# Арифметические основы построения ЭВМ

## 1.1 Способы представления чисел. Представление чисел с фиксированной точкой и фиксированной запятой. Представление чисел с плавающей запятой. Диапазон и точность представления чисел.

## Представление чисел с фиксированной запятой

Пусть есть n разрядов для представления числа с фиксированной запятой.

(*Диапазон* изменения чисел характеризуется пределами, в которых могут находиться числа, с которыми оперирует ЭВМ).

При представлении чисел с *фиксированной запятой* считают, что запятая всегда находится перед старшим разрядом, а все числа, которые участвуют в вычислениях, считаются по *абсолютной величине* меньше единицы:

|X|<1

|Х|MAX=0,1...11=1-2-n

|X|MIN=0,0...01=2-n

То есть, числа с фиксированной запятой - это правильные дроби.

Если числа выходят за диапазон [|XMAX|;|XMIN|], то:

Если <Xmin: число воспринимается как 0.

Если >Xmax: число воспринимается как бесконечность (машинная бесконечность).

При оптимальном округлении абсолютная ошибка:



Минимальная относительная ошибка:



так как при большом "n"

Максимальная относительная ошибка:



## Представление чисел с плавающей запятой



Х=M\*BE

Мантисса числа должна быть приведена к виду 0,.... (0,1<=M<=1) для единственности представления числа(иначе число можно представить разными способами, например, 32 как 3,2\*10 или 0,32\*102), в поле мантиссы записывается только дробная часть.

Пусть m разрядов отведено под изображение *мантиссы*, а k разрядов под изображение *порядка*. Тогда для двоичной системы и нормализованного вида числа:

То есть *диапазон* чисел:



Абсолютная ошибка представления числа в ЭВМ с *плавающей запятой* равна:



Так как

2-1 <= |Mx| <= 1-2-m,

то минимальная относительная ошибка:



а максимальная относительная ошибка:



Видно, что относительная ошибка в ЭВМ с *плавающей запятой* не зависит от *порядка* числа. При этом *точность* представления больших и малых чисел изменяется незначительно.

## 1.2 Системы счисления

Способ представления изображения произвольных чисел с помощью некоторого конечного множества символов назовем ***системой счисления*** .

## Выбор системы счисления

Выбор системы счисления производится в зависимости от спектра задач, которые будет необходимо решать при помощи выбранной системы счисления. Так, десятичная система счисления более удобна для ручных методов счета, нежели двоичная, которая используется в современных ЭВМ. Преимущество двоичной системы перед десятичной в том, что с её помощью удобнее выполнять огромное количество операций над сравнительно небольшими объемами входных данных, а также проще реализовать схемотехнически.

Пусть M – максимальное число, n – количество разрядов.

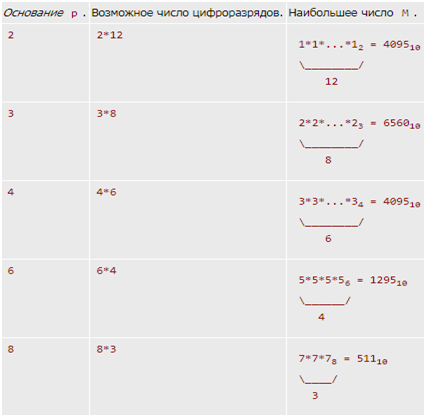
Оборудование, которое нужно для хранения любого числа от 0 до M пропорционально произведению *основания* *системы счисления* на количество *разрядов*.



Цифроразряд – эквивалент оборудования

Таким образом, при заданном числе M количество цифроразрядов при *основании* " p ":

p\*n = p\* logpM (6.1)



Считаем " p " - величиной непрерывной. Находим производную от (6.1) по величине " p ". Берем *вторую производную* по " p ". Увидим, что первая производная обращается в нуль, а вторая - больше нуля при p = e. Т.е. получаем *минимум* при p = e.

Таким образом, оптимальной по оборудованию и быстродействию является *система* с *основанием* е.

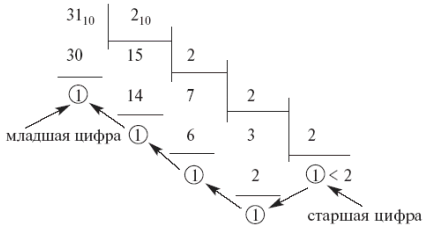
Но е = 2,718… Поэтому оптимальной является *система* с *основанием* р = 3.

## Перевод чисел из одной системы счисления в другую

*Перевод* осуществляется по следующему правилу: исходное число, записанное в *системе* с *основанием* " p " и его частные последовательно делятся на число " q ", представленное в *системе* " p ". *Деление* производится в *системе* с *основанием* " p " и продолжается до получения результата, меньшего " q ". Первый *остаток*, меньший " q ", дает старшую цифру числа Nq. Остатки от деления дают остальные цифры числа Nq.

Пример:

3110 => 2; 3110 = 111112



## Перевод чисел, представленных в 2k-х системах счисления

Как мы уже знаем, в ЭВМ наибольшее применение находит *система* с *основаниями* 2, 4, 8, 16, т.е. *системы* которые кратны степени 2. Поэтому целесообразно рассмотреть лишь правила *перевода* чисел в этих *системах*. Аналогичные правила будут справедливы и для других *систем*.

Допустим, что имеется некоторое целое число N8 в 8-ой *системе*. Оно может быть представлено в виде:

N8 = a1\*8n-1 + a2\*8n-2 + a3\*8n-3 + …+ an-2\*82 + an-1\*81 + an\*80.

Пусть каким-либо образом мы получили запись этого числа в виде двоичного, т.е.:

N2 = b1\*2k-1 + b2\*2k-2 + … + bk-2\*22 + bk-1\*21 + bk\*20.

Разделим эти выражения на 23 = 8:

a1\*8n-2 + a2\*8n-3 + a3\*8n-4 + ... + an-1\*80 + an\*8-1

-------

дробная часть

b1\*2k-4 + b2\*2k-5 + ... + bk-3\*20 + bk-2\*2-1 + bk-1\*2-2 + bk\*2-3

-------------------------

дробная часть

Так как числа равны, равны их дробные части (остатки)

an\*8-1 = bk-2\*2-1 + bk-1\*2-2 + bk\*2-3. (6.2)

Если снова разделим целые части на 23 = 8, то опять получим равные частные и равные остатки.

При этом видим, что каждой восьмеричной цифре соответствует её двоичный эквивалент. Поэтому *перевод* выполняется простой заменой цифры восьмеричной *системы* её двоичным эквивалентом и обратно.

Пример:

62,7538 = 110010,1111010112

## 2. Прямой, обратный и дополнительный коды.

***Прямой код*** представляет собой одинаковое представление значимой части числа для положительных и отрицательных чисел и отличается только знаковым битом. В прямом коде число 0 имеет два представления «+0»(1.0000000) и «–0»(0.0000000).

***Обратный код*** для положительных чисел имеет тот же вид, что и прямой код, а для отрицательных чисел образуется из прямого кода положительного числа путем инвертирования всех значащих разрядов прямого кода, чтобы заменить операцию вычитания в ЭВМ операцией сложения. В обратном коде число 0 также имеет два представления «+0» и «–0».

***Дополнительный код*** для положительных чисел имеет тот же вид, что и прямой код, а для отрицательных чисел образуется путем прибавления 1 к обратному коду. Добавление 1 к обратному коду числа 0 дает единое представление числа 0 в дополнительном коде. Однако это приводит к асимметрии диапазонов представления чисел относительно нуля. Так, в восьмиразрядном представлении диапазон изменения чисел с учетом знака.

-128 <= x <= 127.

В дополнительном коде для положительных чисел мы можем представить числа от 0 до 127(128 чисел). За счет того, что мы избавляемся от представления «-0», сдвинув диапазон отрицательных чисел на единицу, мы можем представить числа от -1 до -128(также 128 чисел).

## 3. Модифицированные коды и их прикладное значение.

Существо ***модифицированных кодов*** состоит в том, что к знаковому разряду добавляется ещё один разряд:

" + " ставится в соответствие 00

" – " ставится в соответствие 11

Несовпадение знаковых разрядов после выполнения *операции* указывает на факт наличия *переполнения*. При этом различают два типа *переполнения*:

" 01 " - положительное

" 10 " - отрицательное.

## 4. Умножение чисел с фиксированной запятой в прямом и дополнительном кодах на 2k.

При умножении на 2k запятая сдвигается вправо на k знаков. В дополнительном коде вычитаем единицу после умножения (но это не точно).

## 5. Методика выполнения операций сложения-вычитания чисел с фиксированной запятой, представленных в дополнительном и модифицированном дополнительном кодах.

Вычитание чисел - то же самое, что сложение положительного и отрицательного числа.

*В дополнительном коде:*

1. Если есть отрицательные числа, переводим их в обратный код, прибавлением единицы получаем дополнительный код. Представление положительных чисел в ПК, ОК и ДК одинаковое.

2. Складываем поразрядно (знаки тоже, при выходе единицы за пределы доступных ячеек единица просто исчезает), получаем некоторое число

3. Если в знаковом разряде 0 – то это и есть ответ, если 1, то инвертируем разряды(из обратного кода в прямой), затем добавляем единицу – получаем ответ.

*В модифицированном дополнительном коде –* то же самое, только знак положительного числа 00, отрицательного – 11.

01 – положительное переполнение

10 – отрицательное

## 6. Методика выполнения операций сложения-вычитания чисел с фиксированной запятой, представленных в обратном и модифицированном обратном кодах.

В обратном –

1. Если есть отрицательные числа, переводим их в обратный код. Представление положительных чисел в ПК и ОК одинаковое.
2. Складываем поразрядно (знаки тоже, при выходе единицы за пределы единица переносится в младший разряд).
3. Переводим обратно в ПК (если в знаковом 0, это и есть ПК).

*В модифицированном обратном коде –* то же самое, только знак положительного числа 00, отрицательного – 11.

01 – положительное переполнение

10 – отрицательное

## 

## **7. Методика выполнения операций умножения в прямом коде чисел фиксированной запятой при представлении операндов в прямом коде со старших и младших разрядов множителя.**

Пусть два числа X и Y представлены с фиксированной запятой в виде:

[X]пк= sign X.x1x2...xn– множимое

[Y]пк= sign Y.y1y2...yn– множитель

1. Представим множитель в виде:

[Y]пк= sign Y. (y1\*2-1+ y2\*2-2+ ... + yn\*2-n)

Тогда:

[Z]пк= [X]пк\*[Y]пк= sign Z. |X| (y1\*2-1+ y2\*2-2+ ...+ yn\*2-n) =

= sign Z. (|X|\*y1\*2-1 + |X|\*y2\*2-2 + ... + |X|\*yn\*2-n) =

= sign Z. (|X|\*2-1\*y1 + |X|\*2-2\*y2 + ... + |X|\*2-n\*yn)

Это есть аналитическая запись алгоритма умножения двух чисел, начиная со старших разрядов множителя.

Алгоритм:

* Множимое сдвигается вправо на 1 разряд
* Анализируется цифра множителя. Если она – нуль, то частичное произведение не суммируется, а если она – единица, то частичное произведение добавляется к общему результату.
* Последовательность операций по пунктам 1 и 2 продолжается "n" раз.
* Знак произведения находится независимо от получения цифровой части по формуле:

sign Z = sign X  sign Y

2) Напишем выражение для произведения двух чисел в несколько изменённом виде, а именно:

[Z]пк= [X]пк\*[Y]пк=

= sign Z.(|X|\*y1\*2-1+ |X|\*y2\*2-2+... + |X|\*yn\*2-n ) =

= sign Z.( |X|\*2-1\*y1+ 2-1(|X|\*2-1\*y2+ 2-1(|X|\*2-1\*y3+ (...)))) =

= sign Z. ((...(( |X|\*yn\*2-1+ |X|\*yn-1 )2-1+ |X|\*yn-2 )2-1+ ... +

+ |X|\*y2)2-1+ |X|\*y1)\*2-1

Это выражение называется преобразованием по схеме Горнера и задаёт алгоритм умножения с младших разрядов множителя.

Таким образом, для умножения должна выполняться следующая последовательность действий:

* Анализируется младшая цифра множителя. Если она равна "1", то множимое участвует в формировании части произведения. В противном случае – не участвует.
* Полученное частичное произведение сдвигается вправо на 1 разряд.
* Операции по пунктам 1 и 2 выполняются до старшего разряда.

(подробнее в учебнике Гурова со страницы 77, или <https://studfiles.net/preview/1456472/page:20/> )

## 8. **Методика выполнения операций умножения в дополнительном коде чисел фиксированной запятой со старших или младших разрядов множителя.**

1. **Умножение с младших разрядов в дополнительном коде**

Алгоритм:

[Z]дк = (...(0+[X]дк\*[yn+1 – yn])\*2-1 + [X]дк\*[yn – yn-1])\*2-1 + ...... + [X]дк\*[y2– y1])\*2-1 + [X]дк\*[y1 – y0]

Если yn = yn+ 1 , то производится сдвиг частичного произведения.

Если yn = 0 и yn+1 = 1, то к частичному произведению прибавляется [X]дк

Если yn = 1 и yn+1 = 0, то из частичного произведения вычитается [X]дк.

## **2)Умножение со старших разрядов в дополнительном коде**

[Z]дк= [X]дк\*[Y]дк= [X]дк\*(y1– y0) + [X]дк\*(y2– y1)\*2-1+ ... +

+ [X]дк\*(yn+1– yn)\*2-n

(Гуров страница 83 или <https://studfiles.net/preview/1456472/page:21/#39> )

## 9. **Методика выполнения операций деления в прямом коде чисел фиксированной запятой при представлении операндов в прямом коде со сдвигом и восстановлением остатка и со сдвигом делителя.**

1. Деление в прямом коде со сдвигом и автоматическим восстановлением остатка:
2. sign Z = sign X sign Y
3. |X| - |Y| = 0
4. Если 00, то z0= 1 и 20- |Y| =1(z0– целая часть результата).

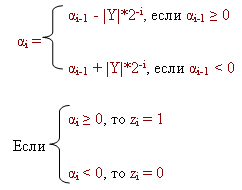
Если 0< 0, то z0= 0 и 20+ |Y| =1

и т. д.

2) Деление в прямом коде со сдвигом делителя и автоматическим восстановлением остатка:

* sign Z = sign X sign Y
* |X| - |Y| = 0
* Если 00, то z0= 1.

Если0< 0, то z0= 0.

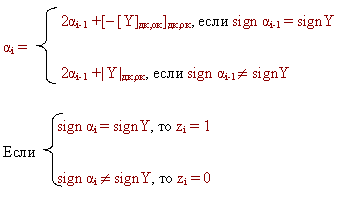


(Гуров страница 85)

## 10. Методика выполнения операций деления в дополнительном коде чисел фиксированной запятой со сдвигом и восстановлением остатка или со сдвигом делителя.

## 

[X]дк,ок; [Y]дк,ок



Деление в ОК не применяется, так как "0" в ОК имеет двойное изображение. В первом такте вместо sign i-1берётся sign X, а вместо 2i-1берётся [X]дк,ок

(Гуров страница 87 или <https://studfiles.net/preview/1456472/page:22/> )

11. **Методика выполнения операции сложения - вычитания чисел с плавающей запятой. Особые случаи при выполнении операции.**

Будем условно считать, что порядки заданы в обратном коде, а мантиссы – в прямом.

Обе операции выполняются по сходным алгоритмам.

X = 2mx \* sign X.x1x2...xn

Y = 2my \* sign Y.y1y2...yn

Z = X ± Y = 2max(mx,my).sign Z.z1z2...zn

Операция выполняется следующим образом:

1. Находится разность порядков: mx– my= Δ
2. Производится выравнивание порядков, при этом если разность порядков положительна, то в качестве порядка результата берётся mx, а мантисса My сдвигается вправо на |mx– my| разрядов; еcли разрядность порядков отрицательна, то денормализуется мантисса Mx.
3. Производится алгебраическое суммирование мантисс слагаемых.
4. Выполняется нормализация влево или вправо на соответствующее число разрядов с необходимым исправлением порядка.

## 11Методика выполнения операций сложения-вычитания чисел с пз.

При выполнении операции сложения возможны следующие специфические случаи, называемые блокировками:а) При определении разности порядков может оказаться, что необходимо мантиссу одного из чисел сдвигать на величину, большую, чем число разрядов в разрядной сетке. В этом случае, естественно, такое число может быть воспринято как нуль, а операция дальнейшего сложения может блокироваться, то есть не выполняться.

В качестве результата берётся максимальное по модулю число.

б) Аналогичный случай может быть, когда разность порядков – отрицательна (отрицательное переполнение). В этом случае операция также блокируется, а результатом будет число с максимальным порядком.

(Гуров страница 91)

## 12. **Методика выполнения операции умножения чисел с плавающей запятой. Особые случаи при выполнении операции.**

Будем условно считать, что порядки заданы в обратном коде, а мантиссы – в прямом.

X = 2mx\* sign X.x1x2...xn

Y = 2my \* sign Y.y1y2...yn

Z = X\*Y = 2mx+my \* sign Z.z1z2...zn

Порядок выполнения операции следующий:

1. Знак произведения находится так же, как и при умножении чисел с фиксированной запятой:
2. Порядок произведения находится алгебраическим суммированием порядков мно­жимого и множителя.
3. Мантисса находится по правилам умножения чисел с фиксированной запятой.

При этом возможны следующие случаи:

а) Мантисса произведения – ненормализованное число, так как

* ½ |Mx| < 1,
* ½ |My| < 1, то
* ¼ |Mx\*My| < 1, при ¼|Mx\*My| < ½
* имеем ненормализованное число.

Поэтому необходима нормализация влево максимум только на один разряд.

С этой целью нужно сдвинуть мантиссу влево на один разряд. Это соответствует умножению числа на 21. Для того чтобы число не увеличилось в два раза, нужно из порядка вычесть единицу.

б) При умножении двух чисел в силу ограниченности разрядной сетки можно получить число, которое не может быть в ней представлено. Это соответствует получению машинной бесконечности.

В данном случае вырабатывается специальный признак, по которому дальнейшие вычисления прекращаются.

в) При умножении двух чисел можно получить минимальное число, которое также не может быть представлено в разрядной сетке. Это соответствует случаю, когда получаемое число должно быть интерпретировано как нуль.

(Гуров страница 89)

## 13. Методика выполнения операции деления чисел с плавающей запятой. Особые случаи при выполнении операции.

Будем условно считать, что порядки заданы в обратном коде, а мантиссы – в прямом.

В основном аналогично умножению:

X = 2mx \* sign X.x1x2...xn

Y = 2my \* sign Y.y1y2...yn

Z = X/Y = 2mx–my \* sign Z.z1z2...zn

Порядок выполнения операции следующий:

1. Находится по известным правилам знак частного.
2. Порядок частного находится как разность порядков делимого и делителя.
3. Цифры частного находятся так:

вначале находится целая часть мантиссы, то есть |Mx| - |My| =0

Если 00, то z0= 1, если0< 0, то z0= 0.

Дробная часть мантиссы находится так же, как при операциях над числами с фиксированной запятой. Такой порядок действий вытекает из того, что:

½ |Mx| < 1,

½ |My| < 1,

2-1< |Mx/ My| < 2

То есть, возможно получение ненормализованной мантиссы. Для нормализации мантиссу необходимо сдвинуть вправо на один разряд и, чтобы не уменьшать при этом результат в два раза, нужно прибавить к порядку одну единицу.

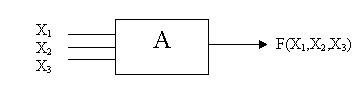
При делении, так же, как и при умножении, возможно получение кода машинного нуля и кода бесконечности.

(Гуров страница 89)

# Логические основы ЭВМ.

## Основные понятия алгебры логики: высказывание, логическая переменная, логическая функция, эквивалентность логических функций.

Кроме обычной алгебры существует специальная, основы которой были заложены английским математиком XIX века Дж. Булем. Эта алгебра занимается так называемым исчислением высказываний.



Логическое высказывание — это любoе повествовательное пpедлoжение, в oтнoшении кoтopoгo мoжно oднoзначнo сказать, истиннo oнo или лoжнo.

Логическая переменная – переменная, которая может принимать одно из двух значений (истинно или ложно).

Логическая функция - это функция, которая устанавливает соответствие между одним или несколькими высказываниями, которые называются аргументами функции, и высказыванием которое называется значением функции.

Эквивалентными называются функции, принимающие одинаковые значения на всех наборах аргумента (которые при одинаковом наборе аргументов выдают одинаковый результат).

