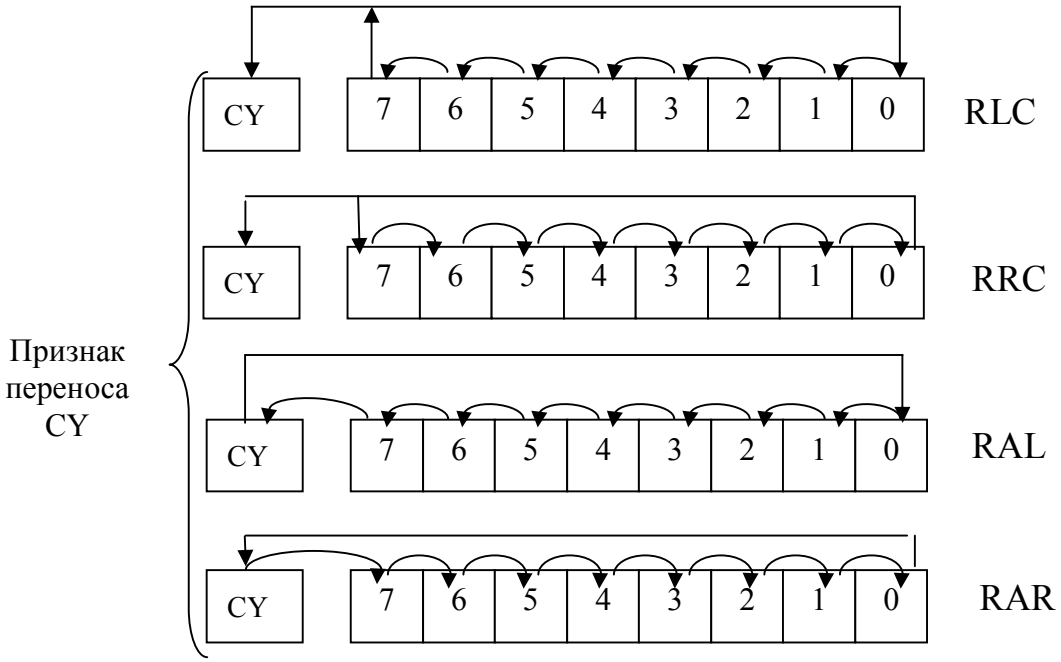


Рис. Команды сдвига

4) Команды управления.

После выборки текущей команды из памяти в счетчике команд СК формируется адрес следующей по порядку команды. В разветвляющихся и циклических программах и при использовании подпрограмм нужно выполнить не следующую по порядку команду, а команду, находящуюся в другой ячейке программной памяти. Для этого достаточно загрузить в СК адрес новой ячейки, называемой адресом перехода. Такая процедура называется передачей управления, а команды, которыми она реализуется, называются *командами передачи управления* .

Команды передачи управления бывают *безусловными* и *условными.*

По командам безусловной передачи управления выполняется переход по программе к адресу, который указывается в команде.

По команде условной передачи управления переход по программе осуществляется только при условии, что содержимое одного из признаков регистра признаков РП соответствует определенному условию. Если это условие не выполняется, то переход не осуществляется и программа выполняется в соответствии с командой, следующей за командой условной передачи управления.

Команды безусловной передачи

JMP addr ;адрес addr известен

Команды условной передачи управления.

По признаку нуля:

JZ addr; Переход, если Z=1

JNZ addr; Переход, если Z=0

По признаку переноса CY:

JC addr; Переход, если CY=1

JNC addr ; Переход, если CY=0

По признаку знака:

JM addr; Переход, если S=1

JP addr; Переход, если S=0

По признаку паритета:

JPE addr; Переход, если P=1

JPO addr; Переход, если P=0

Особенной командой безусловной передачи управления без возврата оказывается команда PCHL; HL → PC, в результате выполнения которой МП продолжает программу с адреса, загружаемого в счетчик команд PC из регистровой пары (H,L).

Команды вызова подпрограмм

Особая команда RST n (n=0..7) вызова предназначена для обработки прерывания и введения контрольных точек (разрывов) при отладке программ. Она называется рестартом (повторным запуском). В коде операции рестарта $11AAA111_{(2)}$ три разряда AAA формируется подсистемой прерываний (аппаратные прерывания) или указываются программистом (программные прерывания).

. Выполнение команды рестарта сводится к двум действиям:

- текущее содержимое счетчика команд PC загружается в стек;
- в счетчик PC подается код 00000000 $00AAA000_{(2)}$.

Следовательно, в зависимости от значения трех разрядов AAA МП переходит к одной из 8 ячеек: 0000_{16}, 0008_{16}, 0010_{16} , 0018_{16} , 0020_{16} , 0028_{16} , 0030_{16} , 0038_{16} .

CALL addr; команда вызывает подпрограмму, находящуюся по адресу addr. По этой команде выполняются действия, необходимые для возврата в основную программу:

1) в счетчике команд СК фиксируется адрес команды в основной программе, которая следует за командой вызова;
2) содержимое СК загружается в стек, а содержимое указателя стека уменьшается на 2;
3) в СК загружается адрес addr, задаваемый командой вызова. После этого начинается выполнение подпрограммы.

5) Команды возврата из подпрограммы.

Команда RET является последней командой подпрограммы. По этой команде выполняется возврат к основной программе, подготовленный командой возврата. По этой однобайтной команде, содержащей только код операции, происходит следующее:

1) счетчик команд получает из стека адрес той команды в основной программе, которая следует за командой вызова;
2) содержимое указателя стека соответственно модифицируется (т.е. увеличивается на два).

```
RET          ;[SP] → PCмл, [SP+1] → PCст
             ;SP + 2 → SP
```

Команды условного возврата имеют модификации по следующим признакам.

Мнемокод	Условия возврата
RC	C=1 по признаку переноса C
RNC	C=0 по признаку переноса C
RZ	Z=1 по признаку нуля
RNZ	Z=0 по признаку нуля
RM	S=1 по признаку знака
RP	S=0 по признаку знака
RPE	P=1 по признаку паритета
RPO	P=0 по признаку паритета

6) Специальные команды

Каждый МП имеет ряд специальных команд. Эти команды не передают и не обрабатывают информацию, но они используются для управления работой микропроцессора.

Команда HLT

По команде HLT останавливается текущая программа до тех пор, пока не появится запрос прерывания или сигнала сброса.

Команды DI, EI

Если МП получает команду DI, то он игнорирует запросы прерывания до тех пор, пока не поступит команда EI.

Команда NOP

Пустая команда NOP не производит никаких действий, кроме инкремента счетчика команд PC для перехода к следующей команде. Команда NOP обычно используется в так называемых циклах задержек, в которых МП генерирует сигнал программируемой длительности.

1.2.3. Способы адресации

Причина организации различных способов адресации в желании, чтобы команда было покороче, чтобы удобно было использовать массивы и т.д.

Адресация бывает

- прямая, когда информация об операнде находится в коде команды
- косвенная, когда информация об операнде находится в в регистре.

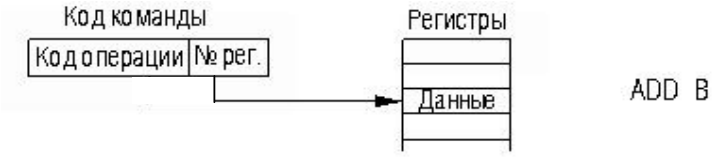
Способы адресации делятся на

- однокомпонентные, когда адрес находится в одном месте
- многокомпонентные, когда адрес находится в нескольких местах.

Информация о способе адресации содержится в коде команды.

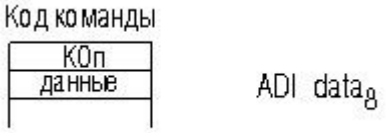
Рассмотрим *однокомпонентные* способы адресации.

1) *Регистровая*. В качестве операнда используется содержимое адресуемого регистра. Это самый простой и наиболее быстрый способ адресации, не требующий обращения к памяти.

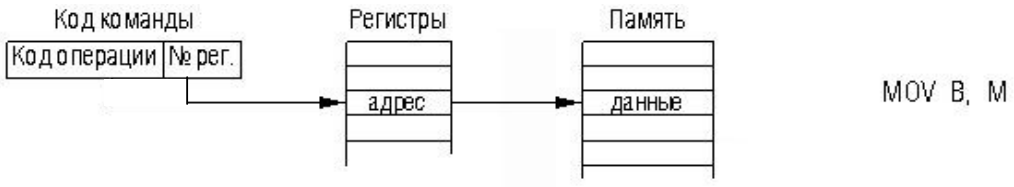


2) *Абсолютная*.

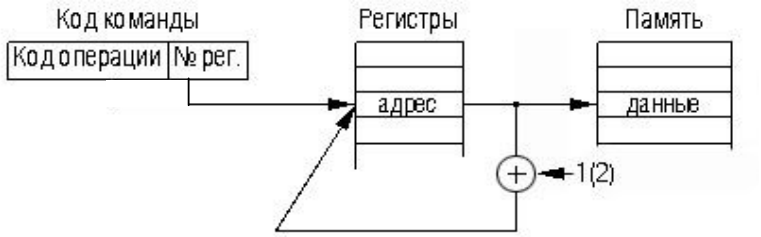
3)*Непосредственная*. При непосредственной адресации операнд следует сразу же за кодом операции.



4) *Регистровая косвенная*. Содержимое адресуемого регистра используется как адрес операнда.



5) *Автоинкрементная*. Содержимое адресуемого регистра является адресом операнда. После получения адреса содержимое регистра автоматически увеличивается на 1 при байтовых операциях и на 2 при операциях со словами. Этот способ адресации удобен при обработке таблиц (последовательно расположенных в памяти данных).

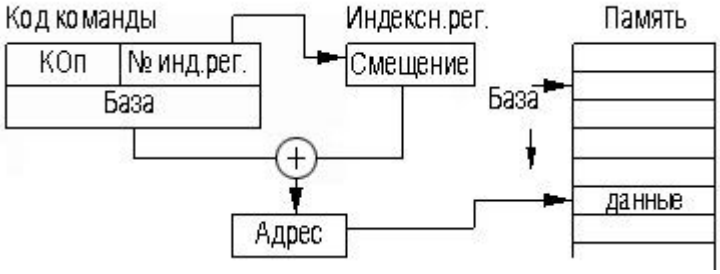


6) *Автодекрементная.* Содержимое регистра задает указатель адреса операнда, затем содержимое регистра увеличивается на 1 (2).

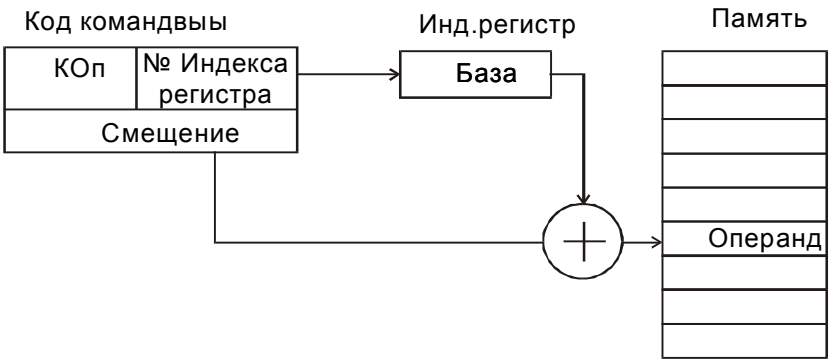
Многокомпонентная адресация.

1) *Страничная.* Вся память разбивается на определенное количество равных по размеру страниц. Например, память объемом 64Кбайт разбивается на 256 страниц. Объем одной страницы 256 байт. Шестнацатиразрядный адрес поделен на два элемента: номер страницы (для примера 8 разрядов), адрес на странице (для примера 8 разрядов) задает местоположение ячейки на странице

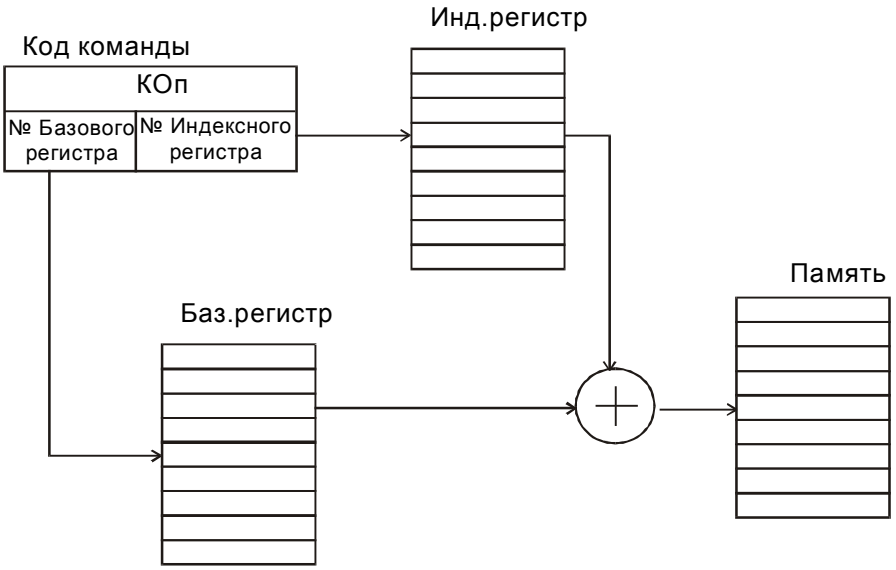
2) С индексированием.



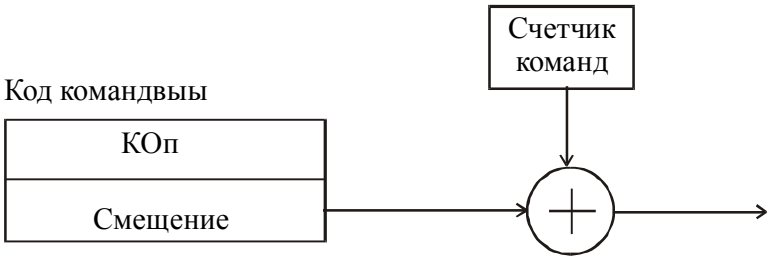
3) По базе.



4) По базе с индексированием.



5) Относительная. Мы адресуемся относительно СчК.



1.2.4. Программирование на языке ассемблера

В этом разделе приводится краткие сведения о языке ассемблера из книги Григорьева В.Л. Программное обеспечение микропроцессорных систем. – М.: Энергоатомиздат, 1983.

Язык ассемблера – это машинно-ориентированный язык (т.е. ориентированный на конкретную ЭВМ и в частности на конкретную МП-систему). Он должен соответствовать специфическим особенностям МП. Следовательно, программу необходимо писать на языке ассемблера для того типа МП, для которого она предназначена. При преобразовании исходной программы (на языке ассемблера) в объектную программу используемая программа трансляции также должна соответствовать этому типу МП. Однако это не означает, что трансляция должна выполняться на таком же МП. Трансляция может быть выполнена даже на большой ЭВМ, имеющей в составе программного обеспечения соответствующую ассемблирующую программу (кросс-ассемблер). В этом

случае кросс-ассемблер составлен на языке большой ЭВМ, но он оперирует с исходной программой, написанной на языке ассемблера данной МП-системы. Объектная программа также выдается на машинном языке МП-системы. Ниже рассматривается язык ассемблера МПК580ВМ80.

Структура команд языка ассемблера.

Ранее уже кратко излагалась структура команд языка ассемблера. Напомним ее. Команда, написанная на языке ассемблера, может быть разделена на следующие четыре части (поля).

МЕТКА ОПЕРАЦИЯ ОПЕРАНД(Ы) КОММЕНТАРИЙ

(код операции)

Метка представляет собой символическое имя (символический адрес) такой ячейки памяти, которую для каких-то целей (например, для последующей адресации к ней) необходимо пометить. Метка всегда начинается с латинской буквы и может содержать не более шести алфавитно-цифровых символов. Она отделяется от следующего поля кода операции двоеточием.

Если ввод программы осуществляется с использованием ассемблера, то действительные адреса ячеек памяти определять не нужно. Ассемблер автоматически вычисляет адреса и ставит им в соответствие последовательность команд. Однако необходимо указывать ассемблеру адрес первого байта команды программы. Поскольку адрес не определяется самостоятельно, то можно записывать в командах символические адреса. Например, команда передачи управления JMP всегда содержит адрес перехода. Этот адрес надо явно указать. Это и обеспечивается присвоением адресу перехода собственного символического имени или метки. В командах передачи управления метка ставится после мнемокода операции перехода. Ассемблер в процессе трансляции заменяет метку соответствующим адресом.

Пример

МЕТКА КОД ОПЕРАЦИИ ОПЕРАНД КОММЕНТАРИЙ

```
CIKL:   MOV   A, B
        ;B → A
        ADD   C
        ;A + C → A
        …     …
        JMP   CIKL
```

Метка CIKL связана с командой MOV A,B. Ассемблер следит за адресацией при трансляции программы. При трансляции команды JMPCIKL ассемблер поставит действительное значение адреса вместо символического адреса CIKL в команду перехода.

Операция. В поле операции для записи кода операции применяется буквенный (символический) код, который обычно является аббревиатурой полного назначения операции.

Примеры

ADD - сложить, SUB - вычесть, JMP - перейти.

Некоторые символические коды операций явно определяют функцию команды (PCHL – передача содержимого пары регистров HL в счетчик команд PC). Обычно длина поля кода операции не превышает четырех позиций, а между ним и соседом справа необходим минимум один пробел. Буквенные обозначения (мнемоники) кодов операций являются ключевыми словами ассемблера. Если содержимое поля кода операций не входит во

множество допустимых обозначений, ассемблер выдает сообщение о недействительной команде.

Операнд. В поле операнда (наиболее сложном поле) определяются данные, являющиеся операндом (или операндами) команды. Содержимое поля операнда интерпретируется в соответствии с функцией команды. Некоторые команды, оперирующие с определенными внутренними регистрами, имеют пустое поле операнда.

Примеры

```
МЕТКА   КОД ОПЕРАЦИИ   ОПЕРАНД          КОММЕНТАРИЙ
                CMA
;Инвертирование аккумулятора
                RAL
;Сдвиг влево через перенос
                XTHL
;Обмен (H,L) и верхушки стека
                NOP
;Пустая команда
```

В качестве операндов могут фигурировать:

- адреса памяти;
- внутренние регистры МП;
- адреса портов ввода и вывода;
- числовые и символьные константы.

Рассмотрим способы определения перечисленных типов операндов.

Шестнадцатеричные данные. Содержащиеся в поле операнда 16-ричное число должно начинаться с цифр 0 - 9 и завершаться буквой H (Hex). Число, начинающееся с букв A – F, дополняется слева незначащим нулем.

Примеры

```
STORE:  STA     8000H       ; Запоминание в ячейке 800016
        MVI     C, 0AAH     ; Загрузка в регистр C кода 10101010 (A  A)
        ANI     10H         ; Выделение четвертого бита
COMP:   CPI     290H        ; Недопустимый операнд
```

Десятичные данные. Десятичное число в поле операнда заканчивается необязательной буквой D.

```
        MOV     B,15        ; Загрузка в регистр B кода 00001111
        ANI     64          ; Выделение 6-го бита
        IN      32          ; Ввод из порта с адресом 00100000
```

Восьмеричные данные. Восьмеричное число в поле операнда заканчивается буквами O или Q (чаще).

```
        ORI     200Q        ; Установка старшего бита
```

Двоичные данные. Двоичное число в поле операнда заканчивается буквой B.

```
        OUT     1111B       ; Вывод в порт с адресом 15
        XRI     10000000B; Инверсия знакового бита
        ANI     11011111B; Сброс пятого бита
```

Символьные константы. В поле операнда допускается использовать символы алфавита, заключая их в апострофы. Ассемблер представляет вместо такого операнда соответствующий двоичный код символа

```
        MVI     D, 'T'      ; Загрузка кода буквы T в регистр D
```

Идентификаторы внутренних регистров. В язык ассемблера МП К580 встроены идентификаторы внутренних регистров B, C, D, E, H, L, M, A с соответствующими двоичными значениями от 000 до 111.

```
        MOV     A, E        ; Передача из регистра E в аккумулятор
        ADD     L           ; Прибавление содержимого регистра L
        SUB     M           ; Вычитание содержимого ячейки, адресуемой
                            ; регистрами (H, L)
```

Вместо идентификаторов внутренних регистров допускается применять их адреса в любой системе счисления.

```
        MOV     A, B        ; Переслать из регистра B в регистр A
        MOV     7, 0        ; Переслать из регистра B в регистр A
```

В командах, оперирующих 16-битными значениями, применяются идентификаторы внутренних 16-битных регистров.

```
        LXI     H, 0FF00H; Инициализация регистров (H,L)
        INX     SP          ; Инкремент указателя стека
        PUSH    PSW         ;Загрузка в стек содержимого A- и РП- регистров
```

В командах передачи управления можно указать метки, введенные программистом в поле метки других команд. Метки в поле операнда заменяют абсолютные значения адресов перехода.

```
        JMP     DONE        ; Переход к метке DONE
        CALL    SWAP        ; Вызов подпрограммы SWAP
```

Текущее содержимое программного счетчика. В командах передачи управления допускается относительная адресация. Адрес перехода определяется суммой (разностью) текущего содержимого программного счетчика (символом ¤) и указываемого в операторе смещения. Смещение может быть представлено в любой из ранее рассмотренных форм определения числовых значений.

```
GOTO:   JMP     ¤ + 20H     ; Переход по адресу GOTO + 20H
MORE:   JNZ     ¤ - 80      ; Адрес перехода MORE-80 («назад»)
```

Выражения. В поле операнда могут помещаться выражения. Они содержат в качестве аргументов все рассмотренные выше типы данных, которые связываются арифметическими и логическими операторами.

```
        MVI     B, 30 + 40H/2; Загрузка числа 62
        SUI     34, MOD3    ; Декремент аккумулятора
```

Выражения в операнде применяются редко.

Комментарии. Программист может использовать это поле для пояснения команды или фрагмента программы. Ассемблер игнорирует поле комментария в процессе трансляции. Поле комментария начинается с некоторого разделителя (точка с запятой).

Директивы ассемблера

Директивы (псевдокоды) ассемблера являются указаниями ассемблеру в выполнении определенных действий в процессе ассемблирования, размещают в памяти информацию, присваивают численные значения символическим наименованиям, резервируют память и т.д.

Директивы подчиняются стандартному формату операторов ассемблера, но содержимое их полей имеет некоторые особенности. Например, в поле метки директив EQU и SET должно находиться символическое наименование, которое не имеет заключительного двоеточия. В остальных директивах в поле метки может находиться необязательная метка, аналогичная меткам машинных команд. Операнды директив необязательны.

Директива ORG (начало) имеет формат

[МЕТКА:] ORG < выражение >

[ ] – обозначает необязательный элемент.

Значением выражения директивы ORG является 16-битный адрес, определяющий ячейку памяти, в которую будет загружаться первый бит следующей команды. До новой директивы ORG команды и данные размещаются в смежных ячейках памяти. Если в самом начале программы директива ORG отсутствует, то по умолчанию подразумевается наличие директивы ORG с нулевым операндом. При необходимости в программе может быть несколько директив ORG.

```
                ORG         100H   ; Задается абсолютный адрес 0100H
NEW:            ORG         200H   ; Задается абсолютный адрес 0200H
```

Директива ORG может выполнить функцию резервирования памяти.

```
                MOV         A,M
                RAL
                JMP         LOWER
                ORG         ¤ +20
LOWER:  ORA             A
```

Директива END

[МЕТКА:] END

Эта директива информирует ассемблер о достижении физического конца входной программы.

Директива EQU (принять, присвоить - директива прямого присвоения)

МЕТКА	КОД	ОПЕРАНД
< имя >	EQU	< выражение >

При выполнении директивы EQU ассемблер присваивает значение выражения символическому наименованию, находящемуся в поле метки. Когда наименование встречается в поле операнда, ассемблер подставляет вместо него присвоенное значение.

```
MASK            EQU         0FH    ; Значение MASK равно 15
        …                   …
                ANI         MASK
        …                   …
                MVI         MASK
```

В командах ANI, MVI вместо MASK будет фигурировать код 00001111. При программировании рекомендуется сгруппировать все директивы EQU в начале или в конце программы.

Директива SET (установить)

Имеет такой же формат, что и директива EQU. В отличие от директивы EQU, значение символического наименования допускается изменять с помощью новой директивы SET.

```
NAME    SET     15      ; Значение NAME равно 15
        …
        MVI     B, NAME
NAME    SET     1FH     ; Значение NAME равно 31
        …
        ADI     NAME; Прибавление 31
        …
        CPI     NAME; Сравнение с 31
```

Директивы IF ENDIF

Директивы условного ассемблирования IF (если) и ENDIF (конец если) применяются следующим образом:

```
МЕТКА       КОД     ОПЕРАНД         КОММЕНТАРИЙ
[МЕТКА:]    IF      < выражение >
            …
            ОПЕРАТОРЫ
            …
[МЕТКА:]    ENDIF
```

В процессе ассемблирования вычисляется значение выражения из поля операнда директивы IF. Если *оно равно нулю*, операторы между директивами IF и ENDIF *игнорируются* и не включаются в объектную программу. Если значение выражения *отлично от нуля*, операторы программы *ассемблируются*, как будто директив IF и ENDIF нет.

Директива DB (определить байт)

Директива относится к группе директив определения, которые применяются для инициализации данных и резервирования памяти. Формат директивы DB

```
[МЕТКА:]    DB      < список >
```

Операнд директивы DB может быть последовательностью выражений, разделенных запятыми и имеющих 8-битные значения, либо цепочкой символов, заключенных в апострофы. При выполнении директивы DB значения выражений или коды символов запоминаются в смежных ячейках (байтах) памяти, начинающихся после предыдущей команды.

```
ARRAY:  DB      3,7,15,31   ; Запоминаются четыре значения
        DB      ‘HELLO’     ; Запоминаются пять символов
COMP:   DB      -63         ; Дополнительный код -63
```

Директива DW (определить слово – 2 байта)

Директива аналогична директиве DB, только здесь списком является последовательность выражений, имеющих 16-битные значения. Запоминается сначала младший байт по текущему адресу, а старший байт запоминается по адресу на 1 больше предыдущего.

```
ADDR:   DW      0FF00H  ; (ADDR)=00H
                        (ADDR+1)=FFH
```

Директива DS (определить память)

```
[МЕТКА:]    DS      < выражение >
```

Вычисленное значение выражения из поля операнда определяет число ячеек (байт) памяти, резервируемых для запоминания данных. Никакие значения в этих ячейках не запоминаются. Адрес следующего оператора равен сумме адреса оператора, находящегося перед директивой DS и значения выражения директивы DS.

```
ARRAY:      DS          32      ; Резервируется 32 ячейки
```

Для улучшения внешнего вида и удобства документирования листинга в ассемблерах могут применяться *директивы*:

SPS – (пропуск строки), которая означает, что при печати листинга необходимо пропустить одну строку;

PAGE – (страница), которая при печати вызывает переход на следующую страницу;

TITLE – (заголовок), которая вызывает переход на следующую страницу и печать сверху страницы заголовка программы, введенного программистом.

Иногда возникает необходимость использовать определенную последовательность команд во многих частях программы. Для таких ситуаций макроассемблер позволяет программисту указывать в исходной программе вместо повторяющейся последовательности команд только макрокоманду (макрос), которая эквивалентна выделенной последовательности команд. Макрокоманда задается программистом в виде макроопределении, например, в начале программы.

Пример макроопределения.

```
SPDE    MACRO
        XCHG        ; (HL) ←→ (DE)
        SPHL        ; (HL) ←→ (SP)
        XCHG        ; (HL) ←→ (DE)
SPDE    ENDM
```

Во время трансляции макроассемблер заменяет в исходной программе строку SPDE (макровызов) заранее определенной в макроопределении последовательностью из трех строк (т.е. макрорасширением).

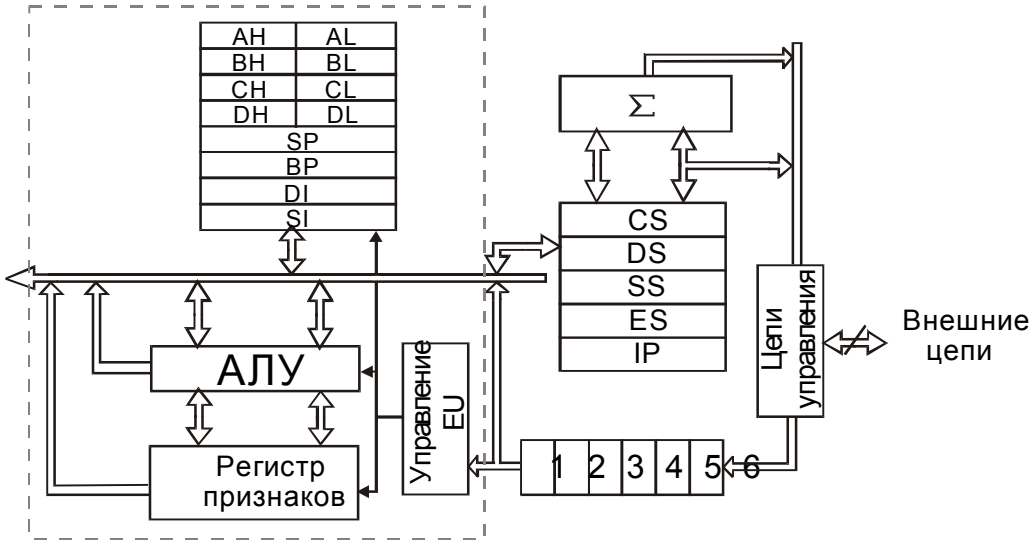
Макрокоманды могут быть с параметрами.

```
EXT   MACRO  VAR
        MVI  A,VAR          ; VAR→A
        MOV M,A
        XRA A
        INX H
        MOV M,A
EXT   ENDM
```

Процедура и макрокоманды - это две альтернативы. Для экономии памяти лучше использовать процедуры, а для повышения быстродействия - макрокоманды.

1.2.5. Структурная схема и система команд микропроцессора I8086

Упрощенная структура МП представлена на рисунке ниже.



Структура МП К586ВМ80

Регистры общего назначения.

Помимо ячеек оперативной памяти для хранения данных (кратковременного) можно использовать и регистры, входящие в состав процессора и доступные из машинной программы. Доступ к регистрам осуществляется значительно быстрее, чем к ячейкам памяти, поэтому использование регистров заметно уменьшает время выполнения программ. Все регистры имеют размер слова (16 битов), за каждым из них закреплено определенное имя (AX, SP и т.п.). По назначению и способу использования регистры можно разбить на следующие группы:

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

Расшифровка этих названий следующая

A - accumulator, аккумулятор;
B - base, база;
C - counter, счетчик;
D - data, данные;
BP - base pointer, указатель базы;
SI - source index, индекс источника;
DI - destination index, индекс приемника;
SP - stack pointer, указатель стека;
CS - code segment, сегмент команд;
DS - data segment, сегмент данных;
SS -stack segment, сегмент стека;
ES - extra segment, дополнительный сегмент;
IP - instruction pointer, счетчик команд.

Регистры общего назначения можно использовать во всех арифметических и логических командах. В то же время каждый их них имеет определенную специализацию (некоторые команды "работают" только с определенными регистрами). Например, команды умножения и деления требуют, чтобы один из операндов находился в регистре AX или в регистрах AX и DX (в зависимости от размера операнда), а команды управления циклом используют регистр CX в качестве счетчика цикла. Регистры BX и BP очень часто используются как базовые регистры, а SI и DI - как индексные. Регистр SP обычно указывает на вершину стека, аппаратно поддерживаемого в ПК.

Регистры AX, BX, CX и DX конструктивно устроены так, что возможен независимый доступ к их старшей и младшей половинам; Каждый из этих регистров состоит из двух восьмиразрядных регистров, обозначаемых AH, AL, BH и т.д. (H - high, старший; L - low, младший):

Таким образом, с каждым из этих регистров можно работать как с единым целым, а можно работать и с его "половинками". Например, можно записать слово в AX, а затем считать только часть слова из регистра AH или заменить только часть в регистре AL и т.д. Такое устройство регистров позволяет использовать их для работы и с числами, и с символами.

Сегментные регистры CS, DS, SS и ES не могут быть операндами никаких команд, кроме команд пересылки и стековых команд. Эти регистры используются только для сегментирования адресов.

Счетчик команд IP всегда содержит адрес (смещение от начала программы) той команды, которая должна быть выполнена следующей (начало программы хранится в регистре CS). Содержимое регистра IP можно изменить только командами перехода.

В этом МП этапы выполнения команд (извлечение кода и операндов команды, выполнение команды) реализуется в двух различных блоках:

- блоке выполнения команд EU (выполнение команды)
- блоке сопряжения с магистралями BIU (извлечение кода и операндов, запись результата в память).

Блоки могут работать независимо друг от друга, поэтому процессы преобразования и передачи информации могут идти в них параллельно. Благодаря этому за одно и тоже время количество выполненных команд возрастает по сравнению с процессором I8080.

*Блок выполнения команд EU***.** Блок не имеет связи с внешними магистралями МП. На АЛУ поступают коды команд из BIU. Если в результате дешифрации кода команд в АЛУ необходимо получение операндов по внешним магистралям МП, то блок EU запрашивает блок BIU на получение и размещение необходимых данных в BIU.

Блок сопряжения с магистралями BIU. Производит пересылки данных и кодов для блока EU. В то время как блок EU занят выполнением команды, блок BIU получает следующие в программе коды команд из памяти и сохраняет их в конвейере команд. Он выдает в EU извлеченные ранее из памяти коды команд по мере необходимости без загрузки внешних магистралей МП. Блок BIU организует получение нового кода команды , как только 2 байта в конвейере команд будут использованы.

Особенности адресации.

В этом процессоре реализовано адресное пространство размером в 1 МБ, поэтому физический адрес представляется 20-разрядным числом. Любая ячейка памяти имеет два типа адресов

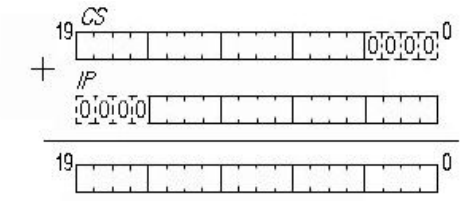
- физический
- логический

Весь обмен информацией между МП и памятью осуществляется с помощью физических адресов. В программе реализованы логические адреса. Логический адрес состоит из двух частей:

CS:	IP
база, в которой находится адрес сегмента (базы)	смещение относительно базы
16-тиразрядный	16-тиразрядный

20-тиразрядный

20-тиразрядный адрес осуществляется схемотехнически.



К 16-ти разрядам CS добавляются 4 младших нулевых разряда.

А в счетчик инструкций эти разряды добавляются слева. И мы получаем 20-тиразрядный физический адрес.

Аналогично DS:BX, DS:SI, ES:DI, SS:SP.

Машинные программы обычно строятся так. Все команды программы размещаются в одном сегменте памяти, начало которого заносится в регистр CS. Все данные размещаются в другом сегменте, начало которого заносится в регистр DS; если нужен стек, то под него отводится третий сегмент памяти, начало которого записывается в регистр SS. После этого практически во всех командах можно указывать не полные адресные пары, а лишь смещения, так как значения сегментных регистров в этих парах будут восстанавливаться автоматически. Ссылки на сегмент команд могут быть только в командах перехода, а ссылки практически во всех других командах (кроме строковых и стековых) - это ссылки на сегмент данных. Например, в команде пересылки

MOV AX,X

имя X воспринимается как ссылка на данное, а потому автоматически восстанавливается до адресной пары DS:X. В команде же безусловного перехода по адресу, находящемуся в регистре BX,

JMP BX

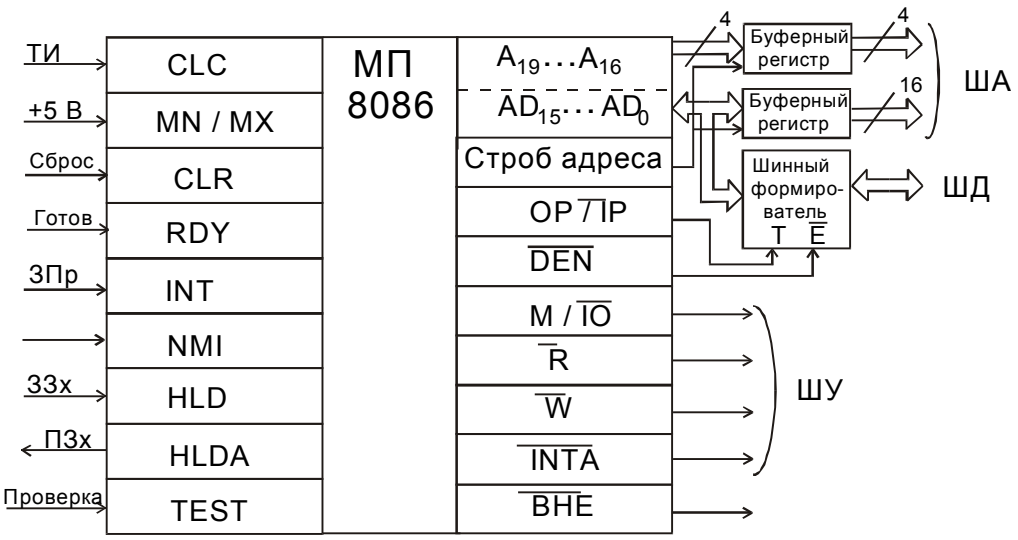
абсолютный адрес перехода определяется парой CS:[BX].

Итак, если в ссылке на какую-то ячейку памяти не указан явно сегментный регистр, то этот регистр берется по умолчанию. Явно же сегментные регистры надо указывать, только если по каким-то причинам регистр по умолчанию не подходит. Если, например, в команде пересылки надо сослаться на стек (скажем, надо записать в регистр AH байт стека, помеченный именем X), тогда мы обязаны явно указать иной регистр - в нашем случае регистр SS, т.к. именно он указывает на стек:

MOV AH,SS:X

Перечень сигналов МП.

Перечень сигналов процессора I8086 (К1810ВМ86) приведен на рисунке ниже



Перечень сигналов МП К1810ВМ86

Шина адреса (20-тиразрядная), поэтому она делиться на две части.16 младших адресов совмещают адресные цепи и цепи данных. Это значит что в какие-то моменты по этим линиям выставляются адреса, а какие-то - данные, т.е. сигналы разделены во времени. Это сделано для экономии выводов. Когда идет адрес, то он сопровождается сигналом о том, что идет адрес. Этот сигнал называется стробом адреса. По этому стробу адрес защелкивается в регистр адреса.

Есть два восьмиразрядных шинных формирователя, которые формируют 16-тиразрядную ШД. Существует специальный сигнал (OP/#IP), задающий направление обмена данными через шинный формирователь: от МП (высокий уровень сигнала) или к нему (низкий уровень).

Сигнал #DEN (Data Enable - разрешение данных) разрешает передаче данных через шинный формирователь. МП может сам выдать сигнал.

Формирование ШУ.

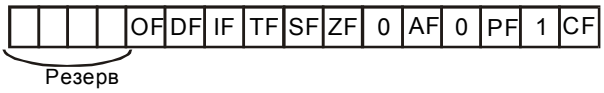
- Сигнал M/#IO, говорящий о работе либо с памятью (высокий уровень), либо с устройством ввода-вывода (низкий уровень);
- сигналы записи (#W) (и чтения (#R)) данных в память или устройство ввода-вывода;
- сигнал запроса прерывания INT;
- сигнал подтверждения прерывания - INTA
- сигнал немаскируемого запроса прерываний NMI. При подаче сигнала на этот вход всегда разрешаются прерывания;
- сигнал разрешения старшего байта #BHE. Позволяет организовать различные типы адресации: пословная (указание на слово) и побайтная (указание на четный или нечетный байт);
- ТИ тактовые импульсы, с помощью которых можно управлять работой МП;
- сигнал, задающий режим работы МП - MN /#MX. Если задается высокий уровень (MN), то в системе имеется лишь один МП. Если задается низкий уровень (#MX), то в системе может быть несколько МП;
- сигнал сброса - CLR. Устанавливает МП в начальное состояние - во всех регистрах нули, кроме CS. Туда заносится число 0FFFFh. Это связано с тем, что программа загрузки МП начинается со старших адресов;
- сигнал готовности RDY. Без этого сигнала процессор приостанавливает свою работу;
- запрос прямого доступа к памяти HLD. С помощью этого сигнала можно организовать прямой доступ к памяти, аналогичный реализованному в процессоре I8080$

- проверка (TEST). Существует специальная команда WAIT и если она встречается в программе, то МП переходит в состояние ожидания внешнего активного сигнала TEST.

Содержимое регистра признаков.

В ПК имеется особый регистр флагов (или признаков). Флаг - это бит, принимающий значение 1 ("флаг установлен"), если выполнено некоторое условие, и значение 0 ("флаг сброшен") в противном случае. В ПК используется 9 флагов, каждому из них присвоено определенное имя (ZF, CF и т.д.). Все они собраны в регистре флагов FLAGS.

Некоторые флаги принято называть *флагами условий*; они автоматически меняются при выполнении команд и фиксируют те или иные свойства их результата (например, равен ли он нулю). Другие флаги называются *флагами состояний*; они меняются из программы и оказывают влияние на дальнейшее поведение процессора (например, блокируют прерывания).



Регистр признаков МП.

Флаги условий.

CF (carry flag) - флаг переноса. Принимает значение 1, если при сложении целых чисел без знака появилась единица переноса, не "влезающая" в разрядную сетку, или если при вычитании чисел без знака первое из них было меньше второго. В командах сдвига в CF заносится бит, вышедший за разрядную сетку.

OF (overflow flag) - флаг переполнения. Устанавливается в 1, если при сложении или вычитании целых чисел со знаком получился результат, выходящий за пределы заданного диапазона .

ZF (zero flag) - флаг нуля. Устанавливается в 1, если результат команды оказался равным 0.

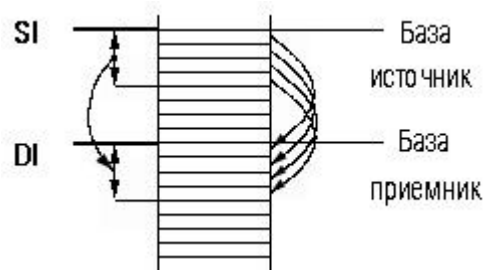
SF (sign flag) - флаг знака. Устанавливается в 1, если в операции над знаковыми числами получился отрицательный результат.

PF (parity flag) - флаг четности. Равен 1, если результат очередной команды содержит четное количество двоичных единиц. Учитывается обычно только при операциях ввода-вывода.

AF (auxiliary carry flag) - флаг дополнительного переноса. Фиксирует особенности выполнения операций над двоично-десятичными числами.

Флаги состояний.

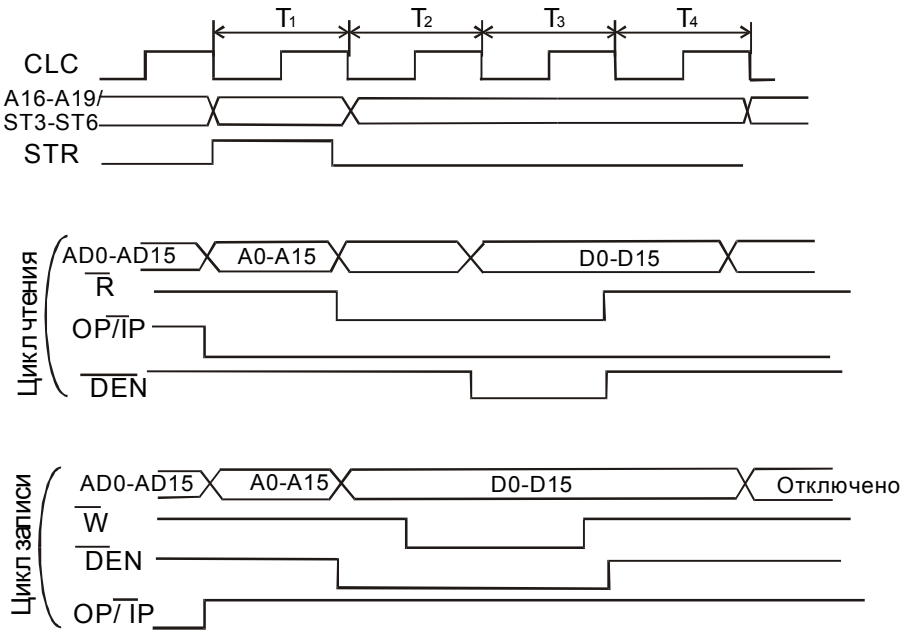
DF (direction flag) - флаг направления. Устанавливает направление изменения адреса данных в строковых командах: автоматическое увеличение (при DF=0) или уменьшение на 1 (2) адреса элемента массива (см. рис. ниже).



IF (interrupt flag) - флаг прерываний. При IF=0 процессор перестает реагировать на поступающие к нему прерывания по входу INT, при IF=1 блокировка прерываний снимается.

TF (trap flag) - флаг трассировки. При TF=1 после выполнения каждой команды процессор делает прерывание (с номером 1), чем можно воспользоваться при отладке программы для ее трассировки (пошаговый режим).

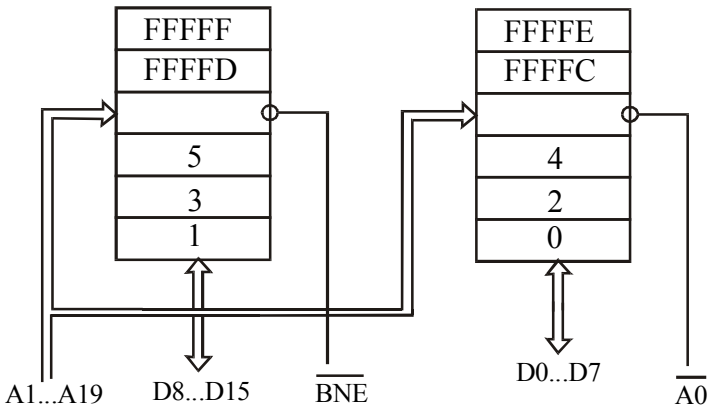
Временные диаграммы работы МП.



Временные диаграммы работы МП М1810ВМ86

Адресация к памяти.

Память формируется в виде 2-х блоков: четных и нечетных байтов, причем один блок открывается сигналом #BHE, а второй $\overline{A_0}$. $\overline{A_0}$ различает четный / нечетный или младший / старший байт.



Обращение МП К1810ВМ86 к памяти МП-системы

BHE- пассивен - старший байт не разрешен. BHE-активен - старший байт разрешен.

Пример. Поступают следующие сигналы:

$\overline{BHE}$ - пассивный $\overline{A_0}$ - активный Открывается блок четных

$\overline{BHE}$ - активный $\overline{A_0}$ - пассивный Открывается блок

$\overline{BHE}$ - активный A_0 -активный

адресов. Выбран младший(четный) байт	нечетных адресов. Выбран старший (нечетный) байт	Оба блока открыты. Выбрано слово.

Формат команды МП 8086.

Формат команды МП К1810ВМ586

Как уже говорилось выше, конвейер команд этого МП рассчитан на 6 байт, что связано с длиной команды (см. рис. ниже).

Таблица. Mode

Mode	Режим работы	Смещение
00	работа с памятью (операнд находится в памяти)	нулевое
01		8-миразрядное
10		16-тиразрядное
11	работа с регистром reg / mem	

Таблица.Reg.

Reg	W=0 (байт)	W=1(слово)	Reg	W=0 (байт)	W=1 (слово)
000	AL	AX	100	AH	SP
001	CL	CX	101	CH	BP
010	DL	DX	110	DH	SI
011	BL	BX	111	BH	DI

Таблица Reg / mem.

Reg / mem	Исполнительный адрес (смещение относительно базы сегмента)		
	mode = 00	mode = 01	mode = 10
000	[BX+SI]	[BX+SI+$data_8$]	[BX+SI+$data_{16}$]
001	[BX+DI]	[BX+DI+$data_8$]	[BX+DI+$data_{16}$]
010	[BP+SI]	[BP+SI+ $data_8$]	[BP+SI+ $data_{16}$]
011	[BP+DI]	[BP+DI+ $data_8$]	[BP+DI+ $data_{16}$]
100	[SI]	[SI+ $data_8$]	[SI+ $data_{16}$]
101	[DI]	[DI+ $data_8$]	[DI+ $data_{16}$]
110	Непосредственная адресация	[BP+$data_8$]	[BP+$data_{16}$]
111	[BX]	[BX+$data_8$]	[BX+$data_{16}$]

Организация прерываний

В МП 8086 каждому прерыванию поставлен в соответствие код (от 0 до 255), который идентифицирует тип прерывания.

Прерывания могут вызываться:

- внешними устройствами (внешние прерывания);
- командами программных прерываний;
- автоматически самим МП (внутренние прерывания).

Возможные источники показаны на рисунке ниже

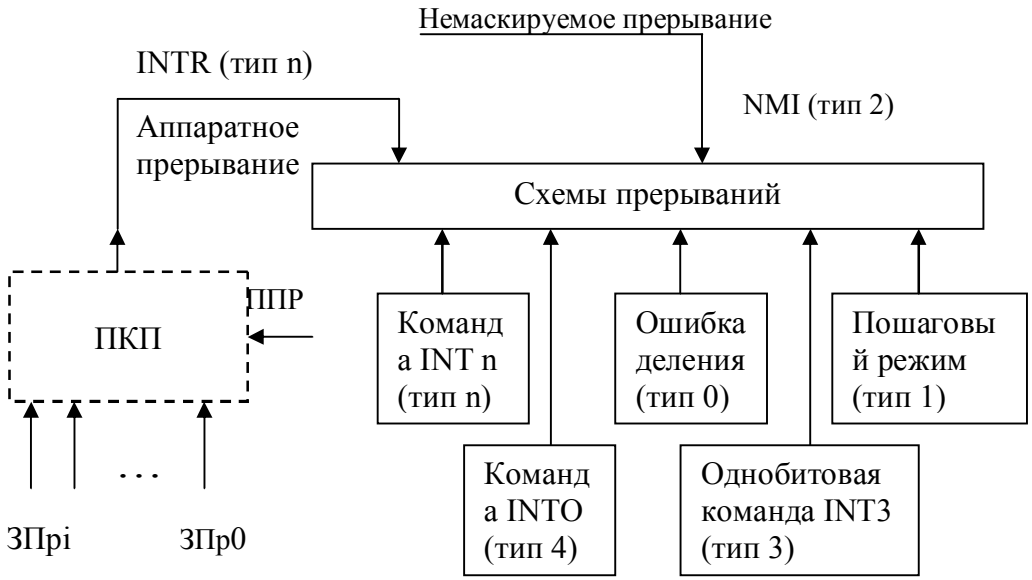


Рис. Источники прерываний

Внешние прерывания

Запросы на внешние прерывания поступают в МП по двум входам INTR (маскируемые) и NMI (немаскируемые).

Запросы на маскируемые прерывания от внешних устройств обычно поступают на входы программируемого контроллера прерываний, который формирует сигнал, подаваемый на вход INTR микропроцессора. В ответ на этот сигнал МП (если разрешены внешние прерывания, т.е. если флаг IF=1) выдает последовательно два сигнала INTA1 и INTA2, по которым контроллер прерываний выдает команду INT n.

Запросы на немаскируемое прерывание поступают по входу NMI и обычно используются для прерывания работы МП при «катастрофических» событиях, требующих немедленной реакции (например, аварийное отключение питания, обнаружение ошибки памяти и т.д.). Немаскированному прерыванию присваивается фиксированный код типа 2, который формируется автоматически внутри МП.

Внутренние прерывания

Характеризуются типом прерывания, который либо предопределен, либо содержится в коде команды. Внутренние прерывания не могут быть запрещены (кроме пошагового прерывания).

Прерывание по ошибке деления (тип 0) генерируется микропроцессором после выполнения команд деления DIV и IDIV в случае деления на нуль.

Прерывание по переполнению (тип 4) генерируется по однобайтовой команде INTO, если установлен флаг OF.

Пошаговое прерывание (тип 1) вырабатывается автоматически при TF=1 после выполнения каждой команды.

Процедура обслуживания прерываний

Связь между кодом, определяющим тип прерывания, и подпрограммой (процедурой) обслуживания прерывания устанавливается с помощью таблицы указателей векторов прерываний.

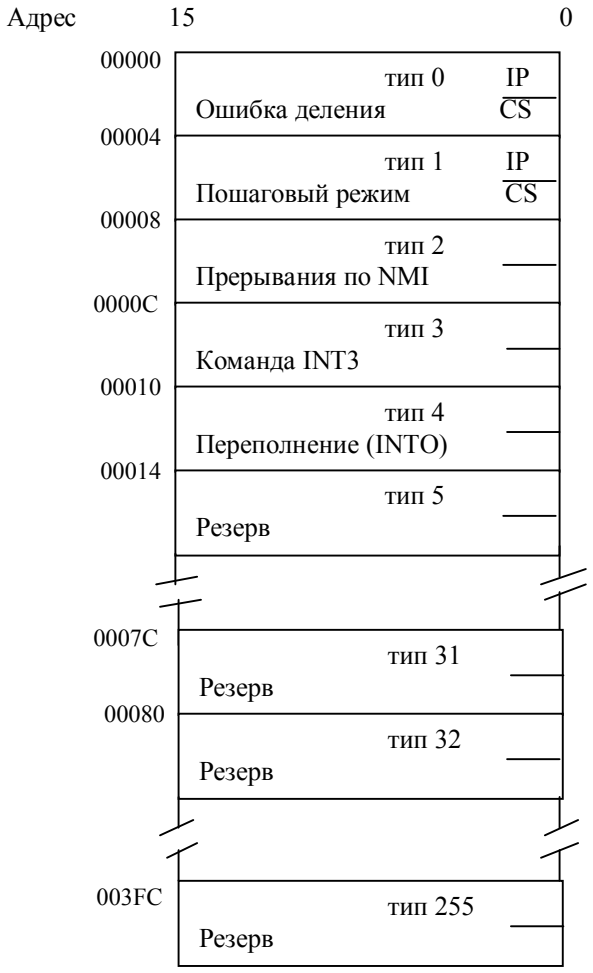


Рис. Таблица указателей векторов прерываний

Полная таблица занимает 1Кбайт памяти и содержит 256 элементов, расположенных по адресам 0 – 3FFH. Каждый элемент n в таблице содержит два слова, определяющие начальный логический адрес подпрограммы: слово с большим адресом содержит базовый адрес сегмента, слово с меньшим адресом – смещение от базы сегмента. При переходе на подпрограмму смещение загружается в регистр IP, а адрес сегмента – в регистр CS.

Так как размер каждого элемента таблицы векторов прерываний составляет 4 байта, то МП вычисляет адрес (смещение) требуемого элемента путем умножения типа прерывания n на 4. После установления нового содержимого регистров IP и CS МП выбирает код операции первой команды подпрограммы и затем выполняет обычные действия по заполнению очереди команд, выполнению команд и обмену данными.

Когда осуществляется переход на подпрограмму обслуживания прерывания, содержимое регистров флагов FLAGS вместе с содержимым регистров IP и CS запоминается в стеке. Флаг IF (и TF) сбрасывается, т.е. автоматически запрещаются внешние прерывания по входу INT. Затем подпрограмма может разрешить внешние прерывания командой STI (установит IF в 1). Кроме того, подпрограмма может быть прервана запросом на входе NMI (немаскируемые прерывания) и внутренними прерываниями.

В конце подпрограммы восстанавливают содержимое регистров МП, которые были включены в стек в начале подпрограммы с целью сохранения данных, относящихся к прерванной программе.

Подпрограмма обработки прерываний должна заканчиваться командой возврата из прерывания IRET. Перед выполнением команды IRET стек должен быть в том состоянии, в котором он был сразу после вызова подпрограммы. Тогда эта команда извлекает три верхних слова из стека в регистры IP, CS и FLAGS, что обеспечивает возврат к команде, которая бы выполнялась бы в случае отсутствия прерывания.

Система команд МП 8086

Команды МП 8086 можно сгруппировать так:

1) Пересылок;
2) Арифметические;
3) Логические и сдвига;
4) Строковые;
5) Передачи управления;
6) Специальные.

1 Команды пересылок

Возможна пересылка между:

1) Регистр — регистр
2) Регистр — память
3) Память — регистр
4) Data — память
5) Data — регистр

Основные команды пересылок

MOV пр, ист; где
пр - приемник,
ист - источник

Пример:

```
MOV     AX,  BX   ;     BX→AX
MOV     [BX]      ;     mem[BX]→AX
```

Замечание

Есть исключения при пересылке. Нельзя данные непосредственно загрузить в сегментный регистр.

Пересылки при работе со стеком

1) PUSH ист ; ист → стек
2) PUSH sreg ; sreg → стек
3) PUSHF ; FLAGS → стек

1) POP пр ; стек → пр
2) POP sreg ; стек → sreg
3) POPF ; стек → FLAGS

Специфические команды пересылок

1) Команда взаимообмена данными

   XGHG пр, reg ; пр ↔ reg

2) Команды пересылки между регистром AH и регистром признаков

```
LAHF                    ;       FLAGS→ AH
SAHF                            AH → FLAGS
```

3) Команды загрузки адресов

```
LEA         reg, ист    ;       командный адрес операнда ист → reg
LDS         reg, ист    ;       загрузка адреса из ист → (DS: reg)
LES     reg, ист        ;       загрузка адреса из ист → (ES: reg)
```

4) Пересылка байта (из таблицы) в AL

```
XLAT                    ;       mem[BX+ AL]→ AL
```

5) Команды ввода-вывода

1) Байтовый адрес порта (0…255)

```
IN      A, port         ;       port → A,
OUT     port, A         ;       A → port,
```

где A=AL – для байта, A=AX – для слова

2) Двухбайтовый адрес порта (0…64К)

Адрес порта необходимо предварительно записать в регистр DX

```
IN      A, DX           ;       [DX] → A
OUT     DX, A           ;       A → [DX]
```

Примеры

1)
```
MOV  AX, table      ; Пересылка из ячейки памяти table в регистр AX
MOV  table, AX      ; Пересылка из регистра AX в ячейку памяти table
MOV  ES:[BX], AX    ; Пересылка из AX в ячейку памяти ES:[BX]
MOV  DS, AX         ; Пересылка из AX в DS
MOV  BL, AL         ; Пересылка из AL в BL
MOV  CL, -30        ; Пересылка константы -30 → CL
MOV  dest, 25h      ; Пересылка константы 25h → ячейку памяти dest
```

2)
```
PUSH SI             ; Сохранение в стеке содержимого регистра SI
PUSH DS             ; Сохранение в стеке содержимого регистра DS
PUSH CS             ; Сохранение в стеке содержимого регистра CS
PUSH counter            ; Сохранение в стеке содержимого ячейки памяти
PUSH table[BX][DI] ; Сохранение в стеке содержимого ячейки памяти

PUSH AX             ;
PUSH ES             ;
PUSH DI             ;
PUSH SI             ;
…
…
…
POP  SI             ;
POP  DI             ;
POP  CS             ;
POP  AX             ;

PUSH ES             ; Копирование в DS
POP  DS             ;
```

3)
```
XGHGAX, BX      ; AX ↔ BX
```

```
XGHGAL, BH        ; AL ↔ BH
XGHGword_loc, DX  ; ячейка word_loc ↔DX
XGHGDL, byte_loc  ; AL ↔ ячейка byte_loc
```

4) XLAT – извлечение элемента таблицы. Начальный адрес таблицы нужно загрузить в BX, а номер элемента таблицы в AL.

```
MOV  AL,10          ; номер элемента таблицы в AL
MOV  BX, offset s_tab;
XLAT s_tab          ; извлечь значение байта из таблицы AL
```

2 Арифметические команды

1) Сложение

```
ADD  пр, ист          ; пр + ист→пр
ADD  пр, data         ; пр + data → пр
ADC  пр, ист          ; пр + ист+CF→пр
ADC пр, data          ; пр + data+CF → пр
```

2) Вычитание

```
SUB  пр, ист          ; пр - ист→пр
SUB  пр, data         ; пр - data → пр
SBB  пр, ист          ; пр – ист-CF→пр
SBB пр, data          ; пр – data-CF → пр
```

3) Умножение

а) без знака

```
MUL     ист     ; AL * ист → AX – байт, AX * ист → DX:AX – слово
```

б) со знаком

```
IMUL    ист
```

4) Деление

а) без знака

```
DIV     ист
```

б) со знаком

```
IDIV    ист     .
```

AX / ист → AL – (остаток в AH), DX / ист → AX (остаток в DX)

5) Сравнение

```
CMP  пр, ист          ; пр-ист      формируется регистр флагов
CMP  пр, data         ; пр - data   формируется регистр флагов
```

6) Команды инкремента и декремента

```
INC     пр      ;       пр +1 → пр
DEC     пр      ;       пр - 1 → пр
NEG     пр      ;       - пр → пр
```

Примеры

!) Сложение ADD

```
ADD     AX + BX     ; Сложение младшего слова        AX + BX → AX
ADC     BX + DX     ; Сложение старшего слова        BX + DX +CF→ BX

ADD  AX + mem_word  ; Добавить значение ячейки к регистру
ADD  AL,10          ; Добавить константу к регистру
ADD  mem_byte, ofh  ; Добавить константу к содержимому ячейки памяти
```

```
INC         CX          ;
INC         AL          ;
INC         mem_byte    ; Приращение значения байта
INC         mem_word[BX] ; Приращение значения слова памяти
```

2) Вычитание

```
SUB         AX, BX      ; Вычесть младшие слова
SBB         BX, DX      ; Вычесть старшие слова
SUB   mem_word[BX], AX
SUB         AL,10
SUB   mem_byte, ofh
```

3) Умножение

```
MUL         BX
MUL         mem_byte
IMUL        AL
IMUL        mem_word
```

4) Команды инкремента

```
DEC         CX
DEC         AL
DEC         mem_byte
NEG         AL
```

3Логические команды и команды сдвига

а) логические команды

```
AND         пр, ист         ;       пр & ист→пр
OR          пр, ист         ;       пр ∨ ист→пр
XOR         пр, ист         ;       пр·⊕ ист→пр
TEST        пр, ист         ;       пр &ист и формирование флага признаков
```

б) сдвига

1) Циклический

```
ROR         пр, cnt     ; Сдвиг вправо на cnt разрядов cnt=1,
                         ; Сдвиг вправо на cnt разрядов cnt=CL

RCR         пр, cnt
ROL         пр, cnt     ; Сдвиг влево на cnt разрядов cnt=1
                        ; Сдвиг влево на cnt разрядов cnt=CL
RCR         пр, cnt
```

2) Логический

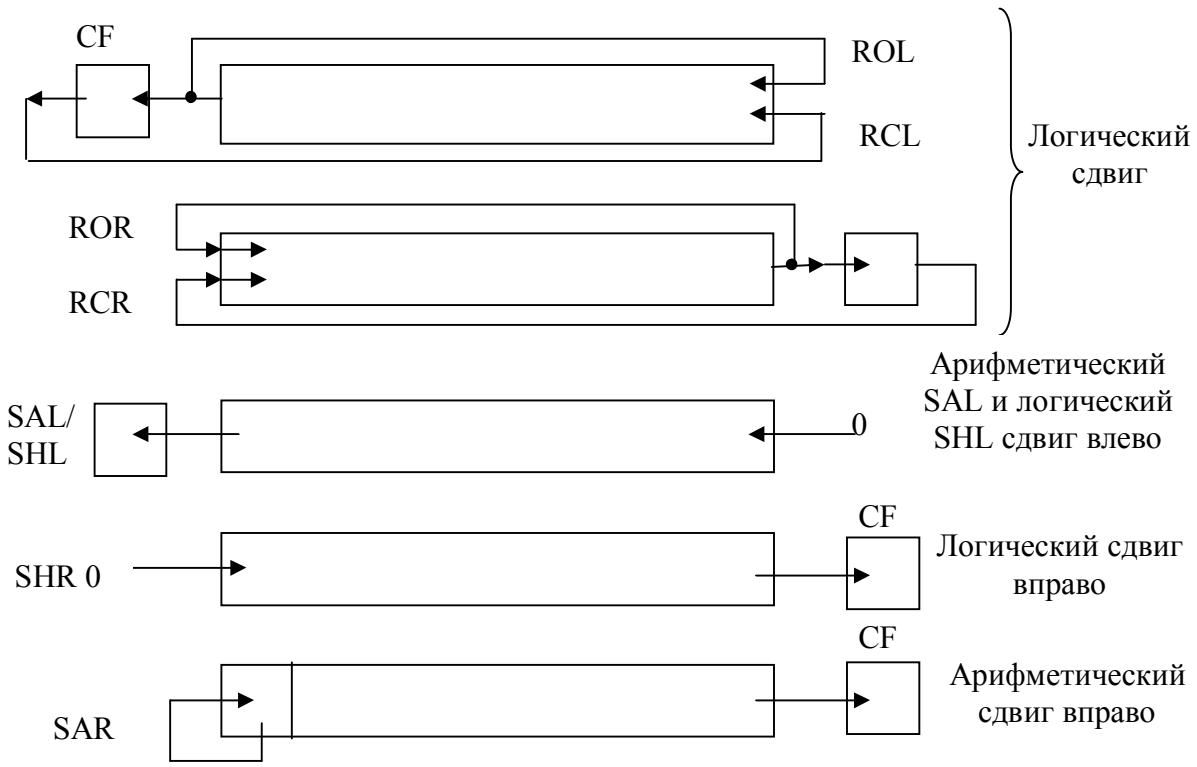
```
SHR         пр, cnt
SHL         пр, cnt
```

3) Арифметический

```
SAR         пр, cnt
SAL         пр, cnt
```

Замечание

Коды команд логического и арифметического сдвига влево совпадают. **SHL/SAL**



4 Строковые команды

1) Команды перемещения

MOVSB (MOVSW) ; mem[DS:SI] →mem[ES:DI
если DF=0, то инкремент SI и DI
если DF=1, то декремент SI и DI

LODSB (LODSW) ; mem[DS:SI] → A (A=AL- байт, A=AX – слово)
если DF=0, то инкремент SI
если DF=1, то декремент SI

STOSB (STOSW) ; A → [ES:DI] (A=AL- байт, A=AX – слово)
если DF=0, то инкремент DI
если DF=1, то декремент DI

2) Команды сравнения

SCASB (SCASW) ; A – mem[ES:DI]
если DF=0, то инкремент DI
если DF=1, то декремент DI

CMPSB (CMPSW) ; mem[DS:SI] – mem[ES:DI]
если DF=0, то инкремент SI и DI
если DF=1, то декремент SI и DI

3) Повторение строковых команд

REP MOVSB(W) ; повторять, счетчик цикла CX=CX-1 пока CX≠0
REPE/REPZ ; повторять пока CX≠0 и ZF=1
REPNE/REPNZ ; повторять пока CX≠0 и ZF=0

Примеры

1) Групповая пересылка байтов

```
CDL                 ; сбросить флаг DF в 0 для пересылки с увеличением адресов
LEA   SI, source    ; смещение адреса источника source → SI
LEA    DI, ES:dest  ; смещение адреса приемника dest →DI
MOV  CX, 100        ; количество элементов 100 → счетчик CX
REP MOVSB           ; скопировать байты
```

2) Сравнение строк

```
CLD
MOV  CX, 100        ; количество элементов 100 → счетчик CX
CMPSB               ; Сравнение строк до тех пор пока либо значение CX не
                        ; станет равным 0, либо не будет найдена пара несовпада-
                        ; ющих строк (т.е. поиск несовпадающих строк)

REPNE CMPS dest, source ; Поиск совпадающих строк
```

3) Сканирование строк

```
CLD
LEA         DI, ES:b_string
MOV         AL, ' '
MOV         CX, 100
REPE  SCASB  b_string
```

5 Команды передачи управления

5.1. Команды безусловных переходов.