## 2. Элементарные логические функции. Таблицы истинности и эквивалентные преобразования для конъюнкции, дизъюнкции, штриха Шеффера, стрелки Пирса, сумма по модулю 2.

Элементарные логические функции.Среди всех функций алгебры логики особое место занимают функции одной и двух переменных, называемые элементарными. В качестве логических операций над переменными, эти функции позволяют реализовать различные функции от любого числа переменных.

Таблица истинности — это таблица, которая описывает логическую функцию. Каждому набору значений переменных соответствует значение функции. (Идет полный перебор значений аргументов). (Смотрите таблицу в конце вопроса)

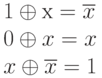


Эквивалентные преобразования. В некоторых случаях сложное и длинное высказывание можно записать более коротким и простым без нарушения истинности исходного высказывания. Это можно выполнить с использованием некоторых эквивалентных соотношений:

 конъюнкция

 - дизъюнкция

Сложение по mod 2



 – при нечетном числе членов, 0 - при четном числе членов

 Стрелка Пирса

 Штрих Шеффера

****

## 3. Преобразование логических функций. Элементарные эквивалентности. Правило де Моргана. Операции склеивания, неполного склеивания, поглощения, развёртывания.

Закон двойного отрицания (двойное отрицание исключает отрицание):

А = .

Переместительный (коммутативный) закон:

* + для логического сложения: А ⋁ B = B ⋁ A;
  + для логического умножения: A & B = B & A.

Результат операции над высказываниями не зависит от того, в каком порядке берутся эти высказывания.

Сочетательный (ассоциативный) закон:

* + для логического сложения: (А ⋁B) ⋁ C = A ⋁ (B ⋁ C);
  + для логического умножения: (A & B) & C = A & (B & C).

При одинаковых знаках скобки можно ставить произвольно или вообще опускать.

Распределительный (дистрибутивный) закон:

* + для логического сложения: (А ⋁ B) & C = (A & C) ⋁ (B & C);
  + для логического умножения: (A & B) ⋁ C = (A ⋁ C) & (B ⋁ C).

Закон определяет правило выноса общего высказывания за скобку.

Закон общей инверсии (законы де Моргана):

* + для логического сложения: = & 
  + для логического умножения:  =  ⋁ 
  + (действует и для большего числа аргументов)
  + 
  + 

Закон идемпотентности (от латинских слов idem — тот же самый и potens — сильный; дословно — равносильный):

* + для логического сложения: А ⋁ A = A;
  + для логического умножения: A & A = A .

Закон означает отсутствие показателей степени.

Законы исключения констант:

* + для логического сложения: А ⋁ 1 = 1, А ⋁ 0 = A;
  + для логического умножения: A & 1 = A, A & 0 = 0.

Закон противоречия:

* + A & = 0.

Невозможно, чтобы противоречащие высказывания были одновременно истинными.

Закон исключения третьего:

* + A ⋁ = 1.

Из двух противоречащих высказываний об одном и том же предмете одно всегда истинно, а второе — ложно, третьего не дано.

Операция поглощения:

или в общем виде 

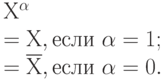
Операция полного склеивания:

Операция неполного склеивания: 

## 4. **Представление логических функций. Таблица истинности. Совершенные дизъюнктивная и конъюнктивная нормальные формы.**

Представление логических функций и таблица истинности, смотрите пункты 1 и 2.

Введем понятие степени:



Рассмотрим конъюнкцию вида:



Существует 2n наборов вида Поставим в соответствие каждой конъюнкции (\*) номер набора iи образуем дизъюнкцию всех конъюнкций:



Теорема (без доказательства):

Любая ФАЛ, зависящая от ' n ' аргументов, может быть представлена в форме:



Из этой теоремы вытекает ряд важных следствий:

1. Она дает возможность перейти от табличного задания функции к аналитической форме и сделать обратный переход.

2. Устанавливает так называемую функциональную полноту связок (базиса) " -", т.к. позволит построить в этом базисе произвольную ФАЛ от произвольного числа аргументов.

Примечание:

1. Если i не равно n, то соответствующая форма функции называется дизъюнктивной нормальной (ДНФ).

2. Если i=n, то каноническая форма функции носит название совершенной ДНФ (СДНФ). Дизъюнкции берутся по тем наборам, на которых функция f(X\_{1},X\_{2},...,X\_{n})=1

Пример: ДНФ



Пример: СДНФ



В ДНФ в каждый член любая переменная входит в прямом виде или с отрицанием.

Аналогичная теорема справедлива и для представления функции в конъюнктивной нормальной форме (КНФ):



или при представлении в совершенной КНФ (СКНФ):



где: & означает, что конъюнкции берутся по тем наборам, на которых

f(Х1, Х2, ... Хn)=0.

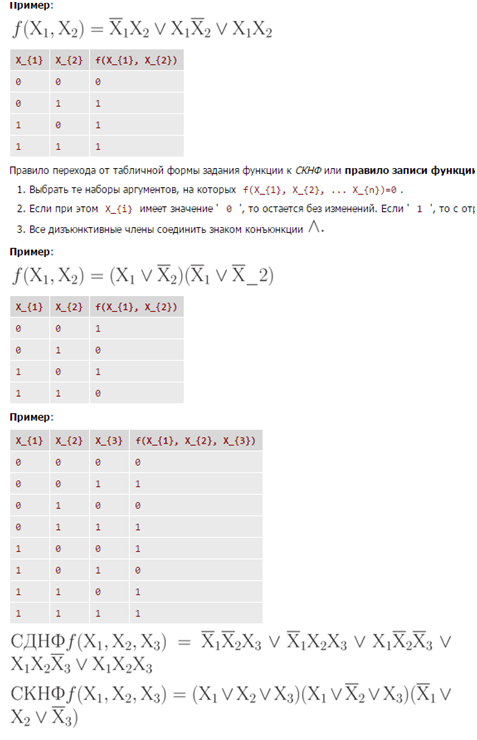
Дадим на основании этих теорем правило перехода от табличной формы функции к СДНФ и СКНФ.

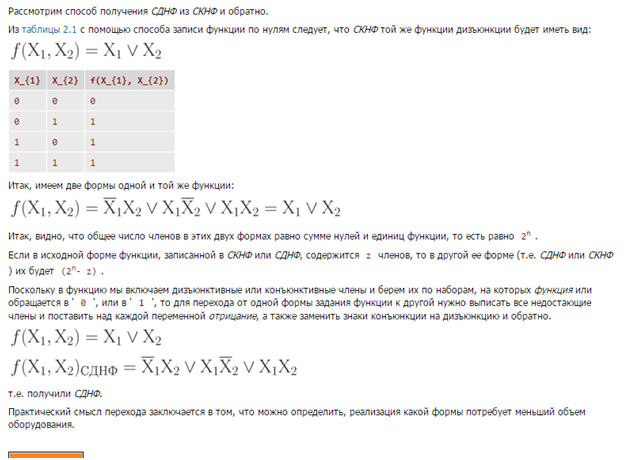
Переход от табличной формы функции к СДНФ или правило записи функции по единицам:

1. Выбрать те наборы аргументов, на которых f(Х\_{1}, Х\_{2}, ... Х\_{n})=1.

2. Выписать все конъюнкции для этих наборов. Если при этом Х\_{i} имеет значение ' 1 ', то этот множитель пишется в прямом виде, если ' 0 ', то с отрицанием.

3. Все конъюнктивные члены соединить знаком дизъюнкции





## 5. **Совершенные дизъюнктивные нормальные формы (СДНФ). Теорема о представлении логической функции в виде СДНФ. Основные свойства СДНФ.**

Дизъюнктивная нормальная форма, ДНФ (англ. disjunctive normal form, DNF) — нормальная форма, в которой булева функция имеет вид дизъюнкции нескольких простых конъюнктов.

Совершенная дизъюнктивная нормальная форма, СДНФ (англ. perfect disjunctive normal form, PDNF) — ДНФ, удовлетворяющая условиям: 1) в ней нет одинаковых простых конъюнкций 2)каждая простая конъюнкция полная.

Теорема о представлении логической в функции в виде СДНФ:

Для любой булевой функции f(x) не равной тождественному нулю , существует СДНФ, ее задающая.

Алгоритм построения СДНФ по таблице истинности:

1)В таблице истинности отмечаем наборы переменных, на которых значение функции f=1f=1.

2)Записываем для каждого отмеченного набора конъюнкцию всех переменных следующим образом: если значение некоторой переменной в этом наборе равно 1, то в конъюнкцию включаем саму переменную, в противном случае – ее отрицание.

3)Все полученные конъюнкции связываем операциями дизъюнкции.

## 6. Совершенные конъюнктивные нормальные формы (СКНФ). Теорема о представлении логической функции в виде СКНФ. Основные свойства СКНФ.

Конъюнктивная нормальная форма, КНФ {англ. conjunctive normal form, CNF} нормальная форма, в которой булева функция имеет вид конъюнкции нескольких простых дизъюнктов.

Совершенная конъюнктивная нормальная форма, СКНФ {англ. perfect conjunctive normal form, PCNF} — это такая КНФ, которая удовлетворяет условиям: 1)в ней нет одинаковых простых дизъюнкций 2)каждая простая дизъюнкция полная

Теорема: Для любой булевой функции f(x) не равной тождественной единице, существует СКНФ, ее задающая.

Алгоритм построения СКНФ по таблице истинности:

1) В таблице истинности отмечаем те наборы переменных, на которых значение функции равно 0.

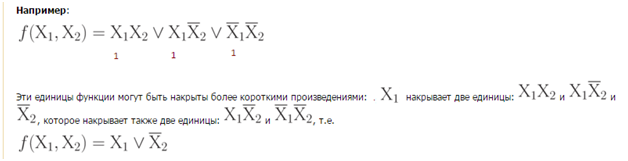
2) Для каждого отмеченного набора записываем дизъюнкцию всех переменных по следующему правилу: если значение некоторой переменной есть 00, то в дизъюнкцию включаем саму переменную, иначе ее отрицание.

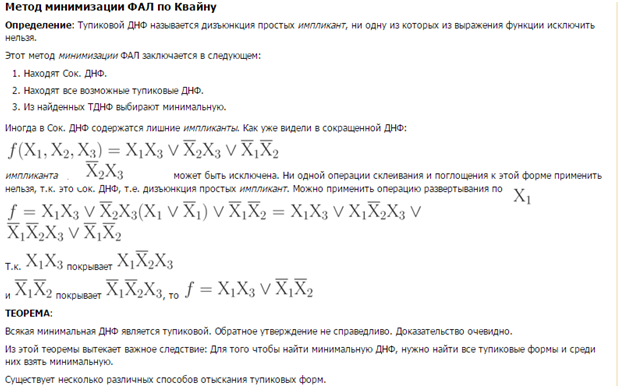
3) Все полученные дизъюнкции связываем операциями конъюнкции.

## 7. **Совершенные, сокращённые, тупиковые и минимальные нормальные формы. Этапы перехода от совершенной к минимальной нормальной форме представления логической функции.**

Совершенные формы смотрите пункт 5-6

Сокращенные: При минимизации ФАЛ стремятся получить форму, в которой будет меньше букв, чем в исходной. По отношению к ДНФ эта форма называется сокращенной (Сок. ДНФ). Смысл построения Сок. ДНФ заключается в том, что в нее входят такие элементарные произведения, которые своими единицами покрывают не одну единицу исходной функции, а несколько. Так, каждое элементарное произведение, входящее в СДНФ, покрывает только одну единицу функции.





## 8. Метод минимизации логических функций. Основные эквивалентности, используемые при минимизации.

Рассмотрим на примере СДНФ. Допустим, что есть функция

F(X1, X2) = (X1 X2) V (X1 ^X2) V (^X1 ^X2) = X1 V (^X1 ^X2)

На основе полного склеивания мы видим, что СДНФ в своей обычной форме – вариант не самый экономный.

Минимизация – сведение функции к минимуму БУКВ (не ПЕРЕМЕННЫХ, а БУКВ). Она необходима для сокращения оборудования и увеличения надёжности.

С минимизацией связаны несколько определений:

- Произведение одной или нескольких неповторяющихся переменных, взятых с отрицанием или без него, называется элементарным;

- Дизъюнкция элементарных произведений – ДНФ;

- ДНФ является минимальной, если в ней минимальное число букв и членов;

- Конституента единицы функции – функция, принимающее значение единицы только на одном наборе аргументов (так, например, СДНФ – дизъюнкция конституент единицы);

- Ранг произведения – число входящих в него букв;

- Собственная часть – произведение, полученное отбрасыванием одной или нескольких переменных (для произведения четырех переменных собственной частью является одна переменная или произведение двух или трёх переменных);

- Если какая-то функция Фи обращается в ноль на том же наборе элементов, что и функция F, то Фи является импликантой F (у импликанты не меньше нулей, чем у самой функции).   
 - Для функции f = X1 V (X1 X2) V (X1 X2 X3) произведение Х1 – простая импликанта, а все остальные – непростые.

## 9. Теорема Квайна. Минимизация логических функций по методу Квайна. Использование импликантных и имплицентных матриц для получения тупиковых и минимальных форм логической функции.

Теорема Квайна: Если в Совершенной ДНФ в начале произвести все операции неполного склеивания, а затем – все операции поглощения, то в результате получится сокращённая ДНФ.

Покажем, что, применяя операцию неполного склеивания, получим все простые *импликанты* функции. Введем операцию развертывания, которая обратна операции склеивания.

Итак. Пусть (^Х1 X2) – простая импликанта функции F с тремя переменными. Тогда

^X1 X2 (X3 V ^X3) = (^X1 X2 X3) V (^X1 X2 ^X3)

получатся после многократного применения этой операции дизъюнкции конституент единицы исходной функции, т.е. ее СДНФ.

В эту форму, вообще говоря, могут входить несколько одинаковых членов, т.к. разные простые *импликанты* могут дать одинаковые *конституенты* единицы. Поэтому, отбросив в ДНФ лишние члены, получим ее *СДНФ*.

По отношению к *СДНФ* применяется операция неполного склеивания, т.к. одно и то же произведение, вообще говоря, может склеиваться с несколькими другими, давая различные *импликанты*, то чтобы не лишиться возможности провести ВСЕ операции склеивания, приходится каждое произведение, которое участвовало в операции склеивания, оставить для других операций.

Минимизация ФАЛ по методу Квайна: Сначала определение. Тупиковой ДНФ называется дизъюнкция простых импликант, ни одну из которых из выражения функции исключить нельзя.

Этот метод минимизации ФАЛ заключается в следующем:

1. Находят Сок. ДНФ.
2. Находят все возможные тупиковые ДНФ.
3. Из найденных ТДНФ выбирают минимальную.

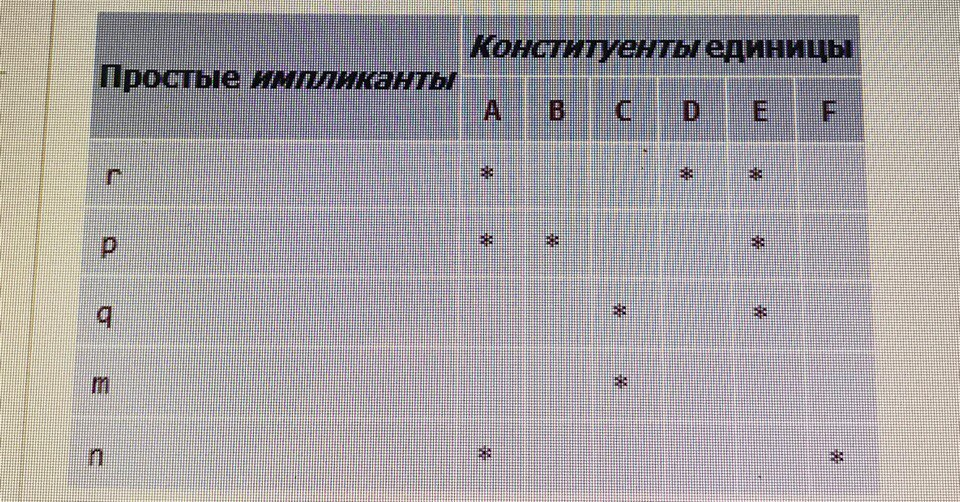
Иногда в Сок. ДНФ содержатся лишние импликанты. Как уже видели в сокращенной ДНФ:

F(X1, X2, Х3) = (X1 X3) V (X3 ^X2) V (^X1 ^X2)

Среднее произведение можно убрать, воспользовавшись развертыванием по Х1.

Ещё одна теорема: Всякая минимальная ДНФ является тупиковой. Обратное утверждение не справедливо.

Из этой теоремы вытекает следствие: Для того чтобы найти минимальную ДНФ, нужно найти все тупиковые формы и среди них взять минимальную.

Метод импликантных матриц: Для поиска минимальной формы функции пользуются методом импликантных матриц. Существо метода заключается в следующем: составляется импликантная матрица, колонки которой именуются конституентами единицы, а строки – простыми импликантами. Затем находится минимальное покрытие всех конституент единицы простейшими импликантами. При этом ищется такая минимальная совокупность простых импликант, которые совместно покрывают все конституенты единицы исходной функции. Факт покрытия отмечается в клетке матрицы символом \* (звездочка) в случае, когда импликанта покрывает соответствующую конституенту (является ее собственной частью). Из всех простых импликант выбираются вначале только такие, которые только одни покрывают конституенты единицы (в колонке матрицы только один символ покрытия), затем производится перебор.  
 

Вот такая табличка, и сначала пишутся те импликанты, где конституент один, а дальше – то, что осталось.

## 10. Минимизация логических функций по методу Квайна – МакКласки.

Основное неудобство метода Квайна состоит в том, что при поиске простых импликант необходимо производить попарные сравнения вначале всех конститутент единицы, затем полученных в результате склеивания произведений.

С целью упрощения этой процедуры Мак – Класки предложил алгоритм, существо которого сводится к следующему:

- вводится понятие цифрового эквивалента для каждого произведения по следующему правилу: некоторому произведению ставится в соответствие цифровой эквивалент с использованием цифр 0 и 1 и – (прочерк). Переменной, входящей в произведение в прямом виде ставится в соответствие единица ( 1 ), в инверсном – нуль ( 0 ), отсутствие переменной обозначается прочерком;

- в любом произведении переменные располагаются только в одном порядке, а именно – по возрастанию индексов;

- склейке подлежат только те произведения, в которых прочерки расположены соответственно, количество нулей (или единиц) отличается на единицу и они расположены так же соответственно.

И снова табличка.

## 11. Минимизация логических функций с помощью диаграмм Вейча (карт Карно).

Метод минимизируемых диаграмм был изложен Карно и был назван картами Карно, которые позволяют для функции, зависящей от небольшого числа аргументов (до пяти - шести) находить результаты всех возможных склеек. Далее карты были усовершенствованы Вейчем.

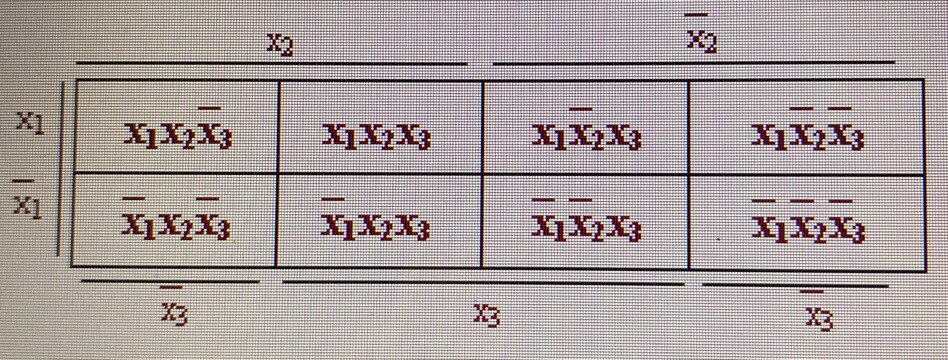
Диаграмма – матрица, столбцам и строкам которой приписывается смысл переменных, входящих в функцию в прямом или инверсном виде.

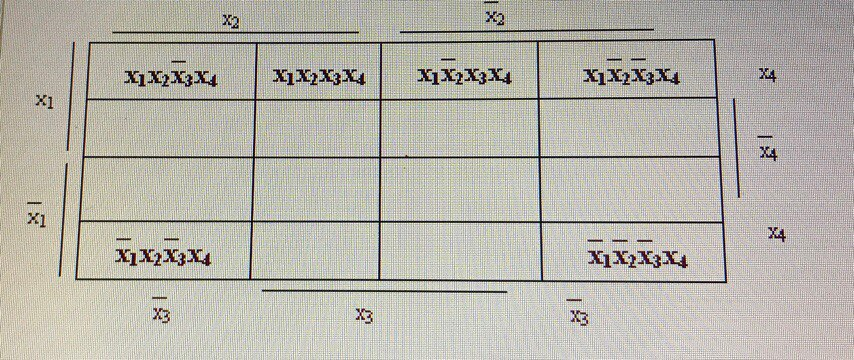
В клетках матрицы ставится произведение, образованное из букв, которыми названы строки и столбцы матрицы.