Производят модификацию регистра инструкций IP или регистров IP и CS без сохранения прежних значений этих регистров.

а) команды передачи управления

```
JMP         addr
JMP         ист
```

б) Переход на процедуру

```
CALL        addr        ; переход на процедуру по адресу addr CS:IP → стек
CALL        ист         ; переход на процедуру по адресу ист  CS:IP → стек
```

в) Возврат из процедуры

```
RET                     ; возврат из процедуры стек → CS:IP
RET         n           ; возврат из процедуры стек → CS:IP, SP=SP+n
```

5.2 Условные

```
Jcc         offset      ; переход, если условие cc выполняется
                            ; где cc = C и NC, Z и NZ и т.д.
```

5.3 Организация циклов

```
LOOP offset             ; CX=CX-1, возврат по смещению offset, если CX≠0
```

Разновидности

```
LOOPE/LOOPZ offset ; CX=CX-1, возврат по смещению offset, если CX≠0, ZF=1
LOOPNE/LOOPNZ offset ; CX=CX-1, возврат по смещению offset, если CX≠0, ZF=0
JCXZ  offset            ; переход по смещению, если CX=0
```

5.4 Команды программного прерывания

```
INT         n           ;  где n=0…255,  FLAGS → стек, 0 →IF, 0 →TF,
                        ; вектор прерывания → CS:IP
IRET                    ; стек → CS:IP, стек → FLAGS
```

Примеры

1 Условная передача управления

Вычислить c=a*a+b , но если ответ превосходит размер байта, то передать управление на метку error

```
MOV     AL, A
MUL     AL
JC      error       ; если a*a>255, то CF=1 и переход на error
ADD     AL, B
JC      error       ; если a*a+b>255, то CF=1 и переход на error
MOV     C, AL
```

2. Управление циклом

Вычислить факториал числа байтовой переменной n (значение 0…8)

n! = n(n-1)(n-2)..1

```
        MOV     AX, 1
        MOV     CL, n
        MOV     CH, 0       ; CX=n
        JCXZ    F1          ; при n=0 обойти цикл
        MOV     SI, 1
F:      MUL     SI          ; (AX*SI →(DX,AX))
        INC     SI
        LOOP    F
F1:
```

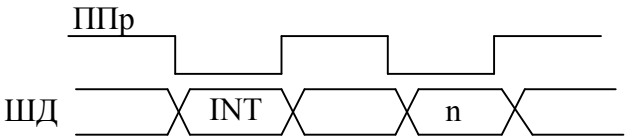
3. Усреднение чисел без знака

```
        LEA     BX, table
        MOV     CX, 100
        CALL        average

average PROC
        SUB     AX, AX      ; Присвоить делимому 0
        SUB     DX, DX
        PUSH    CX          ; сохранить значение счетчиков в стеке
Add_w:
        ADD     AX, [BX]    ; добавить текущее значение к сумме
        ADC     DX, 0
        ADD     BX, 2
        LOOP    add_w       ; все ли слова просуммированы ?
        POP     CX          ; Да
        DIV     CX          ; вычислить среднее
average RET
        ENDP
```

Общие сведения об аппаратных и программных прерываниях (командаINT n)

Все возможные прерывания нумеруются числами от 0 до 255. Для каждого прерывания при необходимости составляется своя процедура обработки прерывания (ПОП). Начальные адреса этих процедур (векторные прерывания) записываются в самые первые ячейки памяти (первый Кбайт памяти). Эти начальные адреса попадают сюда при загрузке ОС в оперативную память. Четыре ячейки (байта) выделяются для адресации каждой ПОП (младшие 2 байта – значение IP, старшие 2 байта – значение CS).

Устройство, в котором произошло событие, требующее внимания ОС, посылает в МП сигнал прерывания (ЗПр) и номер этого прерывания, точнее команду INT n (n=0..255) по сигналу от микропроцессора ППр.



Получив номер вектора прерывания n, МП сохраняет в стеке регистр флагов FLAGS и адрес возврата и передает управление по адресу n-го вектора прерывания, т.е. передает управление n-ой ПОП, которая начинает обрабатывать данное прерывание (Подробнее см. раздел Ввод-вывод по прерыванию).

ПОП – это, в общем, обычная процедура. Если она использует какие-то регистры, то она в начале своей работы сохраняет в стеке значения этих регистров, а в конце работы восстанавливает их. Поэтому, когда будет возобновлена работа прерванной программы, в регистрах останутся те же значения, которые были в них в момент прерывания. Для возобновления работы прерванной программы по команде IRET из стека восстанавливаются регистры IP, CS и регистр флагов.

Функции DOS

Рассмотренный механизм прерываний первоначально был предназначен для того, чтобы ОС могла следить за событиями во внешних устройствах ЭВМ (принтере, дисководах и др.). Но затем этот механизм стали использовать и в других целях. Например, в различных программах приходится выполнять одни и те же действия: выводить символы на экран или вводить символы с клавиатуры и т.д.

Чтобы избавить авторов программ от необходимости программировать эти действия в каждой программе заново, такие часто повторяющиеся действия описывают один раз в виде соответствующих процедур и включают их в состав ОС. Теперь все программы могут пользоваться этими процедурами. Эти процедуры ОС можно вызывать с помощью программных прерываний, т.е. с помощью команды

INT n, n=0…255

Эта команда в программе вызывает искусственное (насильственное) прерывание с номером n. Хотя по команде INT n можно вызвать ПОП для любого n (например, n=0), но следует использовать эту команду только для вызова процедур ОС. В состав ОС (точнее MS DOS) входит много процедур и для них не хватает номеров прерываний. Поэтому эти процедуры объединены в группы, чтобы процедуры из одной группы вызывались по прерываниям с одним и тем же номером.

Процедуры одной группы принято называть функциями соответствующего прерывания. Чтобы различить их, перед выполнением команды INT n в регистр AH записывают номер нужной функции

```
MOV     AH, <номер функции>
INT     < номер прерывания>
```

Получив по команде INT n управление, ОС по номеру из регистра AH определяет, к какой именно функции произошло обращение и передает ей управление. Для выполнения функции может потребоваться определенная информация, например, для функции вывода символа на экран нужно указать код этого символа. Такая информация передается через регистры. Какие именно параметры надо передавать и через какие регистры – зависит от конкретной функции.

Некоторые функции прерывания 21h

Чтобы после завершения программы вернуть управление операционной системе, эта программа обращается к функции 4ch прерывания 21h/

```
MOV     AL, <код завершения>
MOV     AH, 4ch
INT     21h
```

где <код завершения> - некоторое целое число, передающее информацию о успешной работе вызванной программы или об возникшей ошибке.

Вывод на экран (в текстовом режиме)

Для вывода одного символа на экран ПК используется информация 02 прерывания 21h.

```
MOV     DL, <код выводимого символа>
MOV     AH, 2
INT     21h
```

Выводимый символ высвечивается в позиции курсора, после чего курсор сдвигается на одну позицию вправо (или на начало новой строки из последней позиции текущей строки).

Для вывода на экран строки (последовательности символов) можно с помощью функции 09 прерывания 21h.

```
DS:DX – начальный адрес строки
MOV     AH, 9
MOV     21h
```

Перед обращением к этой функции в регистр DS должен быть помещен номер того сегмента памяти, в котором находится выводимая строка, а в регистр DX – смещение строки внутри этого сегмента. При этом в конце строки должен находиться символ '$', который служит признаком конца строки и который сам не выводится.

6 Специальные команды

1) Манипуляции с флагами регистра FLAGS

а) с флагом переноса CF

```
CLC             ; 0 → CF
STS             ; 1 → CF
CMC             инвертирование CF
```

б) с флагом направления DF

```
CLD             ; 0 → DF
STD             ; 1 → DF
```

в) с флагом прерываний IF

```
CLI             ; 0 → IF
STI             ; 1 → IF
```

2) Команда останова

```
HALT            ; останов и ожидание сигнала сброса или сигнала
                ; прерывания
```

3) Команда ожидания

```
WAIT            ; переход МП в состояние ожидания внешнего
                 ; активного сигнала
```

4) Холостая команда

```
NOP
```

1.2.6. Программирование на языке ассемблера микропроцессора I8086

В этом разделе приводится в основном краткий конспект книги Григорьева В.Л. Программирование однокристальных микропроцессоров. – М.: Энергоатомиздат, 1987. 288 с.

В языке ассемблер есть командные операторы, т.е. операторы которые транслятор переводит в команды МП (они впоследствии будут выполняться), операторы распределения данных и директивы ассемблера - команды для транслятора.

Исходный модуль программы представляет собой последовательность операторов (или предложений) языка ассемблера, которые классифицируются на:

- командные операторы, определяют генерируемые ассемблером машинные команды. Они содержат мнемонику и (необязательно) один или два операнда;
- операторы распределения данных, резервирующие ячейки памяти для программных данных;
- директивы ассемблера (или псевдокоманды), содержащие специальные указания для ассемблера.

Иногда операторы распределения данных относят к директивам.

Формат операторов.

Операторы имеют свободный формат, т.е. любое поле оператора может начинаться с любой позиции строки, но поля должны быть отделены от предыдущего поля одним или несколькими пробелами. Формат командных операторов имеет вид:

{Метка:} {Префикс} Код операции {Операнд(ы)} {; Комментарии}

Фигурными скобками обозначены необязательные поля.

Директива ассемблера и операторы распределения данных имеют следующий формат:

{Имя} Директива {Операнд(ы)} {; Комментарии}

Имя директивы никогда не заканчивается двоеточием. Некоторые директивы требуют обязательного наличия имени, например, SEGMENT, ENDS, PROC и другие. В других директивах поле имени должно быть пустым, например NAME, ASSUME, ORG, PUBLIC и другие. В операторах распределения данных DB, DW, DD имя является необязательным.

Поле директивы содержит одно из ключевых неизменяемых слов ассемблера и определяет его действия в процессе ассемблирования. Операнды директив аналогичны операндам командных операторов.

В языке ассемблера имеются три пары взаимосвязанных директив , требующих согласованного употребления :

SEGMENT/ENDS(сегмент / конец сегмента)

PROC/ENDP (процедура или подпрограмма/конец процедуры)

MACRO/ENDM (начало определения макрокоманды/ конец определения макрокоманды)

Элементы операторов .

Операторы состоят из:

- ключевых слов;
- идентификаторов (меток и переменных);
- численных констант;
- символьных цепочек;
- специальных символов;
- комментариев.

Ключевые (или зарезервированные) *слова* являются именами, имеющими для ассемблера определенный смысл. Примерами служат мнемоники команд и директив (MOV, PROC и др.).

Идентификатор (общий термин меток и имен переменных) - это определяемая программистом последовательность символов. Первым символом в последовательности должна быть буква или один из символов *@, -, ?*, но один знак ? не может быть идентификатором. Остальные символы могут начинаться с прописной или строчной буквы латинского алфавита или одним из следующих специальных символов *@, -, ?* . Ассемблер различает идентификаторы по первым символам, количество этих символов зависит от версии языка ассемблер.

Операнды команд (регистры, переменные, метки, константы) имеют атрибут типа , который сообщает ассемблеру некоторую информацию об операнде. Например, тип abc сообщает ассемблеру, что операнд является численной константой, а не адресом ячейки памяти.

Ассемблер определяет для программиста набор следующих регистров МП:
общие типа BYTE - регистры AL, AH, BL, BH, CL, CH, DL, DH
общие типа WORD - регистры AX,BX,CX,DX,SP,BP,SI,DI
сегментные типа WORD - CS,DS,SS,ES.

Переменная - это единица программных данных, имеющая символическое имя. Переменная имеет три атрибута:

1) сегмент (SEG) , идентифицирует сегмент , содержащий переменную;
2) смещение (OFFSET) представляет собой расстояние в байтах переменной от начала (базы) содержащего ее сегмента (диапазон 0..65535);
3) тип (TYPE) , идентифицирует единицу памяти, выделяемую для хранения переменной, т.е. 1 (байт), 2 (слово), 4 (двойное слово).

Ассемблер использует атрибуты переменной для определения формата генерируемой машинной команды.

Метка - это имя, относящееся к ячейке памяти, в которой находится команда, и предназначенное для использования как операнда в командах управления .Метка имеет 4 атрибута :

-сегмента (аналогичен такому же атрибуту переменной);
-смещения (аналогичен такому же атрибуту переменной);
-расстояние (дистанция), определяет возможность перехода по команде передачи к метке, находящейся или внутри сегмента (т.е. управления с помощью двухбайтного смещения -тип NEAR) или вне текущего сегмента (т.е. с помощью 4-байтного указателя сегмент: смещение - тип FAR);
-предположение о сегментном регистре CS. Когда при трансляции (1-ый проход) ассемблер встречает метку как операнд, он не знает ее смещение. Он делает предположение о ее расположении: близкая - NEAR , если знает, что смещение в пределах ±128байт, т.е. SHORT

Ассемблер считает имя меткой, если выполняется одно из следующих правил:

-имя перед кодом операции заканчивается двоеточием, например,

```
NED:    NOP
```

-имя находится в поле имени директивы LABEL; например,

```
NED  LABEL    FAR
```

-имя находится в поле директивы EQU, например,

```
NED     EQU     THIS NEAR
```

-имя находится в поле имени пары директивPROC/ENDP

```
NED     PROC    NEAR
.
.
```

.
NED ENDP

Численная константа - это численное значение, вычисляемое во время ассемблирования по заданному выражению. Константа отличается от адреса памяти (метка или переменная) тем, что она определяет число. Численные константы можно представлять в системах счисления с основание 2 (111001B), 8 (16Q), 10 (15 или 15D), 16 (38H, 0A3H). Если константа начинается с символа, то число предваряется нулем (0A3H). Диапазон всех чисел соответствует диапазону 16-тиразрядных двоичных чисел (включая знаковый разряд). Отрицательные числа представляются в дополнительном коде. В качестве численных констант допускается использовать односимвольные (типа BYTE) и двухсимвольные (типа WORD) цепочки, например фрагмент

```
NED   EQU   'AB'
                    MOV  DX, NED
```

обеспечивает загрузку 16-тиричного числа 4142H в регистр DX (в код ASCII буква A имеет код 41H, а буква B - 42H.

Символьные цепочки (или символьные константы) - заключаются в апострофы и обычно имеют максимальную длину до 255 символов (за исключением символов возврата каретки и перевода строки).Например,

```
FAM   DB    'IVANOV'
```

Символьные цепочки, содержащие более двух символов , применяются только для инициализации памяти. Ассемблер представляет цепочку в виде последовательности байт, соответствующих кодированию символов цепочки в коде ASCII.

Директивы.

Директивы определения переменных.

Директивы DB,DW,DD.

Рассмотрение директив начнем с директив определения данных DB (определить байт), DW (определить слово), DD (определить двойное слово).
Формат директив.

```
                    DB
<имя переменной>    DW  <нач. значение>, [<нач. значение>]…
                    DD
```

Мнемокоды директив фактически определяют тип переменных (байт, слово, двойное слово). Поле операнда директив идентифицирует, сколько байт, слов, двойных слов распределяют директивы и какими должны быть их начальные значения. Атрибутом сегмента переменной является тот сегмент, в котором она определена. Атрибут смещения переменной равен числу байт от начала сегмента до ячейки с переменной.

В качестве начального значения может фигурировать выражение, содержащее значение инициализации для одной единицы памяти. Выражения могут быть числовые и адресные.

Замечание! .кроме директив DB,DW,DD есть еще следующие директивы для 386,486, 586 .

DP - определить указатель - 32-разрядный указатель
DF - определить дальний указатель - 48- разрядный указатель
DQ - определить учетверенное слово (8байтов) - вещественные числа

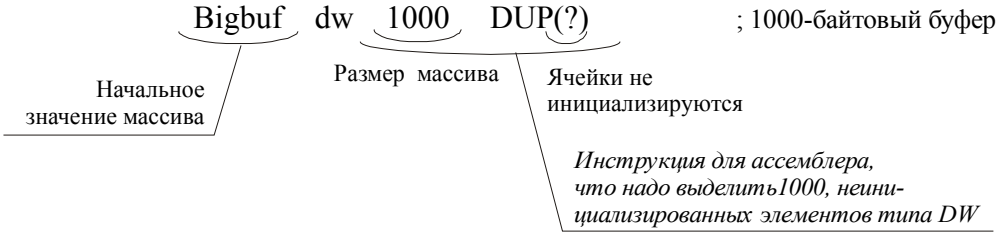
DT - определить 10 байтов - BCD числа

Инициализированные и неинициализированные слова.

Если первоначальное значение переменных не важно, то их можно определить как *неинициализированные переменные* .

```
stud        db      ? ; байт с неизвестным значением
morestud    dw      ?;  слово с неизвестным значением .
```

Для получения резервирования большого инициированного пространства следует использовать DUP выражения со знаком ? в скобках.

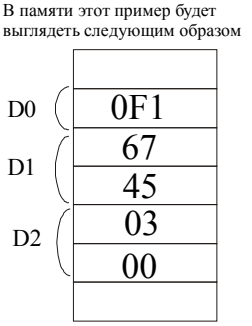


При запуске программы созданный этой директивой 1000-байтовый буфер состоит из байтов, не имеющих определенных значений. Главная причина для использования неинициализированных переменных состоит в уменьшении размера исполняемого файла. Вместо записи бесполезных байтов на диск во время исполнения выделяется неинициализированное пространство. Для этого необходимо придерживаться двух правил:

1) Размещать все неинициализированные переменные последними в сегменте данных.
2) Предварять неинициализированные переменные директивой UDATASEG. Встретив директиву UDATASEG Turbo Assembler автоматически поместит эту часть за последнюю инициализированную страницу.

Примеры:

```
DATA    SEGMENT
D0          DB      0F1H            ;один байт равен F1H
D1          DW      4567H           ;одно слово , равное 4567H
D2          DW      D2              ;одно слово, равное 0003
D3          DD      ?               ;двойное слово без инициализации
D4          DB      20DUP(?)        ; двадцать байтов без значений
D5          DW      10DUP(0)        ;десять нулевых слов
DATA    ENDS
```



Директивы DW с именем D2, определяют слово, равное 0003 - это смещение переменой D2 относительно начала сегмента, т.е. слово в этой директиве представляет собой адрес переменной в сегменте. Одиночный вопросительный знак в поле операнда означает, что содержимое ячеек памяти не задается в директиве. Конструкция DUP применяется для распределения и инициализации заданного числа единиц памяти. В качестве элементов в конструкциях DUP могут фигурировать численные и адресные выражения, вопросительный знак, список или повторная конструкция DUP.

Строковые переменные.

Хотя директива DB может создавать строковые переменные, в языке ассемблера отсутствуют строковые команды для чтения и записи строк, удаления символов, сравнения одной строки с другой. Эти действия могут быть реализованные с помощью процедур.

Существует несколько строковых форматов, например,

ASCII$ -строка - это набор ASCII-символов , заканчивающийся знаком доллара $

ASCIIZ -строка - - это набор ASCII-символов , заканчивающийся нулевым байтом. Для создания ASCII$ - строки следует использовать директиву DB.

```
str_$       db      'Hello','$'
str_$       db      'Hello $'
```

Более распространены ASCIIZ- строки, они создаются также директивой DB

```
str_0       db      'Hello',0
```

ASCII - строки могут иметь длину от одного до тысяч байт.

Первый байт строки - это либо ASCII символ, либо нулевой байт, называемый также нулевым символом ASCII.

Если первый символ равен 0, то строка нулевая, поэтому легко создавать переменные нулевой длины.

```
sbuf    db          81  dup(0)          ;80- символьный буфер
```

При создании строки таким способом всегда следует устанавливать счетчик DUP на 1 больше максимального числа символов, которое планирует записывать в эту строку, чтобы оставалось место для нуля, которым строка всегда должна закачиваться.

Замечание!

Кавычки в кавычках. Строки, описанные с помощью директивы DB могут браться либо в апострофы ('), либо в двойные кавычки (").

Простейший метод включения кавычек внутрь строк состоит в использовании различных типов кавычек для выделения всей строки и ее отдельных частей:

```
st1         db          ' "Прекрасно" сказал Иван', 0
st1         db          " ' Отлично' сказал Григорий", 0
```

Локальные метки.

Различают глобальные и локальные метки. Глобальные метки доступны из любого места программы. Такие метки должны быть все разные, что создает серьезные неудобства при реализации коротких переходов , например таких

```
        CMP   AX,9              ;AX=9?
              JE        SK                ;пропустить строку и
перейти вниз, если AX=9
        ADD   CX,10             ;Иначе прибавить к CX 10
        SK:
```

Короткие переходы, например JE, на метку SK являются очень часто используемыми для программирования на языке ассемблера. Скорее всего никакие другие команды не будут выполнять переходы на эту же метку, следовательно, метка SK не нужна нигде, кроме этого места. В больших программа таких переходов может существовать очень много и придется придумывать (изобретать) для каждого из них новые имена . Для упрощения этого TURBO ASSEMBLER позволяет создавать *локальные метки*, существующие в той части программы, в которой они нужны.

Локальные метки идентичны любым другим меткам, за исключением того, что они начинаются с двух знаков @@. Например,

```
@@ 20:
@@ TWO:
```

Внимание! Локальная метка, существует только до следующей глобальной метки.
Пример:

```
                JMP   THERE          ; Переход на глобальную метку
@@ 20:    INC   AX
                CMP   AX, 30
                JE          @@20          ; Переход на локальную
метку вверх
THERE:    CMP   AX, 30
                JE          @@20          ; Переход на локальную
метку вниз
                XOR   CX, CX
@@20:
```

Массивы в языке Ассемблера.

Не существует специальных команд, структур или методов описания и использования массивов на языке Ассемблера. Хотя работать с массивами можно. Так, например, массив целых чисел можно создать используя оператор DUP:

```
AR    DB    10 DUP (?) -          ;массив из 10 целых элементов.
```

Можно определить последовательность из 10 значений:

```
AR    DB          0,1,2,3,4,5,6,7,8,9
```

Можно создавать и массивы из составных элементов, например строк:

```
AR          DB          20 DUP(?),0          : AR [0]
            DB          20 DUP(?),0          : AR [1]
            DB          20 DUP(?),0          ; AR [3]
```

При создании больших массив следует использовать такие директивы :
LABEL - метка
REPT - повторить

```
LABEL          AR          BYTE
REPT      100
                      DB                20 DUP(?),0
ENDM
```

Первая строка описывает метку AR типа BYTE . Здесь можно использовать и другие типы WORD, DWORD и т.д.

Директива LABEL указывает ассемблеру, как адресовать последующие данные. В памяти она не резервирует никакого пространства.

В приведенном примере следующие данные являются строками, которые всегда адресуются как одиночные байты.

Директива REPT (повторить определенное количество раз) (в данном случае 100 раз) повторяет любой оператор языка ассемблера. Все, что находится между REPT и ENDM (конец макро) повторяется так, как если бы эти строки набрали необходимое количество раз.

Рассмотрим удобный прием изменять каждый раз описание в определении. Например для создания массива из десяти целых элементов и задания каждому элементу массива значений от 0 до 9 можно использовать следующее описание:

```
VAL = 0              ;                          директива присваивания
LABEL     AR                  WORD
REPT      10
                DW    VAL
                VAL = VAL+1        ;директива присваивания
ENDM
```

Индексирование массивов.

Рассмотрим способы чтения и записи значений в массивы. Например, как обратиться к четвертому элементу массива? Для этого нужно понимать, что в языке ассемблера индексы массивов - это обычные адреса. Поэтому независимо от типа данных, записанных в массив, индексация отдельных значений (т.е. отдельных элементов) сводится к двум шагам:

1) умножить размер элемента массива на индекс массива; I
2) прибавить результат к базовому адресу массива.

Пример. В простом массиве байтов, если I=0, тогда I*1=0*1=0 плюс адрес ar (база) определяет первый элемент массива ar [0] Второй элемент ar [1] (I=1) расположен по базовому адресу плюс 1 (I*1=1) и т.д.

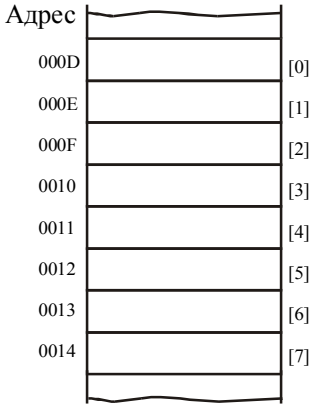


Рис. 1

Итак, нужно значение индекса массива преобразовать в адрес памяти. Пример приведен на рис 1.

Из рисунка видно, что индекс 0 эквивалентен адресу 000D , т.е. базовому адресу всего массива. Индекс 1 соответствует 000E, индекс 2 - 000F и так до индекса 7, который определяет элемент с адресом 0014.

Пример. Работа с элементами 100 - битового массива. Режим TURBO ASSEMBLER IDEAL.

```
DATASEG
AR          DB                  100 DUP(?)
CODESEG
```

```
MOV     AL, [AR+63]  ; Добавление 64-того элемента массива к AL
```

В большинстве случаев под индексы массивов предпочитают использовать регистры или переменные в памяти. Применяется базовый способ адресации, можно записывать значения индексов массивов в регистр BX.

Например, в регистр AL надо загрузить элемент массива AR с индексом index. Это можно сделать так:

```
DATASEG
INDEX       DW      ?
ARRAY       DB              10 DUP(?)
CODESEG
            MOV     BX, [INDEX]     ; получить значение индекса.
            MOV     AL, [BX+AR]     ; AL←AR [INDEX]
```

Для этих же целей можно использовать регистры SI и DI

```
MOV   SI, [INDEX]       ; Получить значение индекса
MOV   AL, [SI+AR]       ; AL←AR [INDEX]
MOV   DI, [INDEX]       ; Получить значение индекса
MOV   AL,[DI+AR]        ; AL←AR [INDEX]
```

Операторы TYPE, LENGTH, SIZE.

Эти встроенные операторы используются в выражениях, где запрашивается тип информации.

Оператор TYPE (тип) сообщает, сколько байт отведено для переменной (1 (DB) , 2 (DW) или 4 (DD)).Это тип элемента.

Оператор LENGTH (длина) сообщает число основных единиц памяти (байт, слов или двойных слов), распределенных в строке с определенной переменной. Количество элементов в массиве.

Оператор SIZE (размер) сообщает, сколько байт памяти распределено при определении переменной. Для этих операторов справедливо соотношение.

SIZE<имя> = LENGTH<имя>* TYPE<имя>

Пример применения рассматриваемых операторов:

```
DATASEG
AR          DW    20 DUP(?)         ; память, выделяемая в сегменте данных
под массив,
                                    ; нужно подсчитать, какую сумму он занимает в
                                    ; памяти
        SUB   AX, AX                        ; обнулить AX
        MOV   CX, LENGTH AR     ; число элементов массива AR в CX
        MOV   SI, SIZE AR               ; индекс конца массива
        SUM:  SUB   SI, TYPE AR             ; чтобы осуществить переход на
предыдущий
                                    ; элемент, вычитается размерность элемента массива.
              ADD   AX, AR [SI]     ; прибавить элемент
              LOOP  SUM             ; цикл будет выполняться до тех пор, пока
CX не
                                    ; обратится в нуль.
```

Замена атрибутов переменных.

Переменная имеет три атрибута: тип, сегмент, смещение в сегменте.

Оператор замены типа имеет формат:

BYTE
WORD PTR < имя переменной >
DWORD

Конструкция <тип> PTR < имя переменной > означает, что в команде следует использовать сегмент и смещение с именем <имя переменной>, но с явно заданным типом.

Пример:

```
MOV  BYTE PTR    DW5, 7;
ADD  AL, BYTE    PTR    DW5;
```

Оператор замены типа требуется для устранения неоднозначности, которая может возникнуть в обращениях типа:

```
INC    [BX]          ;инкремент байта или слова ?
```

При оперировании байтом нужно явно определить тип операнда:

```
INC    BYTE PTR    [BX]
```

Атрибут сегмента заменяется явным указателем сегментного регистра в команде

```
ADD   CX, ES:[SI]; явно обозначен тип сегмента.
```

Директива RECORD.

В языке ассемблера можно осуществить символическое определение отдельных битов и битовых цепочек внутри байта или слова. Такое определение называется *записью (RECORD)***.** Каждая наименованная битовая цепочка в записи (в частном случае один бит), называется *полем*. Директива RECORD, определяющая запись, имеет формат

```
Имя    RECORD      <Имя поля:длина>,{<Имя поля : длина>}
```

Максимальное число бит в данной директиве равно 16, а минимальное - 1.

Оператор WIDTH (ширина), примененный к записи, сообщает ее ширину в битах, т.е. сумму длин всех составляющих ее полей.

Размер записи равен числу байт, необходимых для ее размещения: 1 (если ширина записи равна 1..8 бит) или 2 (если ширина записи равна 9..16 бит).

Директива RECORD позволяет давать имена битовым полям в байтах или словах. TURBO ASSEMBLER сам подсчитывает положение поля. Например

```
RECORD SBYTE  SIGN: 1 , VAL: 7
```

Нельзя создавать записи, больше чем слово. Переменные типа RECORD создаются следующим образом:

```
DATASEG
```

```
V1    SBYTE    <>          ;начальное значение этим полям не присваиваются
V2    SBYTE    <1>         ;SIGN=1, VAL - по умолчанию
V3    SBYTE    <,5>        ;SIGN- по умолчанию, VAL =5
V4    SBYTE    <1,127>     ;SIGN=1, VAL=127
V5    SBYTE    <3,300>     ;SIGN=1,VAL=44 (по модулю 128,т.к. 300=256+44
```

Обычно по умолчанию поля переменных RECORD имеют значение 0.Чтобы это изменить нужно добавить к ширине поля знак равенства и новое желаемое значение .Например:

```
RECORD    MINUSBYTE    MSIGN:1=1,MVAL:7=5
```

TURBO ASSEMBLER считает имена полей глобальными, т.е. активными во всех местах программы или модуля . Хотя в будущих версиях имена полей могут стать локальными.

Описав тип RECORD и несколько переменных данного типа можно использовать различные способы для чтения и записи значений в битовые поля этих переменных.

Директива STRUCTURE (STRUC) .

Эта директива обеспечивает определение и доступ к переменным со сложными типом данных. Структура, ограниченная директивами STRUC и ENDS с одинаковыми именами, представляет собой шаблон (карту), который присваивает имена и атрибуты(тип, длина , размер) набору полей. Единицами полей структуры являются байт, слово, двойное слово. Следовательно, каждое поле в структуре определяется директивами DB,DW,DD. Поля такого шаблона используются для определения переменных с новым типом данных, который порождает директива STRUC. Обращение к различным полям в таких переменных осуществляется с использованием оператора в виде точки (.)

Память структуре как шаблону не распределяется. Структура связывается с конкретной областью памяти, если имя поля структуры фигурирует в команде вместе с базовым адресом, в качестве которого может быть имя переменной или один из базовых регистров BX, BP. Определяя различные базовые адреса можно связывать структуру с различными областями памяти.

Приведем еще один пример структуры:

```
STRUC    DATE
DAY      DB      1       ; Поле ДЕНЬ - начально в 1
MONTH    DB      ?       ; Поле МЕСЯЦ - значение неустановлено
YEAR     DW  2004  ; Поле ГОД - начальное значение 2000
ENDS     DATE            ; "DATE" - здесь является необязательным
```

Этот тип в дальнейшем можно использовать.

Описание переменных в структурах.

Для использования структур нужно зарезервировать в памяти пространство под их поля. Результатом будет переменная, имеющая тип структуры. Каждое такое описание переменной начинается с метки, за которой следует имя структуры (DATE) и заканчивается списком начальных значений в угловых скобках < >. Чтобы использовать начальные значения, определенные ранее при описании структуры, скобки нужно оставить пустыми.