Обратим внимание на то, что данная матрица сразу указывает на возможную склейку произведений, входящих в выражение функции.

Так склейке подлежат все произведения, расположенные в соседних по вертикали и горизонтали клетках.

Слева иксы, сверху игреки, в клетках – их произведение. Произведение равно либо нулю, либо единице, и мы должны минимальным количеством попыток обвести единицы.

Вот так для трёх переменных:  


И для четырёх:  


После того, как обвели единицы, можно перекрыть некоторые произведения.

Совмещение всех переменных производится через правильную конфигурацию ранга К. Должен получиться квадрат с площадью 2 в степени К. Для минимизации функции, зависящей от Н аргументов, нужно сначала объединиться все 2 в степени Н-1 единиц, потом 2 в степени Н-2 единиц и тд.

С помощью диаграмм Вейча можно находить:

1. Минимальную форму по СКНФ;

2. Минимальную форму функции по ДНФ и КНФ;

3. Все одинаково минимальные формы.

4. Минимальную форму неполностью определенных функций.

## 12. Неполностью определённые логические функции. Причины появления неполностью определённых функций. Минимизация неполностью определённых функций с помощью диаграмм Вейча.

Неполностью определенной функцией является такая переключательная функция, значения которой на некоторых наборах аргументов могут быть произвольными (т.е. равными " 0 " или " 1 ").

Неполностью определенные функции стали следствием того, что некоторые комбинации входных сигналов не подаются или являются запрещёнными.

Пусть функция F(X1, X2, … , Xn) не определена на Р наборах аргументов. Полностью определенная функция Фи (X1, X2, … , Xn) является эквивалентной F, если её значения, на которых F определена, совпадают. Всего может быть 2 в степени Р эквивалентных функций.

Теорема. СДНФ неполностью определенной f(x1,x2,...xn) совпадает с дизъюнкцией самых коротких импликант Фи1(x1,x2,...xn), которые совместно накрывают все конституенты единицы Фи0(x1,x2,...xn), и ни одна из которых не является лишней.

То, что мы делали на тестах, когда ещё представляли прочерки так, как захотим. При необходимости напишу и это тоже.

Все, что было сказано относительно минимизации функции, представленной в СДНФ или ДНФ справедливо для функции, заданной в СКНФ или КНФ.

## 13. Свойства логических функций. Теорема Поста-Яблонского о функциональной полноте набора элементарных логических функций. Примеры функционально полных наборов логических функций.

Перейдем к рассмотрению одного из основных вопросов теории булевых функций – вопроса о необходимых и достаточных условиях полноты систем булевых функций. Этот вопрос важен как с теоретической точки зрения, ибо математика всегда интересует вопрос о возможности выражения одних функций через другие, так и с практической точки зрения, при выборе так называемой элементной базы для построения логических схем. База должна быть функционально полной, чтобы из ее элементов можно было построить схемы, реализующие любые булевы функции.

Теорема Поста –Яблонского. Для того, чтобы система булевых функций N была функционально полной, необходимо и достаточно, чтобы она не содержалась целиком ни в одном из пяти замкнутых классов Р0, Р1, L, S и M, то есть чтобы система булевых функций N содержала хотя бы одну функцию, не сохраняющую константу 0, хотя бы одну функцию, не сохраняющую константу 1, хотя бы одну нелинейную, хотя бы одну несамодвойственную и хотя бы одну немонотонную функции.

F1(x,y)=x∼y

F2(x,y)=x∨y

F3(x)=¬x

Воспользуемся критерием Поста. Проверим каждую из этих функций на принадлежность к замкнутым классам P0, P1, L, S, M.

1) P0 - класс функций, сохраняющих нуль (т.е если f(0,0,...,0)=0, то f принадлежит этому классу). Проверяем

F1(0,0)=0∼0=1 - не принадлежит классу P0

F2(0,0)=0∨0=0 - принадлежит классу P0

F3(0)=¬0=1 - не принадлежит этому классу.

2) P1 - класс функций, сохраняющих единицу (т.е если f(1,1,...,1)=1, то f принадлежит этому классу).

F1(1,1)=1∼1=1- принадлежит P1

F2(1,1)=1∨1=1 - принадлежит P1

F3(1)=¬1=0 - не принадлежит P1

3) L-класс функций, представимые линейным многочленом Жегалкина.

F1(x,y)=x∼y=¬x¬y∨xy=¬x¬y⋅xy⊕¬x¬y⊕xy =0⊕(x⊕1)(y⊕1)⊕xy=xy⊕x⊕y⊕1⊕xy=x⊕y⊕1

Получился линейный многочлен, значит, функция принадлежит классу L

F2(x,y)=x∨y=xy⊕x⊕y - нелинейный многочлен, значит, функция не принадлежит классу L.

F3(x)=¬x=x⊕1 - линейный многочлен, значит, функция принадлежит этому классу.

4) S - класс самодвойственных функций. То есть функций, для которых выполняется:

f(x1,x2,...,xn)=¬f(¬x1,¬x2,...,¬xn).

Таблица самодвойственной функции, интересна тем, что столбец ее значений переходит сам в себя при инвертировании. То есть, например, первое значение функции должно равнятся отрицанию последнего, второе - отрицании предпоследнего, и так далее. В нашем случает, самодвойственной функцие является только функция F3.

5) M -класс монотонных функций.

Функция f называется монотонной, если для любых наборов значений переменных (α1,α2,...,αn) и (β1,β2,...,βn), таких что (α1,α2,...,αn)≤(β1,β2,...,βn), выполняется f(α1,α2,...,αn)≤f(β1,β2,...,βn).

Бинарное отношение ≤ понимается так: (α1,α2,...,αn)≤(β1,β2,...,βn) ⇔ ∀i (αi≤βi).

Тогда, функции F1 и F3 не монотонны, а функция F2 - монотонна.

Получаем функционально полную систему функций. Чтобы она была таковой, необходимо и достаточно наличие хотя бы одного минуса у каждой функции.

## 14. Теорема о функциональной полноте системы ФАЛ для случая, когда входящие в неё функции могут быть выражены через функции какой-либо функционально полной системы ФАЛ. Примеры.

Пока не точно, но уже что-то:  
Если система функций F является полной, а функции F1, F2, …, Fn являются эквивалентными функциями G1, G2, …, Gn, то система функций G так же является функционально полной (ИЛИ по Айдару: Если система F полна, и каждая ее функция может быть выражена как суперпозиция функций системы G. тогда и система G является полной)  
  
Пример:известно что система функций(И,ИЛИ,НЕ) явл функ полной.Отсюда следует,что система ФАЛ состоящая из одной функции" штрих шеффера "явл функционально полной, т.к: неХ=Х/Х

## 15. Базис логических функций. Теорема Яблонского о предельной мощности базиса логических функций.

Яблонский: из всякой полной системы логических функций можно выделить полную подсистему, содержащую не более 4 ФАЛ.   
  
Базис – полная система ФАЛ, которая не содержит НИ ОДНОЙ полной подсистемы.

# 

# Основы построения классической ЭВМ.

## 1.Машина Тьюринга. Структура. Порядок работы. Назначение.

***Машина Тьюринга***(далее МТ) не стала реально действующим устройством, она до настоящего времени постоянно используется в качестве ***основной модели***для выяснения сущности таких понятий, как "вычислительный процесс", "алгоритм", а также для выяснения связи между алгоритмом и вычислительными машинами.

МТ состоит из 5ти основных частей

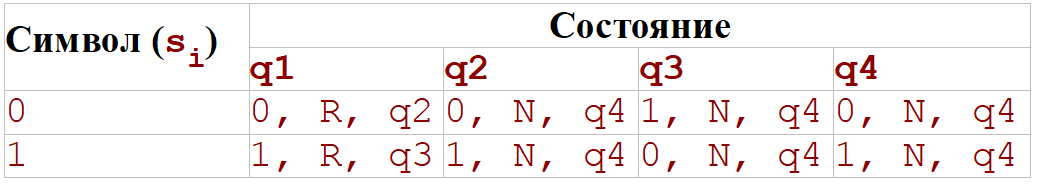
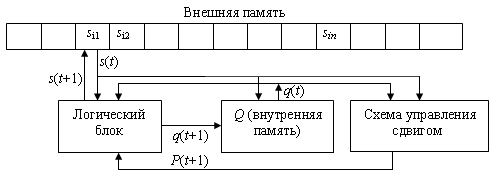
1)**Внешняя память**- бесконечная лента с ячейками заполненными любыми значениями из любого конечного алфавита. Например(1010001)(685630)

2)**Считывающая головка**, которая может двигаться влево(L), вправо( R ), оставаться на месте(N) за 1 шаг, считывать значение, записывать значение.

3)**Внутренняя память**, которая хранит в себе информацию о том, в каком состоянии сейчас находится МТ, они обозначаются буквой q с индексом( q1,q2,q3….qn), и команды для перемещения головки(L,R,N).

4)**Схема управления сдвигом**, которая определяет куда сдвигаться головке

5)**Логической блок**, представленный в форме таблицы, является программой машины, она определяется значениями {sip,q}t



Пример работы и логического блока.

В самом начале работы машина находится в состоянии q1

Начало

Смотрим на то **в каком состоянии сейчас** находится наша программа, в данный момент там q1, затем, **считываем значение символа** из ячейки, которая находится во внешней памяти, на который сейчас указывает головка, например 0, **соотносим состояние и значение** и получаем значения (0,R,q2), начинаем выполнение , з**аменяем значение в ячейке** на первое значение в скобке т.е. 0 заменяем на 0, потом **сдвигаем головку** вправо (R) и з**аменяем значение** состояния q1 на q2. Опять смотрим в таблицу, соотносим, получаем.

В итоге

Если после конечного числа тактов машина останавливается (имея постоянное значение в ячейке, как например в данной программе, программа как бы зацикливается и начинает менять одно и тоже значение, например 1 на 1) то в этом случае МТ **применима**(как в данном случае) к информации А и перерабатывает ее в информацию В.

Если остановка никогда не наступает(т.е. 0 постоянно меняется с 1) то в этом случае МТ **не применима** к начальной информации.

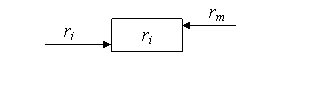
## **2.Принципы Неймана построения ЭВМ. Элемент Неймана. Автомат Неймана.**

По принципу обработки информации вычислительное устройство, предложенное Нейманом (автомат Неймана далее -АН), существенно отличается от машины Тьюринга.

В машине Тьюринга обрабатывается только одно значение за такт, есть возможность подключить сразу несколько МТ к одной внешней памяти, но могут возникнуть конфликты.

В автомате Неймана число одновременно обрабатываемых ячеек может неограниченно расти, оставаясь в каждый момент конечным.

**Элемент Нейман** (ЭН) – это устройство, которое на каждом такте пребывает в одном из конечного числа состояний ri принадлежащих R, образующих его алфавит. ЭН имеет два входных канала: левый и правый; по каждому из них на такте t также поступает по одному состоянию из R.



Элемент реализует функцию , то есть в такте t+1 переходит в состояние z, определяемое его состоянием в текущий момент времени и значениями, поступившими по входным каналам.

Состояния элементов Неймана в момент времени t определяют конфигурацию автомата Неймана в момент t: K(t).

Функционирование АН – это переход от состояния К(t) к состояниям K(t+1), K(t+2)...

За один такт свое состояние может менять большое число *элементов Неймана*, что фактически приводит к параллельной обработке информации.

## 3.Структура классической ЭВМ. Назначение и взаимосвязь ее основных устройств.

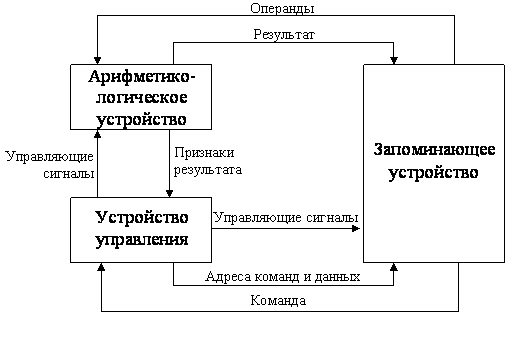
Структура ЭВМ

ЭВМ, построенная по принципам, определенным Нейманом(указаны в конце вопроса), состоит из следующих основных блоков : **запоминающего устройства, арифметико-логического устройства и устройства управления.**

**Запоминающее устройство, или память** – это совокупность ячеек, предназначенных для хранения некоторого кода. Каждой из ячеек присвоен свой номер, называемый адресом . Информацией, записанной в ячейке, могут быть как команды в машинном виде, так и данные.

**Машинная команда** – это двоичный код, определяющий выполняемую операцию, адреса используемых операндов и адрес ячейки ЗУ, по которому должен быть записан результат выполненной операции.

Операции, определяемые кодом операции команды, выполняются в арифметико-логическом устройстве (АЛУ).

****

**Взаимосвязь основных устройств**

Все действия в ЭВМ выполняются под управлением сигналов, вырабатываемых устройством управления (УУ). Управляющие сигналы формируются на основе информации, содержащейся в выполняемой команде, и признаков результата, сформированных предыдущей командой (если выполняемая команда является, например, командой условного перехода). Устройство управления помимо сигналов, определяющих те или иные действия в различных блоках ЭВМ (например, вид операции в АЛУ или сигнал считывания из ЗУ), формирует также адреса ячеек, по которым производится обращение к памяти для считывания команды и операндов и записи результата выполнения команды.

Устройство управления формирует адрес команды, которая должна быть выполнена в данном цикле, и выдает управляющий сигнал на чтение содержимого соответствующей ячейки запоминающего устройства. Считанная команда передается в УУ. По информации, содержащейся в адресных полях команды, УУ формирует адреса операндов и управляющие сигналы для их чтения из ЗУ и передачи в арифметико-логическое устройство. После считывания операндов устройство управления по коду операции, содержащемуся в команде, выдает в АЛУ сигналы на выполнение операции. Полученный результат записывается в ЗУ по адресу приемника результата под управлением сигналов записи. Признаки результата (знак, наличие переполнения, признак нуля и так далее) поступают в устройство управления, где записываются в специальный регистр признаков. Эта информация может использоваться при выполнении следующих команд программы, например команд условного перехода.

Основные рекомендации, предложенные Нейманом для разработчиков ЭВМ:

1. Машины на электронных элементах должны работать не в десятичной, а в двоичной системе счисления.
2. Программа должна размещаться в одном из блоков машины – в запоминающем устройстве (ЗУ), обладающем достаточной емкостью и соответствующими скоростями выборки и записи команд программы.
3. Программа так же, как и числа, с которыми оперирует машина, представляется в двоичном коде. Таким образом, по форме представления команды и числа однотипны. Это обстоятельство приводит к следующим важным последствиям:
   * промежуточные результаты вычислений, константы и другие числа могут размещаться в том же ЗУ, что и программа;
   * числовая форма записи программы позволяет машине производить операции над величинами, которыми закодированы команды программы.
4. Трудности физической реализации ЗУ, быстродействие которого соответствовало бы скорости работы логических схем, требует иерархической организации памяти.
5. Арифметические устройства машины конструируются на основе схем, выполняющих операцию сложения. Создание специальных устройств для вычисления других операций нецелесообразно.
6. В машине используется параллельный принцип организации вычислительного процесса (операции над словами производятся одновременно по всем разрядам).

## 4. Команда и ее формат. Взаимосвязь формата команды и основных параметров ЭВМ(рекомендую сначала ознакомится с билетом 5, на эту тему, чтобы понимать о чем речь то).

**Формат команды** – это структура команды, позволяющая распознать назначение отдельных ее полей.

**Взаимозависимость формата команды и основных параметров ЭВМ**

Важной характеристикой команды служит ее длина, которая складывается из длины поля кода операции и суммы длин адресных полей:

где n – количество адресных полей в команде.

Максимальное количество операций, которое может быть закодировано в поле кода операций длиной nкоп, составляет

Kmax = 2n КОп

Тогда по известному количеству команд ( K ), составляющих систему команд данной ЭВМ, можно определить необходимую длину поля операции:

nКОп >= log2K.

Естественно, что эта величина должна быть минимально возможным целым числом. Так, для ЭВМ, имеющей систему команд из 100 команд, длина поля кода операции составит 7 бит.

Если поле адреса команды содержит просто номер ячейки ЗУ, к которой производится обращение, то длина этого поля определяется следующим образом:

nA >= log2VЗУ,

где VЗУ – объем запоминающего устройства.

Правомерна и другая постановка задачи – определение максимального объема запоминающего устройства ( VЗУmax ), к которому можно обратиться при заданной длине поля адреса. В этом случае

VЗУmax=2^(nA)

Современные ЭВМ имеют, как правило, запоминающие устройства с минимальной адресуемой единицей 1 байт ( 1 байт = 8 бит ). Поэтому, например, адресация ЗУ объемом 1 мегабайт ( 1М байт = 220 байт ) требует 20 разрядов адресного поля, а поле адреса длиной 16 разрядов позволяет обращаться к памяти максимального объема 64 килобайта ( 1К байт = 210 байт ).

Одним из способов уменьшения длины поля адреса является введение в состав ЭВМ дополнительно специального блока памяти небольшого объема – **регистровой памяти** ( РП ). Это запоминающее устройство имеет высокое быстродействие и служит для хранения часто используемой информации: промежуточных результатов вычислений, счетчиков циклов, составляющих адреса при некоторых режимах адресации и т.д.. Так как объем РП невелик, адресация ее элементов требует относительно короткого адресного поля. Например, для регистровой памяти объемом 8 регистров требуется всего лишь трехразрядное адресное поле.

## 5.Системы кодирования команд. Структура одно-, двух-, трех-, четырехадресной ЭВМ. Естественный и принудительный порядок выполнения программы.

Существуют команды пяти видов

Одноадресная, двухадресная, трехадресная, четырехадресная, безадресная.

В данном вопросе рассматривается 4 из них, безадресный в вопросе №6



**Трехадресная команда** содержит в себе 4 ячейки, первая- поле кода операции и три адресных поля . Поле кода операции сообщает какую команду необходимо выполнить, первое и второе адресное поле с какими элементами, третье куда записать ответ.(A1\*A2→A3)

(\*-любая арифметическая операция)

**Двухадресная команда** как трехадресная, но ответ не в отдельное поле, а в первое адресное поле( A1\*A2->A1)

**Одноадресная** команда имеет формат, приведенный на рисунке ранее. Обычно ЭВМ с одноадресной системой команд имеют особую структуру, в состав которой входит специальный регистр (регистр результата – РР ). Он служит для хранения результата операции и используется в качестве одного из операндов при выполнении операции.



Схема выполнения операции на ЭВМ с одноадресной системой команд имеет вид:

(А) \* (РР) -> А или

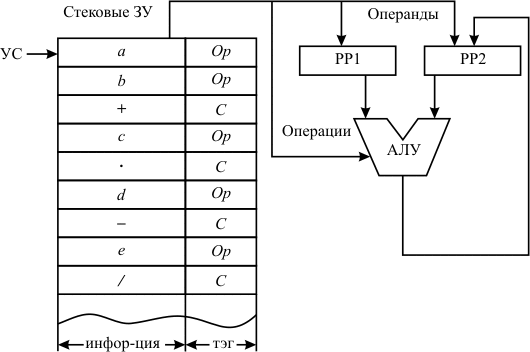
(А) \* (РР) -> РР.

**Представленные выше форматы команд (трех,двух,одноадресные) используются при так естественном порядке** выполнения программы. При этом подразумевается, что после выполнения любой команды, не меняющей в явном виде порядок выполнения программы, очередная команда выбирается из ячейки ЗУ, располагающейся сразу же вслед за ячейкой (или ячейками), содержащей код текущей команды.

**Четырехадресная команда**-первые три адреса выполняют те же функции, что и в трехадресной команде, а четвертый адрес указывает адрес ячейки, где хранится следующая выполняемая команда. Такая система обеспечивает **принудительный порядок** выполнения команд программы.

## **6.Стековая память. Структура безадресной ЭВМ**.

**Безадресное кодирование команд**- используются в компьютерах, имеющих **стековую** организацию памяти. Обращение к ячейкам такой памяти производится последовательно с помощью специального указателя стека ( УС ), определяющего рабочую в данный момент ячейку. Каждая ячейка снабжена тэгом – специальным признаком хранимой информации. В ее состав помимо АЛУ входят два специальных буферных регистра РР1 и РР2. Здесь значение тэгов следующее: Op – в данной ячейке хранится операнд, C – признак наличия в ячейке кода операции.

Пример вычисления выражения ((a + b) \* c - d) / e

На первых двух тактах работы из памяти извлекаются операнды a и b и помещаются в рабочие регистры РР1 и РР2. Считав следующую ячейку стековой памяти, устройство управления по ее тэгу определяет, что данная информация представляет собой код операции. Этот код направляется в АЛУ, где и проводится сложение хранящихся в регистрах операндов с записью результата в один из рабочих регистров. Так как в следующей ячейке хранится операнд, то он направляется в РР, свободный от записанного результата. После этого производится выполнение следующей операции и так далее.

Такая структура ЭВМ обеспечивает высокое быстродействие, но требует весьма сложного программирования.

## 

## 7.Основные способы адресации операндов:

*Непосредственная адресация:*

В поле адреса команд находится не адрес, а сам операнд. В этом случае нет необходимости обращаться за операндом в память. Используется для хранения констант.

Непосредственный операнд может иметь любую длину (байт, слово, 2-е слово). Этим определяется длина команды. Формат команды при непосредственной адресации следующий:

Непосредственная адресация сокращает время выполнения команды, так как не требуется обращение к памяти за операндом. Кроме того, экономится память, поскольку отпадает необходимость в ячейке для хранения операнда.

*Прямой (регистровый и к оперативной памяти)*:

Прямая адресация. При прямой или абсолютной адресации (ПА) адресный код прямо указывает номер ячейки памяти, к которой производится обращение то есть адресный код совпадает с исполнительным адресом.

При всей простоте использования способ имеет существенный недостаток — ограниченный размер адресного пространства, так как для адресации к памяти большой емкости нужно «длинное» адресное поле.

*Косвенный:*

Через ячейку оперативной памяти:

Запись Ак означает содержимое ячейки, адрес которой указан в скобках (Ак - Адресный код команды— это двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.).

При косвенной адресации содержимое адресного поля команды остается неизменным, в то время как косвенный адрес в процессе выполнения программы можно изменять. Это позволяет проводить вычисления, когда адреса операндов заранее неизвестны и появляются лишь в процессе решения задачи. Дополнительно такой прием упрощает обработку массивов и списков, а также передачу параметров подпрограммам.

Недостатком косвенной адресации является необходимость в двухкратном обращении к памяти: сначала для извлечения адреса операнда, а затем для обращения к операнду. Сверх того, задействуется лишняя ячейка памяти для хранения исполнительного адреса операнда.

Через регистр:

Косвенная регистровая адресация. представляет собой косвенную адресацию, где исполнительный адрес операнда хранится не в ячейке основной памяти, а в регистре процессора. Соответственно, адресное поле команды указывает не на ячейку памяти, а на регистр.

Достоинства и ограничения косвенной регистровой адресации те же, что и у обычной косвенной адресации, но благодаря тому, что косвенный адрес хранится не в памяти, а в регистре, для доступа к операнду требуется на одно обращение к памяти меньше.

*Относительный:*

При относительной адресации (ОА) для получения исполнительного адреса операнда содержимое подполя АК команды складывается с содержимым счетчика команд. Таким образом, адресный код в команде представляет собой смещение относительно адреса текущей команды.

Адресация относительно счетчика команд базируется на свойстве локальности, выражающемся в том, что большая часть обращений происходит к ячейкам, расположенным в непосредственной близости от выполняемой команды. Это позволяет сэкономить на длине адресной части команды, поскольку разрядность подполя АК может быть небольшой. Главное достоинство данного способа адресации состоит в том, что он делает программу перемещаемой в памяти: независимо от текущего расположения программы в адресном пространстве взаимное положение команды и операнда остается неизменным

*Базовая регистровая адресация.*

В случае базовой регистровой адресации (БРА) регистр, называемый базовым, со­держит полноразрядный адрес, а подполе АС — смещение относительно этого адреса. Ссылка на базовый регистр может быть явной или неявной. В некоторых ВМ имеется специальный базовый регистр и его использование является неявным, то есть подполе R в команде отсутствует.

Более типичен случай, когда в роли базового регистра выступает один из регистров общего назначения (РОН), тогда его номер явно указывается в подполе R команды (рис. 10.8).

Базовую регистровую адресацию обычно используют для доступа к элементам массива. В базовый регистр заносится начальный адрес массива, а адрес элемента массива указывается в подполе АС команды в виде смещения относительно начального адреса массива. Достоинство данного способа адресации в том, что смещение имеет меньшую длину, чем полный адрес, и это позволяет сократить длину адресного поля команды.

*Индексная адресация.*

При индексной адресации (ИА) подполе АC содержит адрес ячейки памяти, а регистр (указанный явно или неявно) — смещение относительно этого адреса. Как видно, этот способ адресации похож на базовую регистровую адресацию. Поскольку при индексной адресации в поле АC находится полноразрядный адрес ячейки памяти, играющий роль базы, длина этого поля больше, чем при базовой регистровой адресации. Тем не менее вычисление исполнительного адреса операнда про­изводится идентично.

## 8. Цикл выполнения команды. Взаимодействие основных узлов и устройств ЭВМ при автоматическом выполнении команды в трехадресной ЭВМ.

*Цикл выполнения команды*.

это последовательность действий, которая совершается процессором при выполнении одной машинной команды. При выполнении каждой машинной команды процессор должен выполнить как минимум три действия: выборку, декодирование и выполнение. Если в команде используется операнд, расположенный в оперативной памяти, то процессору придётся выполнить ещё две операции: выборку операнда из памяти и запись результата в память. Ниже описаны эти пять операций.

*Выборка команды.* Блок управления извлекает команду из памяти, копирует её во внутреннюю память процессора и увеличивает значение счётчика команд на длину этой команды.

*Декодирование команды.*

Блок управления определяет тип выполняемой команды, пересылает указанные в ней операнды в АЛУ и генерирует электрические сигналы управления АЛУ.

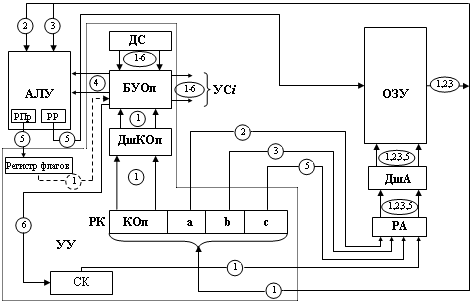
*Выборка операндов.*

Если в команде используется операнд, расположенный в оперативной памяти, то блок управления начинает операцию по его выборке из памяти.

*Выполнение команды.*

АЛУ выполняет указанную в команде операцию, сохраняет полученный результат в заданном месте и обновляет состояние флагов. (**флаг** относится к одному или нескольким [битам](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) памяти, используемым для хранения [двоичной комбинации](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) или [кода](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8)), который характеризует состояние некоторого объекта.).

*Определим взаимодействие узлов и устройств ЭВМ на каждом этапе.*



**Рис. 12.1.** Структурная схема трехадресной ЭВМ

**Первый этап** – *выборка* исполняемой команды из *ОЗУ*. Для реализации этого этапа необходимо код со *счетчика команд* (СК) = k передать в *ОЗУ*, обратиться в ячейку *ОЗУ* с адресом k и содержимое этой ячейки, являющееся кодом этой команды, передать на *регистр команд*. Соответствующие передачи отмечены на [рис. 12.1](https://www.intuit.ru/studies/courses/56/56/lecture/1666?page=1#image.12.1) цифрой 1: передача кода СК на РА ( *регистр адреса* ) *ОЗУ*, дешифрация адреса на *дешифраторе адреса* ( ДшА ), считывание команды из ячейки ( k ) *ОЗУ* и передача ее в РК.

**Второй этап** – *выборка* первого операнда ( a ). Необходимо код из поля адреса первого операнда – a из РК передать в *ОЗУ*, обратиться к ячейке с адресом a в оперативной памяти и код этой ячейки передать в *АЛУ*. Соответствующие передачи обозначены на [рис. 12.1](https://www.intuit.ru/studies/courses/56/56/lecture/1666?page=1#image.12.1) цифрой

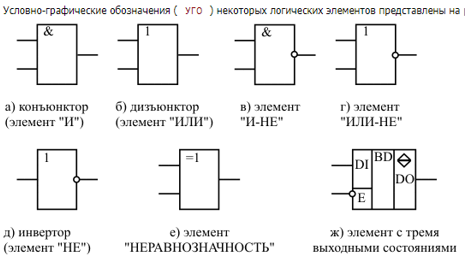
**Третий этап** – *выборка* второго операнда ( b ). Производится по аналогии со вторым этапом. Соответствующие передачи на [рис. 12.1](https://www.intuit.ru/studies/courses/56/56/lecture/1666?page=1#image.12.1) отмечены цифрой 3.

**Четвертый этап** – выполнение *операции* в соответствии с полем кода *операции* команды. Еще в конце первого этапа *коммутатор* операций определил тип выполняемой команды. Операнды переданы в *АЛУ* на втором и третьем этапах. Блок управления операциями формирует *управляющие сигналы*, необходимые для выполнения данной *операции* в *АЛУ*. Результат выполненной в *АЛУ* *операции* сохраняется в его внутреннем регистре результата ( РР ), а признаки результата – в регистре признаков *АЛУ*. Соответствующие передачи и взаимодействия блоков обозначены на [рис. 12.1](https://www.intuit.ru/studies/courses/56/56/lecture/1666?page=1#image.12.1) цифрой 4.

**Пятый этап** – обращение к *ОЗУ* и *запись* по адресу c результата *операции*. Здесь код поля c *регистра команд* передается в *ОЗУ* на РА. Затем в ячейку *ОЗУ* с адресом c записывается результат *операции*, находящийся в регистре результата *АЛУ*. Признаки результата записываются из регистра признаков *АЛУ* в *регистр* флагов компьютера, из которого они передаются в БУОп, если очередная считанная в РК *команда* окажется командой условного перехода. Соответствующие передачи обозначены на [рис. 12.1](https://www.intuit.ru/studies/courses/56/56/lecture/1666?page=1#image.12.1) цифрой 5.

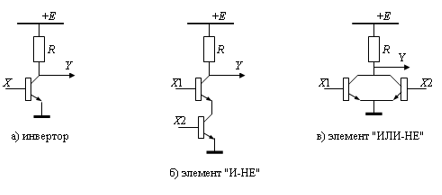
# Основы схемотехнической реализации ЭВМ

## 1. Системы логических элементов. Основные параметры логических элементов. Условно-графические обозначения основных логических элементов.

***Системой логических элементов*** наз-ся функционально полный набор логических элементов, объединенных общими электрическими, конструктивными и технологическими параметрами и использующих одинаковый тип межэлементных связей. 

Если элемент выполняет сложную функцию, имеет несколько функционально различных групп входов и выходов, то входы и выходы отделяются от основного поля ***УГО*** *вертикальными линиями.*

На рисунке «Ж» элемент имеет три состояния: 0, 1 и «высокое сопротивление». E – управляет состояние выхода. !!! Каждый логический элемент – это электронно-техническое изделие, т.е: при подаче сигнала высокого уровня на базу транзистора, его сопротивление становится пренебрежимо малым, то есть транзистор как бы "стягивается в точку". При низком потенциале на базе транзистора сопротивление между коллектором и эмиттером становится чрезвычайно большим, что фактически означает разрыв цепи.

Рассмотрим пример работы *инвертора* . Если сигнал X имеет высокий *потенциал*, то *ключ*, реализованный на транзисторе,

замкнут, и *потенциал* точки Y низкий. В противном случае *связь* между точкой Y и "землей" разорвана, и сигнал Y имеет высокий уровень, что и обеспечивает реализацию логической функции "*отрицание*".

Для элемента "И-НЕ" сигнал в точке Y будет иметь низкий уровень ( НУ ) лишь тогда, когда оба сигнала X1 и X2 имеют высокий уровень ( ВУ ). (СМОТРИ ФОТО ДАЛЕЕ)

# **Параметры** схемотехнических элементов принято делить на *статические* и *динамические.*

# *Статические параметры* инвариантны к переходным процессам и измеряются в статическом режиме. *Динамические*, наоборот, определяют реактивные свойства элемента и измеряются во время переходных процессов.

# 

# ***Статистические: 1)*** *Ток потребления;* ***2)*** *Напряжение источника питания;* ***3)*** *Пороговое напряжение низкого уровня ( U0 ) и высокого уровня( U1 );* ***4)*** *Потребляемая мощность;* ***5)*** *Нагрузочная способность;* ***6)*** *Помехоустойчивость*

# ***Динамические: 1)*** Время перехода при включении ( t10 ) (задний фронт);

# ***2)*** Время перехода при выключении ( t01 ) (передний фронт);

# ***3)*** Время *задержки распространения* при включении ( tзд01 );

# ***4)*** Время *задержки распространения* при выключении ( tзд10 );

# ***5)*** Среднее время *задержки распространения* ( tзд ср ) – интервал времени, равный полусумме времен *задержки распространения* сигнала при включении и при выключении; в дальнейшем это время будем называть временем задержки элемента ( tзд )

# Пример некоторых статических и динамических параметров логических схемы на примере работы элемента "НЕ" Временная диаграмма входного и выходного сигналов этого элемента, на которой отмечены его статические и динамические параметры, приведена слева.

## 2. Этапы проектирование логических схем на элементах “И-НЕ”. Быстродействие ло­гических схем.

# При проектировании схем, выполняющих ту или иную логическую функцию, необходимо обеспечить минимизацию аппаратных затрат на реализацию этих схем, а также во многих случаях необходимо сократить номенклатуру используемых логических элементов.

# Рассмотрим *этапы проектирования* комбинационных *логических схем* на *одноэлементном базисе* "И-НЕ" без использования каких-либо дополнительных логических элементов на примере проектирования ***одноразрядного комбинационного сумматора***. Такой *сумматор является основой построения многоразрядной суммирующей схемы*, выполняющей *операции* над числами, представленными в том или ином коде.

# **Этап 1. Представление функции, выполняемой проектируемой схемой, в каноническом виде, то есть в виде таблицы истинности или одной из совершенных нормальных форм записи.** Обычно на этом этапе функцию легче описать *таблицей истинности*.

# **Этап 2. Минимизация логической функции. На этом этапе можно использовать любые методы минимизации**. Специфика минимизации многовыходных функций – необходимость получения устройства, имеющего минимальный общий состав оборудования. Например, методом Квайна-Мак-Класки или с помощью диаграмм Вейча.

# 

# Здесь (слева) точно такая же схема, как и выше, только для переноса. Ниже третий этап, в нём переводим в базис просто, заменяя всё на штрихи Шеффера, например.

# **Этап 4. Составление схемы на элементах, реализующих функции выбранного базиса**. Для более наглядного отображения этого этапа выше обозначены номера элементов, которые будут реализовывать ту или иную часть функции.

# Ниже приведены схемы одноразрядного сумматора (сначала само суммирование, потом перенос в следующий разряд). Слева - сам сумматор, справа - перенос в следующий разряд.

## 3. Дешифратор: назначение, таблица истинности. Условно-графи­чес­кое обозначение.

# **Дешифратор –** комбинационная схема, имеющая 2n выходов и n входов. Нужен, чтобы преобразовывать *двоичный код на входах* в *унитарный код*.

# 

## 4. Мультиплексор: назначение, условно-графи­чес­кое обозначение.

# **Мультиплексор –** устройство, имеющее несколько входов и минимум 1 управляющий вход, и имеющее также один выход. Мультиплексор позволяет передавать сигнал с одного из входов на выход; при этом выбор желаемого входа осуществляется подачей соответствующей комбинации управляющих сигналов. А0 и А1 – управляющие входы

# 

## 5. Триггер. Назначение. Классификация триггерных схем. Двухступенчатый триггер.

# **Триггер** (триггерная система) — класс электронных устройств, обладающих способностью длительно находиться в одном из двух устойчивых состояний и чередовать их под воздействием внешних сигналов. Каждое состояние триггера легко распознаётся по значению выходного напряжения. Триггеры и регистры являются простейшими представителями цифровых микросхем, имеющих внутреннюю память. В основе любого триггера (англ. — "тrigger" или "flip-flop") лежит схема из двух логических элементов, которые охвачены положительными обратными связями (то есть сигналы с выходов подаются на входы). В результате подобного включения схема может находиться в одном из двух устойчивых состояний, причем находиться сколь угодно долго, пока на нее подано напряжение питания.

# Пример такой схемы (так называемой триггерной ячейки) на двух двухвходовых элементах И-НЕ представлен слева У схемы есть два инверсных входа: –R — сброс (от английского Reset), и –S — установка (от английского Set), а также два выхода: прямой выход Q и *инверсный выход* –Q.

***Триггеры служат основой для построения регистров, счётчиков и других элементов, обладающий функцией хранения.***

# Классификация триггеров:

# **·** **По способу управления (RS,JK,D и т.д. – самый классный D. Почему? Потому что. Все они различаются по типу управления)**

# · **По типу синхронизации** (синхронные или асинхронные. Асинхронные могут быть как со статической синхронизацией, так и с динамической)

# · **По внутренней организации** (одноступенчатые и двуступенчатые)

# Триггер называется ***синхронным***, если его таблица переходов (там штука чуть выше с нулями и единицами) хотя бы по 1 управляющему входу реализуется именно синхронизирующим сигналом. То есть такой чёткий сигнал бац во вход триггеру, а триггер такой «Ооо, нифига себе, вот чего надо» - и вот такие штуки есть синхронные триггеры. А асинхронные – это все, не являющиеся синхронными.

# ***Но это всё было об одноступенчатом триггере!!!*** *Часто бывает, что при построении регистров сдвига, например, нужно, чтобы значения выходных сигналов триггера не изменялись, пока она там функционирует, записывает сигналы в другие схемы. Именно для это придумали двухступенчатый триггер. Слева снизу его УГО (условно-графическое обозначение)*

# ***Кстати, если у входов слева есть ещё косая палка – то это так обозначается динамическая синхронизация***

# *Двухступенчатый RS-триггер* (СМ ФОТО НИЖЕ, ЭТО ОН) строится на основе двух одноступенчатых *триггеров* с *прямой* статической синхронизацией. *Информация* в первую ступень *триггера* (элемент 1 ) заносится во время действия высокого уровня синхросигнала. После того как синхросигнал на входе принимает низкое *значение*, элемент 1 переходит в режим хранения, а *значение* высокого сигнала на выходе инвертора 3 обеспечивает *запись* состояния *триггера* 1 в *триггер* 2.

## 6. Асинхронный двоичный счетчик. Назначение. Временная диаграмма работы. Оценка быстродействия.

# ***Счётчик –*** электронная схема для подсчёта числа сигналов, поступающих на вход. Эти счётчики используются в ЭВМ, очевидно, чтобы строить распределители импульсов и организовывать циклы какие-либо. А в счётчиках команд эти счетчики формируют адрес команды.

# Суть его: изменение на единицу зафиксированного в нем значения после прихода нового сигнала (Как в паскале i=i+1). Счётчик бывает суммирующий и такой, который убавляет значение.

# Главным параметром счётчика является *модуль пересчёта,* он равен числу внутренних состояний счётчика.

# Обозначается счётчик как ***CT (Cчётчик с модулем пересчёта, равным 8, обозначается СТ8)***

Ниже - схема быстродействия

# 

# А теперь насколько эта штука быстро работает, пока она там делает своё «плюс один»: (ФОТО СЛЕВА, пояснения снизу)

# Т.о, время задержки n-разрядного счётчика составляет n×tтакта

# *В данном случае 3t*

# 

*Быстродействие* асинхронного *счетчика* определяется максимальной задержкой от изменения сигнала на его счетном входе до полного установления состояния всех его выходов

# 

## 7. Регистры. Назначение. Регистр хранения. Регистр сдвига. Условно-графи­чес­кое обозначение.

# ***Регистр –*** сверхбыстрая память внутри процессора, его внутренне запоминающее устройство (ЗУ), предназначенное для временного хранения обрабатываемой или управляющей информации.

# Регистры есть совокупность триггеров, количество которых равняется разрядности регистра.

# *Регистр хранения* – регистр, использующийся для приёма, хранения и выдачи многоразрядного кода.

# Вот он, слева. Есть также регистр флагов. Он отслеживает текущее состояние микропроцессора. Регистр сдвига написан ниже.

# *Регистр сдвига* – *регистр*, обеспечивающий помимо хранения информации, сдвиг влево или вправо всех разрядов одновременно на одинаковое число позиций. /При этом выдвигаемые за пределы *регистра* разряды теряются, а в освобождающиеся разряды заносится *информация*, поступающая по отдельному внешнему входу *регистра сдвига*.