Пример.

```
DATASEG
BIRTHDAY        DATE        < >                     ; 1-0-2004
TODAY                   DATE        <5,10>          ;5-10-2004
DayInDayOut DATE <11,12,1915>       ;1-12-1915
```

Использование структурированных переменных.

Так как имена полей содержаться в определении структуры, то для обращения к ним помимо их имени необходимо использовать имя структурированной переменной, разделяя их точкой. Следовательно, для задания нового значения поля для переменой TODAY можно присвоить ему непосредственное значение с помощью команды

```
MOV  [TODAY.DAY], 5          ; Изменить день на 5
```

Можно загрузить значения полей в регистры, например, так

```
MOV  AX, [TODAY.YEAR]        ; Записать в AX значение ГОДА.
```

Замечание! В режиме IDEAL транслятора TURBO ASSEMBLER имена полей являются локальными и уникальными для структуры, в которой они определены. Татим образом, можно создавать многочисленные определения STRUC с одинаковыми именами полей. Например, можно иметь две различные структуры, каждая из которых содержит поля DAY, MONTH, YEAR. Нельзя аналогично поступить в более ограниченном режиме MASM, в котором все имена полей глобальные, т.е. каждое имя может встречаться только в одном определении структуры на протяжении всей программы. По этой причине в режиме MASM нельзя иметь две структуры с одинаковым названием полей. В режиме IDEAL в этом случае использовать структуры проще.

Выражения.

Адресные и строковые выражения .

Выражения являются наиболее сложными объектами полей операндов. Выражение представляет собой совокупность допустимых элементов этих полей, связанных разрешенными в языке ассемблера операндами (арифметическими, логическими и т.д.)

Важнейшими элементами выражений являются символические имена двух классов:

- переменных и меток;
- числовых значений.

Переменная - это имя (точнее адрес) ячейки памяти, содержимое которой считается данными. Напомним, что переменная определяется директивами DB,DW,DD распределения данных и имеет 3 атрибута: сегмент, смещение и тип.

Метка - это имя (точнее адрес) ячейки памяти, содержимое которой считается машинной командой. Напомним о метках следующее. Чаще всего метка определяется указанием в поле метки имени с последующим двоеточием. Но метки могут определяться и без завершающего двоеточия, например, директивами LABEL и в определениях процедур. Метки имеют 4 атрибута: сегмент, смещение, расстояние (тип NEAR или FAR), предположение о регистре CS.

Числовые выражения с помощью директивы EQU могут обозначаться символическими именами. В этом случае имя заменяет число и может быть названо "числовым именем". Числовое имя имеет тип NUMBER (число). Соответственно двум классам символических имен выделяются выражения: адресные и числовые.

Адресным называется выражение которое вычисляется для получения адреса памяти (данных и команд). Адресные выражения имеют обязательные атрибуты: сегмент, смещение, тип. Каждый из атрибутов адресного выражения является числом, но само выражение числом не является.

Числовым называется выражение, в результате вычисления которого получается число. Например, в команде

```
MOV  AL,MUR
```

символическое имя MUR является адресным выражением, поэтому эта команда загружает в регистр AL содержимое из ячейки памяти с адресом MUR .
В команде

```
MOV  AL, OFFSET MUR
```

Выражение OFFSET MUR использует встроенный оператор OFFSET (смещение), который формирует в виде числа смещение адресного выражения MUR. Следовательно, эта команда загрузит в регистр AL смещение адресного выражения MUR.

Таким образом, использование адресного выражения в поле операнда команд преобразований данных означает , что операндом будет содержимое адресуемой ячейки памяти. Адресное выражение в поле операнда команд передачи управления определяет встраиваемый в команду адрес перехода. Значение числового выражения (число), заданного в команде, превращается в непосредственный операнд.

Директивы управления сегментами.

Директивы SEGMENT/ENDS.

Напомним, что физический адрес формируется при сложении смещения и умноженного на 16 сегментного адреса, хранящегося в одном из сегментных регистров: CS (для сегмента кода), DS (для сегмента данных), SS(для сегмента стека), ES (для сегмента дополнительных данных).

Структура сегмента кода имеет вид:

```
Имя сегмента        SEGMENT    [<список атрибутов>]
                    Команды и относящиеся к
                    ним директивы.
Имя  сегмента       ENDS
```

Список атрибутов является необязательным, но он необходим, если программа состоит из нескольких модулей.

Структура сегмента данных имеет вид:

```
Имя сегмента        SEGMENT    [<список атрибутов>]
                    Директивы определения памяти,
              распределения и выравнивания
Имя  сегмента       ENDS
```

Именем сегмента может быть любой допустимый идентификатор. Назначение сегментов сегментным регистрам осуществляется директивами.

Директива ASSUME.

Формат директивы имеет вид:

ASSUME НАЗНАЧЕНИЕ,.., НАЗНАЧЕНИЕ,

где НАЗНАЧЕНИЕ имеет формат:

Имя сегментного регистра**:** Имя сегмента.

Пример определения сегмента кода:

```
CODSEG      SEGMENT   ; Сегменту COD назначим сегментный регистр CS
            ASSUME    CS:CODSEG, DS:DATSEG
START:      MOV       AX, DATSEG
            MOV       DS, AX
                      -
                -     команды и директивы
                      -
CODSEG      ENDS
```

Директива ASSUME не загружает сегментные адреса в сегментные регистры. Загрузка сегментных регистров, кроме регистра CS, производится явными передачами, обычно командой MOV. Так. как команда MOV не может передавать непосредственный операнд в сегментный регистр, поэтому для каждого регистра требуется две команды. Имя DATSEG является именем сегмента (постоянным значением), а не переменной, поэтому ассемблер транслирует первую команду MOV в команду, у которой операнд - источник является непосредственным значением (сегментным адресом).

Поскольку размещение сегментов в памяти осуществляет редактор связей, то операнд - источник фактически формирует редактор связей. Один и тот же сегмент можно назначать несколькими сегментным регистрам .Пример :

ASSUME CS:CODE, DS:CODE

Тогда загрузка сегментного регистра DS осуществляется командами:

```
MOV  AX, CS
MOV  DS, AX
```

Сегментный регистр CS обычно загружается командой межсегментного перехода при инициировании сегментного кода. Если директива ASSUME отсутствует , то во всех обращениях к переменной в памяти необходимо явно указывать сегментный регистр, содержащий базу для конкретного обращения к памяти.

Директива ORG.

Основной внутренней переменной ассемблера является счетчик ячеек (адресов), который при ассемблировании выполняет функцию, аналогичную функции программного счетчика (счетчика команд) при выполнении программы.

Счетчик ячеек сообщает ассемблеру адрес следующей ячейки памяти (точнее смещение в сегменте), в которую будет помещен следующий байт команды или данных.

Первое появление директивы SEGMENT определяет начало сегмента с заданным именем. При этом организуется новый счетчик ячеек, в который загружается нулевое смещение.

Размещение данных можно начать не с нулевой ячейки (по умолчанию), а с требуемой.

Текущее значение счетчика ячеек может быть изменено программистом с помощью директивы ORG (начало), имеющий формат

ORG <выражение>

При выполнении директивы ORG вычисляется значение выражения, а результат загружается в счетчик ячеек. Вычисление выражения производится по модулю 64 К.

Директива определения имен EQU.

Директива EQU позволяет программисту определять символические имена для часто используемых выражений и имеет формат:

Имя EQU <выражение>

Поле <выражение> может определять:
-константы;
-адреса;
-регистры;
-мнемокоды команды.

В поле "имя" находится имя, которое программист использует для представления выражения. Определяемые с помощью директивы EQU имена не разрешается переопределять. т.е. имя может появиться только в одной директиве EQU. Например:

```
CR      EQU  0DH          ; численная константа
ADR  EQU  AR [SI]+3    ;адресное выражение
COUNT   EQU  CX           ; регистр
```

Замечание!

Директива PURGE (удалить, освободить) удалить имена из таблицы имен: После этого это имя можно определить заново.

Директивы процедур PROC и ENDP.

Директива PROC (procedure) отмечает точку входа процедуры, а директива ENDP - окончание процедуры. Формат директив

```
Имя   PROC <тип> ;NEAR, по умолчанию, или FAR -
      .
      тело процедуры
      .
      .
Имя   ENDP
```

Имя является меткой, указывающей точку входа процедуры. Типом процедуры является NEAR или FAR (по умолчанию принимается тип NEAR). Ассемблер использует тип процедуры для определения того, какую команду CALL генерировать при вызове процедуры. Для процедур типа NEAR ассемблер генерирует команду внутрисегментного вызова. При обращении в процедуре в стеке запоминается , а затем восстанавливается только содержимое счетчика команд, т.е. для процедур типа FAR ассемблер генерирует

длинную команду межсегментного вызова, которая содержит базу и смещение точки входа. При обращении к этой процедуре в стеке запоминается, а затем восстанавливается содержимое регистров CS и IP.

Процедуры на языке ассемблера не имеют ограниченной области действия, т.е. из любого места программы можно обращаться к переменным и меткам внутри процедуры (т.е. пары директив PROC/ENDP).

Следует помнить, что командная последовательность, находящаяся сразу перед директивой PROC, в случае появления ошибки войдет в тело процедуры. Для предотвращения случайного выполнения процедуры без ее вызова рекомендуется размещать процедуры перед теми, которые их вызывают.

Входными параметрами могут быть численные значения и адреса. Приведем два способа передачи параметров процедурам. В одном способе значения параметров размещаются в регистрах МП перед обращением к процедуре. В другом способе передача параметров процедурам организуется через стек с помощью регистра BP, для которого сегментным является регистр SS.

В языке ассемблера МП I8086 рекомендуется придерживаться следующего соглашения о возвращаемом процедурой значением выходного параметра.

-значение переменной типа WORD возвращается в аккумуляторе AX;
-значение переменной типа BYTE возвращается в аккумуляторе AL;
-короткий указатель (смещение) возвращается в регистре BX;
-длинный указатель (база:смещение) возвращается в регистрах ES:BX.

Директива LABEL

Директива LABEL выполняет несколько функций, например, обеспечивает несколько точек входа в процедурах или может создавать имена для любых ячеек памяти независимо от их содержимого и предполагаемого использования . Она содержит информацию о типе определяемого имени, а тип однозначно указывает допустимое употребление имени. Формат директивы:

```
Имя    LABEL <тип>
```

В качестве типа фигурирует одно из пяти ключевых слов: BYTE, WORD, DWORD, NEAR, FAR. Первые три типа указывают на то, что имя представляет собой переменную. Имена переменных допустимы во всех команда, оперирующих данными, но их нельзя употреблять в командах передачи управления. Имя с типом NEAR или FAR .является меткой, которая может быть операндом в командах передачи управления, но недопустима в командах, оперирующих данными. Например:

```
BUF   LABEL     WORD
              DW    1234H
```

Этот фрагмент программы аналогичен директиве:

```
BUF   DW    1234H
```

Директива

```
NED   LABEL     NEAR
```

определяет имя NED как метку.

Директива LABEL удобна для определения значения указателя стека (т.е. вершины стека), например,

```
STACK_SEG       SEGMENT   WORD      STACK
                          DW        64 DUP(?)          ;
выделение размера стека
VSTK                      LABEL     WORD
;вершина стека
```

STACK_SEG ENDS

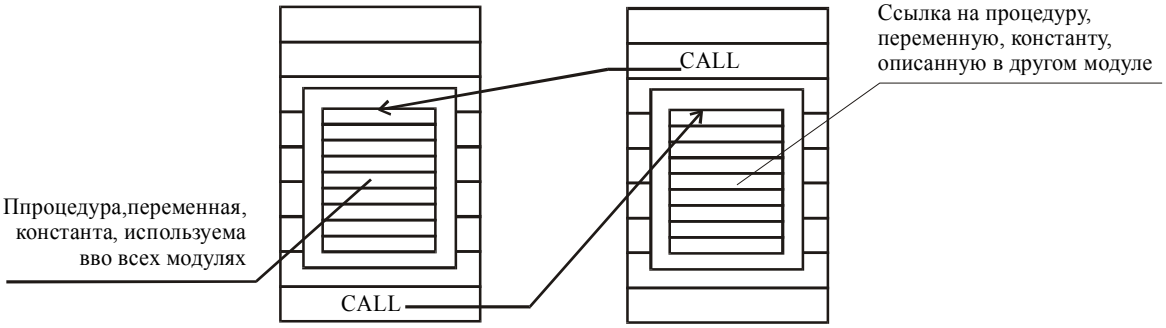
Директива DW резервирует 64 слова памяти без их инициализации. Директива LABEL присваивает имя VSTK слову, находящемуся после 64 слов от начала сегмента STACK_SEG. Смещение VSTK от начала сегмента STACK_SEG представляет собой то значение , которое будет содержать указатель стека SP когда стек пустой.

Директива связи модулей и сегментов.

Директивы связи модулей.

При структурном программировании сложная программа разделяется на несколько сравнительно автономных блоков (модулей). Модульная программа состоит или из нескольких исходных модулей, допускающих ассемблирование, независимо друг о друга, или может содержать несколько процедур. При модульном программировании возникают проблемы объединения (связи, связывания, редактирования связей, компоновки) отдельных модулей в единую программу. Для этого необходимы соответствующие директивы ассемблера и специальная программа, называемая редактором связей или компоновщиком.

Одна из задач объединения модулей заключается в том, что из данного модуля возможны обращения (ссылки) к именованным элементам (переменным, меткам и др.) в других модулях, а другие модули могут обращаться к таким же элементам данного модуля (см. рис. ниже).



С помощью межмодульных обращений модули объединяются в единую программу.

При ассемблировании каждого исходного модуля порождается объектный модуль (объектный файл), содержащий служебную информацию о межмодульных обращениях. Редактор связей использует эту информацию для объединения отдельных объектных файлов в единый объектный файл, в котором учтены все межмодульные обращения. Рассмотрение директив связей модулей и сегментов начнем с директив PUBLIC и EXTRN

Директивы PUBLIC и EXTRN .

Директива PUBLIC объявляет имя, определенное в данном модуле, глобальным, т.е. доступным для обращений из других модулей. Директива EXTRN, сообщает, что используемое в данном модуле имя определено в другом месте, т.е. является для данного модуля внешним. Директивы PUBLIC и EXTRN имеют формат:

PUBLIC Имя, [Имя]
EXTRN . Имя:тип,<Имя :тип>

Имя в обеих директивах может быть именем переменной, метки или константы, ,определенной директивой = . Типом для переменной может быть BYTE, WORD, DWORD.

Типом для метки может быть NEAR, FAR, а для константы -ABS (абсолютное значение)

Каждое имя, определяемое в данном модуле, может быть указано в директиве PUBLIC только один раз. Рекомендуется директивы PUBLIC и EXTRN сгруппировать в начале модуля.

Тип, фигурирующий в директиве EXTRN, должен совпадать с типом имени , определенным в другом модуле.

Примеры:

```
PUBLUC          NED, MUR, SVAL
EXTRN           DAD:BYTE, IVN:FAR, KRN:ABS
```

Если ассемблер встречает обращение к внешнему имени, он в команде резервирует необходимое место (с учетом типа имени), а в объектном файле делается отметка об этом. При объединении нескольких объектных файлов редактор связей просматривает такие отметки и, используя доступную ему информацию о глобальных именах, подставляет необходимые адреса или значения.

Директива END.

Каждый модуль должен заканчиваться этой директивой, для указания транслятору, что текст данного модуля закончен.

Директива END имеет формат

END [<пусковой адрес>]

В каждом исходном модуле может быть только одна директива END в последней строке модуля. Пусковым адресом является метка той команды, с которой должно начаться выполнение программы. При объединении нескольких модулей только в одном из модулей (главном) должен быть указан *пусковой адрес.*

Директивы объединения логических элементов.

При объединении модулей в одну программу возникает задача объединения логических сегментов, из которых состоит каждый модуль. Если эту задачу не решать, то при передаче управления одного модуля к другому необходимо изменять содержимое сегментного регистра CS, а при обращении к данным - сегментного регистра DS. Возникают сложности и при организации стека при объединении модулей.

Поэтому, если объединенный размер сегментов кода объединенных модулей размером меньше 64 Кбайт, то сегменты кодов модулей лучше объединять так, чтобы к содержащимся в них командам можно было обращаться при одном и том же содержимом регистра CS.

Аналогично следует объединять сегменты данных и сегменты стеков.

Необходимая информация о способах объединения логических сегментов объединяемых модулей содержится в директивах SEGMENT в виде списка атрибутов. Полный формат директивы SEGMENT имеет вид:

Имя SEGMENT [<тип выравнивания>] [<тип объединения>] [<'имя класса'>]

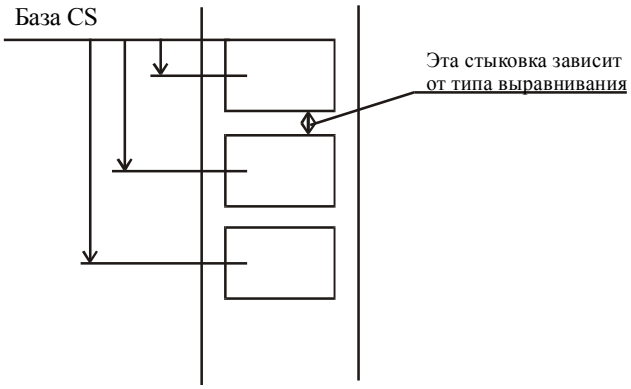
При этом объединяемые сегменты должны иметь одинаковые имена. Каждое из указанных полей операндов директивы является необязательным, но если они есть, то они должны следовать в указанном порядке.

Рассмотрим более подробно атрибут типа объединения .

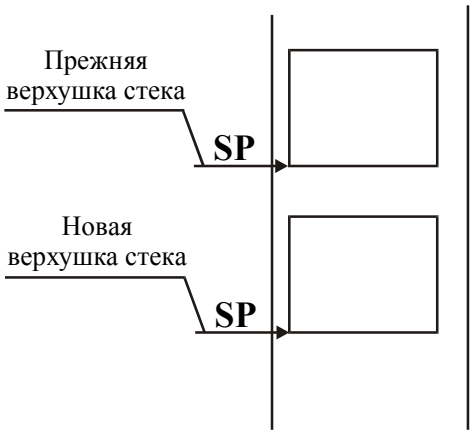
Тип объединения.

Этот атрибут показывает, каким образом данный логический сегмент должен объединяться с другим логическим сегментом имеющим такое же имя. Если для сегмента этот атрибут не определен, то сегмент считается необъединяемым, даже с сегментами, имеющими такое же имя. Тип объединения в случае необходимости нужно указывать в первом определении сегмент. В последующих директивах SEGMENT для сегментов с данным именем этот атрибут можно опускать. Могут использоваться следующие типы объединения : *PUBLIC, STACK, AT, COMMON, MEMORY.*

При атрибуте *PUBLIC* сегменты объединяются путем конкатенации, т.е. размещаются один за другим. При этом смещения внутри сегмента корректируются с учетом суммарного размера уже объединенных сегментов, что позволяет адресовать объединяемые сегменты при одном и том же содержимом сегментного регистра.



Тип объединения *STACK* применяется для логических сегментов, составляющих стек. Сегменты с атрибутом STACK объединяются так, чтобы размер полученной области памяти был равен сумме размеров объединяемых сегментов. Смещение внутри каждого из объединяемых сегментов с атрибутом STACK корректируются так, чтобы последний байт (байт с наибольшим адресом) в каждом сегменте совпадал с последним байтом объединенной области, т.е. все сегменты стека начинаются с одного и того же базового адреса .



Тип объединения *AT* с конструкцией

AT <выражение>

позволяет программисту задать начальный адрес логического сегмента т.е. фиксировать сегмент в нужной области памяти. Значение выражения задает номер параграфа памяти, т.е. это значение загружается в соответствующий сегментный регистр. Например:

```
DISPLAY SEGMENT     AT 0B800H
```

База сегмента DISPLAY равна AT 0B8000H.

Тип объединения *COMMON* (общий) означает, что данный сегмент разделяет одинаковые ячейки памяти со всеми другими сегментами с тем же именем из других модулей.

Атрибут *MEMORY* действует аналогично атрибуту *COMMON*, но сегменты с данным типом объединения размещаются в памяти после всех других сегментов. В объединяемых модулях должен быть один сегмент с атрибутом *MEMORY* .

Тип выравнивания.

Атрибут типа выравнивания определяет границу (байт - BYTE, слово - WORD, параграф - PARA, страница - PAGE), на которой должен быть размещен логический сегмент. Ассемблер использует его для формирования в объектном файле служебной информации, используемой редактором связей для выравнивания сегментов. Границы выравнивания задают следующий начальный адрес сегмента:

1) BYTE - адрес любой - кончился один сегмент и сразу же начался другой.

2) WORD - адрес четный - все начинается с четного адреса, независимо от того, где закончился предыдущий адрес.

3) PARA - адрес кратен 16, 4 младших разряда нулевые.

4) PAGE - адрес кратен 256 - 8 начальных разрядов нулевые.

Имя класса.

При наличии этого атрибута редактор связей собирает вместе все области с одинаковыми именами классов. Этот атрибут просто указывает, что некоторые (уже объединенные) области должны быть размещены друг за другом в физической памяти. Атрибут "Имя класса" не обеспечивает адресацию областей при одном и том же содержимом сегментного регистра.

Все сегменты одного класса загружаются до загрузки сегментов другого класса. Само имя класса должно быть заключено в одиночные кавычки. Имена класса воспринимаются без учета вида регистра (верхний или нижний (прописная или строчная буква)), но это можно учитывать, если при ассемблировании использовать опции /ML или /MX или /NOIGNORECASE (отмена игнорирования).

Сегменты с одинаковыми именами класса принадлежат к одному классу и копируются в выполняемый файл в виде непрерывных блоков. Например:

```
DATAX           SEGMENT   'DATA
                .
                .
                .
DATAX           ENDS
TEXT1           SEGMENT   'CODE'
                .
                .
                .
TEXT1           ENDS
DATAY           SEGMENT   'DATA'
                .
                .
                .
DATAY       ENDS
```

Здесь два сегмента DATAX и DATAY имеют тип класса 'DATA', следовательно, оба копируются в выполняемый файл перед сегментом TEXT1.

Сегменты, для которых никакого имени класса в явном виде нет, имеют имя "нулевого класса" и будут записываться как непрерывные блоки вместе с другими сегментами, имеющими имя нулевого класса. Общий размер всех сегментов в классе не более 64 К.

Замечание.

Порядком, в котором компоновщик будет загружать, сегменты можно управлять, например, задав и ассемблировав фиктивный программный файл, содержащий пустые определения сегментов в том порядке, в котором нужно загрузить реальные сегменты. После ассемблирования этого файла его можно задать в качестве первого объектного файла при любом запуске компоновщика LINK. В этом случае компоновщик автоматически загрузит объединенные в классы сегменты в заданном порядке.

Директива GROUP.

Группа - это множество программируемых областей, объединенных так, что они используют общую базу. Группа представляет собой "объединение объединений", так как группа состоит из отдельных областей, каждая из которых построена из отдельных логических сегментов. Информация ассемблеру для объединения в группу нескольких областей содержится в директиве GROUP , имеющей формат:

Имя группы GROUP Имя сегмента, [Имя сегмента]…..

Директива GROUP ассоциирует указанное 'имя' группы с одним или более сегментами и переопределяет адреса всех меток и переменных, определенных в заданном сегменте таким образом, что они отсчитываются относительно начала данной группы, а не относительно начала сегмента, в котором они определены.
Указанное имя сегмента должно представлять собой имя какого-либо сегмента, определенного с помощью директивы SEGMENT или выражения SEG .

Директива GROUP не влияет на порядок загрузки сегментов той или иной группы. Порядок зависит от класса каждого из сегментов или о того порядка, в котором объектные модули заданы компоновщику.

Сегменты в группе не обязательно должны быть непрерывными. Не принадлежащие к определенной группе сегменты могут быть загружены между сегментами, принадлежащими группе. Единственное ограничение - смещение от первого байта в группе не более 64К.

Имена групп можно использовать с директивой ASSUME и в качестве префикса операнда с оператором переопределения сегмента (:).

В любом исходном файле имя группы не должно использоваться более чеи в одной директиве GROUP. Если в пределах отдельного исходного файла к одной группе принадлежат несколько сегментов, то имена всех этих сегментов должны быть заданы в одной и той же директиве GROUP. Например:

```
DGROUP    GROUP       ASEG, BSEG
          ASSUME    DS:DGROUP
ASEG      SEGMENT   BYTE PUBLIC  'DATA1'

SUM_A:
ASEG      ENDS
BSEG      SEGMENT   BYTE PUBLIC  'DATA2'

SUM_B:
BSEG      ENDS
CSEG      SEGMENT   BYTE PUBLIC  'DATA3'

SUM_C:
CSEG      ENDS
```

Порядок, в котором компоновщик LINK выполняет загрузку этих сегментов показан на рисунке ниже.

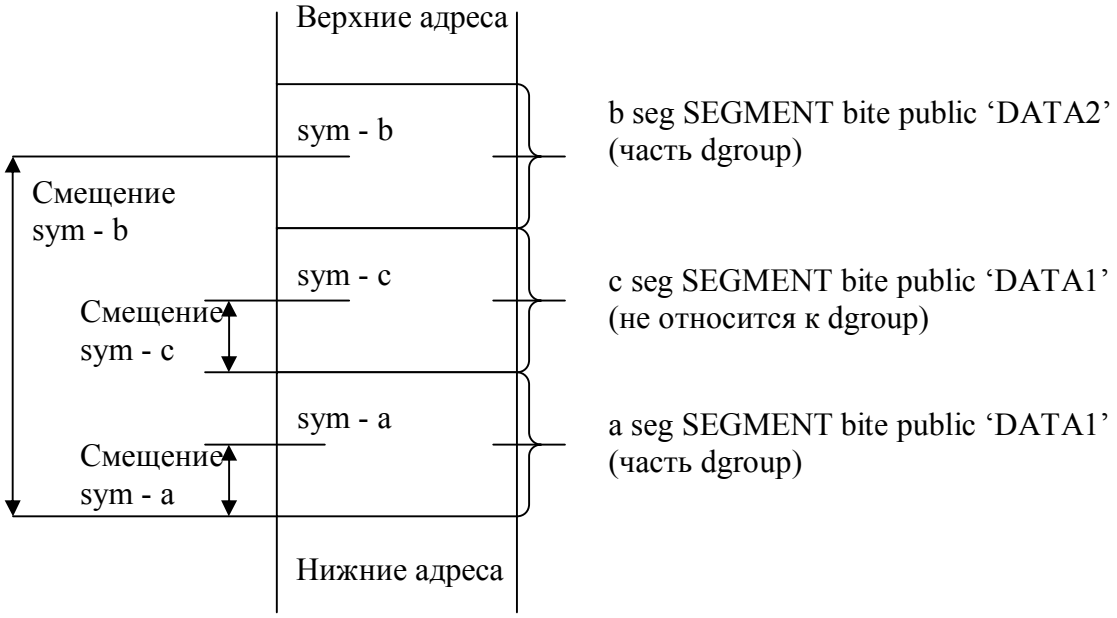


Рис. Порядок загрузки компоновщиком сегментов

Первым LINK загружает ASEG, т.к. он стоит первым в исходном файле, Затем - CSEG т.к. он имеет тот же тип класса, что и ASEG. Последним компоновщик загружает BSEG. В тоже время ASEG и BSEG объявлены как часть одной группы, несмотря на их размещение в памяти. Это означает, что смещение для символьных имен SUM_A и SUM_B отсчитывается от начала этой группы. Оно же является началом для сегмента ASEG. Смещение для имени SUM_C отсчитывается о начала сегмента CSEG.

1.3.Интерфейс ввода-вывода

1.3.1. Общие сведения: синхронный и асинхронный обмен, ввод-вывод с квитированием

Различают два варианта - синхронный и асинхронный обмен данными.

Синхронный обмен данными.

Синхронным обменом данными называется такой обмен данными, когда устройства ввода - вывода работают синхронно по отношению к МП. Синхронный обмен данными может быть реализован, если устройства являются достаточно быстродействующими.

На рисунке ниже приведена типичная схема портов ввода и вывода. Портам присвоен 8-разрядный адрес, по которому МП может отличить их от других портов. Хотя порт ввода и порт вывода имеют один и тот же адрес, однако они отличаются сигналами "Ввод" и "Вывод" на ШУ. При выполнении команд ввода - вывода на 8 младших разрядов ША подается адрес порта, указанный во втором байте команды. Специальная схема, реализующая операцию И (&), распознает адрес данного порта. Входы этой схемы подключены к 8 младшим линиям ША.

Синхронный ввод.

Порт ввода имеет 8-разрядный регистр, на который поступает информация от внешнего устройства. Выводы этого регистра подключены к ШД через 8 тристабильных формирователей. Сигналом разрешения чтения этих формирователей является результат логического умножения сигналов "Выборка" и управляющего сигнала "Ввод".

1.На шину адреса поступает адрес порта ввода. На выходе дешифратора (на рис. схема И) появляется активный сигнал ‘Выборка’(S).

2.На шине управления появляется активный сигнал ‘Ввод’.

3.На выходе схемы ‘И’ появляется активный сигнал, который подключает выходы Q7… Q0 порта ввода к линиям D7… D0 шины данных, т.е. на шине данных появился внешний мир (в это время МП защелкивает данные в аккумулятор). На шине данных данные из внешнего мира находятся до тех пор, пока активен сигнал ‘Ввод’.

МП снимает сигнал ‘Ввод’, и связь между выходами Q7… Q0 порта ввода . и шиной данных прерывается.

Порт ввода передает данные на шину данных только во время активного сигнала ‘Ввод’.

Синхронный вывод.

1.На шину адреса поступает адрес порта вывода.

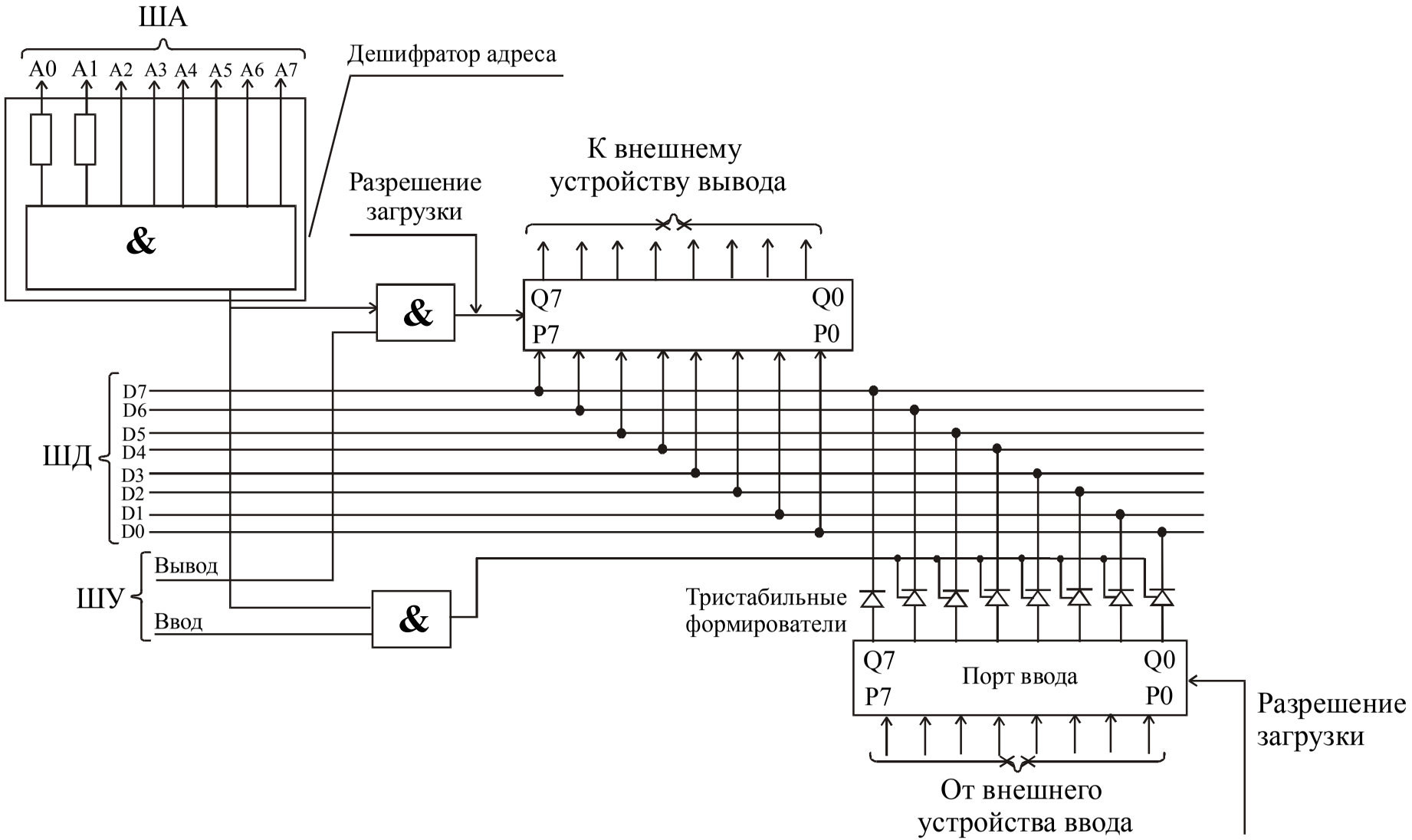
2.На выходе дешифратора (на рис. схема И) появляется активный сигнал ‘Выборка’ (S).

3.Из аккумулятора выставляется число на шину данных. Данные находятся на входе D7… D0 8-разрядного регистра-защелки.

4.На шину управления подается активный сигнал ‘Выход’, который поступает на схему ‘И’. В итоге формируется сигнал ‘Разрешение загрузки’.

Информация защелкивается и появляется на Q7… Q0 порта вывода.

Выведенные данные будут сохраняться на Q7… Q0 порта вывода, пока не будут выведены следующие данные.



Схемотехническая реализация синхронного обмна данными

Асинхронный (условный) ввод-вывод.

Этот способ обмена применяется в том случае, когда устройство работает асинхронно по отношению к процессору. Если, например, устройство ввода - АЦП, то после запуска АЦП на его выходе появляются данные для ввода в МП, но эти данные никак не связанные с работой МП. Поэтому возникает задача согласования моментов срабатывания внешнего устройства и МП. В противном случае при вводе передача может произойти в тот момент, когда данные еще не готовы, а при выводе - когда предыдущие данные еще не были восприняты внешним устройством.

Асинхронный обмен данными между МП и внешним устройством состоит из трех операций:

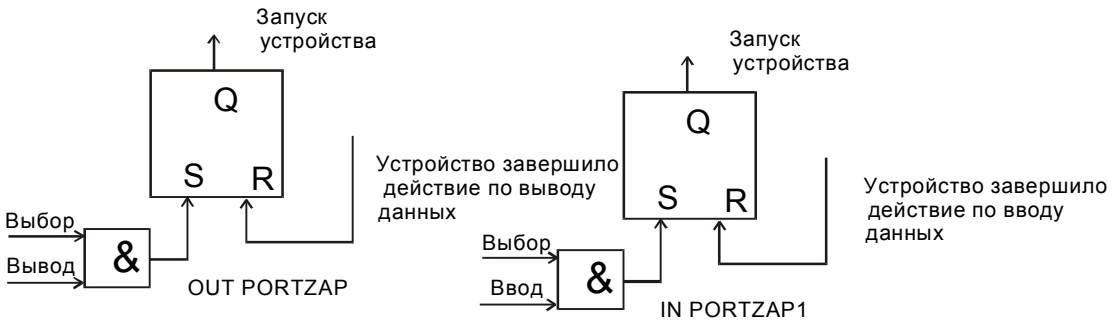
Запуск устройства.

Преобразование аналогового сигнала в цифровой. Мы постоянно спрашиваем устройство - закончило оно преобразование или нет.

Все готово и мы вводим данные.

Рассмотри случаи, когда запуск осуществляется МП в моменты времени, определяемые либо программой, либо самим внешним устройством.

1) Пусть запуск устройства осуществляет программа - внешнее устройство, получив сигнал от МП, выдает очередную порцию информации в порт ввода или принимает слово из порта вывода. Через некоторое время, заранее известное программисту и достаточное для завершения действий в устройстве, программа вновь может обращаться к порту. На рис. ниже показан вариант формирования программой сигналов запуска устройств с помощью триггеров (командой OUT PORTZAP для устройства вывода и командой IN PORTZAP1 для устройства ввода)



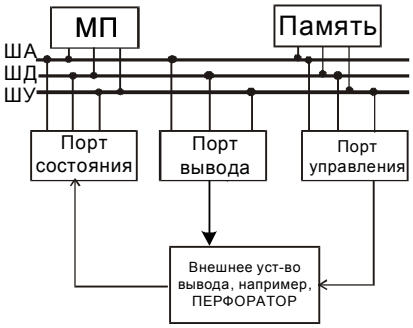
После завершения действия устройство подает сигнал сброса соответствующего триггера.

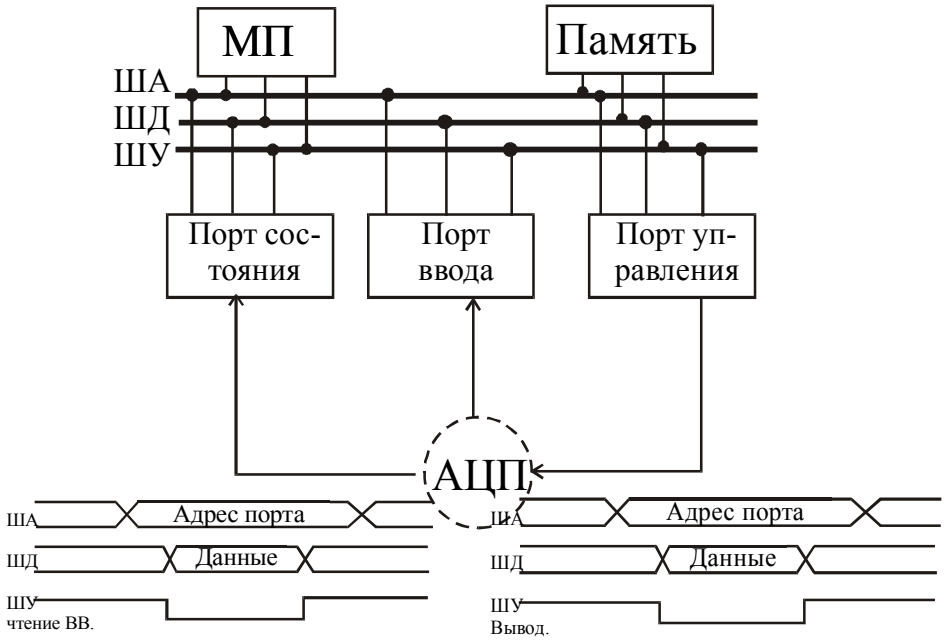
2) Пусть действие в устройстве запускается самим устройством, а программа при этом получает информацию о завершении действия в устройстве и о готовности к очередной передаче данных.

В этом случае программой используется *признак готовности порта*, устанавливаемый устройством, и *порт состояний*, разряды которого характеризуют условия, относящиеся к устройствам. В подпрограмме ввода-вывода через некоторый порт сначала выясняется готовность устройства путем ввода содержимого соответствующего порта состояния и проверки бита готовности.

```
Met:    IN      portst;
        ANI     maskvv      ; накладываем маску готовности
        JNZ     met     ; если устройство не готово, опрашиваем его еще раз
                IN      portvv; ввод данных
```

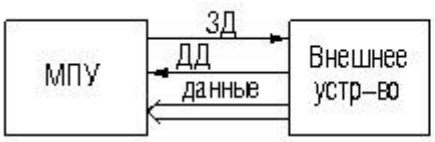
В реальной жизни устройство и запускается, и проверяется программой (см. на рисунках ниже).

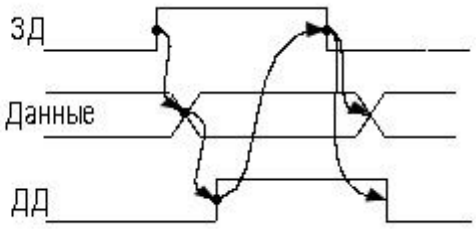




Ввод-вывод с квитированием.

Схема обмена и соответствующая временная диаграмма приведены на рис. ниже.





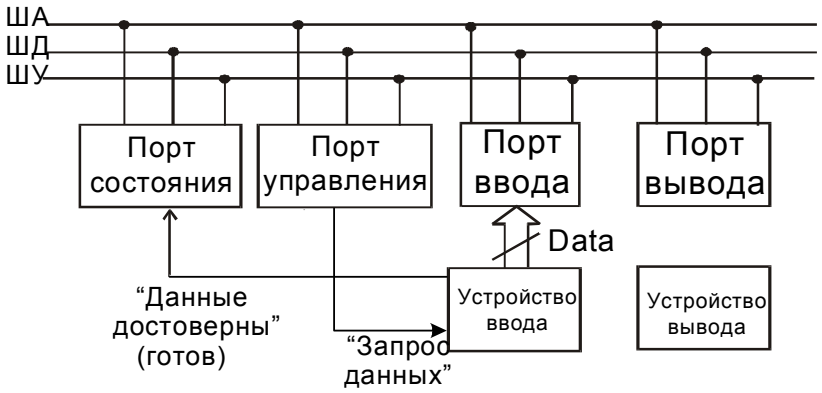
При таком вводе-выводе (см. рис. выше) приемник выставляет сигнал"Запрос данных", а *источник* информации выдает не только данные, но и сигнал "Данные достоверны" (ДД)

Приемник, получив сигнал "Данные достоверны", читает данные и снимает сигнал "Запрос данных".

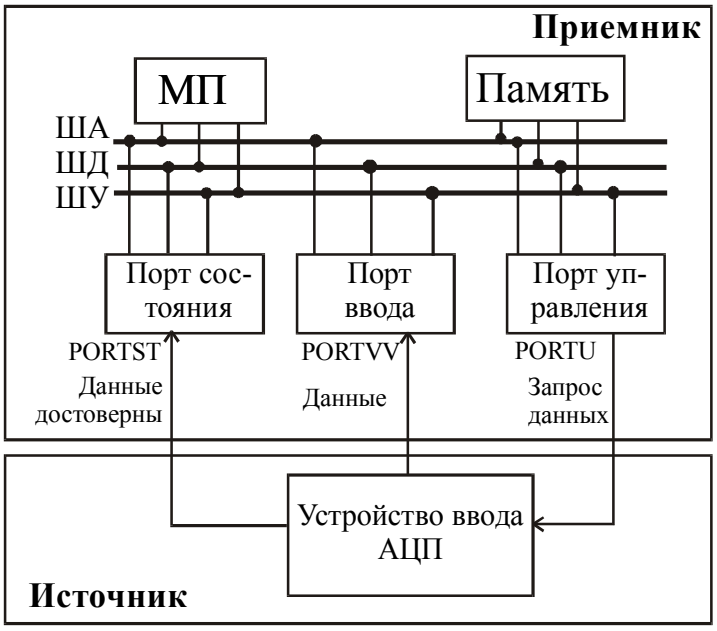
После снятия сигнала "Запрос данных" *источник* "понимает", что данные получены и снимает данные и сигнал "Данные достоверны".

Рассмотрим *ввод данных* с квитированием.

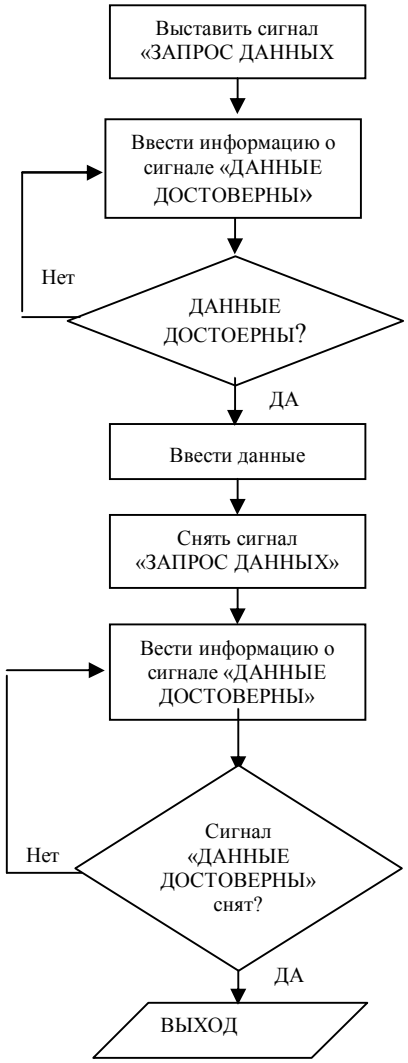
На рисунках ниже приведены более подробные схемы ввода данных с квитированием.



Ввод данных с квитированием



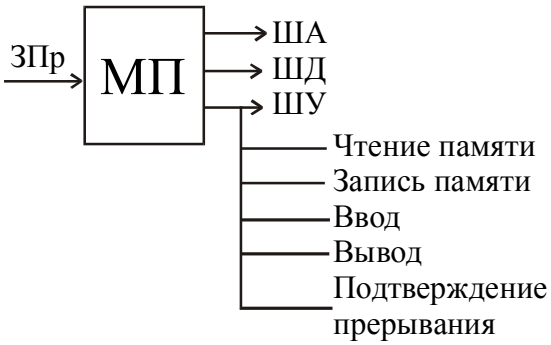
Упрощенная подпрограмма реализации ввода приведена ниже




```
        MVI   A, zapros
        OUT   portu

MET1:   IN    portst

        ANI   maskdd; накладывается
;маска
        JZ MET1; формируем признак
;0,переход на метку
        IN    portvv
        MOV C,A; убрали данные в
;порт A
        MVI   A, 0
        OUT   portu

MET2:   IN    portst

        ANI maskdd
        JNZ MET2

        RET
```

1.3.2. Ввод-вывод по прерыванию

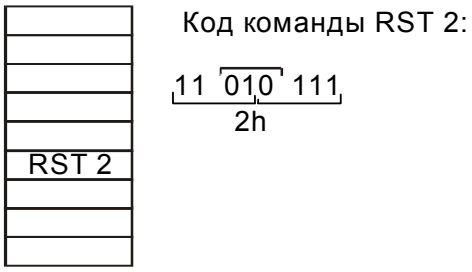
У МП есть сигнал Запрос Прерывания (ЗПр). Это некий эквивалент бита готовности, который МП проверяет в конце каждой выполненной команды.



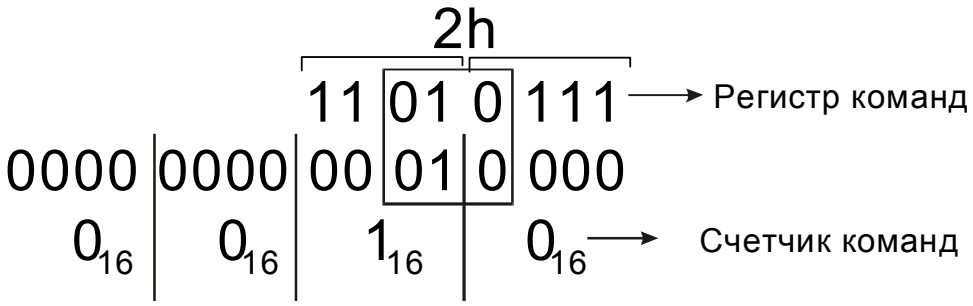
Получив сигнал ЗПр, МП завершает выполнение текущей команды и, определив номер устройства, выставившего ЗПр, передает управление подпрограмме обслуживания этого устройства. Точнее при получении сигнала ЗПр МП приостанавливает выполнение текущей программы, запоминает состояние внутренних регистров и переходит к подпрограмме, обмена информацией с запросившим устройством. Подпрограмма

обслуживания обработки прерывания заканчивается командой возврата, после чего продолжается выполнение прерванной программы.

Для процессора I8080 используется команда RST n, где n принимает значения от 0 до 7.

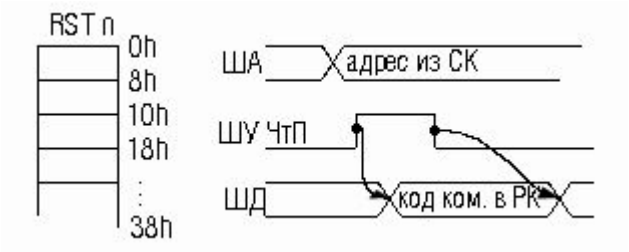


Дадим пояснения на примере выполнения команды RST 2. При чтении этой команды из памяти код команды поступает в регистр команд, затем этот код преобразуется в адрес, в котором все единицы (кроме содержащих номер прерывания разрядов, которые остаются без изменений) заменяются нулями (см. рис. ниже). Это адрес



записывается в счетчик команд и становится адресом, по которому МП уходит на подпрограмму. При этом значение адреса следующей команды прерываемой программы из СК сохраняется в стеке.

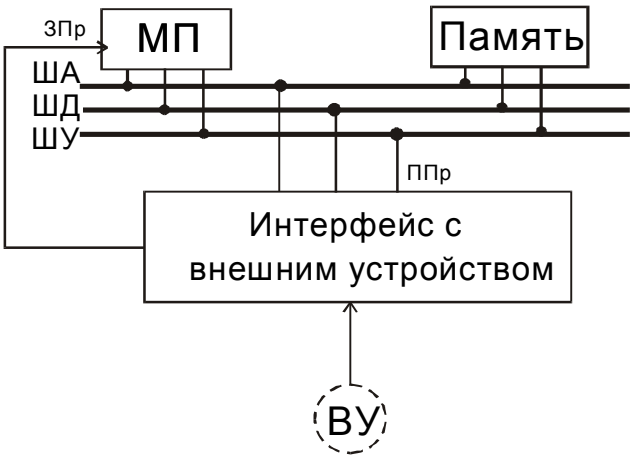
Начальные ячейки памяти, адреса которых формируются из кодов команд RST 0-RST 7, располагаются по следующим адресам: 0000h для RST 0, 0008h для RST 1, 0010h для RST 2, 0018h для RST 3, 0020h для RST 4, 0028h для RST 5, 0030h для RST 6, 0038h для RST 7.



Временная диаграмма чтения из памяти программного прерывания приведена на рисунке выше:

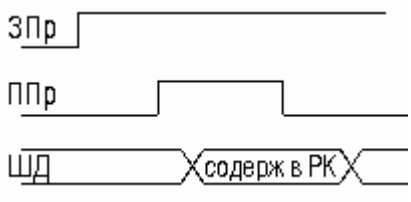
- На ША из СК подается адрес команды RST 2
- На ШУ подается команда "Чтение памяти"
- Читается содержимое ячейки памяти и код RST2 (11010111) с ШД защелкивается в регистр команд.

Рассмотрим способ передачи кода команды RST 2 в регистр команд

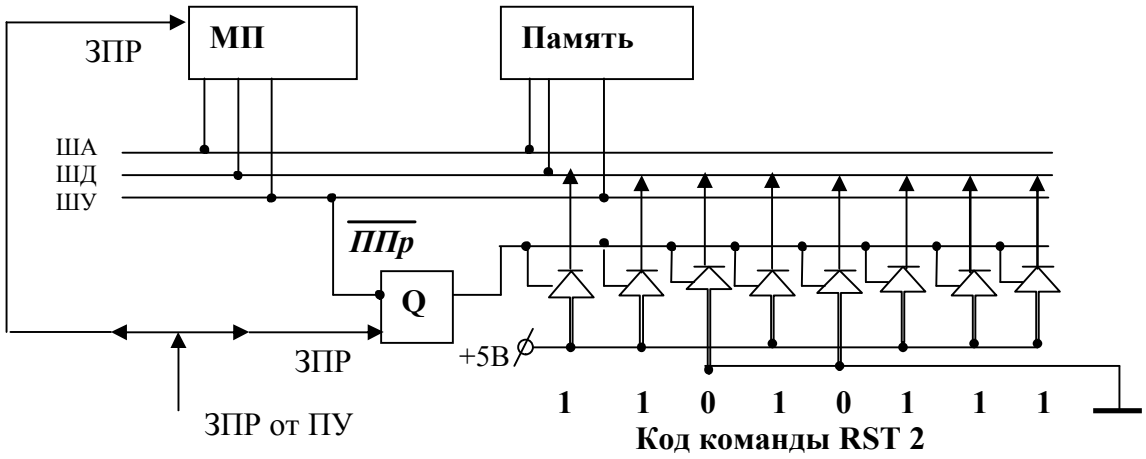


Алгоритм передачи кода команды в регистр команд.

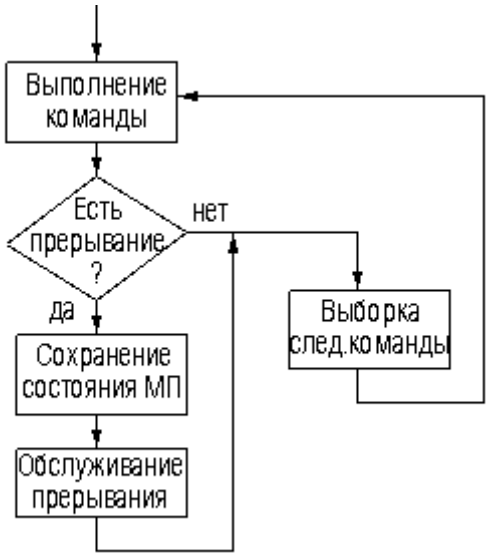
микропроцессора по сигналу ЗПр. При готовности устройства к обмену интерфейс внешнего устройства выдает сигнал ‘Устройство готово к обмену‘ и формирует сигнал ЗПр. МП, получив этот сигнал, формирует на шине управления ШУ сигнал подтверждения прерывания ППр, по которому содержимое с ШД микропроцессор записывает в свой регистр команд (см. рис. ниже). Задача состоит в том, чтобы в этот момент на ШД оказался код команды RST 2.



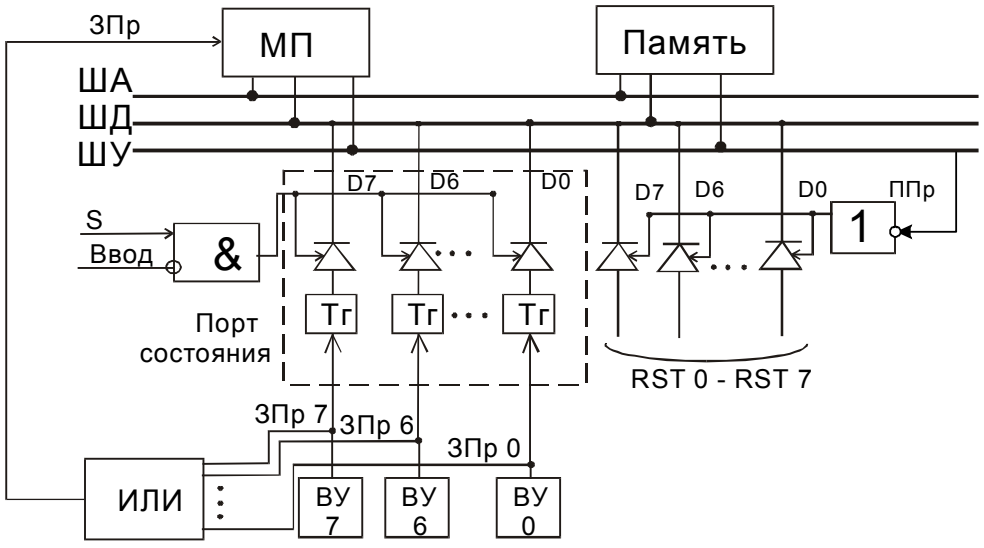
Вариант формирования на шине данных кода команды RST 2 приведен на рисунке ниже



Алгоритм обслуживания прерываний приведен на рисунке ниже.



Рассмотрим теперь случай, когда несколько внешних устройств ВУ0..ВУ7 могут выставлять ЗПр0..ЗПр7. Эти запросы прерываний объединяются по схеме ИЛИ и передаются в МП (см. рис. ниже). Следовательно, процессор перейдет на подпрограмму. Далее МП в этой подпрограмме идентифицирует устройство, выставившее ЗПр, например, следующим образом.

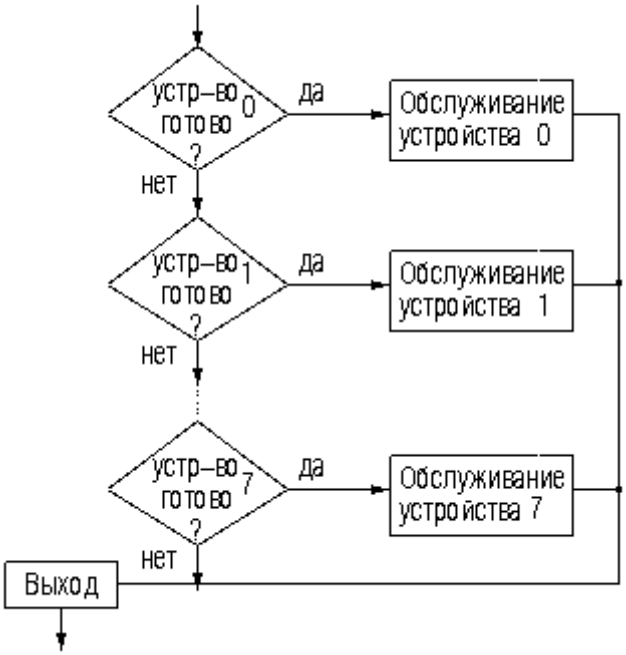


Запрос прерываний от нескольких внешних устройств

Все внешние устройства формируют бит готовности. Опрашивая порт состояния, МП узнает, какое из устройств нужно обслуживать, используя, например, программный поллинг.

Программный поллинг (опрос)

В подпрограмме обслуживания прерывания реализуется алгоритм, которой приведен ниже.



Достоинство программного поллинга заключается в простоте его реализации, почти не требующей дополнительных аппаратных средств. Основным недостатком программного поллинга является достаточно большое время, затрачиваемое на опрос состояния всех устройств, даже тех из них, которые не требуют обслуживания. Это приводит к потерям времени МП и замедлению его реакции на запросы прерывания.

Если программный поллинг не удовлетворяет, то используется *векторное прерывание.* Существуют 3 способа реализации векторного прерывания:

1) аппаратный поллинг
2) использование шифратора приоритетов
3) выдача первой команды обслуживания прерываний.

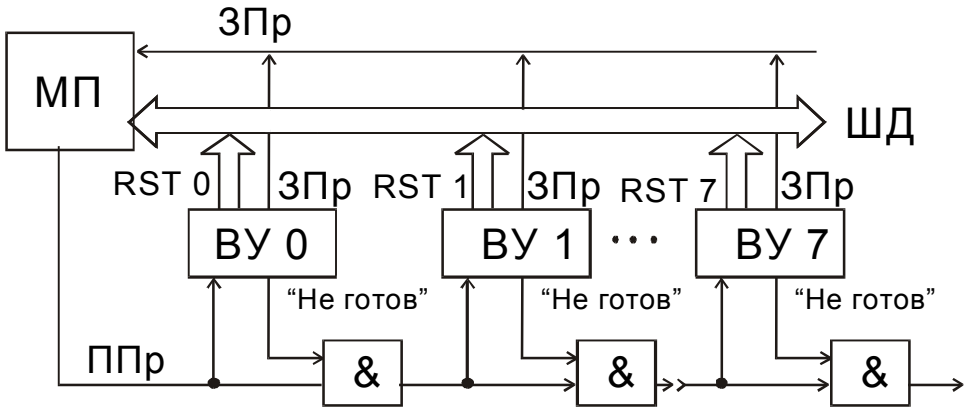
Рассмотрим каждый способ реализации в отдельности.

Аппаратный поллинг.

При аппаратном поллинге схемы прерываний внешних устройств присоединяются к МП в виде приоритетной цепочки таким образом, что МП может осуществлять автоматический опрос с целью идентификации прерывающего устройства (см. рис. ниже). Получив сигнал ЗПр, МП формирует сигнал на линии ППр, который последовательно проходит через все устройства. При прохождении сигнала ППР по цепочке проверяется состояние триггеров готовности устройств. Если устройство не требует прерывания, то сигнал ППр проходит в следующее устройство, пока не встретится прерывающее устройство. Следовательно, приоритетность устройств определяется их близостью к МП по линии ППр. Получив сигнал ППр, прерывающее устройство передает по шине данных свой адрес (вектор), который имеет однозначное соответствие с начальным адресом подпрограммы обслуживания прерывания данного устройства.

Достоинства: увеличена реакция системы, так как нет необходимости проверять бит готовности.

Недостатки: аппаратно (т.е. жестко) закреплена приоритетность устройства. - Больший приоритет имеет устройство, расположенное ближе к МП по линии ППр. Этот недостаток можно устранить путем использования шифратора приоритетов .

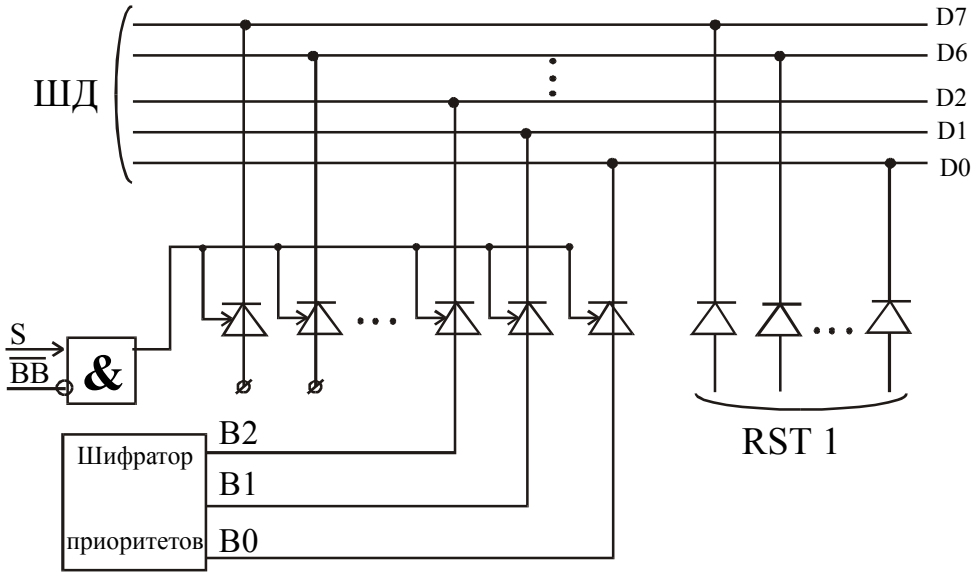


Схемотехника аппаратного поллинга.

Использование шифратора приоритетов.

В этой системе прерываний МП сообщается номер прерывающего устройства. Прерывание вызывает переход на фиксированную ячейку, где начинается главная программа обработки прерываний. Эта программа выводит данные из специального порта, содержащего номер устройства, выставившего ЗПр.

При наличии нескольких ЗПр управление передается устройству с более высоким приоритетом. Номер наиболее приоритетного устройства определяется с помощью шифратора приоритетов. На его вход подаются запросы прерываний ЗПр от отдельных устройств, входы которых пронумерованы, начиная с 0. Номер входа соответствует номеру подключенного устройства.



Схемотехника использования шифратора приоритетов

Пр0	000	I0	$b_0 = I_7 + \bar{I}_6(I_5 + \bar{I}_4(I_3 + \bar{I}_2 I_1))$
Пр1	001	I1	$b_1 = I_7 + I_6 + \bar{I}_5\bar{I}_4(I_3 + I_2)$
			$b_2 = I_7 + I_6 + I_5 + I_4$

Пр2	010	I2	При такой прошивке приоритетов - наивысший приоритет у I7 наименьший - I0.
Пр3	011	I3	
Пр4	100	I4	
Пр5	101	I5	
Пр6	110	I6	
Пр7	111	I7	

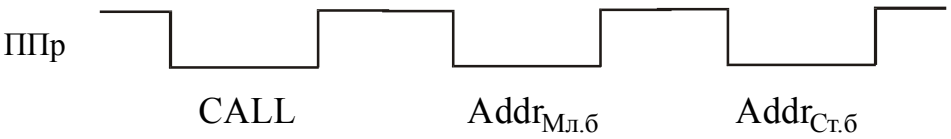
Пример. Пусть поступили следующие запросы прерывания - ЗПр5, ЗПр3.,ЗПр1.Тогда,

b2	b1	b0
1	0	1

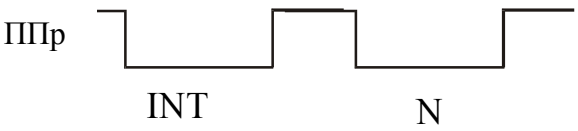
Получаем код команды RST5.

Выдача первой команды обслуживания прерываний.

Кроме команды RST n е VG Ш8080 есть еще команда CALL $addr_{16}$. В этом случае сигнал ППр будет выставляться 3 раза, т.к. команда CALL $addr_{16}$ трехбайтная. В первый раз передается собственно код команды CALL, второй и третий раз передается 16-тиразрядный адрес (см. рис. ниже).



У МП I8086 есть команда INT n, n=0…255. Эта команда 2-х байтовая. Для ввода этой команды сигнал ППр будет выставляться два раза (см. рис. ниже).