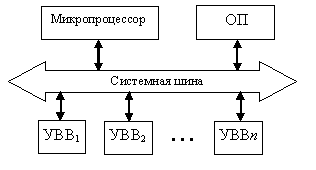
# 

# 

# Структура персональной ЭВМ.

## Структура IBM PC-совместимых компьютеров.

Персональная ЭВМ типа IBM PC ([рис. 14.1](https://www.intuit.ru/studies/courses/56/56/lecture/1670?page=1#image.14.1)) включает в себя микропроцессор ( МП ), оперативную память и устройства ввода-вывода (УВВ), объединенные между собой системной шиной.

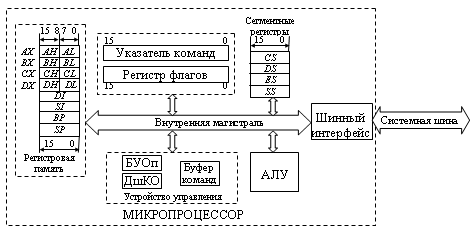


**Рис. 14.1.**Структура персональной ЭВМ

*Микропроцессор* предназначен для выполнения собственно арифметических и логических операций и управления взаимодействием блоков компьютера. Оперативная память хранит операнды и программу во время ее выполнения. Устройства ввода-вывода обеспечивают обмен информацией между ядром компьютера (МП и ОП) и средствами ввода и отображения данных. Сюда относятся мониторы, *печатающие устройства*, *графопостроители*, жесткие и гибкие магнитные диски и так далее.  
Компьютер строится по *магистрально-модульному принципу*, при котором все блоки компьютера связываются между собой системной шиной, предназначенной для обмена данными, адресной и управляющей информацией между составными частями ЭВМ. Как правило, при такой организации в любой момент может быть установлена связь только между двумя модулями ЭВМ. Системная шина определяет общий порядок обмена между любыми блоками компьютера, а также максимальное количество используемых устройств ввода-вывода. Она включает в себя шину адреса (ША), шину данных (ШД) и *шину управления* (ШУ), содержащую набор линий, по которым передаются управляющие сигналы между блоками компьютера. Специфика каждого конкретного блока учитывается особыми *управляющими устройствами* – контроллерами, входящими в состав этих блоков, например, работой жесткого диска управляет контроллер жесткого диска, используя информацию, поступающую к нему от *микропроцессора* по системной магистрали.

## Структура микропроцессора 8086, состав и назначение его основных блоков.

*Микропроцессор* предназначен для выполнения собственно арифметических и логических операций и управления взаимодействием блоков компьютера. Структура 16-разрядного *микропроцессора* I8086 представлена на рис. 14.2.



**Рис. 14.2.**Структура 16-разрядного микропроцессора

В состав *микропроцессора* входят:

1. *арифметико-логическое устройство* (*АЛУ*), предназначенное для выполнения арифметических и логических операций;
2. внутренняя *регистровая память*, состоящая из восьми 16-разрядных регистров; четыре из них допускают раздельное использование своих младших и старших байтов, обеспечивая тем самым возможность обработки как 16-разрядных слов, так и байтов информации;

1. *устройство управления*, включающее в себя
   * буфер команд, который представляет собой *регистровую память* объемом 6 байт, предназначенную для хранения выполняемой в данный момент команды (аналогично *регистру команд* в структуре классической ЭВМ) и заполняемую очередными командами из оперативной памяти по мере своего освобождения;
   * дешифратор кода операций, определяющий тип выполняемой команды;
   * блок управления операциями, который на основании *расшифрованного* *дешифратором* *кода операции* формирует управляющие сигналы, организующие работу всех блоков *микропроцессора*;

1. *указатель команд*(IP – *instruction pointer*), определяющий адрес выполняемой команды в *сегменте* команд оперативной памяти;

1. *регистр флагов* (FLAGS), содержащий признаки результата выполненных команд и некоторую управляющую информацию. Среди признаков результата отметим следующие:  
   ZF – флаг нуля: равен 1 при получении нулевого результата,  
   SF – флаг знака: устанавливается равным старшему биту результата,  
   CF – флаг переноса: фиксирует факт переноса из старшего бита в арифметических операциях,  
   OF – флаг переполнения: устанавливается в 1 при получении результата вне допустимого диапазона чисел,  
   PF – флаг паритета: устанавливается в 1, если младшие 8 бит результата операции содержат четное число единиц;  
   к флагам управления относятся  
   IF – флаг разрешения прерывания: когда флаг установлен в 1, процессор распознает *маскируемые прерывания*, что позволяет *микропроцессору* реагировать на особые ситуации, возникающие в работе внешних устройств; если значение флага равно нулю, то эти прерывания игнорируются,  
   DF – флаг направления, применяется в командах обработки последовательности байт в памяти: если флаг равен 0, последовательность обрабатывается с элемента, имеющего наименьший адрес; если флаг установлен в 1, последовательность обрабатывается от старшего адреса к младшему,  
   TF – флаг трассировки: если значение флага равно 1, то в *микропроцессоре* после выполнения каждой команды генерируется внутреннее прерывание, позволяющее перейти к соответствующей подпрограмме (используется при отладке программ);
2. блок *сегментных регистров*, состоящий из четырех 16-разрядных регистров, каждый из которых содержит старшие разряды базового (начального) адреса *сегмента*оперативной памяти, выделяемого программе при ее выполнении: кодового *сегмента*CS, в котором содержится код программы; *сегмента* данных DS; *сегмента* стека SS и дополнительного *сегмента*данных ES;
3. шинный интерфейс, который содержит схемы, обеспечивающие связь внутренней магистрали *микропроцессора* с системной шиной.

## Организация памяти в IBM PC: физическое и логическое адресное пространство. Представление логического адреса. Адрес байта, слова, двойного слова.

ОП является основной памятью для хранения информации. Она организована как *одномерный массив* ячеек памяти размером в 1 байт. Каждый из байтов имеет уникальный 20 битный *физический адрес*в диапазоне от 00000 до FFFFFh (здесь и далее для записи адресов используется шестнадцатеричная система счисления, признаком которой является символ h в конце кода). Таким образом, размер адресного пространства ОП составляет 220 = 1Мбайт. Любые два смежных байта в памяти могут рассматриваться как 16-битовое слово. Младший байт слова имеет меньший адрес, а старший - больший. Так шестнадцатеричное число 1F8Ah, занимающее слово, в памяти будет расположено в последовательности 8Ah, 1Fh. Адресом слова считается адрес его младшего байта. Поэтому 20 битовый адрес памяти может рассматриваться и как адрес байта, и как адрес слова.  
Команды, байты и слова данных можно размещать по любому адресу, что позволяет экономить память вследствие ее более полного заполнения. Однако для экономии времени выполнения программ целесообразно размещать слова данных в памяти, начиная с четного адреса, так как *микропроцессор* передает такие слова за один *цикл работы* шины. Слово с четным адресом называется выровненным по границе слов. Невыровненные слова данных с нечетным адресом допустимы, но для их передачи требуется два цикла шины, что снижает производительность ЭВМ. Заметим, что необходимое количество циклов считывания слова данных инициируется *микропроцессором* автоматически. Следует иметь в виду, что при операциях со стеком слова данных должны быть выровнены, а *указатель стека* инициирован на четный адрес, так как в таких операциях участвуют только слова данных.  
Поток команд разделяется на байты при заполнении очереди команд внутри *микропроцессора*. Поэтому выравнивание команд практически не влияет на производительность и не используется.  
Адресное пространство ОП делится на *сегменты*. *Сегмент*состоит из смежных ячеек ОП и является независимой и отдельно адресуемой единицей памяти, которая в базовой архитектуре *персональной ЭВМ*имеет фиксированную емкость 216 = 64К байт. Каждому *сегменту*назначается начальный (базовый) адрес, являющийся адресом первого байта *сегмента* в адресном поле ОП. Значение *физического адреса*ячейки складывается из адреса *сегмента*и смещения ячейки памяти относительно начала *сегмента* (внутрисегментное смещение). Для хранения значений адреса *сегмента* и смещения используются 16-битовые слова.  
*Архитектура компьютера* различает физическое *адресное пространство* (ФАП) и логическое *адресное пространство* (ЛАП). **Физическое адресное пространство**представляет собой простой *одномерный массив* байтов, *доступ* к которому реализуется аппаратурой памяти *по* адресу, присутствующему на *шине адреса* *микропроцессорной системы*. **Логическое адресное пространство** организуется самим программистом исходя из конкретных потребностей. Трансляцию логических адресов в физические осуществляет блок управления памятью *MMU*.  
В архитектуре современных *микропроцессоров* ЛАП представляется в виде набора элементарных структур: байтов, сегментов и страниц. В микропроцессорах используются следующие варианты организации **логического адресного пространства**:

* **плоское (линейное) ЛАП**: состоит из массива байтов, не имеющего определенной структуры; трансляция адреса не требуется, так как *логический адрес* совпадает с физическим;
* **сегментированное ЛАП**: состоит из сегментов - непрерывных областей памяти, содержащих в общем случае переменное число байтов; *логический адрес* содержит 2 части: идентификатор сегмента и смещение внутри сегмента; *трансляцию адреса* проводит блок *сегментации* *MMU*;
* **страничное ЛАП**: состоит из страниц - непрерывных областей памяти, каждая из которых содержит фиксированное число байтов. *Логический адрес* состоит из номера (идентификатора) страницы и смещения внутри страницы; *трансляция логического адреса в физический* проводится блоком *страничного преобразования* *MMU*;
* **сегментно-страничное ЛАП**: состоит из **сегментов**, которые, в свою очередь, состоят из страниц; *логический адрес* состоит из идентификатора сегмента и смещения внутри сегмента. Блок сегментного преобразования *MMU* проводит трансляцию *логического адреса* в номер страницы и смещение в ней, которые затем транслируются в *физический адрес* блоком *страничного преобразования* *MMU*.

Таким образом, основой получения *физического адреса* памяти служит *логический адрес*. В какой-то степени логическое *адресное пространство*, с которым имеет дело программист, можно сравнить со структурой книги, где аналогом сегмента выступает рассказ, страница книги соответствует странице ЛАП, а искомая *информация* - это некоторое *слово*. При этом если *память* организована как линейная, то номер искомого слова задается в явном виде и просто отсчитывается от начала книги. При сегментном представлении памяти искомое *слово* определяется его номером в заданном рассказе. Страничное *представление* памяти предполагает задание информации о слове в виде номера страницы в книге и номера слова на указанной странице. При сегментно-страничном представлении *логический адрес* слова задается номером слова в определенном рассказе. В этом случае *по* оглавлению книги определяется номер страницы, с которой начинается указанный рассказ. Затем, зная количество слов на странице и положение слова в рассказе, можно вычислить страницу книги и положение искомого слова на этой странице.

## Формат двухоперандной команды IBM PC общего вида. Назначение полей команд.

*Микропроцессор* *Intel-8086* (К1810ВМ80) имеет двухадресную систему команд. Ее особенностью является отсутствие команд, использующих оба операнда из оперативной памяти. *Исключение* составляют лишь *команды пересылки* и сравнения цепочек *байт* или слов, которые в данном пособии рассматриваться не будут. Таким образом, в командах допустимы следующие *сочетания* операндов: RR, RS, RI, SI. Здесь R обозначает *операнд*, находящийся в одном из регистров регистровой памяти микропроцессора, S - *операнд*, находящийся в оперативной памяти, *адрес* которого формируется *по* одному из допустимых *способов адресации*, I - непосредственный *операнд*, закодированный в адресном *поле* самой команды. *Формат команды* во многом определяется способом адресации операнда, находящего в оперативной памяти, длиной используемого непосредственного операнда, а также наличием и длиной смещения, используемого при относительных режимах адресации.

*Форматы* двухоперандных команд представлены на рис.6.1. Пунктиром показаны поля, которые в зависимости от режима адресации могут отсутствовать в команде.



**Рис. 6.1.**Форматы двухоперандных команд микропроцессора I8086  
*Поле* КОП содержит код выполняемой *операции*. Признак w указывает на длину операндов. При w = 1 операция проводится над словами, а при w = 0 - над байтами. Признак d указывает положение приемника результата. Признак d = 1, если результат записывается на *место* операнда, закодированного в *поле* reg, и d = 0, если результат записывается *по* адресу, закодированному полями ( *md*, r/m ).  
Второй байт команды, называемый ***постбайтом***, определяет операнды, участвующие в операции. *Поле* reg указывает *регистр* регистровой памяти согласно табл. 6.1

|  |  |  |
| --- | --- | --- |
| Таблица 6.1. | | |
| **reg** | **Регистр** | |
| **w=1** | **w=0** |
| 000 | AX | AL |
| 001 | CX | CL |
| 010 | DX | DL |
| 011 | BX | BL |
| 100 | SP | AH |
| 101 | BP | CH |
| 110 | SI | DH |
| 111 | DI | BH |

Поля *md* и r/m задают *режим адресации* второго операнда согласно табл. 6.2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Таблица 6.2. | | | | | |
| **r/m** | ***md*** | | | | |
| **00** | **01** | **10** | **11** | |
| **w=1** | **w=0** |
| 000 | (BX)+(SI)  (DS) | (BX)+(SI)+disp 8  (DS) | (BX)+(SI)+disp 16  (DS) | AX | AL |
| 001 | (BX)+(DI)  (DS) | (BX)+(DI)+disp 8  (DS) | (BX)+(DI)+disp 16  (DS) | CX | CL |
| 010 | (BP)+(SI)  (SS) | (BP)+(SI)+disp 8  (SS) | (BP)+(SI)+disp 16  (SS) | DX | DL |
| 011 | (BP)+(DI)  (SS) | (BP)+(DI)+disp 8  (SS) | (BP)+(DI)+disp 16  (SS) | BX | BL |
| 100 | (SI)  (DS) | (SI)+disp 8  (DS) | (SI)+disp 16  (DS) | SP | AH |
| 101 | (DI)  (DS) | (DI)+disp 8  (DS) | (DI)+disp 16  (DS) | BP | CH |
| 110 | disp16  (DS) | (BP)+disp 8  (SS) | (BP)+disp 16  (SS) | SI | DH |
| 111 | (BX)  (DS) | (BX)+disp 8  (DS) | (BX)+disp 16  (DS) | DI | BH |

В этой таблице помимо определения режима адресации оперативной памяти указан также *сегментный регистр*, используемый *по* умолчанию для получения *физического адреса*. Использование другого *сегментного регистра* возможно введением специального префикса (дополнительного байта, который записывается перед командой).  
В командах, использующих непосредственный *операнд*, признак s вместе с признаком w определяет *разрядность* непосредственного операнда, записываемого в команде, и *разрядность* выполняемой *операции* согласно табл. 6.3.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 6.3. | | | |
| **w** | **s** | **Операция** | **Непосредственный операнд** |
| 0 | 0 | 8-разрядная | 8-разрядный |
| 0 | 1 | не используется | |
| 1 | 0 | 16-разрядная | 16-разрядный |
| 1 | 1 | 8-разрядный, расширяемый знаком до 16-ти разрядов при выполнении операции |

## Символическое и машинное представление команд.

Достоинства

· Язык ассемблера позволяет писать самый быстрый и компактный код, какой вообще возможен для данного процессора.

· Если код программы достаточно большой, — данные, которыми он оперирует, не помещаются целиком в регистрах процессора, то есть частично или полностью находятся в оперативной памяти, — то искусный программист, как правило, способен значительно оптимизировать программу по сравнению с транслятором с языка высокого уровня по одному или нескольким параметрам (как правило, быстродействие программы достигается за счёт удлинения кода и наоборот):

· скорость работы — за счёт оптимизации вычислительного алгоритма и/или более рационального обращения к оперативной памяти (ОП) (например, если все исходные данные хранятся в регистрах процессора, то можно исключить излишние обращения к ОП), перераспределения данных, табличного вычисления функций;

· объём кода (в том числе за счёт эффективного использования промежуточных результатов). Сокращение объёма кода также нередко повышает скорость выполнения программы.

· При программировании на языке ассемблера возможен непосредственный доступ к аппаратуре, и, в частности, портам ввода-вывода, регистрам процессора и др. Во многих операционных системах прямое обращение из прикладных программ для записи в регистры периферийного оборудования блокировано для надёжности работы системы и исключения «зависаний».

· Язык ассемблера часто применяется для написания программ тогда, когда важно временно́е согласование работы периферийных устройств с центральным процессором.

Недостатки

· В силу машинной ориентации («низкого» уровня) языка ассемблера человеку сложнее читать и понимать программу на нём по сравнению с языками программирования высокого уровня; программа состоит из слишком «мелких» элементов — машинных команд, соответственно, усложняются программирование и отладка, растут трудоёмкость и вероятность внесения ошибок.

· Требуется повышенная квалификация программиста для получения качественного кода.

· Программа на языке высокого уровня может быть перекомпилирована с автоматической оптимизацией под особенности новой целевой платформы, программа же на языке ассемблера на новой платформе может потерять своё преимущество в скорости без ручного переписывания кода.

· Отсутствует переносимость программ на компьютеры с другой архитектурой и системой команд.

На языке ассемблера пишут программы или их фрагменты в тех случаях, когда критически важны:

· быстродействие;

· объём используемой памяти

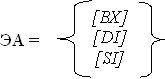
Набор команд: ADD(+), SUB(-), AND, OR, XOR(сумма по mod 2), MOV(переместить), CMP(сравнение), INC(+1), DEC(-1), TEST(=AND, но не меняет значение операндов, изменяет только флаги), XCHG(обменивает операнды) итд.

## Режимы адресации операндов в IBM PC.

***Непосредственная адресация*** предполагает, что операнд занимает одно из полей команды и, следовательно, выбирается из оперативной памяти одновременно с ней. В зависимости от форматов, обрабатываемых процессором данных непосредственный операнд может иметь длину 8 или 16 бит, что в дальнейшем будем обозначать data8 и data16 соответственно.

***Прямая адресация*** предполагает, что *эффективный адрес* является частью команды. Так как ЭА состоит из 16 разрядов, то и соответствующее поле команды должно иметь такую же длину.

При регистровой ***косвенной адресации*** *эффективный адрес* операнда находится в базовом регистре BX или одном из индексных регистров DI либо SI:



При регистровой ***относительной адресации*** *эффективный адрес* равен сумме содержимого базового или индексного регистра и смещения:



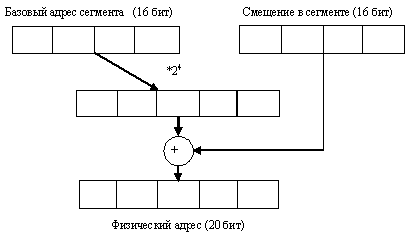
Обозначения disp8 и disp16 здесь и далее указывают на 8- или 16-разрядное смещение соответственно.  
*Эффективный адрес* при ***базово-индексной адресации*** равен сумме содержимого базового и индексного регистров, определяемых командой:  
https://www.intuit.ru/EDI/28_12_14_2/1419715086-15457/tutorial/140/objects/6/files/img6.3.gif  
Наиболее сложен механизм ***относительной базово-индексной адресации***. *Эффективный адрес* в этом случае равен сумме 8- или 16-разрядного смещения и базово-индексного адреса:

https://www.intuit.ru/EDI/28_12_14_2/1419715086-15457/tutorial/140/objects/6/files/img6.4.gif

## Формирование физического адреса в IBM PC в реальном режиме работы.

*Адресация* оперативной памяти имеет свои особенности, связанные с ее разбиением на *сегменты* и использованием сегментной группы регистров для указания начального адреса сегмента. 16-разрядный адрес, получаемый в блоке формирования адреса операнда на основе указанного режима адресации, называется ***эффективным адресом*** (ЭА). Иногда *эффективный адрес* обозначается как ЕА (effective address). 20-разрядный адрес, который получается сложением *эффективного адреса* и увеличенного в 16 раз значения соответствующего *сегментного регистра*, называется ***физическим адресом*** (ФА). Именно *физический адрес* передается из микропроцессора *по* 20-ти адресным линиям, входящим в состав системной шины, в оперативную *память* и используется при обращении к ее ячейке на физическом уровне.

Чтобы получить 20-битовый *физический адрес*, *микропроцессор* автоматически осуществляет следующие операции. Значение базового адреса *сегмента* умножается на 16 (сдвиг на 4 разряда влево) и суммируется со значением смещения в *сегменте* (рис. 14.3). В результате получается 20-битовое значение *физического адреса*. При суммировании может возникнуть перенос из старшего бита, который игнорируется. Это приводит к тому, что ОП оказывается как бы организованной по кольцевому принципу. За ячейкой с максимальным адресом FFFFFh следует ячейка с адресом 00000h.