Далее выполняется введенная команда INT n (или CALL addr16)

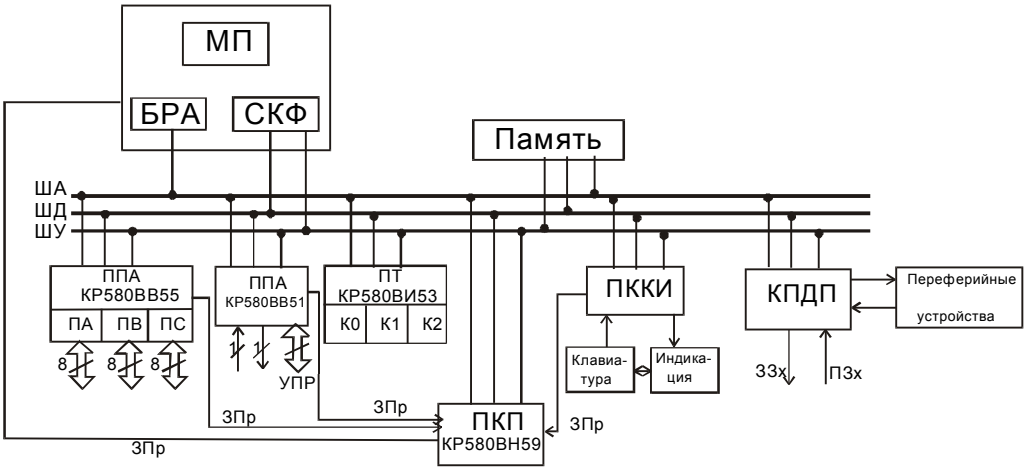
1.3.3. Ввод-вывод с использованием прямого доступа к памяти

Если нужно ввести массив с высокоскоростного устройства, например, с быстродействующего АЦП то может оказаться, что время генерирования байтов данных сравнимо с временем выполнения подпрограммы прерывания. В этом случае МП может не успеть за устройством ввода. В подобных случаях применяется ввод-вывод с прямым доступом к памяти (ПДП) с помощью контроллера ПДП, т.е. обмен данными между внешним устройством и памятью выполняется без участия МП.

Обмен в режиме ПДП выполняется, как правило, под управлением специальной микросхемы, называемой *контроллером ПДП*. Начальный адрес ячейки обмена в памяти и количество передаваемых байтов заранее программно записывается в контроллер ПДП. При возникновении необходимости обмена в режиме ПДП устройство ввода-вывода выдает сигнал «Запрос ПДП» на вход контроллера ПДП. Если данному устройству разрешена передача данных в режиме ПДП, то контроллер ПДП после получения сигнала «Запрос ПДП» подает на МП сигнал «Запрос захвата» (ЗЗх). В ответ на этот сигнал МП приостанавливает свою работу и отключается от системной магистрали (т.е. от ША, ШД, ШУ), выдавая контроллеру сигнал «Подтверждение захвата» (ПЗх). Получив сигнал «Подтверждение захвата», котроллер ПДП выдает устройству ввода-вывода сигнал «Подтверждение ПДП». Затем он организует обмен данными между памятью и устройством ввода-вывода путем подачи на ША адресов ячеек памяти, а на ШУ управляющих сигналов Запись (Чтение) и Ввод (Вывод). По окончании передачи контроллер ПДП снимает сигнал «Запрос захвата» и МП продолжает свою работу.

1.3.4. Микропроцессорный комплект семейства КР580

Неполный состав микропроцессорного комплекта КР580 приведен на рисунке ниже.



Состав МПК КР580

БРА – буфер адреса.

СКФ – системный контроллер и шинный формирователь.

ППА - программируемы параллельный адаптер - имеет три восьмиразрядных порта ввода - вывода.

ПСА - программируемый последовательный адаптер - осуществляет последовательный обмен данными.

ПТ - программируемый таймер - счетчик, на базе которого можно построить интерфейс частотного датчика, подсчитать частоту или период. Имеется возможность

выдавать программируемые частоты, подсчитывать входные события. В каждом канале есть 16-ти разрядный вычитающий счетчик.

ПКП - программируемый контроллер прерываний - принимает запросы от внешних устройств и сообщает процессору о том, что поступил запрос прерывания и выдает адрес подпрограммы обслуживания прерываний от соответствующего устройства .

ПККИ - программируемый контроллер клавиатуры и индикации.

КПДП - контроллер прямого доступа к памяти.

Обобщенная структура интерфейсной БИС.

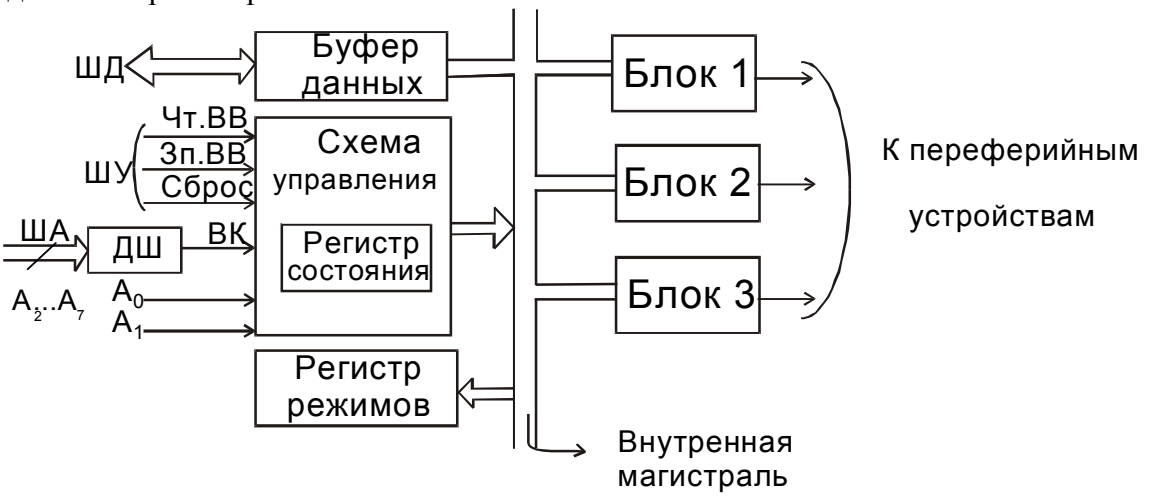
Обобщенная схема интерфейсной БИС приведена на рисунке ниже.

Рассмотрим назначение основных элементов интерфейсной БИС.

Буфер данных - согласует интерфейсную БИС с ШД МП-системы.

Схема управления. - управляет работой всей БИС. Она содержит регистры состояний, в которых проверяется сигнал готовности (готов / не готов)

Регистр режимов - в него записываются кодовые слова, которые настраивают БИС на определенный режим работы.



Обобщенная схема интерфейсной БИС

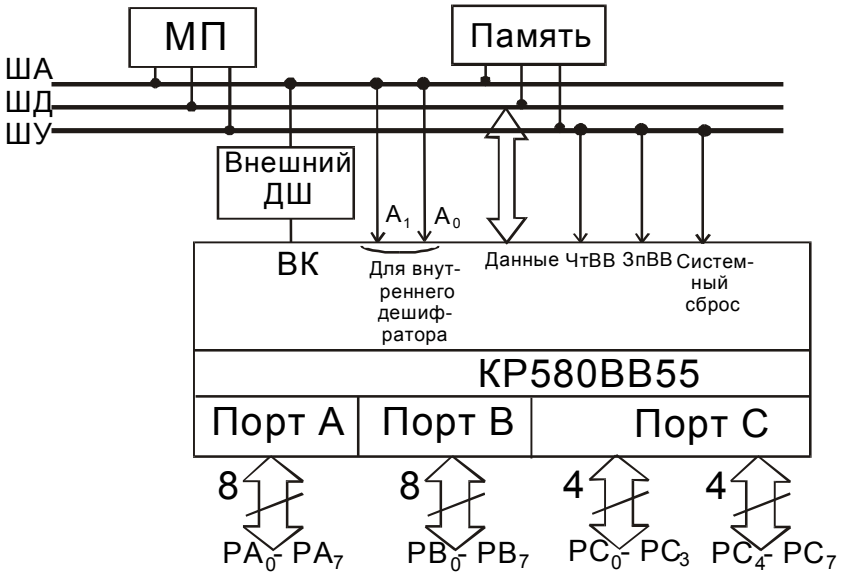
ДШ – дешифратор адреса.

Операционные блоки (или просто блоки)- в них собственно и выполняются действия по сопряжению с периферийными устройствами. Состав операционных блоков (блоки 1…N) определяются конкретным значением БИС. В параллельном адаптере операционными блоками являются буферные регистры, работающие в режиме входного порта, выходного порта и порта ввода-вывода. В последовательном адаптере к магистрали подключены сдвигающие регистры, преобразующие параллельный формат данных на внутренней магистрали в последовательный на выходной линии адаптера и наоборот. Формирователь временных интервалов в качестве операционных блоков содержит управляемые пересчетные схемы, а контроллер прерывания – регистры, фиксаторы и блоки логической обработки. Операционные блоки контроллера прямого доступа представляют собой счетчики для формирования массива адресов оперативной памяти с дополнительными схемами генерации управляющих сигналов для обмена.

Ниже рассмотрены некоторые интерфейсные БИС.

1.3.5. Программируемый параллельный адаптер КР580ВВ55

Программируемый параллельный адаптер (ППА) обеспечивает обмен информацией между МП-системой и периферийными устройствами в параллельном коде. Он имеет три многорежимных порта ввода-вывода, которые обозначены как A, B и C.



Программируемый параллельный адаптер

Адресация к портам следующая:

A1	A0	
0	0	порт A
0	1	порт B
1	0	порт C
1	1	регистр управляющего слова.

Порты A,B,C можно запрограммировать в одном из 3-х режимов:

Обозначение режима	Используемые порты	Реализуемый обмен данными
Режим 0	A,B,C,	синхронный обмен данными
Режим 1	A,B	ввод/вывод с квитированием
Режим 2	A	двунаправленный ввод/вывод с квитированием.

Если порты A и B запрограммированы в режим 1, то порт C используется для формирования служебных сигналов (см. рис. ниже), а также используется в качестве порта состояний.

Форматы управляющих слов для ППА.

ППА управляется двумя управляющими словами.

Формат управляющего слова, задающего режимы работы портов ППА (первое управляющее слово).

D7	D6	D5	D4	D3	D2	D1	D0

D7 –1 –всегда в управляющем слове задание режима.
D6, D5 - задают режим порта А
0 0- режим 0
0 1 - режим 1
1 X - режим 2
D4 - направление обмена порта А
0 - вывод
1- ввод
D3 - направление обмена старшего полубайта порта С
0- вывод
1 - ввод
D2 - режим работы порта В
0 - режим 0
1- режим 1
D1 направление обмена порта В
0 - вывод
1- ввод
D0 - направление обмена младшего полубайта порта С.
0- вывод
1 - ввод

Пример. Допустим, что надо запрограммировать порт А на ввод в режиме 0, порт В на вывод в режиме 0, PC4…PC7 – на вывод в режиме 0, PC0…PC3 – на ввод в режиме 0.

D7 D6 D5 D4 D3 D2 D1 D0

1 0 0 1 0 0 0 1

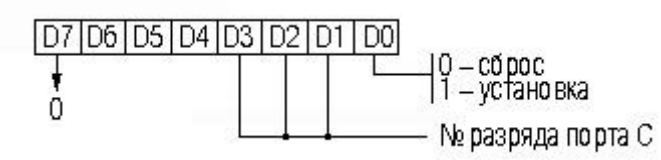
91h

Программирование ВВ55

MVI A, 91h; управляющее слово →аккумулятор
OUT portRUS; управляющее слово из аккумулятора → регистр управляющего слова

Формат управляющего слова установки-сброса разрядов порта С (второе управляющее слово)

Осуществляет установку и сброс разрядов порта С. Можно побитно управлять разрядами порта С.

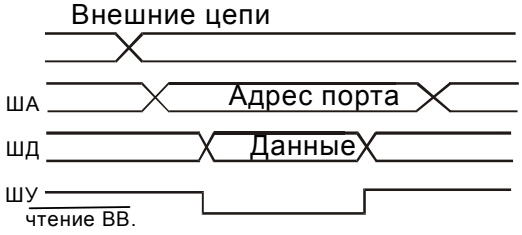


Пример: установить второй разряд порта С в единицу.

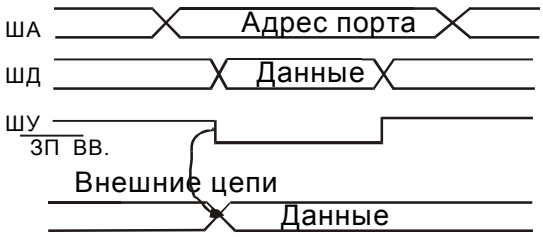
D7 D6 D5 D4 D3 D2 D1 D0
0 x x x | 0 1 0 1

05h

Временные диаграммы 0-го и 1-го режима.



Ввод в режиме 0.



Вывод в режиме 0

Из временной диаграммы ввода в режиме 0 видно, что в этом режиме порт не фиксирует вводимую информацию (т.е. выполняет только функцию вентиля). Следовательно, вводимое слово должно присутствовать на входе порта во время действия активного сигнала ЧтВВ.

При выводе информация фиксируется на выходных линиях порта до ее замены в следующем цикле вывода (т.е. порт выполняет функции простого буферного регистра).

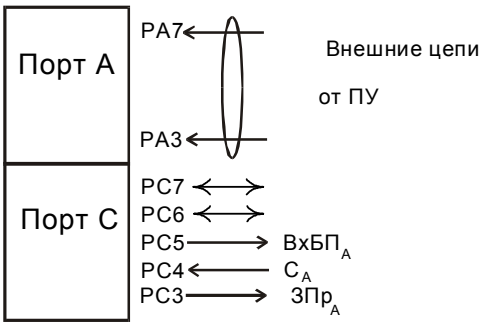
Режим 1 (ввод или вывод с квитированием) обеспечивает возможность организации однонаправленной асинхронной передачи информации между МП-системой и периферийным устройством. Временная диаграмма ввода в режиме 1 для портов А и В приведены на рисунке ниже, где используются следующие обозначения сигналов:

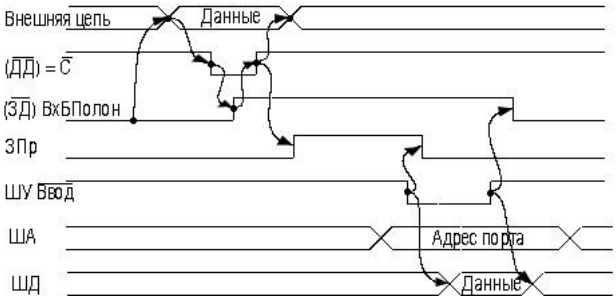
С - Строб приема;

ВхБП Входной буфер полон;

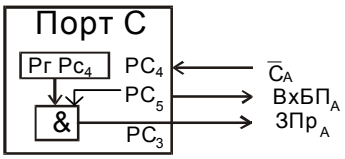
ЗПр Запрос прерывания;

При вводе данных в режиме 1 сигнал «Строб» выполняет функции сигнала «Данные достоверны», а сигнал «Входной буфер полон» выполняет функции сигнала «Запрос данных».





Формирование служебных сигналов с помощью старшего и младшего полубайтов порта С приведено на рисунке ниже.



Если РгРС4 сбросить в 0, то будут замаскированы аппаратные сигналы запроса прерывания порта А (ЗПрА)

РгРС4 - запрещает / разрешает прерывания от порта А при вводе.

РгРС6 - запрещает / разрешает прерывания от порта А при выводе.

Ниже приведен формат слова состояния ППА (читается по адресу порта С) для случая ввода данных в режиме 1 через порты А и В.

D7	D6	D5	D4	D3	D2	D1	D0

D7,D6 - резервные разряды, которые могут быть использованы для ввода / вывода информации

D5 - Входной буфер порта А полон

D4 - РгРС4

D3 - Запрос прерывания от порта А

D2 - РгРС2

D1 - Входной буфер порта В полон

D0 - Запрос прерывания от порта В.

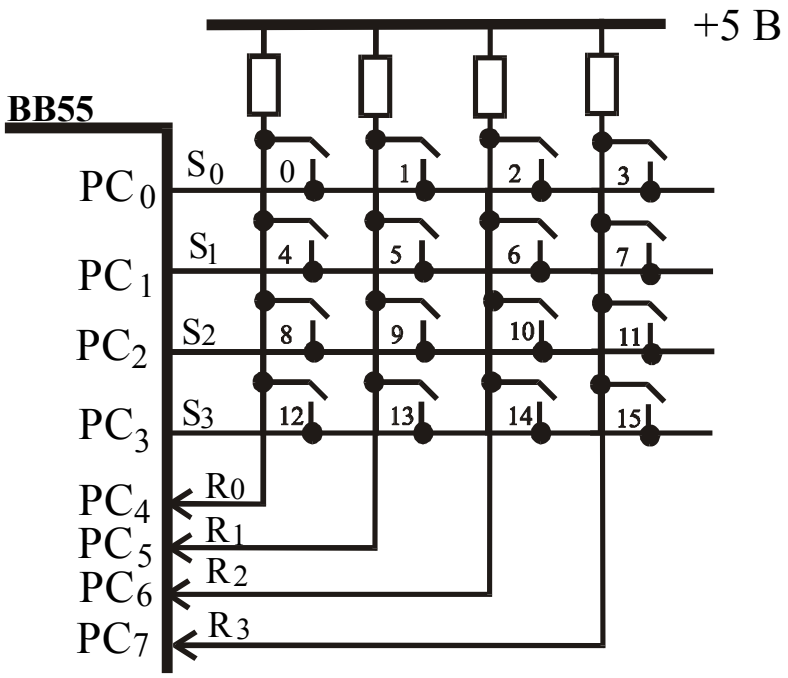
1.3.6. Интерфейс клавиатуры и индикации

Интерфейс клавиатуры и индикации рассмотрим на примере клавишного пульта ввода, содержащего для конкретности матрицу из 4х4 ключей и цифровой индикатор. В качестве интерфейсной схемы используется программируемый параллельный адаптер (ППА) КР580ВВ55.

Ключи в матрице располагаются так, как показано на рисунке ниже

На рисунке ниже приведена схема

Рассмотрим, как МП-система определяет, какая клавиша нажата, на примере клавиатуры 4х4.



Пример схемотехнической реализации клавиатуры 4х4.

Согласно приведенному выше рисунку, порты ППА надо запрограммировать следующим образом:

Порт С	младший полубайт - на вывод старший полубайт - на ввод
Порт А Порт В	В режиме 0

Из приведенного выше рисунка видно, что один вывод каждой клавиши соединен с одной из входных линий R0, R1, R2, R3, а другой – с одной из выходных линий S0, S1, S2, S3. Если на все выходные линии S0, S1, S2, S3 подавать высокий уровень 1, на всех входных линиях R0, R1, R2, R3 всегда будет уровень 1, вне зависимости от того есть ли нажатая клавиша или нет (см. табл. ниже).

S0	S1	S2	S3
1	1	1	1
R0	R1	R2	R3
1	1	1	1

Чтобы обнаружить нажатую клавишу, надо руководствоваться следующим правилом. В каждый момент времени только одна из выходных линий S0, S1, S2, S3 (т.е.

один из разрядов PC0..PC3 порта С ППА) может иметь уровень 0, а остальные должны иметь уровень 1. В то время как единственная выходная линия имеет уровень 0, входные линии R0, R1, R2, R3 проверяются МП-системой путем ввода разрядов PC4..PC7 порта С. Если при этом есть нажатая клавиша, то соответствующая ей выходная линия будет иметь уровень 0

Пока не нажата ни одна клавиша, везде на выходных линиях R0, R1, R2, R3 сохраняется высокий уровень напряжения.

Пусть в первой строке матрицы появился нулевой уровень напряжения, т.е.

S0	S1	S2	S3
0	1	1	1

Если при этом будет нажата клавиша в первом столбце матрицы, тогда на входы PC4-PC7 порта С ППА будет поступать комбинация сигналов

R0	R1	R2	R3
0	1	1	1

Следовательно, нажатая клавиша дает уровень 0 на соответствующей выходной линии R0 R1 R 2 R3.

номер клавиши	R0	R1	R2	R3
0	0	1	1	1
1	1	0	1	1
2	1	1	0	1
3	1	1	1	0

Аналогично при нулевом уровне во второй строке матрицы

S0	S1	S2	S3
1	0	1	1

можно определить нажатые клавиши второго столбца матрицы

номер клавиши	R0	R1	R2	R3
4	0	1	1	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Аналогично определяются номера других нажатых клавиш.

Сигналы уровня 0 поочередно выдаются на выходные линии S0, S1, S2, S3 т.е.

S0	S1	S2	S3
0	1	1	1
1	0	1	1

1	1	0	1
1	1	1	0

.Это постоянно делается в цикле.

Номер нажатой клавиши определяется по формуле:

$$N_{кл}=4i+j$$

i- номер строки матрицы

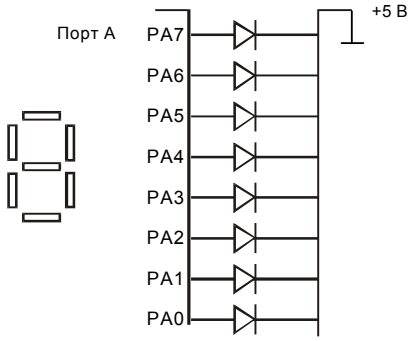
j-номер столбца матрицы

Сделаем еще одно важное замечание. Необходимо исключить дребезг клавиш. Для этого одна и та же клавиша опрашивается несколько раз (от 50 до 100). Если все опросы подтвердились, значит, клавиша действительно была нажата.

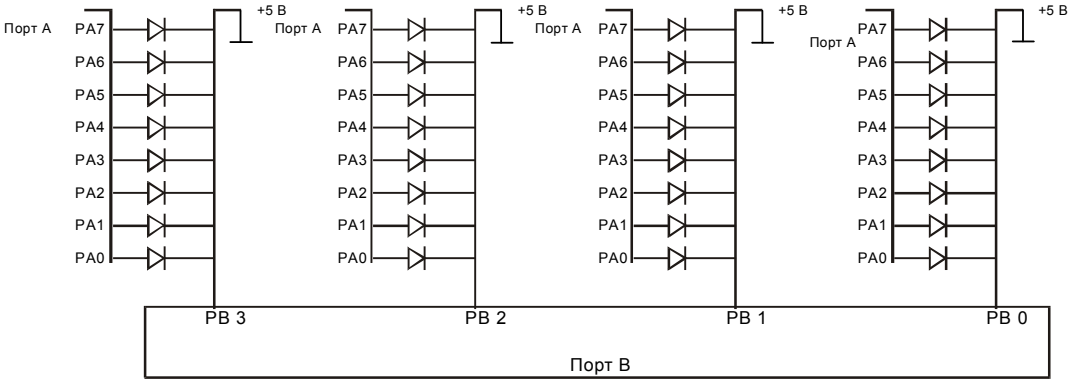
Индикация.

Индикация бывает двух типов: *статическая* и *динамическая*.

. При статической индикации на каждый элемент требуется отдельный порт(см. рис. ниже).



При динамической индикации возможное число подключаемых элементов возрастает до 8. Это достигается за счет параллельного подключения выводов всех 8-ми индикаторов к порту А. Еще один вывод подключается к порту В, как показано на рисунке ниже. При таком подключении через порт А будет осуществляться управление выводимой информацией, а через порт В - выбор светового индикатора, на который будет выводиться информация. Если подавать питание с большой частотой, то можно добиться одновременной индикации всех элементов.



1.3.7. Программируемый контроллер прерываний КР580ВН59. Пример применения

Напомним, что *система прерываний* — это совокупность программных и аппаратных средств, реализующих *механизм прерываний.*

Аппаратные средства системы прерываний включают выводы микропроцессора:

INTR — вывод для входного сигнала внешнего прерывания. На этот вход поступает выходной сигнал от микросхемы контроллера прерываний КР580ВН59;

INTA — вывод микропроцессора для выходного сигнала подтверждения получения сигнала прерывания микропроцессором. Этот выходной сигнал поступает на одноименный вход INTA микросхемы контроллера прерываний КР580ВН59;

NMI — вывод микропроцессора для входного сигнала немаскируемого прерывания.

Кроме того, аппаратные средства системы прерываний содержат микросхему программируемого контроллера прерываний КР580ВН59. Она предназначена для фиксирования сигналов прерываний от восьми различных внешних устройств. В силу ее важной роли при работе всей вычислительной системы, мы ее подробно рассмотрим ниже;

Итак, центральное место в схеме обработки аппаратных прерываний занимает программируемый контроллер прерываний (ПКП), например, микросхема КР580ВН59 (или КР580ВН59А), позволяющий организовать богатую систему обслуживания прерываний. Эта микросхема может обрабатывать запросы от восьми источников внешних прерываний. Перечислим функции, выполняемые микросхемой контроллера прерываний:

фиксирование запросов на обработку прерывания от восьми источников, формирование единого запроса на прерывание и подача его на вход INTR микропроцессора;

формирование номера вектора прерывания и выдача его на шину данных;

организация приоритетной обработки прерываний;

запрещение (маскирование) прерываний с определенными номерами.

На рисунке ниже показано схематическое представление внутренней структуры и физических выводов микросхемы КР580ВН59.

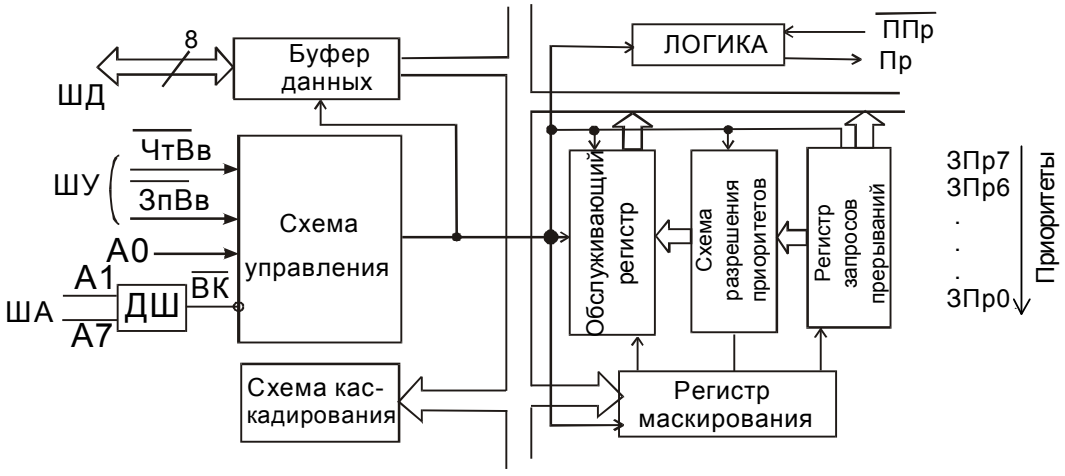


Схема ПКП КР580ВН59

Рассмотрим назначение основных структурных компонентов контроллера прерываний

регистр запросов на прерывания (обозначение в литературе IRR - Interrupt Request Register) - восьмиразрядный регистр, фиксирующий поступление сигнала на один из

входов ЗПр0-ЗПр7. Фиксация выражается в установке соответствующего бита в единичное состояние;

регистр маскирования прерываний (или IMR - Interrupt Mask Register — восьмиразрядный регистр, с помощью которого можно запретить обработку запросов на прерывания, поступающих на соответствующие входы (уровни) ЗПр0-ЗПр7. Для запрещения (маскирования) определенных уровней прерываний необходимо установить соответствующие биты регистра IMR. Эта операция осуществляется путем программирования порта 21h;

регистр обслуживаемых прерываний ISR (Interrupt Service Register) — восьмиразрядный регистр, единичное состояние разрядов которого показывает, прерывания каких уровней обрабатываются в данный момент в микропроцессоре;

схема разрешения приоритетов — функцией данного блока является разрешение конфликта при одновременном поступлении запросов на входы ЗПр0-ЗПр7;

схема управления — основной функцией данного блока является организация информационного обмена контроллера прерываний и микропроцессора через шину данных.

Рассмотрим возможное прохождение и обработку сигнала прерывания от некоторого внешнего устройства.

1) Контроллер получает запрос на прерывания.
2) Этот запрос фиксируется в регистре запросов прерываний.
3) Схема разрешения приоритетов выявляет запрос на прерывание с максимальным приоритетом и пропускает его на вход регистра обслуживания прерываний.

4) Сигнал поступает на схему управления, которая формирует сигнал на выводе Пр . Этот вывод связан с входом микропроцессора INTR. Таким образом, сигнал на входе контроллера достигает микропроцессора. При поступлении сигнала на вход INTR в микропроцессоре происходят следующие процессы:

1. Анализируется флаг IF. Единичное состояние этого флага говорит о том, что аппаратные прерывания разрешены, нулевое — запрещены.

2. Если прерывания запрещены, то запрос на прерывание «повисает» до момента установки IF в единицу.

3. Если прерывания разрешены, микропроцессор выполняет следующие действия:

сбрасывает флаг IF в ноль;

формирует сигнал подтверждения прерывания на выводе микропроцессора INTA. Этот вывод микропроцессора замкнут на вывод ППр контроллера.

5) Данный вывод внутри контроллера прерываний замкнут на его схему, которая выполняет сразу несколько действий при поступлении этого сигнала:

1. Сбрасывает бит в регистре запросов прерываний, соответствующий уровню поступившего запроса прерывания.

2. Устанавливает в регистре обслуживания прерываний 1 в разряде, соответствующем поступившему запросу прерываний, тем самым, фиксируя факт обработки прерывания в микропроцессоре.

3. Формирует с помощью блока управления номер вектора прерывания, значение которого формируется в буфере данных и далее поступает на выводы контроллера D0.. .D7. Выводы D0.. ,D7 замкнуты на шину данных, по которой номер вектора поступает в микропроцессор. В микропроцессоре этот номер используется для вызова соответствующей процедуры обработки прерывания.

Очень важный момент связан с процессом завершения обработки прерывания. Проблема здесь состоит в следующем. После принятия микропроцессором запроса на обслуживание прерывания в контроллере устанавливается бит в регистре ISR, номер этого бита соответствует уровню прерывания. Установка бита с данным номером блокирует все прерывания уровня, начиная с текущего, и менее приоритетные Если процедура прерывания закончит свою работу, то она сама должна этот бит сбросить, иначе все прерывания этого уровня и менее приоритетные будут игнорироваться. Для осуществления такого сброса необходимо послать код 20h в порт 20h.

MVI A,20h

OUT 20h

Другой не менее интересный момент заключается в том, что микропроцессор при принятии к обработке запроса на прерывание сбросил флаг IF в ноль, тем самым запретив все последующие аппаратные прерывания. Этим обстоятельством программист может пользоваться по своему усмотрению. Поскольку все запросы на прерывания с приоритетом, равным или меньшим текущему, будут запрещены в любом случае (это обусловлено логикой работы контроллера КР580ВН59), программист должен решить, насколько его замыслам могут помешать запросы на более приоритетные прерывания. Если это некритично, то лучше сразу, в начале процедуры обработки прерывания установить флаг IF в единицу, В большинстве случаев эту операцию нужно делать как можно раньше. Для установки флага IF в единицу в системе команд микропроцессора есть специальная команда, не имеющая операндов:

STI - разрешить аппаратные прерывания.

В процессе загрузки компьютера и в дальнейшем во время работы контроллер прерываний настраивается на работу в одном из режимов приоритетов

1.Жесткий приоритет (в литературе также встречается обозначение FNM (Fully Nested Mode) или режим *вложенных прерываний.*) В этом режиме каждому входу присваивается фиксированное значение приоритета, причем уровень ЗПр0 имеет наивысший приоритет, а ЗПр7 — наименьший. Приоритетность прерываний определяет их право на прерывание обработки менее приоритетного прерывания более приоритетным (при условии, конечно, что IF = 1).

2.Циклический приоритет. (В литературе также встречается обозначение ARM (Automatic Rotation Mode))— режим *циклической обработки прерываний.* В этом режиме значения приоритетов уровней прерываний также линейно упорядочены, но уже не фиксированным образом, а изменяются после обработки очередного прерывания по следующему принципу: значению приоритета последнего обслуженного прерывания присваивается наименьшее значение. Следующий по порядку уровень прерывания получает наивысшее значение, и поэтому при одновременном приходе запросов на прерывания от нескольких источников преимущество будет иметь этот уровень. Это дает возможность обеспечить «равноправие» при обработке прерываний.

Например. До поступления запроса на прерывания:

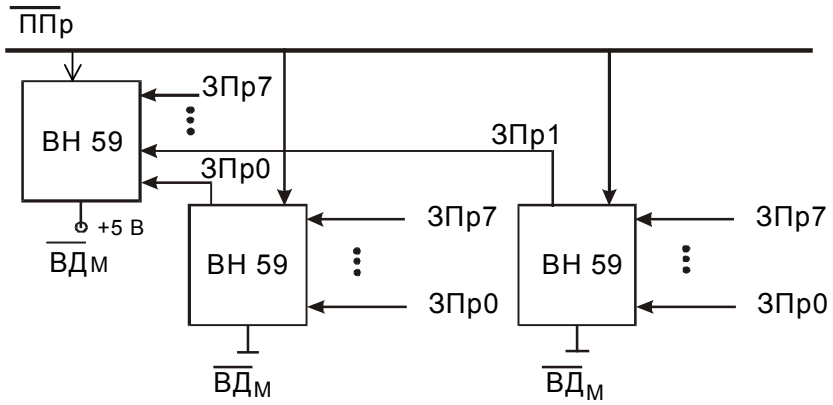
01234567

Поступил ЗПр3. После его выполнения очередь на обслуживание прерываний будет выглядеть следующим образом:

45670123

Как уже было сказано выше, указатели на процедуры обработки прерываний записаны в таблице векторов прерываний. Но как же МП определяет адрес каждой процедуры обработки? В памяти компьютера таблица векторов прерываний выглядит следующим образом. Имеется общий базовый адрес, от которого через равное количество ячеек (либо 4 - либо 8) находятся адреса соответствующих процедур обработки запросов прерываний. Базовый адрес задается двумя первыми начальными управляющими словами при начальной инициализации контроллера. Бывают ситуации, когда восьми входов ЗПр

контроллера недостаточно. Тогда систему прерываний реализуют каскадированием. К одному контроллеру, называемому ведущим, подключают до восьми ведомых контроллеров. Таким образом, при желании число входов ЗПр можно увеличить до 64.



Пример каскадирования с двумя ведомыми МС.

Как же здесь обслуживаются ЗПр? Ведущему (на его вход ВДм подается высокий уровень +5В) с помощью третьего начального управляющего слова сообщается о том, что к нему подключены ведомые (на их входы ВДм подается низкий уровень 0). При поступлении сигнала ЗПр ведущий знает, кто выставил запрос прерывания, и выдает кодовую комбинацию команды CALL и по шине каскадирования в соответствии с приоритетами сообщает, какой из ведомых должен выставить адрес addr16.

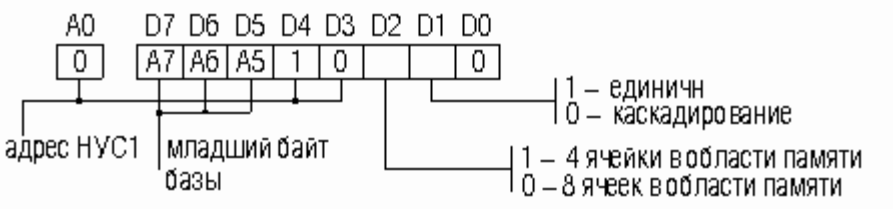
Программирование контроллера прерываний осуществляется через адресное пространство ввода-вывода посредством двух 8-битовых портов с адресами 20h и 21 h. Управление контроллером осуществляется путем посылки в определенной последовательности в эти порты управляющих слов, которые также управляют работой дешифраторов адресов. Управляющие слова делятся на две группы. Это начальные управляющие слова (НУС), задающие начальные установки работы ПКП и текущие управляющие слова (ТУС), позволяющие программировать ПКП во время его работы.

Начальные управляющие слова.

НУС 1.

Осуществляет сброс всех разрядов регистра запросов прерываний, очищает регистр маски, присваивает низший приоритет входу ЗПр7, сбрасывает триггеры обслуживающего регистра.

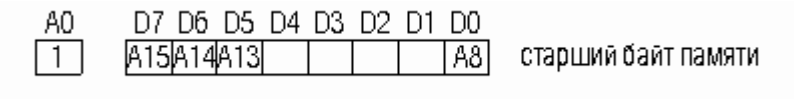
Формат НУС 1:



После НУС 1 нужно обязательно загружать НУС 2.

НУС 2.

Задает старший байт базового адреса подпрограмм обслуживания прерываний.



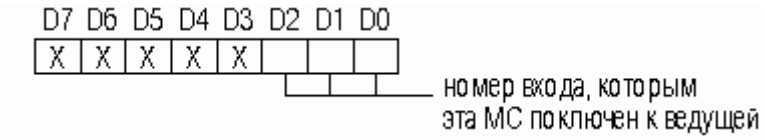
Если есть каскадирование, то за НУС 2 должно обязательно следовать НУС 3, которое является разным для ведущего и ведомого контроллеров.

Для ведущего: единицы устанавливаются в тех битах, к которым подключены ведомые.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	1	0	1

Для ведомого: в последних трех битах выставляется номер входа ЗПр ведущего, к которому он подключен.

Ведущий ПКП выставляет на шину каскадирования ту трехразрядную кодовую комбинацию, которая записана в D2 D1D0 третьего начального управляющего слова

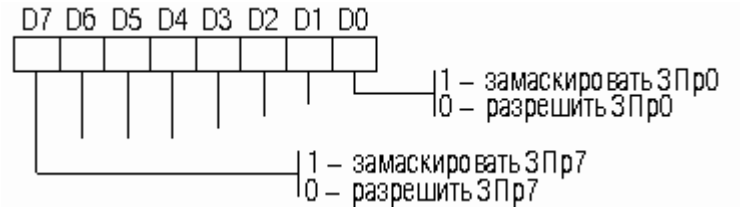


ведомого ПКП, и тот ПКП, у которого эта комбинация совпадает, выставляет соответствующий адрес подпрограммы обработки прерывания.

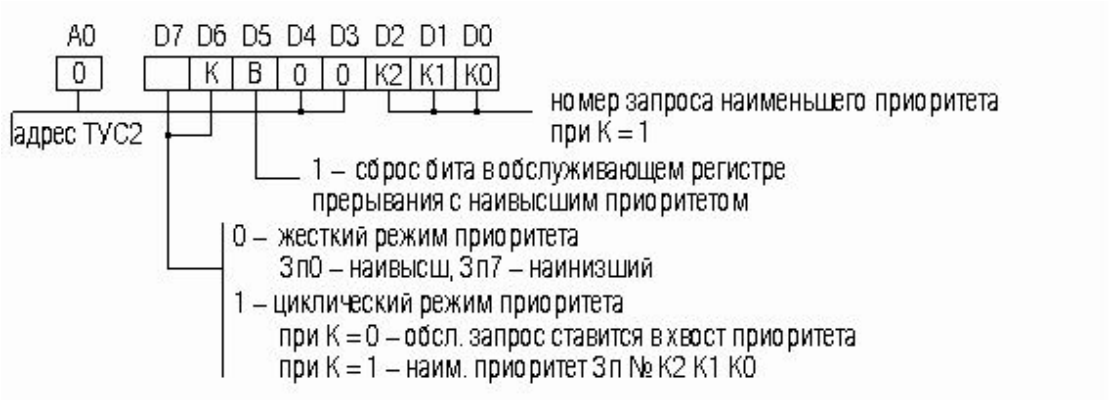
Текущие управляющие слова.

ТУС 1

Этим управляющим словом программируется регистр маскирования прерываний.



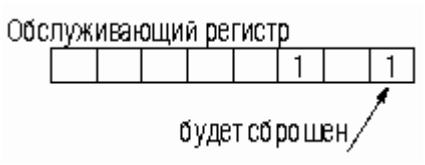
ТУС 2.



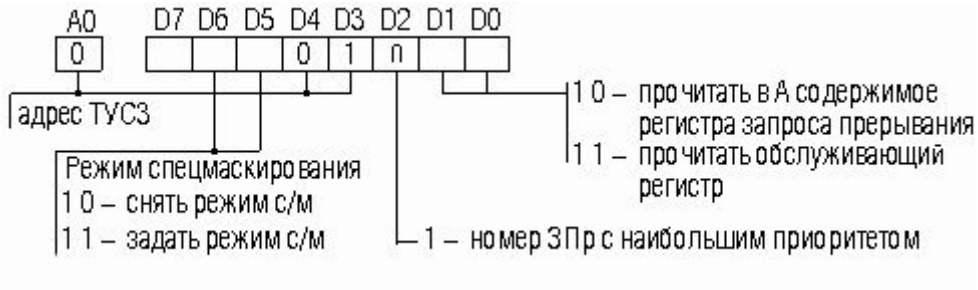
Если К=0 и В=1, то будет сброшен бит в обслуживающем регистре прерывания, имеющего наивысший приоритет.

После обработки прерываний необходимо обязательно сбросить обслуживающий бит.

Если К=1 и В=1, то будет сброшен бит в обслуживающем регистре с номером К2 К1 К0 (номер запроса).

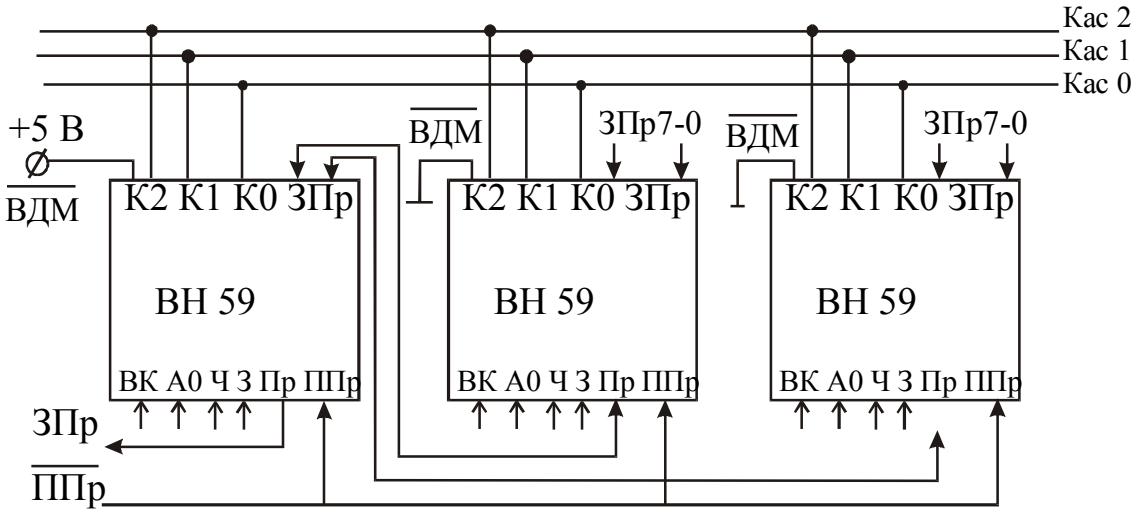


ТУС 3.



Режим спецмаскирования. Если во время обслуживания запроса наивысшего приоритета надо обслужить ЗПр с меньшим приоритетом, а сбрасывать ‘1’ в обслуживающем регистре до конца обслуживания запроса нельзя, то применяется спецмаскирование. На время обслуживания ЗПр меньшего приоритета прерывается обслуживание запроса наивысшего приоритета (таким образом можно принимать все ЗПр).

Рассмотрим теперь более подробную схему каскадирования (см. рис. ниже).

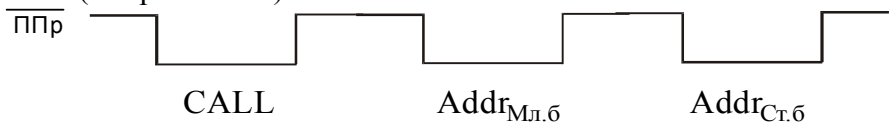


Пример каскадирования с двумя ведомыми МС (с шиной каскадирования)

На этой схеме явно указана двунаправленная шина каскадирования (ведущий выдает, а ведомый принимает). Напомним, какой из контроллеров будет ведущим, а какой ведомым задается схемотехнически с помощью входа ВДм.

Сигналы ППр от МП подаются на все 3 контроллера одновременно. Получив первый сигнал ППр, ведущий контроллер выдает код команды CALL и каскадный код (т.е. номер входа ЗПр, к которому подключен выбранный ведомый) ведомого контроллера

с высшим приоритетом. По второму и третьему сигналам ППр выбранный ведомый контроллер выдает 16-тиразрядный адрес подпрограммы обработки соответствующего прерывания (см. рис. ниже).



Поскольку сигнал ЗПр может быть принят микропроцессором в случайный момент времени, то необходимо помнить о следующих моментах:

- необходимо сохранить в стеке (команда PUSH) все регистры, которые используются в подпрограмме обработки прерываний;
- в конце подпрограммы восстановить содержимое всех используемых в подпрограмме регистров из стека (команда POP);
- для процессора I8080 перед возвратом из подпрограммы обработки прерываний надо выполнить команду EI (разрешить прерывание), так как этот МП после ухода на подпрограмму прерываний автоматически выполняет команду DI (запретить прерывания);
- в контроллере надо сбросить бит обслуживания прерывания после обслуживания прерывания.

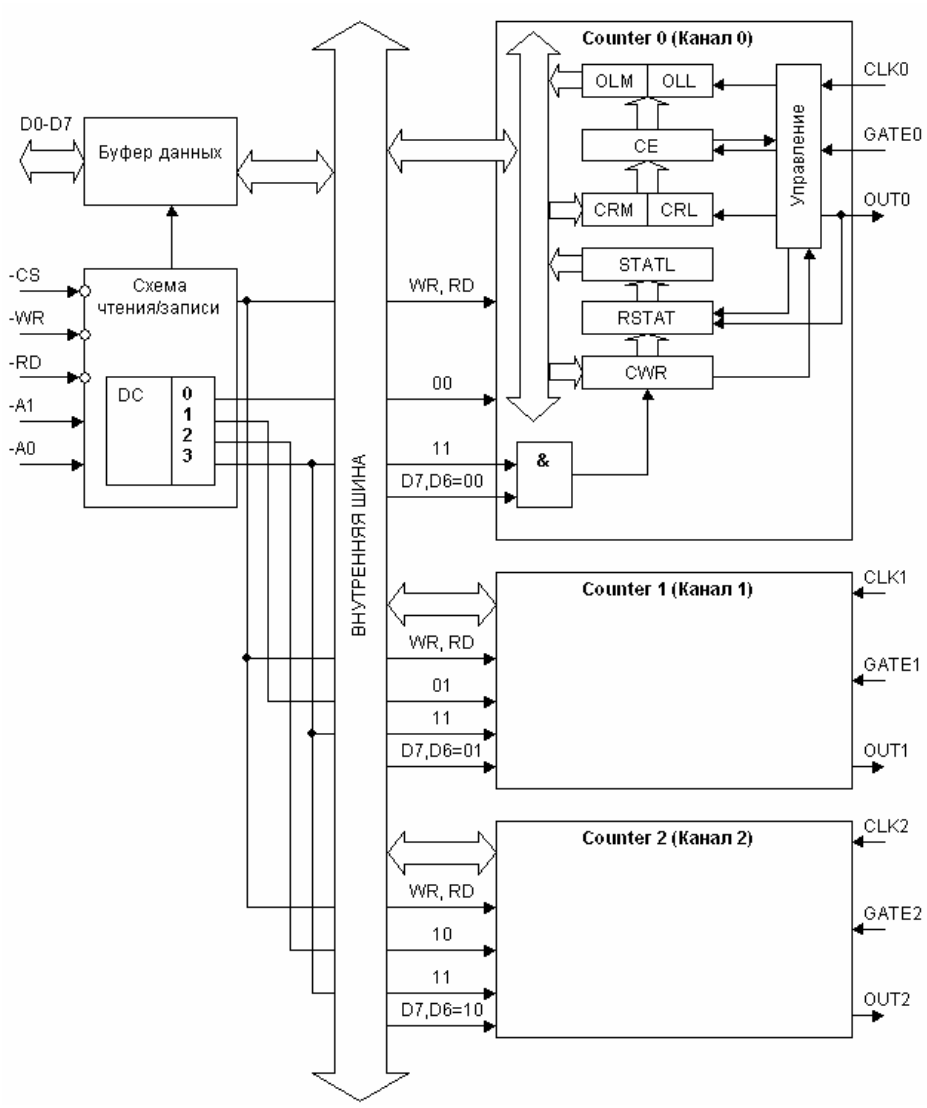
1.3.8. Программируемый таймер КР580ВИ54. Пример применения

Программируемый таймер (ПТ) может вырабатывает временные интервалы, длительность и форма представления которых задаются программно. Специальные режимы ПТ позволяют:

- делить входную частоту;
- вырабатывать однократные временные интервалы по внешнему сигналу или команде МП;
- вести подсчет внешних импульсов или измерять длительность временного интервала.

ПТ состоит из 3-х идентичных и не зависящих друг от друга вычитающих счетчика и могут иметь различные режимы работы и различные типы счета (двоичный или двоично-десятичный).

Любой из названных режимов работы любого из трех счетчиков ПТ может быть запрограммирован загрузкой в таймер управляющего слова. Текущее состояние счетных схем ПТ может быть прочитано микропроцессором. Структурная схема ПТ приведена на рисунке ниже.



Используемые условные обозначение ПТ:

- D0…D7 - двунаправленная шина данных;
- $\overline{RD}$ (ЧТ), $\overline{WR}$ (ЗП) - управляющие сигналы чтения и записи;
- A1, A0 – адресные входы;
- $\overline{CS}$ - выбор кристалла (устройства);
- CLK0 (ТИ0).. CLK2 (ТИ2) - тактовые импульсы;
- GATE0 (Р0)... GATE2 (Р2) - сигналы разрешения работы канала;
- OUT0 (Вых0)...OUT2 (Вых2)- выходные сигналы каналов.
- CE - счетчик/таймер;
- OL - выходная защелка;
- CR - регистр для хранения констант счета;
- RSTAT - регистр состояния;
- CWR - регистр управления.

Назначение основных частей МС.

Буфер шины данных D0…D7 представляет собой двунаправленный 8-разрядный буфер, используемый для сопряжения ПТ с ШД МП-системы. Через буфер принимаются управляющие слова и начальные значения кодов счетчиков и выдаются текущие состояния счетных схем ПТ.

Схема управления чтением и записью обеспечивает выполнение операций ввода и вывода информации в таймер. Адресные сигналы указывают один из счетчиков или регистр управляющего слова. Операции ПТ приведены в таблице.

Таблица операций ПТ КР580ВИ54

Операция	Сигнал				
	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0
Запись данных в счетчик 0	0	1	0	0	0
Запись данных в счетчик 1	0	1	0	0	1
Запись данных в счетчик 2	0	1	0	1	0
Запись управляющего слова	0	1	0	1	1
Чтение содержимого счетчика 0	0	0	1	0	0
Чтение содержимого счетчика 1	0	0	1	0	1
Чтение содержимого счетчика 2	0	0	1	1	0
Отключение таймера от ШД	1	X	X	X	X
(выходы D - в высокоимпедансное состояние)	0	1	1	X	X

Счетчики каналов CE представляют собой 16-разрядные вычитающие счетчики с возможностью установки произвольного начального кода. Счет может выполняться в двоичном или двоично-десятичном коде. Входными сигналами каждого счетчика являются тактовые импульсы CLK, по отрицательным фронтам которых содержимое счетчика уменьшается на 1, и сигнал разрешения работы канала, разрешающий (GATE = 1) и запрещающий (GATE = 0) счет. Выходной сигнал канала формирует временной интервал в соответствии с запрограммированным режимом. Счетчики каналов таймера полностью независимы, Каждый из них может иметь свой режим работы и свой начальный код. Чтение содержимого каждого счетчика выполняется либо при прекращении счета, либо без его приостановки (режим чтения "на лету").

Выходная защелка OL 16-разрядная, необходима для запоминания и хранения содержимого счетчика CE по командам "Чтение ‘на лету’" и "Обратное чтение".

Регистр для хранения констант счета CR 16-разрядный, предназначен для хранения констант счета, содержимое регистра изменяется только при перезагрузке счетчика или при перепрограммировании режима его работы (запись управляющего слова).

Регистр состояния RSTAT 8-разрядный регистр состояния RSTAT содержит код типа счета, код режима работы канала, код чтения/записи констант счета, признак перезагрузки констант и состояние выхода OUT, содержимое RSTAT защелкивается в STATL по команде "Обратное чтение";

Регистр управления CWR 6-разрядный, предназначен для приема и хранения кода управляющего слова (разряды D5-D0), который задает режим работы канала, определяет тип счета и последовательность загрузки констант счета.

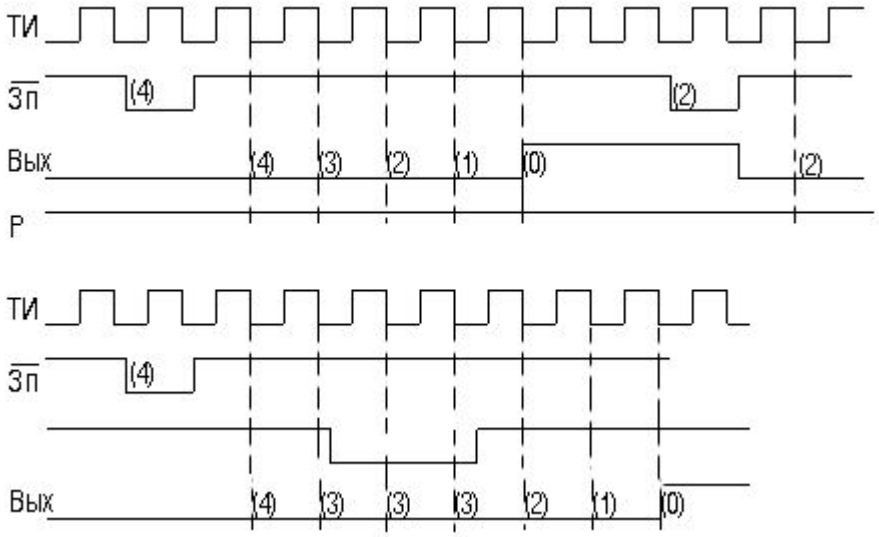
РЕЖИМЫ РАБОТЫ.

Каждый канал таймера может работать в одном из шести режимов.

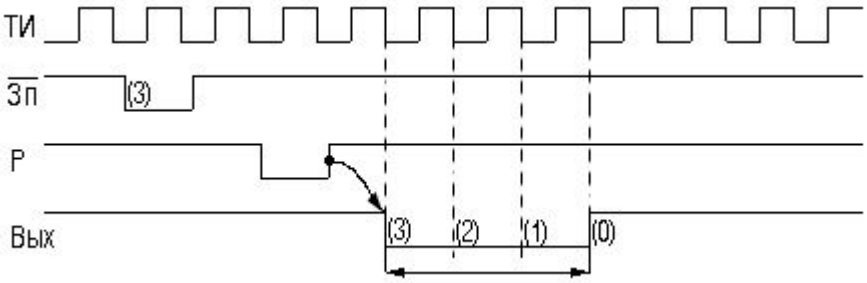
Режим 0. Режим счетчика событий.

Обеспечивает выработку единичного выходного сигнала после поступления на ПТ заданного количества импульсов CLK

Выходной сигнал OUT устанавливается в 0 (низкий уровень) в момент загрузки управляющего слова, задающего данный режим при программировании таймера. После записи начального кода начинается счет импульсов CLK. Отсчитав заданное количество импульсов, выходной сигнал OUT устанавливается в 1 (высокий уровень) по отрицательному фронту CLK и сохраняет это значение до занесения следующего начального кода.



Режим 1. Режим программируемого одновибратора.

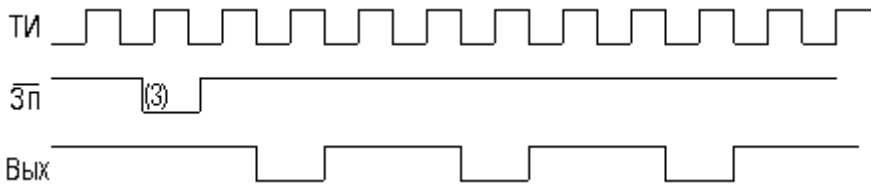


Обеспечивает выработку временного интервала, равного заданному количеству периодов CLK .

Выходной сигнал OUT в исходном состоянии имеет высокий уровень (т. е. 1) и принимает нулевые значения на время, пропорциональное значению начального кода счетчика. Интервал формируется от первого отрицательного фронта сигнала CLK после установки разрешающего входного сигнала GATE в единичном значении.

Режим 2. В режиме делителя частоты.

Таймер генерирует периодический сигнал с частотой в N раз меньшей, чем частота тактовых импульсов CLK, N - начальный код счетчика.

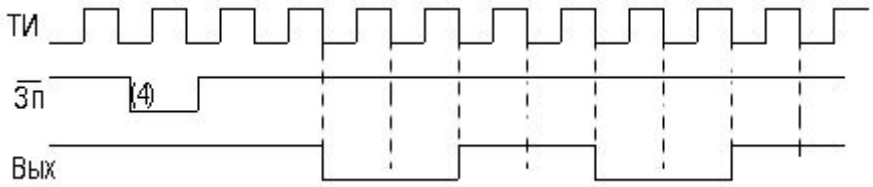


В течение N-1 такта каждого периода выходной сигнал OUT равен 1, на последнем такте периода - нулю. Перезагрузка счетчика между двумя выходными импульсами приводит к изменению величины следующего периода. Формирование выходного сигнала осуществляется при GATE = 1 непосредственно после загрузки начального кода счетчика.

Режим 3. Режим генератора меандра.

Аналогичен режиму делителя частоты за исключением того, что выходной сигнал в течение первой половины периода равен 1, а во второй половине устанавливается в ноль.

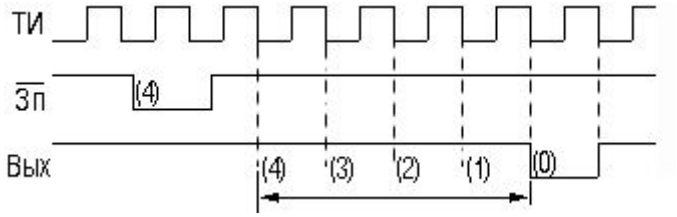
Если начальный код счетчика нечетен, то единичный выходной сигнал на один такт дольше нулевого.



Режим 4. В режиме программного стробирования.

Таймер формирует временной интервал, отсчитываемый от первого отрицательного фронта сигнала CLK после окончания записи кода N в ПТ.

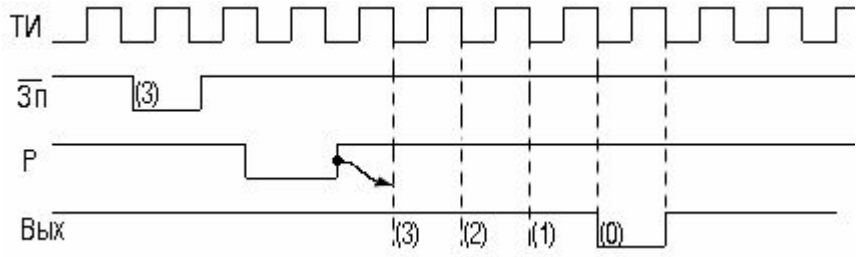
Длительность интервала равна N периодам тактового сигнала. После окончания интервала выходной сигнал принимает нулевое значение на один период сигнала CLK. Если счетчик перезагружается во время счета, то предыдущий интервал прерывается и



формируется новый - с учетом принятого значения начального кода.

Режим 5. Режим аппаратного стробирования.

Аналогичен режиму программного стробирования, однако формирование интервала здесь начинается после установки входного сигнала GATE из 0 в 1. Для генерирования следующего интервала сигнал GATE должен быть предварительно установлен в ноль. Если это произойдет до окончания формирования текущего интервала, то он прерывается,

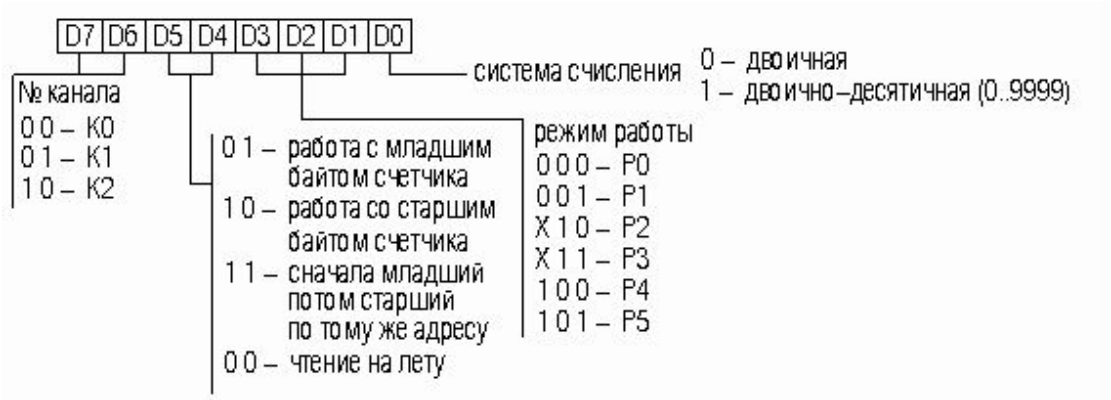


и начинается формирование нового интервала.

Программирование таймера

Программирование таймера осуществляется занесением управляющих слов для каждого счетчика и их начальных кодов. Отдельные каналы могут программироваться в любой последовательности. Начальный код счетчика может быть загружен не сразу после управляющего кода, а в любой последующий момент времени. Загрузка отдельных байтов начального слова может выполняться различным способом в соответствии с запрограммированной последовательностью.

Формат управляющего слова ПТ КР580ВИ54 приведен на рис.



Часто желательно читать содержимое счетчика без нарушения процесса счета. Имеется три метода чтения счетчиков *Простое чтение, "Чтение на 'лету'", "Обратное чтение"*.

1. Простое чтение

Осуществляется путем выполнения обычной операции чтения. Для обеспечения стабильного показания счетчика работа счетчика должна быть приостановлена путем временного снятия сигнала разрешения (т.е. подачей GATE=0), либо путем прекращения подачи сигнала CLK на данный счетчик (с помощью внешней логической схемы)

2. "Чтение на 'лету'"

Осуществляется путем подачи в таймер специальной команды "Чтение на 'лету'" и последующего чтения. Во втором случае программист может обеспечить считывание без прерывания процесса счета путем подачи команды чтения счетчика на "лету", которая характеризуется следующими значениями разрядов управляющего слова:

- D7, D6 содержит код канала (т.е. номер счетчика);
- D5, D4 равны 0 (задают режим чтения счетчика на "лету");
- D3...D0 имеют произвольное значение.
-

Примеры программирования ПТ на разные режимы работы и его инициализации.

1. Программирование ПТ

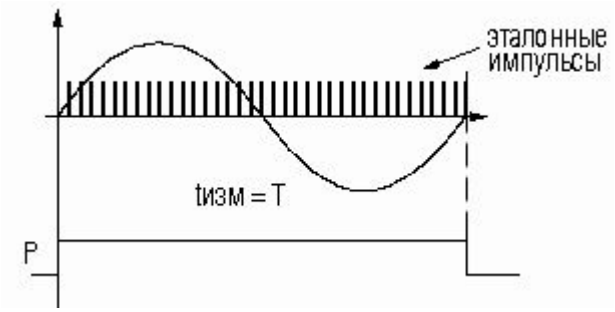
```
        MVI     A,00110101B     ;   Счетчик 0 в режим делителя частоты (режим 2),
        OUT     73H             ;   двоично-десятичный счет
        MVI     A,01010011B     ;   Счетчик 1 в режим программируемого одновибратора
        OUT     73H             ;   (режим 1), двоично-десятичный счет
        MVI     A,10110000B     ;   Счетчик 2 в режим счетчика событий (режим 0),
        OUT     73H             ;   двоичный счет
```

2. Инициализация ПТ

```
        MVI     A,0             ;   Коэффициент деления
        OUT     70H             ;   Счетчик 0 равен 10_10^3
        MVI     A,00010000B     ;
        OUT     70H             ;
        MVI     A,5H            ;   Длительность выделенного интервала составляет 5
                                ;   периодов входной последовательности импульсов
                                ;   ТИ1
        OUT     71H             ;
        MVI     A,0             ;   Счетчик 1 будет считать число входных импульсов
                                ;   ТИ2 до максимально возможного значения 2^16
        OUT     72H             ;
```

Интерфейс с частотным датчиком.

В медленно меняющихся процессах можно измерять частоту сигнала. В быстро изменяющихся процессах частоту можно определить, измерив, период.



Пример схемы сопряжения частотного датчика (сигнал X) с МП-системой с помощью таймера КР580ВИ53 приведен на рисунке ниже.

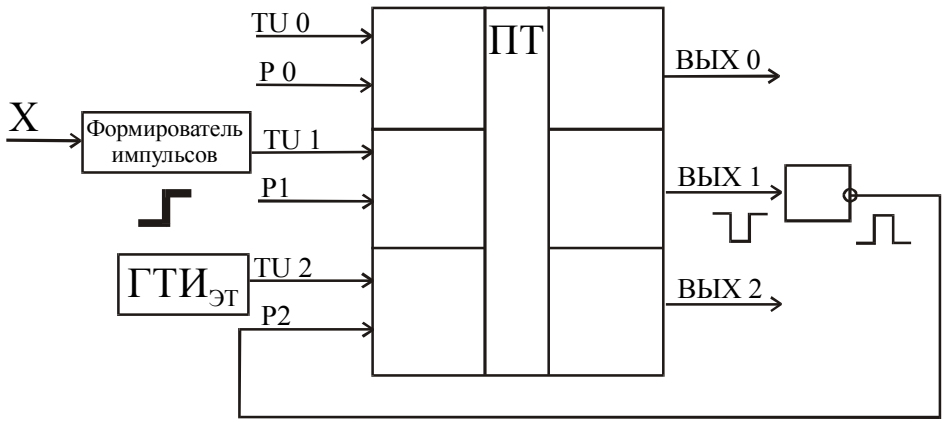


Схема реализации интерфейса с частотным датчиком

Существует два способа измерения частоты.

1. Измерение частоты.

Задаем время измерения $t_{изм}$. Частоту $\mathbf{f_{изм}}$ определяем по количеству периодов $\mathbf{T_{изм}}$, которые уместились в интервал времени $t_{изм}$. Время измерения задается с помощью эталонных импульсов:

$$\mathbf{t_{изм}=T_{эт} * к_{эт}.}$$

$T_{эт}$ – эталонный импульс;

$к_{эт}$ – количество импульсов.

При погрешности порядка 1% необходимо, чтобы в заданный интервал времени уложилось не менее 100 периодов.

2.Измерение периода.