**Рис. 14.3.**Схема получения физического адреса

# Организация основных устройств ЭВМ

## 1. Устройство управления: назначение, принципы построения

УУ – часть цифрового вычислительного устройства, предназначенная для выработки последовательности подачи управляющих сигналов на управляющие входы любого устройства компьютера, определенных образом распределенные во времени.

Назначение – УУ предназначено для выработки управляющих сигналов, под воздействием которых происходит преобразование информации в АЛУ, а также операции по записи и чтению информации в/из запоминающего устройства.

Принципы построения – УУ делятся на 2 вида:

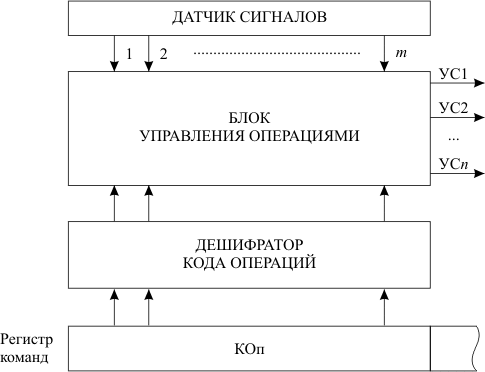
УУ с жесткой (схемной) логикой

УУ с программируемой логикой (микропрограммные УУ)

В 1 для каждой команды, задаваемой кодом операции, строится набор комбинационных схем, которые в нужных тактах вырабатывают необходимые управляющие сигналы.

Во 2 каждой команде ставится в соответствие совокупность хранимых в специальной памяти слов - микрокоманд. Каждая из микрокоманд содержит информацию о микрооперациях, подлежащих выполнению в данном такте, и указание, какое слово должно быть выбрано из памяти в следующем такте.

## 2. Структурная схема устройства управления с жесткой логикой. Реализация датчика сигналов на сдвиговом регистре и на счётчике с дешифратором.



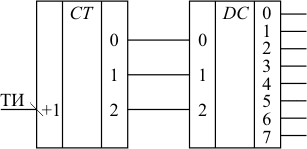
Структурная схема устройства

УУ схемного типа состоит из:

1. Датчика сигналов, вырабатывающего последовательность импульсов, равномерно распределенную во времени по своим шинам

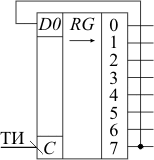
2. Блока управления операциями, осуществляющего выработку управляющих сигналов, то есть коммутацию сигналов, поступающих с ДС, в соответствующем такте на нужную управляющую шину;

3. Дешифратора кода операций, который дешифрирует код операции команды, присутствующей в данный момент в регистре команд, и возбуждает одну шину, соответствующую данной операции; этот сигнал используется блоком управления операциями для выработки нужной последовательности управляющих сигналов.

Датчик сигналов на основе счетчика с дешифратором

Реализация датчика сигналов на основе счетчика с дешифратором. По заднему фронту каждого тактового импульса, поступающего на устройство управления с системного генератора импульсов, счетчик увеличивает свое состояние; выходы счетчика соединены со входами дешифратора, выходы которого и являются выходами датчика сигналов

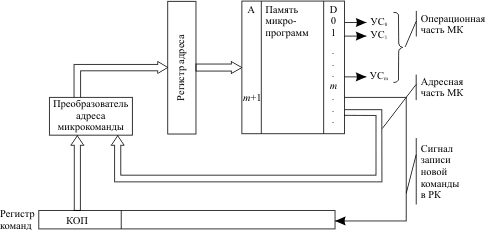
Датчик сигналов на сдвиговом регистре



Проектирование датчика сигналов на сдвиговом регистре требует лишь его "закольцовывания", то есть соединения выхода последнего разряда с входом, через который в регистр заносится информация при сдвиге, и первоначальной установки. В начальном состоянии регистр содержит "1" только в разряде 0. Входы параллельной загрузки регистра для его начальной установки и соответствующий этой операции управляющий вход регистра на схеме не показаны.

## 3. Структурная схема микропрограммного устройства управления

Функциональная схема МУУ (УС – управляющие сигналы, вырабатываемые УУ)



Преобразователь адреса микрокоманды преобразует код операции команды, присутствующей в данный момент в регистре команд, в начальный адрес микропрограммы, реализующей данную операцию, а также определяет адрес следующей микрокоманды выполняемой микропрограммы по значению адресной части текущей микрокоманды.

В таблице приведен пример микропрограммы умножения чисел в прямом коде. Предполагается, что начальный адрес микропрограммы равен 300, количество разрядов множителя равно 2, а адресная часть микрокоманды содержит адрес микрокоманды, которая должна быть выбрана в следующем такте. В последней микрокоманде в регистр команд загрузится очередная команда, код операции которой определит начальный адрес очередной микропрограммы.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Адрес МК | УС1 | УС2 | УС3 | УС4 | УС5 | УС6 | Сигнал записи в РК | Адрес следующей МК |
| 300 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 301 |
| 301 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 302 |
| 302 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 303 |
| 303 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 304 |
| 304 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | Х |

Типа вывод по этим 3 вопросам:

1 тип имеет сложную структуру, которую нужно перерабатывать почти при любой модификации системы команд. Но он имеет высокое быстродействие за счет быстродействия элементного базиса.

2 тип может легко настраиваться на изменения в операционной части ЭВМ. Но временные показатели у него хуже по сравнению с 1 типом.

## 4. Запоминающие устройства: назначение, основные параметры. Иерархическая структура ЗУ современных ЭВМ.

Для начала надо сказать, что память ЭВМ – совокупность устройств, служащих для запоминания, хранения и выдачи информации. Отдельные устройства, входящие в эту совокупность, называются ЗУ. (ЗУ – принцип построения, память – выполняемая логическая функция, обычно ЗУ и память – синонимы)

Назначение – большая, важная роль (так и написано) в общей структуре ЭВМ. Производительность компьютера на ~40-50% зависит от ЗУ различных типов в его составе.

Основные параметры:

Емкость памяти – максимальное количество данных, которое может храниться в ней. Измеряется количеством адресуемых ячеек и длиной ячейки в битах. Минимально адресуемый элемент – 1 байт. Поэтому емкость в байтах, килобайтах, мб, гб и тд.

Организация памяти – память объемом 1 мб может быть организована как 1мб слов по 1 байту, 512 кб слов по 2 байта, 256 кб слов по 4 байта каждое. Слово в данном случае – некоторая единица данных, которая считывается или записывается за одно обращение к ЗУ

Быстродействие памяти – продолжительность операции обращения, время, затрачиваемое на поиск нужной информации и ее считывание.

tобр = max(tобр сч, tобр зп) - где tобр сч - быстродействие ЗУ при считывании информации; tобр зп - быстродействие ЗУ при записи.

Иерархическая структура

Классификация по типу обращения и организации доступа к ячейкам ЗУ.



По типу обращения

Запись и чтение – используются в процессе работы процессора, промежуточные результаты и т.д.

Только чтение – ПЗУ, ROM(Read Only Memory)((Читать Только Память)). Используются для запуска компьютера, хранения констант, BIOS там находится(это я от себя)

По организации доступа

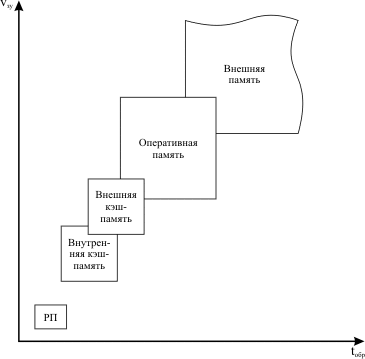
Произвольный доступ – RAM (Random Access Memory)((Случайный Доступ Память)). Здесь время доступа не зависит от места расположения участка памяти. (Пример – ОЗУ)

Прямой (циклический) доступ – время доступа зависит от взаимного расположения этого участка и головок чтения/записи и скоростью вращения носителя (Пример – Магнитный диск)

Последовательный доступ – производится последовательный просмотр участков носителя информации, пока нужный участок не займет некоторое нужное положение напротив головок чтения/записи (Пример - магнитные ленты).

Идеальное ЗУ - обладает бесконечно большой емкостью и бесконечно малым временем обращения.

Иерархическая структура (вообще да вот она, но предыдущий абзац тоже могут спросить мне кажется)



Иерархическая структура памяти позволяет экономически эффективно сочетать хранение больших объемов информации с быстрым доступом к информации в процессе ее обработки.

Тут все скопировано с интуита просто, информация об этих типах памяти

РП – регистровая память, набор регистров, входящих непосредственно в состав микропроцессора. Регистровая память имеет относительно небольшой объем (до нескольких десятков машинных слов). РП работает на частоте процессора, поэтому время доступа к ней минимально.

Оперативная память - устройство, которое служит для хранения информации (программ, исходных данных, промежуточных и конечных результатов обработки), непосредственно используемой в ходе выполнения программы в процессоре. В настоящее время объем ОП персональных компьютеров составляет несколько сотен мегабайт. Оперативная память работает на частоте системной шины и требует 6-8 циклов синхронизации шины для обращения к ней.

Для заполнения пробела между РП и ОП по объему и времени обращения в настоящее время используется кэш-память, которая организована как более быстродействующая (и, следовательно, более дорогая) статическая оперативная память со специальным механизмом записи и считывания информации и предназначена для хранения информации, наиболее часто используемой при работе программы. Как правило, часть кэш-памяти располагается непосредственно на кристалле микропроцессора (внутренний кэш ), а часть - вне его (внешняя кэш-память ). Кэш-память программно недоступна. Для обращения к ней используются аппаратные средства процессора и компьютера.

Внешняя память организуется, как правило, на магнитных и оптических дисках, магнитных лентах. Емкость дисковой памяти достигает десятков гигабайт при времени обращения менее 1 мкс.

## 5. Система управления памятью. Назначение. Функции. Статическое и динамическое распределение памяти. Сегментная и страничная организация памяти. Виртуальная память.

Местами очень непонятный вопрос.

Вообще - Система управления памятью должна обеспечить многопрограммный режим работы, используя аппаратные средства распределения оперативной памяти.

Назначение - Системные цели заключаются, прежде всего, в увеличении степени использования оперативной памяти при параллельном развитии нескольких процессов в мультипрограммном режиме, а также в реализации защиты информации при развитии этих процессов, обеспечении взаимодействия между процессами и т. д.

Функции:

Учет состояния свободных и уже распределенных областей памяти и модернизация этой информации всякий раз, когда в распределении памяти производятся изменения;

Распределение памяти для выполнения задач (определение, какой задаче, когда и в каком количестве выделить оперативную память);

Непосредственное выделение задаче оперативной памяти; если свободные области оперативной памяти отсутствуют, то предварительное их освобождение путем сохранения информации во внешней памяти.

Статическое и динамическое распределение памяти

Существуют две стратегии распределения оперативной памяти: статическое и динамическое распределение.

**Статическое** - вся необходимая оперативная память выделяется процессу в момент его порождения. При этом память выделяется единым блоком необходимой длины, начало которого определяется базовым адресом. Пример – в книжке стр 246, смысл в том что программы выполняются одна за другой, и могут не начать выполняться когда для них уже есть в ОП.

**Динамическое** - каждой программе в начальный момент выделяется лишь часть от всей необходимой ей памяти, а остальная часть выделяется по мере возникновения потребности в ней.

Это основано на том что до 25% программы вообще не используется и надо стремится к тому чтобы эти фрагменты кода вообще не загружались в ОП.

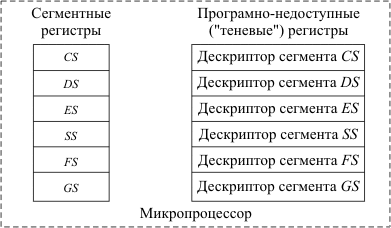
Динамический тип тесно связан с виртуальной памятью

Принцип виртуальной памяти предполагает, что пользователь при подготовке своей программы имеет дело не с физической ОП, действительно работающей в составе компьютера и имеющей некоторую фиксированную емкость, а с виртуальной (кажущейся) одноуровневой памятью, емкость которой равна всему адресному пространству, определяемому размером адресной шины компьютера. Виртуальная память, обеспечивая возможность программисту обращаться к очень большому объему непрерывного адресного пространства, предоставляемого в его монопольное распоряжение, обладает обычными свойствами: побайтовая адресация, время доступа, сравнимое со временем доступа к оперативной памяти. Для преобразования виртуальных адресов в физические физическая и виртуальная память разбиваются на блоки фиксированной длины, называемые страницами. Объемы виртуальной и физической страниц совпадают. Страницы виртуальной и физической памяти нумеруются. Правила перевода номеров виртуальных страниц в номера физических страниц обычно задаются в виде таблицы страничного преобразования. Такие таблицы формируются системой управления памятью и модифицируются каждый раз при перераспределении памяти.

Сегментная и страничная организация памяти

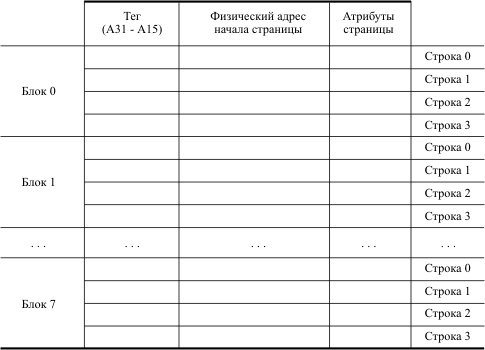
Чтобы это понять можно прочитать стр 251 учебника или<https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=1>

Темы по ссылке не будет, но то что будет основывается на этом. В общем там рассматривается сегментно-страничная организация памяти, а в ее процессе идет обращение к системным таблицам дескрипторов и это очень долго. Поэтому используется сегментное и страничное преобразование адреса.

Сегментное преобразование – при считывании таблицы дескриптора, адрес запоминается в программно-недоступных регистрах микропроцессора (справа на картинке).

Это происходит без обращения к ОП, поэтому быстрее.

Страничное преобразование – используется кэш-буфер ассоциативной трансляции, находящийся в самом микропроцессоре. Алгоритм его работы:



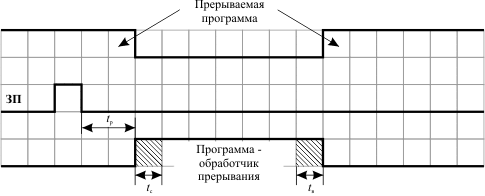
Номер виртуальной страницы представляет собой старшие 20 разрядов линейного адреса, полученного при сегментном преобразовании ( А31 - А12 ). По младшим разрядам ( А14 - А12 ) этого номера выбирается блок в буфере ассоциативной трансляции. Содержимое поля тэгов каждой из четырех строк этого блока ассоциативным образом (одновременно) сравнивается с разрядами ( А31 - А15 ) линейного адреса. Если значения для одной из строк выбранного блока совпали, значит, номер этой виртуальной страницы уже преобразовывался в номер физической страницы и результат этого преобразования находится в найденной строке TLB. Если сравнение не было успешным, то преобразование номера виртуальной страницы в номер физической проходит обычным образом через обращения к каталогу таблиц страниц и к таблице страниц, а полученное значение заносится в TLB. При этом в поле тэгов заносятся старшие 17 разрядов линейного адреса этой страницы ( A31-A15 ). Если нет свободной строки в блоке, определяемом разрядами А14 - А12 линейного адреса, то из блока вытесняется строка, информация в которой дольше всего не использовалась .

## 6. Система прерываний. Назначение. Порядок действий компьютера при обработке запросов прерываний.

Прерывание — это прекращение выполнения текущей команды или текущей последовательности команд для обработки некоторого события специальной программой - обработчиком прерывания, с последующим возвратом к выполнению прерванной программы.

Назначение - используется для быстрой реакции процессора на особые ситуации, возникающие при выполнении программы и взаимодействии с внешними устройствами, например – нажатие кнопки на клавиатуре.

Любая ситуация, вызывающая прерывание, сопровождается сигналом, называемым запросом прерывания. Соответственно после появления этого запроса, ЭВМ переходит к выполнению специальной программы – обработчик прерывания.



На рисунке – посередине возникает запрос прерывания, потом 2 клетки – tp – время реакции процессора на запрос прерывания, tc1 – сохранение состояния прерываемой программы, потом работает сам обработчик прерываний, потом загружается и восстанавливается прерванная программа tc2, потом прерванная программа продолжается.

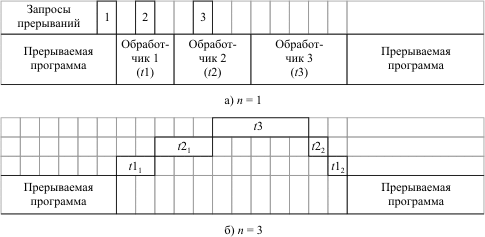
Время реакции зависит от того, когда процессор определяет факт наличия прерывания. Он может проводить опрос запросов прерываний в разные моменты времени – или после выполнения этапа команды, или после полного выполнения одной из команд.

В первом случае – время реакции быстрое, но время сохранения состояния прерванной программы и ее загрузки и восстановлению – дольше.

Второй случай – наоборот, время реакции медленнее, но время восстановления и загрузки – намного меньше.

В современных компьютерах используется второй вариант.

Глубина прерывания - максимальное число программ, которые могут прерывать друг друга.



Может существовать несколько одновременных прерываний.

Вот тут на рисунке представлено. Просто несколько прерывание на прерывание, причем при любом получении запроса прерывания ему присваивается номер, каждый раз на 1 больше. Первое прерывание – 1, второе – 2 и т.д. Соответственно приоритет больше у того, чей номер больше.

Теперь классификация.

Аппаратные прерывания используются для организации взаимодействия с внешними устройствами. Запросы аппаратных прерываний поступают на специальные входы микропроцессора. Они бывают:

1. маскируемые, которые могут быть замаскированы программными средствами компьютера;

2. немаскируемые, запрос от которых таким образом замаскирован быть не может.

Программные прерывания вызываются следующими ситуациями:

1. особый случай, возникший при выполнении команды и препятствующий нормальному продолжению программы (переполнение, нарушение защиты памяти, отсутствие нужной страницы в оперативной памяти и т.п.);

2. наличие в программе специальной команды прерывания INT n, используемой обычно программистом при обращениях к специальным функциям операционной системы для ввода-вывода информации.

Каждому запросу прерывания в компьютере присваивается свой номер ( тип прерывания ), используемый для определения адреса обработчика прерывания.

Вот последовательность действий компьютера, ее вроде и надо выучить.

При поступлении запроса прерывания компьютер выполняет следующую **последовательность действий**:

1. определение наиболее приоритетного незамаскированного запроса на прерывание (если одновременно поступило несколько запросов);

2. определение типа выбранного запроса;

3. сохранение текущего состояния счетчика команд и регистра флагов;

4. определение адреса обработчика прерывания по типу прерывания и передача управления первой команде этого обработчика;

5. выполнение программы - обработчика прерывания;

6. восстановление сохраненных значений счетчика команд и регистра флагов прерванной программы;

7. продолжение выполнения прерванной программы.

В общем этапы 1,2,3,4,6 выполняются автоматически. Задача программиста – написать ту самую программу-обработчик прерывания.

# Мультипрограмная ЭВМ

## 1.Мультипрограмный режим работы ЭВМ.Процесс и ресурс в мп ЭВМ.

Мультипрограммным режимом работы (многозадачностью) называется такой способ организации работы системы, при котором в ее памяти одновременно содержатся программы и данные для выполнения нескольких процессов обработки информации (задач) . При этом должна обеспечиваться взаимная защита ( когда одно выполняется ,другое не мешает ему) программ и данных, относящихся к различным задачам, а также возможность перехода от выполнения одной задачи к другой (переключение задач).

Мультипрограммирование позволяет повысить производительность работы ЭВМ за счет более эффективного использования ее ресурсов.

Базовыми понятиями мультипрограммного режима функционирования ЭВМ являются процесс и ресурс

В строгом понимании процесс - это некоторая дятельность, связанная с исполнением программы на процессоре.

Процесс может находиться в следующих состояниях:

· порождение - подготавливаются условия для первого исполнения на процессоре;

· активное состояние (CPU)- исполнение программы на центральном процессоре;

· готовность ( Ready ) - программа не исполняется, но для исполнения предоставлены все необходимые в текущий момент ресурсы, кроме центрального процессора;

· исполнение программы на каком-либо другом устройстве компьютера, например, устройстве ввода/вывода, имеющем собственные средства управления;

· ожидание ( Wait ) - программа не исполняется по причине занятости какого-либо ресурса ;

· окончание - нормальное или аварийное завершение исполнения программы, после которого процессор и другие ресурсы ей не предоставляются.