Период сигнала датчика вычисляется по тому, сколько эталонных импульсов уместилось в периоде датчика:

$$\mathbf{T_д = T_{эт} * к_{эт}.}$$

$T_{эт}$ –эталонный импульс;

$к_{эт}$ – количество эталонных импульсов внутри измеряемого периода сигнала датчика.

Погрешность метода сравнима с $T_{эт}$. Возникает проблема создания $t_{изм}= T_д$.

Пример.

Измерим период сигнала датчика с помощью программируемого таймера. Необходимо сформировать $t_{изм}$, кратное заданному количеству измеряемых периодов $T_д$. Используем режим одновибратора для формирования $t_{изм}$. Эталонные импульсы будем подсчитывать с помощью канала, запрограммированного в нулевом режиме.

Таким образом:

1.Первый канал надо запрограммировать в режим одновибратора, чтобы сформировать интервал измерения $t_{изм}$, ($к_д$ – число периодов сигнала датчика – задается программно).

2.Вых.1 через инвертор подаем на вход Р2 второго канала, который программируем в режим счетчика событий.

3.Счетчик второго канала обнуляем.

4.Начало измерений запускаем выходным сигналом одного из выходов порта управления (перепад с низкого уровня на высокий, то есть сначала в этот разряд записывается 0, а потом - 1).

5.Процесс измерения заканчивается через интервал времени $t_{изм}$ путем снятия высокого уровня сигнала Р2.

6.Читаем содержимое счетчика канала 2.

Теперь в счетчике будет

0000h 0

0FFFFh -1

0FFFEh -2

.

.

.

$-к_{эт}$

Чтобы получить $к_{эт}$, надо изменить знак у прочитанного из второго канала значения.

1.3.9. Контроллер прямого доступа KP580BT57

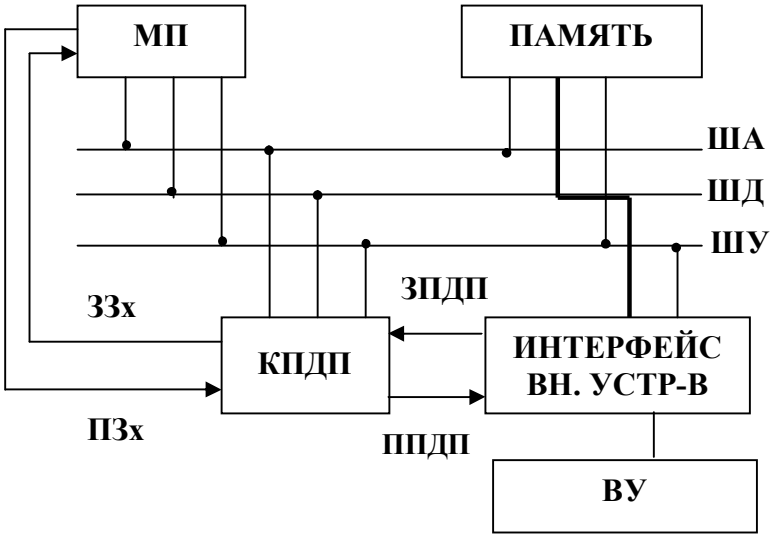
Ввод-вывод в режиме прямого доступа к памяти (ПДП).

Прямой доступ к памяти (ПДП) характерен для высокоскоростных внешних устройств, например, накопителей на магнитных дисках, сетевых карт и т.д. Этот режим ввода-вывода позволяет интерфейсным схемам устройства обмениваться данными с памятью без постоянного участия процессора. Операции ПДП выполняются под управлением схемы, называемой контроллером прямого доступа к памяти (КПДП). Этот контроллер исполняет роль процессора при обращении к памяти. Для каждого пересылаемого слова он формирует адрес памяти и сигналы системной магистрали, управляющие пересылкой данных. Поскольку КПДП производит пересылку блоков данных, он увеличивает адрес памяти, по которому будет записываться (или читаться) каждое следующее слово, и отслеживает количество таких операций. Чтобы инициировать пересылку блока данных, МП пересылает контроллеру начальный адрес этого блока (в регистр адресов), размер блока и направление пересылки (в счетчик циклов или ячеек). Последовательность работы КПДП при пересылке блока данных следующая (см. рис. ниже):

1) От ВУ посылается сигнал ЗПДП (запрос прямого доступа к памяти) к КПДП;
2) КПДП выставляет сигнал ЗЗх (запрос захвата) к МП. МП, получив сигнал ЗЗх, отключается от системной магистрали и выставляет сигнал ПЗх (подтверждение

захвата) к КПДП;

3) КПДП, получив ПЗх, выставляет сигнал ППДП (подтверждение прямого доступа к памяти) для ВУ и сигнал, запрещающий адресацию к ВУ. Далее КПДП начинает выполнять функции процессора по управлению системной магистралью. КПДП узнает адрес ячейки, объем передаваемых данных и направление обмена от МП.



Рассмотрим конкретный КПДП КР580ВТ57

КПДП КР580ВТ57.

Структурная схема контроллера КР580ВТ57 приведена на рисунке ниже, где приняты следующие обозначения:

ЗПДП0..ЗПДП3 – запросы внешних устройств;
ППДП0..ППДП3 – подтверждение прямого доступа для внешних устройств;
ЗЗх – запрос захвата магистрали;
ПЗх - подтверждение захвата микропроцессором;
ГТ - готовность устройства;

ТИ – тактовые импульсы;

Сбр – начальная установка контроллера;

ВК – выбор микросхемы;

А3..А0 – двунаправленная адресная шина для адресации внутренних регистров контроллера (входы) или выдачи младших четырех разрядов адреса памяти (выходы);

А7..А4 – разряды адреса памяти;

Стр А – строб адреса, используется для загрузки выдаваемого контроллером старшего байта адреса памяти во внешний регистр-защелку;

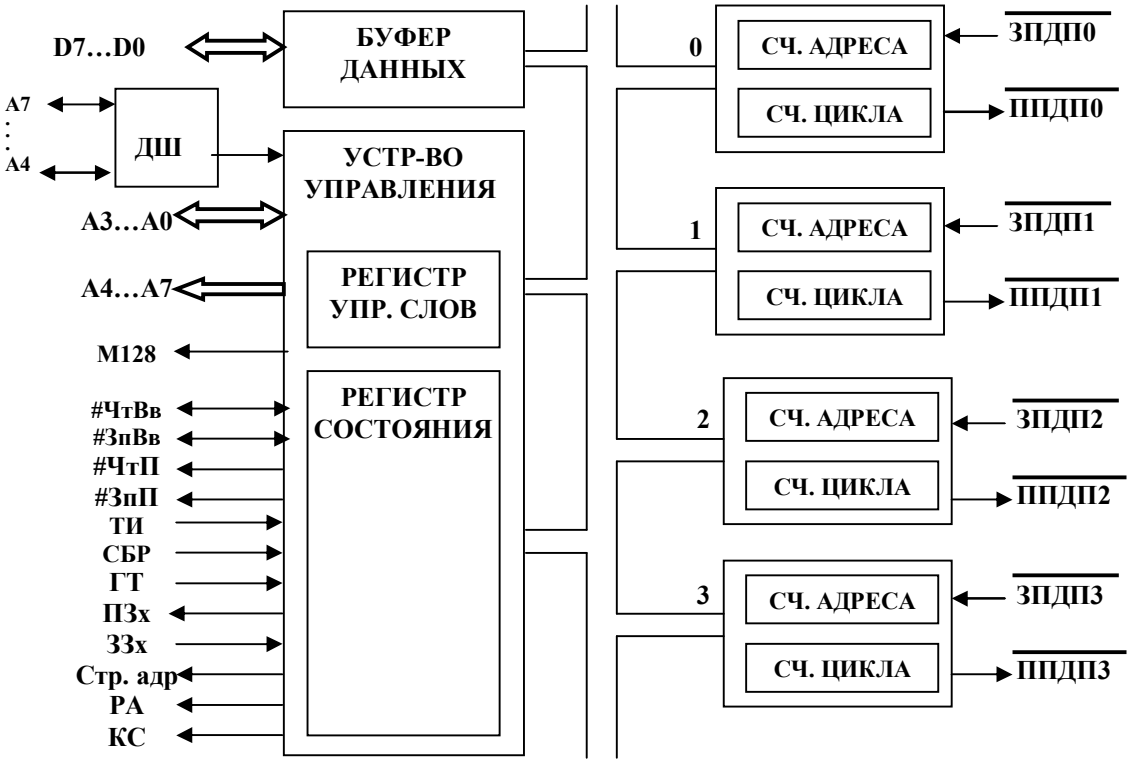
РА – разрешение адреса используется для блокировки во время режима ПДП всех устройств МП-системы, не имеющих прямого доступа к памяти;

ЗпП, ЧтП – выходные линии записи и чтения памяти;

ЗпВВ,ЧтВВ – двунаправленные линии, являющие с одной стороны входами при программировании контроллера и чтения его внутренних регистров, а с другой стороны выходами при обмене ПДП;

КС – конец счета;

М128 – при передаче блоками каждые 128 байт сопровождаются импульсом.



Двунаправленный 8-разрядный буфер данных, сопрягающий КПДП с ШД системы, предназначен для приема управляющего слова, выдачи старшего байта адреса ОЗУ, содержимого счетчика адресов и счетчика циклов и слова состояния контроллера.

Блок управления обеспечивает синхронизацию отдельных узлов КПДП при его программировании, чтении его состояния и собственно обмена.

При программировании блок управления адресует управляющие слова в регистр режима, счетчики адресов и циклов каждого канала и обеспечивает их запись по сигналу на входе $\overline{ЗпВ/В}$. Блок также обеспечивает чтение счетчиков и регистра состояния по входному сигналу $\overline{ЧтВ/В}$. Реализуемые блоком операции чтения и записи приведены в таблице ниже.

Операции чтения и записи, реализуемые блоком управления.

Операции	Сигнал			
	$\overline{ВК}$	$\overline{ЗпВ/В}$	$\overline{ЧтВ/В}$	А3
Запись управляющих слов в счетчики каналов	0	0	1	0
Запись управляющего слова в регистр режима	0	0	1	1
Чтение содержимого счетчика каналов	0	1	0	0
Чтение регистра состояния	0	1	0	1

В режиме обмена блок вырабатывает текущий адрес ячейки ОЗУ и с учетом готовности ПУ формирует управляющие сигналы для ОЗУ и ПУ ($\overline{ЗпП}$, $\overline{ЧтП}$, $\overline{ЗпВ/В}$, $\overline{ЧтВ/В}$), обеспечивающие запись данных в ОЗУ или ПУ. Блок также управляет окончанием передачи информации по исчерпании запрограммированного количества циклов или снятия запроса.

Каналы 0…3 содержат по два 16-разрядных счетчика:

- счетчик адресов;

- счетчик циклов.

Счетчик адресов задает текущий адрес ячейки памяти, с которой выполняется обмен. Содержимое 14-ти младших разрядов счетчика циклов, уменьшенное на 1, определяет количество циклов ПДП, а два старших его бита A_{14} и A_{15} отведены для задания режима обмена. (A_{14}=1 – запись, A_{15}=1 – чтение). Каждый канал включает в себя схему приема запроса на прямой доступ, которая в соответствии с разрешением блока приоритетов генерирует сигнал подтверждения ПДП. Блок приоритетов обеспечивает очередность обслуживания одновременных запросов со стороны нескольких ПУ либо по жесткому приоритету, либо по назначенному приоритету с учетом номера обслуживаемого канала.

Контроллер КР580ВТ57 может быть запрограммирован с помощью МП для работы в различных режимах. При этом МП адресуется к КПДП как к обычному модулю ввода-вывода. Такой режим работы контроллера называется *подчиненным* режимом и служит для пересылки команд и адресов КПДП и для получения от него слова состояния (т.е. информации о состоянии контроллера). В *основном* режиме контроллер сначала запрашивает у МП системные шины с помощью сигнала 33х. После отключения МП от шин контроллер дает команду памяти записать данные с ШД (или выставить данные на ШД) и одновременно периферийному устройству выставить данные на ШД (или считать данные с ШД). Таким образом, информация передается между памятью и ПУ под управлением КПДП с помощью сигналов $\overline{ЗпП}$, $\overline{ЧтП}$, $\overline{ЧтВ/В}$, $\overline{ЗпВ/В}$.

При использовании КПДП требуется две различные конфигурации адресной шины: одна для подчиненного режима, другая – для основного. В подчиненном режиме (сигнал РА контроллера выключен) адрес формирует МП на ША А0…А15. В основном режиме (сигнал РА включен) старший байт адреса на линиях А8…А15 формирует контроллер со своих линий D0…D7, которые мультиплексно подключены к этим адресным линиям. Этот байт адреса фиксируется во внешнем буферном регистре управляющим сигналом СтрА. Младший байт адреса выдается на ША непосредственно линиями А0…А7 контроллера.

Программирование КПДП.

В начале работы на контроллер подается сигнал Сбр, который сбрасывает регистр режима. Это приводит к запрещению использования всех каналов ПДП и предотвращает конфликты на шинах МП-системы при включении питания. Затем загружаются счетчики каналов контроллера исходной информацией. Загрузка (инициализация) выполняется программным способом путем засылки информации в КПДП.

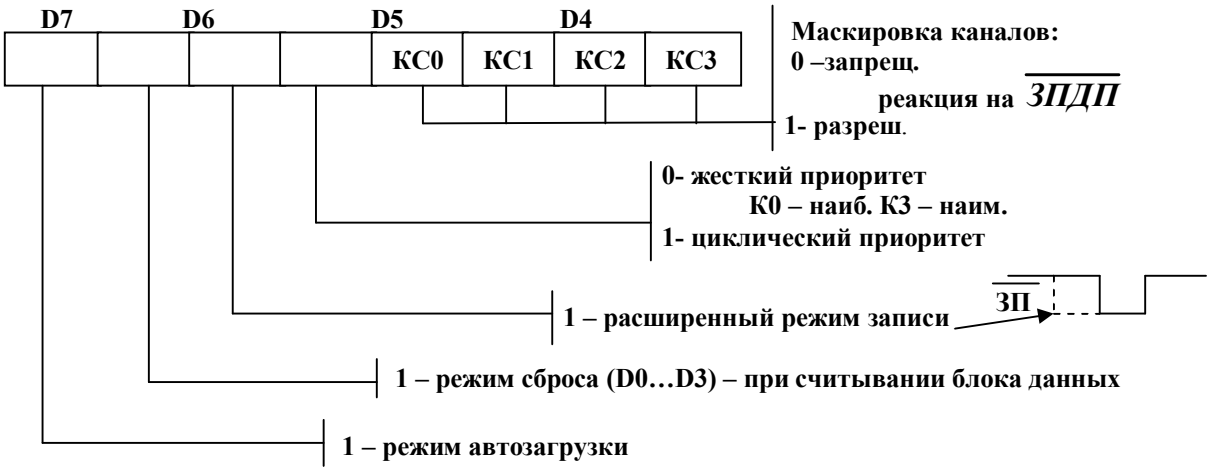
Адресации внутренних регистров КПДП

Регистр	Линии адреса				Регистр	Линии адреса			
	А3	А2	А1	А0		А3	А2	А1	А0
Счетчик А0	0	0	0	0	Счетчик Ц2	0	1	0	1
Счетчик Ц0	0	0	0	1	Счетчик А3	0	1	1	0
Счетчик А1	0	0	1	0	Счетчик Ц3	0	1	1	1
Счетчик Ц1	0	0	1	1	Регистр режима	1	0	0	0
Счетчик А2	0	1	0	0	Регистр состояния	1	0	0	0

Так как адрес ячеек памяти является 16-разрядным числом, то в счетчики адресов и счетчики циклов загружают последовательно два байта, сначала младший байт, затем старший. Оба байта должны быть посланы один за другим. На это время должна быть заблокирована система прерываний, чтобы не допустить передачи посторонних данных.

В соответствии с выше указанными таблицами МП может прочитать текущее содержимое 16-разрядных счетчиков адресов и счетчиков циклов каждого из двух следующих одна за другой команд ввода, сначала младшего байта, а затем старшего байта. После того как счетчики каналов контроллера инициированы, осуществляется загрузка управляющего слова в регистр режима.

Формат слова режима КПДП.



Четыре младших бита D0…D3 определяют состояние соответствующих каналов ПДП. При Di=0 (i й канал запрещен) контроллер не реагирует на запрос ЗПДПi. При Di=1 (i й канал разрешен) контроллер реагирует на запрос ЗПДПi.

Бит D4=0 определяет фиксированный приоритет (канал 0 имеет наивысший, а канал 3 – самый низкий приоритет) или циклический приоритет при D4=1, когда приоритеты каналов меняются после передачи каждого байта.

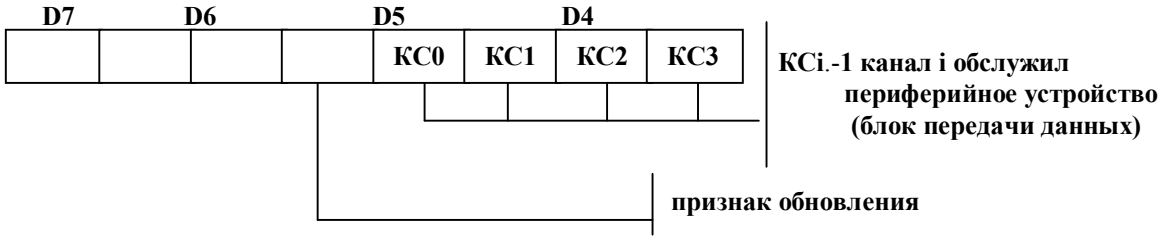
В этом режиме после того, как канал обнаружен, каналу со следующим порядковым номером присваивается наивысший приоритет для передачи следующего байта. Затем присваиваются приоритеты остальным каналам в соответствии с возрастанием их номеров. При этом считается, что канал 0 по приоритету следует за каналом 3.

После установки бита D5 расширенный (удлиненный) записи несколько увеличивается продолжительность управляющих сигналов $\overline{ЗпП}$ и $\overline{ЗпВ/В}$ контроллера. Это позволяет не вводить дополнительного состояния ожидания с помощью сигнала Гт.

Если установлен бит D6 (режим останова КС), то после формирования сигнала КС по окончании передачи ПДП соответствующий канал ПДП запрещается. Дальнейшие передачи в этом канале возможны только после повторной загрузки регистра режима контроллера. Если бит D6=0, то появление сигнала КС не влияет на состояние канала ПДП, и закончить операции ПДП должно ПУ.

Установленный бит D7 автозагрузки позволяет использовать канал 2 для повторных передач предыдущего блока данных или сцепленных блоков данных без программного вмешательства между передачами блоков. Для этого счетчики канала 2 инициализируются обычным образом, а в счетчики канала 3 загружаются параметры повторной инициализации канала 2. После завершения в канале 2 передачи первого блока данных параметры из счетчиков канала 3 загружаются в счетчики канала 2. При этом состояние бита D6 в регистре режима не влияет на работу канала 2. В режиме автозагрузки, во-первых, при программировании канала 2 его начальные параметры автоматически дублируются в счетчиках канала 3, что в дальнейшем и обеспечивает повторяющиеся передачи одного и того же блока данных. Во-вторых, адрес блока, длина блока и информация о режиме для следующей передачи блока по каналу 2 могут быть посланы в канал 3 во время работы канала 2.

Слово состояния контроллера



Слово состояния КПДП содержит четыре бита КС0…КС3. Эти биты указывают какие каналы сгенерировали сигнал КС после того, как слово состояния было прочитано в последний раз. Таким образом, биты состояния КС0…КС3 указывают, была ли завершена передача блока данных в течение этого промежутка времени.

Признак обновления D4 устанавливается, когда в режиме автозагрузки параметры канала 3 передаются в счетчики канала 2, т.е. обновляется содержимое счетчиков канала 2. Следовательно, чтобы предотвратить преждевременное обновление информации в канале 3, необходимо считать слово состояния и проверить признак обновления прежде, чем сделать попытку перезагрузить канал 3. Этот признак можно перевести в нулевое состояние сигналом Сбр или отменой режима автозагрузки. Кроме того, признак обновления автоматически сбрасывается после обновления содержимого счетчиков канала 2 и передачи первого блока данных.

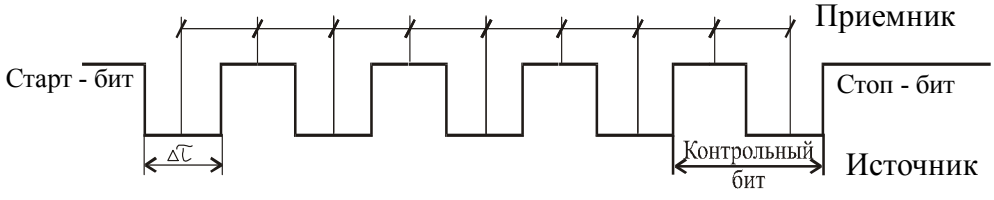
1.3.10. Программируемый связной адаптер КР580ВВ51

Общие сведения

При всем удобстве параллельной передачи информации такой способ хорош лишь при передаче на небольшие расстояния.

Основная проблема при последовательной передачи информации заключается в обеспечении битовой и символьной синхронизации. Эту проблему можно решить с помощью асинхронной посылки.

. Асинхронная посылка начинается старт-битом низкого уровня, а заканчивается стоп-битом высокого уровня (см. рис. ниже). Перепадом с высокого уровня напряжения на низкое источник говорит, что будут передаваться данные. Приемник при этом запускает часы. О длительности интервалов $\Delta\tau$ источник и приемник договариваются заранее. Отсчитав половину длительности предполагаемого старт-бита, приемник проверяет уровень входного сигнала. Если он остается низким, то приемник начинает принимать данные, поскольку вслед за старт-битом идут информационные биты. Об их количестве источник и приемник также договариваются заранее (от 5 до 8 информационных битов). Приемник на середине длительности очередного интервала $\Delta\tau$ считывает уровень сигнала.



После информационных битов идет необязательный контрольный бит. Если запрограммировать использование этого бита, то приемник формирует такой же бит и сравнивает его с пришедшим контрольным битом. Если эти сигналы совпадают, значит, информация принята без искажений, если нет, то выставляется флаг "ошибка паритета".

В конце передачи всегда проверяется значение уровня стоп - бита. Он должен быть высоким, в противном случае выставляется ошибка "отсутствие стоп-бита".

Рассмотрим подробнее, как происходит обмен данными между источником и приемником.

Источник.

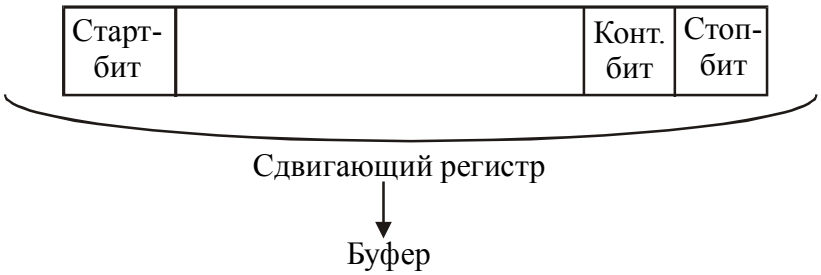
1) Из МП байт информации поступает в буфер передатчика.
2) К этому байту в регистре сдвига передатчика подсоединяются -
   - старт - бит
   - бит контроля
   - стоп бит.

   Теперь окончательно сформирована асинхронная посылка (см. рис. ниже).

3) Асинхронная посылка побитно выталкивается в линию передачи данных приемнику.

Приемник.

1) С входа приемника биты последовательно вталкиваются в сдвиговый регистр приемника (см. рис. ниже).

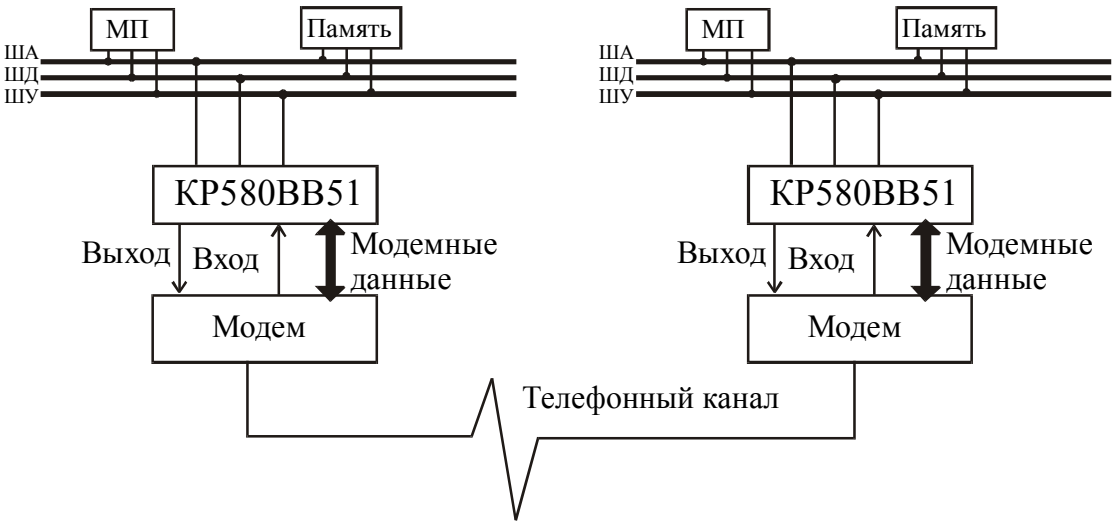


2) Из сдвигового регистра приемника информационная часть принятой асинхронной посылки копируется в буфер приемника, при этом устанавливаются флаги ошибок, если они произошли.
3) Если ошибок нет, МП вводит принятый байт из буфера.

Надо позаботиться о том, чтобы на временном интервале асинхронной посылки $10\Delta\tau$ ошибка рассогласования хода часов источника и приемника ε не превышала значение $1/2\Delta\tau$. Если ε будет больше, то будет либо недополучен бит, либо получен лишний бит. Следовательно, относительная ошибка рассогласования часов источника и приемника не должна превышать 5%. Напомним, асинхронная посылка обеспечивает и побитовую и побайтовую синхронизацию.

Программируемый связной адаптер (ПСА) КР580ВВ51 представляет собой универсальный синхронно - асинхронный приемо - передатчик, который можно запрограммировать в режимы асинхронной и синхронной передачи.

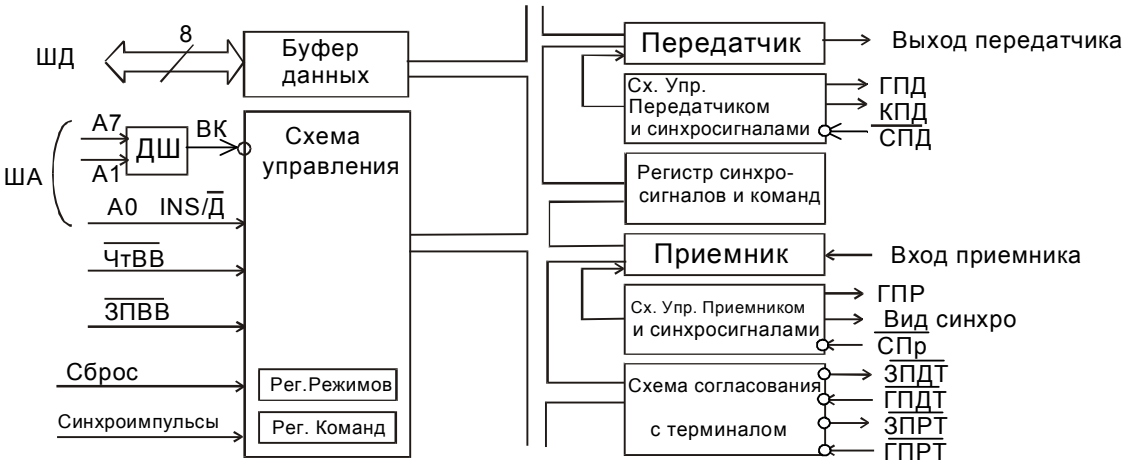
Место КР580ВВ51 в МПС показано на рисунке ниже.



Передача по телефонной линии (или по другой информационной линии) осуществляется в том случае, если расстояние между модемами более 1 км. Если расстояние меньше, то связные адаптеры можно соединять напрямую.

Напомним, что если линия длинная, то в ней происходит затухание сигнала. Для борьбы с этим явлением используют модуляцию с помощью модема, который при передаче данных осуществляет модуляцию передаваемого сигнала, а при приеме модулированного сигнала производит обратную операцию – демодуляцию, т.е. выделение полезного сигнала.

Структурная схема ПСА КР580ВВ51 приведена на рисунке ниже.



Структурная схема ПСА КР580ВВ51

На рисунке приняты следующие обозначения.

Терминал - одним из вариантов терминала может служить модем.

ГПд - сигнал, свидетельствующий о том, что буфер передатчика пуст и передатчик готов к приему новой порции данных из МП.

КПд- сигнал, свидетельствующий о том, что передатчик закончил передачу.

СПд - синхронизация передатчика, управляет скоростью передачи символа. Скорость передачи стандартизована, она задается при инициализации ПСА. В синхронном режиме работы частота синхронизации равна скорости передачи и множитель кратности равен единице (1х), а в асинхронном режиме передачи частота синхронизации выше скорости передачи в целое число раз, т.е. кратна скорости. Значение множителя кратности определяется частью слова режима и может равняться (1х), (16х) или (64х).

ГПр - сигнал, свидетельствующий о том, что буфер приемника полон и приемник адаптера готов передавать полученный символ в МП.

СПр – синхронизация приемника, управляет скоростью приема символа. Скорость приема стандартизована, она задается при инициализации ПСА. В синхронном режиме работы частота синхронизации равна скорости передачи и множитель кратности равен единице (1х), а в асинхронном режиме передачи частота синхронизации выше скорости передачи в целое число раз, т.е. кратна скорости. Значение множителя кратности определяется частью слова режима и может равняться (1х), (16х) или (64х).

Вид синхронизации – синхронизация в синхронном режиме может быть внешней и внутренней.

ЗПдТ - запрос передатчика терминала. Может быть использован как запрос о готовности терминала к передаче данных в адаптер.

ГПдТ - готовность передатчика терминала. Этот сигнал можно трактовать как готовность модема к работе.

ЗПрТ - запрос приемника терминала. Может быть использован для запроса о готовности приемника терминала принять данные от адаптера.

ГПрТ - готовность приемника терминала. Сообщает о готовности приемника терминала принять данные от адаптера.

Программирование ПСА.

При программировании ПСА в него должен быть загружен набор управляющих слов. Управляющие слова подразделяются на два формата:

- слово режима (начальное управляющее слово)
- слово команды (текущее слово)

Слово режима определяет общие характеристики ПСА, поэтому оно обязательно должно следовать сразу же после операции сброса (внешней или внутренней). После записи слова режима в адаптер можно вводить синхросимволы или слова команды.

Слово команды (текущее управляющее слово) управляет реальной работой ПСА в выбранном режиме. Слово команды может записываться в любой момент времени в процессе обработки блока данных. Для возвращения ПСА к формату слова режима в слове команды можно установить в логическую единицу специальный разряд, который имитирует внутреннюю операцию сброса, автоматически переводящую адаптер к формату слова режима. Таким образом, слова команды должны следовать после слова режима или после символов синхронизации для синхронного режима. Типовая структура блока данных приведена на рисунке ниже.

*Формат слова режима***.**

D7	D6	D5	D4	D3	D2	D1	D0

D0,D1 - задание режима

0 0 - синхронный режим

остальные комбинации -асинхронный режим, частота, поступающая на входы СПд и СПр делится на 16 и 64 внутри адаптера

0 1 - x1

1 0 - x16

1 1 - x64

D2,D3 - задание размера информационного блока в асинхронной посылке.

0 0 - 5 бит

0 1 - 6 бит

1 0 - 7 бит

1 1 - 8 бит

D4 - "1" - разрешение контроля по паритету (по какому именно ясно из D5)

D5 - "1" - по четности

"0" - по нечетности

D6,D7 - длительность стоп бита.

0 1 - $\Delta\tau$

1 0 - $1{,}5\Delta\tau$

1 1 - $2\Delta\tau$

Для синхронного режима разряд D6 задает вид синхронизации (внешняя или внутренняя), а D7-количество синхросимволов (один или два).

Формат слова команды.

D7	D6	D5	D4	D3	D2	D1	D0

D0 - 1 - разрешение передачи.

D1 - 1 - запрос о готовности терминала передавать данные (ЗПдТ).

D2 - 1 - разрешение приема.

D3 - 1 – выдать состояние пауза.

D4 - 1 – сброс флагов ошибок.

D5 - 1 - выдать сигнал запроса о готовности принять данные (ЗПрТ).

D6 – программируемый сброс. Чтобы установить ПСА в исходное состояние рекомендуется 3 раза передать эту команду.

D7 - действует только при синхронной передаче. Разрешение поиска синхросимволов.

Формат слова состояния.

D7	D6	D5	D4	D3	D2	D1	D0

D0 – готовность передатчика ГПд.

D1 - готовность приемника ГПр.

D2 - конец передачи КПд.

D3 - ошибка паритета.

D4 - ошибка переполнения (принят из линии связи новый символ, а предыдущий принятый символ микропроцессором еще не прочитан).

D5 – отсутствие стоп-бита.

D6 - только для синхронного обмена. Сигнал «Вид синхро».

D7 - готовность передатчика терминала ГПдТ.