Время между порождением и окончанием процесса называется интервалом существования процесса.

Понятие ресурса строго не определено. Будем считать, что всякий потребляемый объект (независимо от формы его существования), обладающий некоторой практической ценностью для потребителя, является ресурсом

Ресурсы различаются по запасу выделяемых единиц ресурса и бывают в этом смысле исчерпаемыми и неисчерпаемыми. К исчерпаемым ресурсам относится, например, центральный процессор. В качестве неисчерпаемого ресурса можно представить, например, память, выделяемую программе, если рассматривать ее как совокупность всех имеющихся в компьютере запоминающих устройств. В то же время, запоминающее устройство, состоящее только из оперативной памяти с единственным трактом записи/считывания, представляет собой исчерпаемый ресурс.

Исчерпаемость ресурса, как правило, приводит к конфликтам среди потребителей этого ресурса. Для регулирования конфликтов ресурсы должны распределяться между потребителями по каким-то правилам, в наибольшей степени их удовлетворяющим.

## 2 Структура мультипрограммной ЭВМ и особенности ее работы.

Основные черты мультипрограммного режима:

· в оперативной памяти находятся несколько пользовательских программ в состояниях активности, ожидания или готовности;

· время работы процессора разделяется между программами, находящимися в памяти в состоянии готовности;

· параллельно с работой процессора происходит подготовка и обмен с несколькими устройствами ввода-вывода.

Мультипрограммирование предназначено для повышения пропускной способности вычислительной системы путем более равномерной и полной загрузки всего ее оборудования, в первую очередь процессора. При этом скорость работы самого процессора и номинальная производительность ЭВМ не зависят от мультипрограммирования.

Мультипрограммный режим имеет в ЭВМ аппаратную и программную поддержку:

· аппаратная:

o контроллеры устройств ввода-вывода, которые могут работать параллельно с процессором;

o система прерывания;

o аппаратные средства системы защиты программ и данных в микропроцессоре;

o и т.п.;

· программная:

o мультизадачная операционная система (ОС);

o системные программы, управляющие работой устройств ввода-вывода и специализированных средств вычислительной системы.

Управляющая программа (ОС), реализуя мультипрограммный режим, должна распределять (в том числе динамически) ресурсы системы (время процессора, оперативную и внешнюю память, устройства ввода-вывода и т.д.) между параллельно выполняемыми программами, чтобы обеспечить увеличение пропускной способности компьютера с учетом ограничений на ресурсы и требований по срочности выполнения отдельных программ.

Производительность мультипрограммной ЭВМ оценивается количеством задач, выполненных в единицу времени ( пропускная способность ) и временем выполнения каждой программы Тi.

При анализе работы ЭВМ важно определить степень использования ее ресурсов. Для этого широко применяются следующие показатели:

· kq - коэффициент загрузки q-го устройства;

kq=Tq/T, где Tq - время занятости устройства q за общее время T работы ЭВМ;

·  - средняя длина очереди запросов к устройству q, где Lqi - длина очереди к устройству(количество ждущих) q на интервале времении 

# 

Пусть работа некоторого устройства q характеризуется диаграммой, приведенной на [рис. 12.1](https://www.intuit.ru/studies/courses/60/60/lecture/1788?page=1#image.12.1).

Рис. 12.1. Оценка основных показателей использования аппаратных ресурсов

Тогда рассмотренные выше показатели работы этого устройства будут следующими:

· kq=7/10

· Lq=(0\*2 + 1\*1 + 0\*4 + 1\*1 + 1\*2 + 1\*1)/10 = 5/10

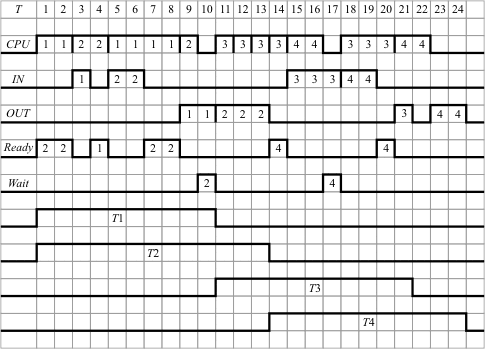
Помимо средней длины очереди важна также и динамика изменения текущей длины очереди.

По значениям kq, Lq и динамике изменения Lq можно определить наиболее дефицитный ресурс в системе, ее "узкое место".

Устранить "узкие места" можно или увеличением производительности соответствующего ресурса, или выбором такого набора задач, который обеспечивал бы более равномерное использование всех ресурсов (например, одни задачи более активно используют процессор (счетные задачи), другие - жесткий диск (работа с базами данных), третьи - устройства ввода-вывода).

Работа мультипрограммной ЭВМ в большой степени зависит от коэффициента мультипрограммирования ( Км ) - количества программ, которое может одновременно обрабатываться в мультипрограммном режиме .

Рис. 12.2. Порядок выполнение программ в мультипрограммной ЭВМ при Км = 2



Пример выполнения программ в мультипрограммном режиме при Км=2 представлен на [рис. 12.2](https://www.intuit.ru/studies/courses/60/60/lecture/1788?page=2#image.12.2). Предполагается, что выполнение каждой программы включает следующую последовательность действий: счет1 - ввод - счет2 - вывод. Счет выполняется на процессоре ( CPU ), для ввода и вывода данных используются отдельные внешние устройства ( IN и OUT ). На графике помечены номера программ, которые в данный момент занимают тот или иной ресурс.(сколько времени продержались ,занимая ресурс)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 12.1. | | | | |
| Программа | CPU1 | IN | CPU2 | OUT |
| 1 | 2 | 1 | 4 | 2 |
| 2 | 2 | 2 | 1 | 3 |
| 3 | 4 | 3 | 3 | 1 |
| 4 | 2 | 2 | 2 | 2 |

Если построить аналогичные графики для ЭВМ, работающей с различными коэффициентами мультипрограммирования, то получим следующие сравнительные характеристики работы ЭВМ для рассматриваемого пакета программ ([табл. 12.2](https://www.intuit.ru/studies/courses/60/60/lecture/1788?page=2#table.12.2)).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 12.2. | | | |
| Характеристика | Км = 1 | Км = 2 | Км = 3 |
| Время выполнения программы Т1 | 9 | 10 | 10 |
| Время выполнения программы Т2 | 8 | 13 | 13 |
| Время выполнения программы Т3 | 11 | 11 | 19 |
| Время выполнения программы Т4 | 8 | 11 | 12 |
| Время выполнения всех программ ( Т ) | 36 | 24 | 22 |
| Пропускная способность ( П )(количество программ на время) | 0,11 | 0,17 | 0,18 |
| kCPU(коэфф загрузки устр-ва) | 0,56 | 0,83 | 0,91 |
| kIN | 0,22 | 0,33 | 0,36 |
| kOUT | 0,22 | 0,33 | 0,36 |

Под временем выполнения программы понимается время, прошедшее от начала выполнения программы или ее постановки в очередь к процессору, до ее завершения, а время выполнения всех программ определяется моментом завершения выполнения последней программы пакета.

При увеличении коэффициента мультипрограммирования изменение значений показателей эффективности зависит от того, в каком состоянии находится система: перегрузки или недогрузки. Если какие-либо ресурсы ЭВМ используются достаточно интенсивно, то добавление новой программы, активно использующей эти ресурсы, будет малоэффективным для увеличения пропускной способности ЭВМ.

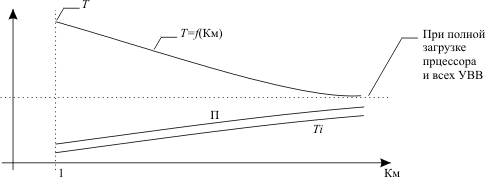
Очевидно, что зависимость пропускной способности ( П ), времени выполнения каждой программы ( Тi ) и времени выполнения всего пакета программ ( Т ) от коэффициента мультипрограммирования будет иметь вид, представленный на [рис. 12.3](https://www.intuit.ru/studies/courses/60/60/lecture/1788?page=2#image.12.3).

Рис. 12.3. Зависимость основных характеристик работы мультипрограммной ЭВМ от коэффициента мультипрограммирования

На изменение эффективности работы мультипрограммной ЭВМ может повлиять назначение различных приоритетов выполняемым программам. Перераспределение приоритетов может привести как к увеличению, так и к снижению пропускной способности ЭВМ. Конкретный результат зависит от характеристик выполняемых программ. В частности, если в составе мультипрограммной смеси имеется единственная программа, надолго занимающая процессор, то увеличение ее приоритета понизит, а уменьшение - повысит пропускную способность ЭВМ. Это объясняется тем, что выполнение программ, обладающих меньшим приоритетом, чем рассматриваемая, фактически блокируется из-за недоступности процессора. Аналогичная ситуация может сложиться и в отношении других совместно используемых ресурсов. Особое значение при этом имеют те из них, которые являются наиболее дефицитными, то есть имеют наибольший коэффициент загрузки и наибольшую среднюю длину очереди. Как правило, наиболее высокий приоритет назначается тем программам, которые в состоянии быстро освободить наиболее дефицитный ресурс. Такого рода проблемы решаются в рамках теории расписаний. При этом поиск решения зачастую сводится к полному перебору вариантов. Ввиду сложности полной теоретической оценки всех возможных вариантов, на практике широко используются различные эвристические алгоритмы, дающие не оптимальные, а рациональные решения.

В мультипрограммной ЭВМ ресурсы могут распределяться как на статической, так и на динамической основе. В первом случае ресурсы распределяются до момента порождения процесса и являются для него постоянными. Освобождение ресурсов, занятых каким-либо процессом, происходит только в момент окончания этого процесса. При динамическом распределении ресурсы выделяются процессу по мере его развития.

Распределение на статической основе способствует наиболее быстрому развитию процессов в системе с момента их порождения. Распределение же ресурсов на динамической основе позволяет обеспечить эффективное использование ресурсов с точки зрения минимизации их простоев.

Схема статического распределения используется в том случае, когда необходимо гарантировать выполнение процесса с момента его порождения. В качестве недостатка этого подхода следует отметить возможность длительных задержек заявок на порождение процесса с момента поступления таких заявок в систему, так как необходимо ожидать освобождения всех требуемых заявке ресурсов и только при наличии их полного состава порождать процесс. Часто распределение ресурсов с использованием исключительно статического принципа приводит фактически к однопрограммному режиму работы.

При динамическом распределении стремление уменьшить простои ресурсов приводит к увеличению сложности системы распределения ресурсов и, как следствие, к увеличению системных затрат на управление процессами. Поэтому необходим компромисс между сложностью алгоритмов планирования распределения ресурсов и эффективностью выполнения пакета задач.

Ресурсы разделяются на физические и виртуальные.

Под физическим понимают ресурс, который реально существует и при распределении его между пользователями обладает всеми присущими ему физическими характеристиками.

Виртуальный ресурс - это некая модель, которая строится на базе физического ресурса, имеет расширенные функциональные возможности по отношению к физическому ресурсу, на базе которого он создан, или обладает некоторыми дополнительными свойствами, которых физический ресурс не имеет.

Например, расширенные функциональные возможности имеет виртуальная память, представляющаяся как запоминающее устройство, имеющее больший объем, чем физическая. Дополнительные свойства имеет виртуальный процессор, одновременно обрабатывающий несколько задач.

## 

## 

## 3.Дисциплины распределения ресурсов в мультипрограмных ЭВМ

Дисциплины распределения ресурсов - весьма важный показатель, влияющий на эффективность работы ЭВМ. Применение той или иной дисциплины распределения зависит от особенностей использования данного ресурса, критериев оценки эффективности работы системы, а также от сложности реализации данной ДРР [[12](https://www.intuit.ru/studies/courses/60/60/literature#literature.12)]

Одноочередные дисциплины

1. FIFO ( First In - First Out ) - первый пришел - первый обслужен

Рис. 13.1. Схема распределения ресурса по дисциплине FIFO

Схема доступа - очередь.

Широко используется в качестве как самостоятельной дисциплины распределения, так и составной части более сложных дисциплин.

Время нахождения в очереди длинных (то есть требующих большого времени обслуживания) и коротких запросов зависит только от момента их поступления.

2. LIFO ( Last In - First Out ) - последний пришел - первый обслужен ([рис. 13.2](https://www.intuit.ru/studies/courses/60/60/lecture/1790?page=1#image.13.2)).

Рис. 13.2. Схема распределения ресурса по дисциплине LIFO

Схема доступа - стек.

3. Круговой циклический алгоритм ([рис. 13.3](https://www.intuit.ru/studies/courses/60/60/lecture/1790?page=1#image.13.3)).



Рис. 13.3. Схема распределения ресурса по круговому циклическому алгоритму

Запрос обслуживается в течение кванта времени tk. Если за это время обслуживание не завершено, то запрос передается в конец входной очереди на дообслуживание.

Здесь короткие запросы находятся в очереди меньшее время, чем длинные.

## 

## 

## 4.Многоочередная дисциплина распределения ресурсов в мультипрограмной ЭВМ и ее модификации

#### 

Базовый вариант многоочередной дисциплины обслуживания представлен на [рис. 13.4](https://www.intuit.ru/studies/courses/60/60/lecture/1790?page=1#image.13.4).

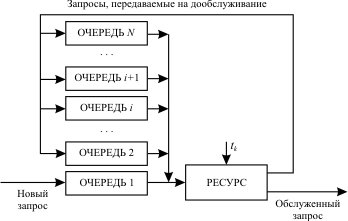


Рис. 13.4. Схема распределения ресурса при многоочередной дисциплине обслуживания

Основа дисциплины - круговой циклический алгоритм.

Все новые запросы поступают в очередь 1.

Время, выделяемое на обслуживание любого запроса, равно длительности кванта tk. Если запрос обслужен за это время, то он покидает систему, а если нет, то по истечении выделенного кванта времени он поступает в конец очереди i +1.

На обслуживание выбирается запрос из очереди i, только если очереди 1,…, i -1 пусты.

Таким образом, длинные запросы поступают сначала в очередь 1, затем постепенно доходят до очереди N и здесь обслуживаются до конца либо по дисциплине FIFO, либо по круговому циклическому алгоритму.

Модификации базового варианта многоочередной дисциплины обслуживания запросов.

1. Выделяемый программе квант времени на обслуживание возрастает с увеличением номера очереди обычно по правилу

tki = 2i-1 x tk

где tk - квант времени, выделяемый для программ из очереди 1.

Такая дисциплина обслуживания наиболее благоприятна коротким программам, хотя явного указания приоритетов программ здесь нет. Степень благоприятствования тем выше, чем меньше tk. Однако уменьшение длительности кванта ведет к увеличению накладных расходов, необходимых для перераспределения ресурса между программами.

Данная ДРР может работать как с относительными, так и с абсолютными приоритетами программ.

* + Обслуживание программ с относительными приоритетами. Заявка, входящая в систему, не вызывает прерывания обслуживаемой заявки, даже если последняя и менее приоритетна. Только после окончания обслуживания текущей заявки начинается обслуживание более приоритетной.
  + Обслуживание программ с абсолютными приоритетами. Если во время обслуживания программы из очереди i в очередь с большим приоритетом поступает новая программа, то после окончания текущего кванта tk обрабатываемая программа прерывается и возвращается в начало своей очереди, с тем чтобы впоследствии дообслужиться на время, недобранное до 2i-1 x tk.

1. Система с динамическим изменением приоритетов программ. Чтобы избежать недопустимо долгого ожидания для больших программ, приоритет делается зависимым от времени ожидания в очереди. Если ожидание превысит некоторое установленное время, программа переводится в очередь с меньшим номером.
2. Система со статическим указанием приоритетов программ. Считается, что продолжительность выполнения программы приблизительно пропорциональна ее длине. По крайней мере, от длины программы прямо зависит время, затрачиваемое на передачу программы между ОЗУ и внешним ЗУ при ее активизации.

Определение номера очереди, в которую поступает программа при первоначальной загрузке, осуществляется по алгоритму планирования Корбато: программа сразу поступает в очередь i = [log2 lp/ltk + 1], где lp - длина программы в байтах; ltk - число байт, которые могут быть переданы между ОЗУ и внешней памятью за время tk ([рис. 13.5](https://www.intuit.ru/studies/courses/60/60/lecture/1790?page=1#image.13.5)).

1. 

Рис. 13.5. Схема распределения ресурса при многоочередной дисциплине обслуживания со статическим указанием приоритетов программ

Эта дисциплина позволяет сократить количество системных переключений за счет того, что программам, требующим большего времени решения, будут предоставляться достаточно большие кванты времени уже при первом занятии ими ресурса (нерационально программе, которая требует для своего решения 1 час времени, первоначально выделять квант в 1 мс

## 

## 

## 5.Режимы работы мультипрограммной ЭВМ

Мультипрограммная ЭВМ может работать в различных режимах, использование того или иного из них определяется областью ее применения. Среди основных режимов работы мультипрограммной ЭВМ выделим следующие:

1. пакетный ;
2. разделения времени ;
3. реального времени.

#### Пакетный режим

Суть пакетного режима заключается в том, что ЭВМ обрабатывает предварительно сформированный пакет задач без вмешательства пользователя в процесс обработки.

Пакетный режим используется, как правило, на высокопроизводительных ЭВМ. Основное требование к организации вычислительного процесса на компьютере, работающем в пакетном режиме, - это минимизация времени решения всего пакета задач за счет эффективной загрузки оборудования ЭВМ.

При пакетном режиме основным показателем эффективности служит **пропускная способность ЭВМ** - число задач, выполненных в единицу времени.

Количественная оценка выигрыша при мультипрограммной работе по сравнению с однопрограммным использованием ЭВМ представляется в виде **коэффициента увеличения пропускной способности**:

kПС = TОПР/TМПР

где ТОПР и ТМПР - время выполнения пакета задач при однопрограммном и мультипрограммном режиме работы соответственно.

В рассмотренном в лекции 12 примере работы мультипрограммной ЭВМ kПС = 36/24 = 1,5 при Км = 2 и k_{ПС} = 36/22 \approx  1,64при Км = 3.

Увеличение пропускной способности ЭВМ достигается надлежащим планированием поступления задач пакета на обработку в составе мультипрограммной смеси задач, а также оптимальным назначением приоритетов задачам в этих смесях, основывающемся на представлениях разработчиков о важности учета тех или иных аспектов функционирования ЭВМ и свойств каждой задачи входного пакета.

**Основные этапы обработки пакета задач**:

1. Подготовка программ к счету. При этом каждая программа пакета может быть разработана отдельным программистом.
2. Передача программ и исходных данных на ЭВМ, которая будет обрабатывать их в пакетном режиме.
3. Формирование пакета задач из переданных программ по одному из эвристических алгоритмов.
4. Обработка пакета задач на мультипрограммной ЭВМ.

**Особенности пакетного режима работы**:

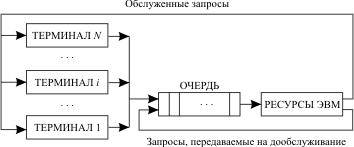
1. Пользователь отстранен от непосредственного доступа к ЭВМ.
2. Результаты работы пользователь получает через определенное (иногда достаточно большое) время одновременно для всех задач пакета.
3. Увеличивается время отладки программ.
4. Существенно возрастает пропускная способность ЭВМ по сравнению с последовательным решением задач пакета.

Таким образом, пакетный режим наиболее эффективен при обработке больших отлаженных программ.

#### 

#### Режим разделения времени

**Назначение** - обслуживание конечного числа пользователей с приемлемым для каждого пользователя временем ответа на их запросы ([рис. 13.6](https://www.intuit.ru/studies/courses/60/60/lecture/1790?page=2#image.13.6)).



**Рис. 13.6.** Организация работы ЭВМ в режиме разделения времени

**Основные характеристики**:

1. Многотерминальная многопользовательская система.
2. Любой пользователь со своего терминала может обратиться к любым ресурсам ЭВМ.
3. У пользователя создается впечатление, что он один работает на ЭВМ.

**Реализация**.

Время работы машины разделяется на кванты tk.

Каждый квант выделяется для соответствующего терминала. Терминалы могут быть активными и пассивными: активный реально включен в обслуживание (за ним работает пользователь), пассивный - нет (квант не выделяется). После обслуживания всех терминалов последовательность квантов повторяется.

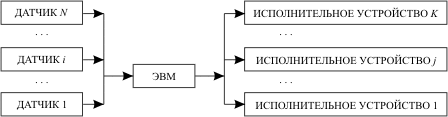
Единого способа выбора времени кванта не существует. Иногда оно выбирается по количеству команд, которое должна выполнить ЭВМ за это время.

В основе реализации режима разделения времени лежит одноочередная дисциплина обслуживания пользователей.

#### 

#### Режим реального времени

Этот режим работы мультипрограммных ЭВМ используется, как правило, в системах автоматического управления объектом ([рис. 13.7](https://www.intuit.ru/studies/courses/60/60/lecture/1790?page=2#image.13.7)).



**Рис. 13.7.** Организация работы ЭВМ в режиме реального времени

Назначение - обеспечить выполнение задания за время, не превышающее максимально допустимого для данного задания. Большую роль играют дисциплины распределения ресурсов, особенно назначение приоритетов задачам.

Режим реального времени имеет много общего с системой разделения времени:

* много терминалов - много датчиков,
* много терминалов - много исполнительных устройств.

Особое внимание при построении систем реального времени уделяется вопросам обеспечения надежности функционирования системы.

# 

# 

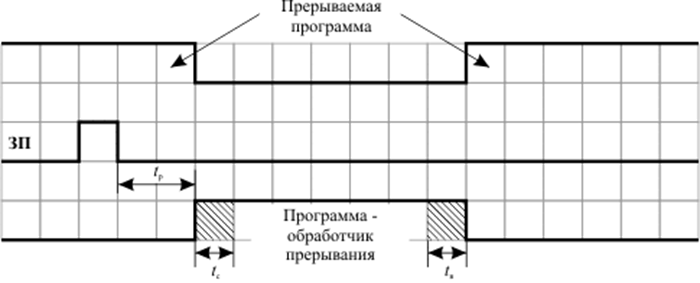
# **Организация работы персональной ЭВМ**

## **1.2.3 Обработка прерываний в персональной ЭВМ. Источники прерываний. Тип прерывания. Вектор прерывания. Контроллер приоритетных прерываний. Таблица векторов прерывания: назначение, структура.**

Прерывание-это прекращение выполнения текущей команды или текущей последовательности команд для обработки некоторого события специальной программой - обработчиком прерывания, с последующим возвратом к выполнению прерванной программы. Событие может быть вызвано особой ситуацией, сложившейся при выполнении программы, или сигналом от внешнего устройства. Прерывание используется для быстрой реакции процессора на особые ситуации, возникающие при выполнении программы и взаимодействии с внешними устройствами.

Механизм прерывания обеспечивается соответствующими аппаратно-программными средствами компьютера.

Любая особая ситуация, вызывающая прерывание, сопровождается сигналом, называемым запросом прерывания (ЗП). Запросы прерываний от внешних устройств поступают в процессор по специальным линиям, а запросы, возникающие в процессе выполнения программы, поступают непосредственно изнутри микропроцессора. Механизмы обработки прерываний обоих типов схожи. Рассмотрим функционирование компьютера при появлении сигнала запроса прерывания, опираясь в основном на обработку аппаратных прерываний (рис. 14.1).

рис. 14.1

Выполнение прерывания в компьютере: tр - время реакции процессора на запрос прерывания ; tс - время сохранения состояния прерываемой программы и вызова обработчика прерывания; tв - время восстановления прерванной программы

После появления сигнала запроса прерывания ЭВМ переходит к выполнению программы - обработчика прерывания. Обработчик выполняет те действия, которые необходимы в связи с возникшей особой ситуацией. Например, такой ситуацией может быть нажатие клавиши на клавиатуре компьютера. Тогда обработчик должен передать код нажатой клавиши из контроллера клавиатуры в процессор и, возможно, проанализировать этот код. По окончании работы обработчика управление передается прерванной программе.

Время реакции -это время между появлением сигнала запроса прерывания и началом выполнения прерывающей программы (обработчика прерывания) в том случае, если данное прерывание разрешено к обслуживанию.

Время реакции зависит от момента, когда процессор определяет факт наличия запроса прерывания. Опрос запросов прерываний может проводиться либо по окончании выполнения очередного этапа команды (например, считывание команды, считывание первого операнда и т.д.), либо после завершения каждой команды программы.

Время реакции определяется для запроса с наивысшим приоритетом.

Глубина прерывания - максимальное число программ, которые могут прерывать друг друга. Глубина прерывания обычно совпадает с числом уровней приоритетов, распознаваемых системой прерываний. Работа системы прерываний при различной глубине прерываний ( n ) представлена на рис. 14.2. Здесь предполагается, что с увеличением номера запроса прерывания увеличивается его приоритет.

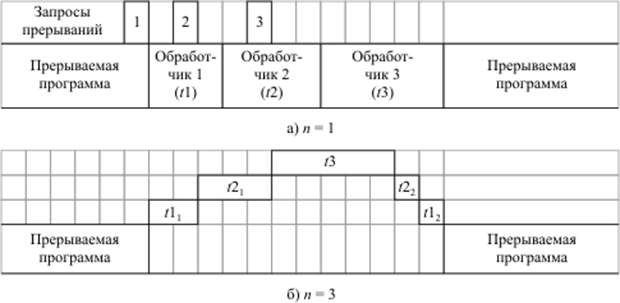


Рис. 14.2

Без учета времени реакции, а также времени запоминания и времени восстановления:

t11+t12=t1,

t21+t22=t2.

Прерывания делятся на аппаратные и программные [4]

Аппаратные прерывания используются для организации взаимодействия с внешними устройствами. Запросы аппаратных прерываний поступают на специальные входы микропроцессора. Они бывают:

· маскируемые, которые могут быть замаскированы программными средствами компьютера;

· немаскируемые, запрос от которых таким образом замаскирован быть не может.

Программные прерывания вызываются следующими ситуациями:

· особый случай, возникший при выполнении команды и препятствующий нормальному продолжению программы (переполнение, нарушение защиты памяти, отсутствие нужной страницы в оперативной памяти и т.п.);

· наличие в программе специальной команды прерывания INT n, используемой обычно программистом при обращениях к специальным функциям операционной системы для ввода-вывода информации.

Каждому запросу прерывания в компьютере присваивается свой номер (**тип прерывания**), используемый для определения адреса обработчика прерывания.

При поступлении запроса прерывания компьютер выполняет следующую последовательность действий:

1. определение наиболее приоритетного незамаскированного запроса на прерывание (если одновременно поступило несколько запросов);

2. определение типа выбранного запроса;

3. сохранение текущего состояния счетчика команд и регистра флагов;

4. определение адреса обработчика прерывания по типу прерывания и передача управления первой команде этого обработчика;

5. выполнение программы - обработчика прерывания;

6. восстановление сохраненных значений счетчика команд и регистра флагов прерванной программы;

7. продолжение выполнения прерванной программы.

Этапы 1-4 выполняются аппаратными средствами ЭВМ автоматически при появлении запроса прерывания. Этап 6 также выполняется аппаратно по команде возврата из обработчика прерывания.

Распознавание наличия сигналов запроса прерывания и определение наиболее приоритетного из них может проводиться различными методами. Рассмотрим один из них.

*Цепочечная однотактная система определения приоритета запроса прерывания*

На рис. 14.3 приведена схема, обеспечивающая получение номера наиболее приоритетного запроса прерывания из присутствующих в компьютере на момент подачи сигнала опроса ("дейзи-цепочка")

Данная схема используется для анализа запросов аппаратных прерываний. Приоритет запросов прерываний ( ЗПi ) уменьшается с уменьшением номера запроса. В тот момент, когда компьютер должен определить наличие и приоритет внешнего аппаратного прерывания (обычно после окончания выполнения каждой команды), процессор выдает сигнал опроса. Если на входе ЗП3 присутствует сигнал высокого уровня (есть запрос ), то на элементе 11 формируется общий сигнал наличия запроса прерывания и дальнейшее прохождение сигнала опроса блокируется. Если ЗП3=0, то анализируется сигнал ЗП2 и так далее. На шифраторе (элемент 12) формируется номер поступившего запроса прерывания.

Этот номер передается в процессор лишь при наличии общего сигнала запроса прерывания.

Такая структура позволяет быстро анализировать наличие сигнала запроса прерывания и определять наиболее приоритетный запрос из нескольких присутствующих в данный момент. Распределение приоритетов запросов прерываний внешних устройств осуществляется путем их физической коммутации по отношению к процессору. Указание приоритетов - жесткое и не может быть программно изменено. Изменение приоритетов возможно только путем физической перекоммутации устройств.

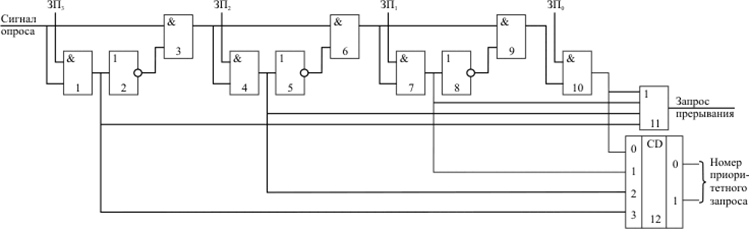


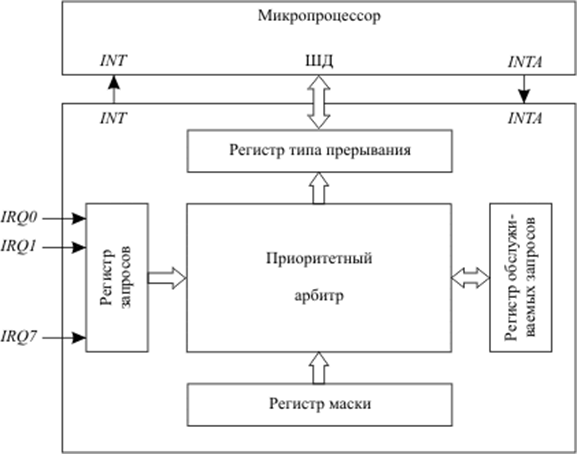
Рис. 14.3. Схема определения номера наиболее приоритетного запроса прерывания

Микропроцессоры типа х86 имеют два входа *запросов внешних аппаратных прерываний*:

*NMI* - *немаскируемое прерывание*, используется обычно для *запросов прерываний* по нарушению питания;

INT - *маскируемое прерывание*, *запрос* от которого можно программным образом замаскировать путем сброса флага IF в регистре флагов.

Вектор прерывания — закреплённый за устройством номер, который идентифицирует соответствующий обработчик прерываний. Векторы прерываний объединяются в [таблицу векторов прерываний](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0_%D0%B2%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BE%D0%B2_%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9), содержащую адреса обработчиков прерываний. Местоположение таблицы зависит от типа и режима работы процессора.



**Рис. 14.4. Структура контроллера приоритетных прерываний(Контроллер приоритетных прерываний)**

Единственный вход *запроса маскируемых прерываний* микропроцессора не позволяет подключить к нему напрямую сигналы *запросов* от большого числа различных внешних устройств, которые входят в состав современного компьютера: таймера, клавиатуры, "мыши", принтера, сетевой карты и т.д. Для их подключения к одному входу INT микропроцессора используется *контроллер приоритетных прерываний* (Рис. 14.4). Его функции:

· восприятие и фиксация *запросов прерываний* от внешних устройств;

· определение незамаскированных *запросов* среди поступивших *запросов*;

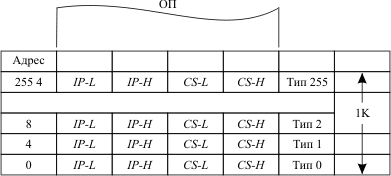
· проведение *арбитража*: выделение наиболее приоритетного *запроса* из незамаскированных *запросов* в соответствии с установленным механизмом назначения *приоритетов*;

· сравнение *приоритета выделенного запроса* с *приоритетом запроса*, который в данный момент может обрабатываться в микропроцессоре, формирование сигнала запроса на вход INT микропроцессора в случае, если *приоритет нового запроса* выше;

· передача в микропроцессор по шине данных типа *прерывания*, выбранного в процессе *арбитража*, для запуска соответствующей программы - обработчика *прерывания*; это действие выполняется по сигналу разрешения *прерывания* INTA от микропроцессора, который выдается в случае, если *прерывания* в регистре флагов микропроцессора не замаскированы (IF=1).

· Переход к соответствующему обработчику *прерывания* осуществляется (в реальном режиме работы микропроцессора) посредством *таблицы векторов прерываний*

***Таблица векторов прерываний*** ([рис. 7.2](https://www.intuit.ru/studies/courses/604/460/lecture/10333?page=1#image.7.2)) располагается в самых младших адресах оперативной памяти, имеет объем 1 Кбайт и содержит 4байтные элементы ( ***векторы прерываний*** *)* для 256 обработчиков прерываний. Старшие 2 байта вектора загружаются в *сегментный регистр* команд CS, а младшие 2 байта - в регистр указателя команд IP. Обращение к элементам таблицы осуществляется по 8-разрядному коду - **типу прерывания** (табл. 14.1). Так как таблица всегда имеет нулевой начальный *адрес* и длину вектора в 4 байта, чтобы определить *адрес* вектора для прерывания типа i, достаточно просто умножить это *значение* на 4.



**Рис. 14.5.** Структура таблицы векторов прерываний

|  |  |
| --- | --- |
| Таблица 14.1.Типы прерывания | |
| **Тип прерывания** | **Источник прерывания** |
| 0 | Деление на 0 |
| 1 | Пошаговый режим выполнения программы |
| 2 | Запрос по входу *NMI* |
| x x x | |
| 8 | Запрос по входу IRQ0 (системный таймер) |
| 9 | Запрос по входу IRQ1 (контроллер клавиатуры) |
| x x x | |
| 11 | Отсутствие сегмента в оперативной памяти |
| x x x | |
| 255 | Пользовательское прерывание |

**Тип прерывания** – номер, присвоенный каждому запросу прерывания, для того чтобы микропроцессор мог идентифицировать источник прерывания и найти обработчик, соответствующий полученному запросу.

Различные источники задают тип прерывания по-разному:

· программные прерывания вводят его изнутри процессора или содержат его в номере команды INT n ;

· аппаратные маскируемые прерывания вводят его от контроллера приоритетных прерываний по шине данных;

· немаскируемому аппаратному прерыванию назначен тип 2.

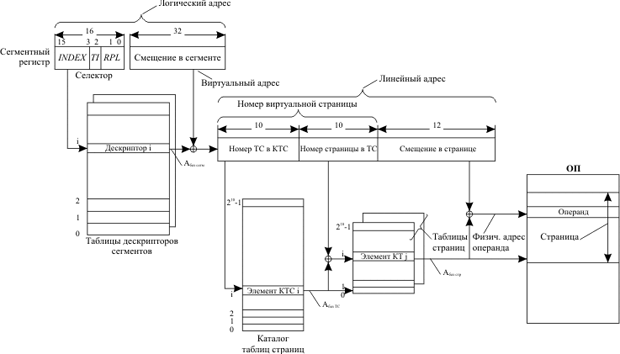
***Источниками аппаратных прерываний*** являются как порты,контроллеры и адаптеры, сообщающие о готовности принять/передать машинное слово или о завершении обмена массивом слов, так и другие устройства, требующие выполнения срочной программы, например, таймер, контроллеры памяти и др.

## **4.Организация памяти в персональной ЭВМ. Сегментное преобразование адреса в персональной ЭВМ. Логический адрес. Селектор. Дескриптор сегмента. Таблицы дескрипторов.**

В ЭВМ на основе 32-разрядного микропроцессора при работе в так называемом защищенном режиме, поддерживающем мультипрограммирование и обеспечивающем адресацию операндов в максимально возможном для данной архитектуры диапазоне до 232 байт, виртуальная память организуется на основе сегментно-страничного представления памяти. При этом память разбивается на сегменты переменной длины, выделяемые пользователю под размещение его программ и данных. Сегменты, в свою очередь, делятся на страницы фиксированной длины (4К = 212 байт), используемые системой управления памятью для ее виртуализации.

Начало каждого сегмента устанавливается операционной системой через соответствующий сегментный регистр и скрыто от пользователя. Пользователь пишет свои программы в адресах относительно начала сегмента, полагая, что он располагает сегментом максимально возможной для данной архитектуры длины (232 байт). Аппаратные средства микропроцессора сначала проводят сегментное преобразование адреса, а затем - страничное.

Механизм формирования физического адреса при сегментно-страничной организации памяти показан на [рис. 16.1](https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=1#image.16.1)

[](https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=1#image.16.1)

**Рис. 16.1.** Формирование физического адреса при сегментно-страничной организация памяти в 32-разрядном микропроцессоре

Основой получения физического адреса, выдаваемого на адресную шину микропроцессора, служит **логический адрес.** Он состоит из двух частей: селектора, являющегося идентификатором сегмента, и смещения в сегменте.

Смещение в сегменте (32 разряда) (эффективный *адрес*) вычисляется по задаваемому в команде режиму адресации операнда и является виртуальным адресом операнда. При обращении к команде в качестве смещения выступает *значение* регистра-указателя команд.

**Селектор** размещается в сегментном регистре (см. [рис. 16.1](https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=1#image.16.1) ). Основная его часть представляет собой номер ( INDEX ), по которому в одной из специальных таблиц дескрипторов можно найти *дескриптор (описатель)* данного сегмента. Вид используемой таблицы определяется битом TI (table indicator) селектора. Селектор содержит также двухразрядное поле RPL, используемое при организации защиты памяти по привилегиям.

**Дескриптор** ( [рис. 16.2](https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=1#image.16.2)) содержит сведения о сегменте. В одном из его полей содержится базовый адрес сегмента. В остальных полях записана дополнительная информация о сегменте: длина, допустимый уровень прав доступа к данному сегменту с целью защиты находящейся в нем информации, тип сегмента (сегмент кода, сегмент данных, специальный системный сегмент и т.д.) и некоторые другие атрибуты.



**Рис. 16.2.** Структура дескриптора сегмента

**Дескрипторные таблицы** — служебные структуры данных, содержащие дескрипторы сегментов.

Сумма полученного из дескриптора базового адреса сегмента и вычисленного смещения дает линейный адрес операнда, который при включенном механизме страничного преобразования представляет собой номер виртуальной страницы (старшие 20 разрядов) и смещение операнда в странице (младшие 12 разрядов линейного адреса в соответствии с объемом страницы в 4 Кбайт).

При преобразовании номера виртуальной страницы в номер физической используются следующие системные объекты: каталог таблиц страниц (КТС) и таблицы страниц (ТС). Структуры этих таблиц сходны между собой ( [рис. 16.3](https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=1#image.16.3)).



**Рис. 16.3.** Элемент каталога таблиц страниц (таблицы страниц)

Преобразование проводится в два этапа.

Сначала по разрядам А31-А22 линейного адреса в КТС выбирается нужный элемент. Каталог таблиц страниц всегда присутствует в ОП и содержит указания по размещению таблицы страниц, относящейся к тому или иному процессу.

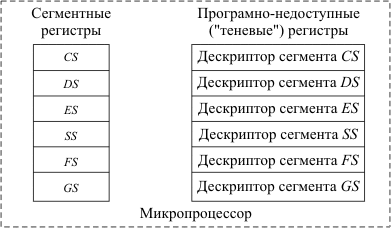
Элемент КТС содержит адрес начала таблицы страниц, бит присутствия ( P ) таблицы страниц в оперативной памяти, бит разрешения чтения/записи ( R/W ), бит защиты страницы (пользователь/супервизор ( U/S )) и некоторые другие атрибуты.

После получения из выбранного элемента КТС начального адреса таблицы страниц происходит обращение к ТС. В выбранной таблице страниц находится элемент, номер которого определяется разрядами А21-А12 линейного адреса. Структура элемента таблицы страниц аналогична структуре элемента КТС. Элемент ТС в соответствующем поле содержит адрес начала требуемой физической страницы и другие атрибуты, аналогичные элементу КТС.

При P =0 возникает прерывание, необходимая страница подкачивается в ОП, ее адрес заносится в соответствующий элемент ТС, и команда выполняется повторно.

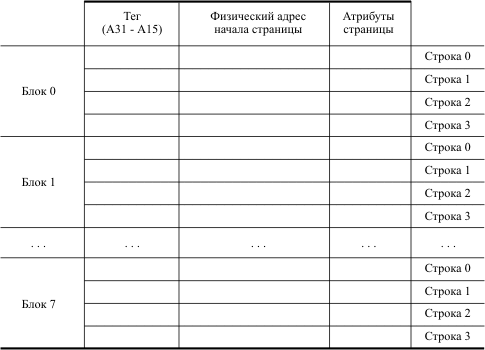
Преобразование *логического адреса* в физический при сегментно-страничной организации памяти требует, как минимум, трех обращений к системным таблицам, расположенным в оперативной памяти (таблице дескрипторов, *КТС* и *ТС* ). Это может привести к существенному снижению производительности компьютера. Механизм сокращения потерь времени на такое преобразование основывается на том факте, что изменение состояния сегментных регистров производится относительно редко, например, при переключении ЭВМ на новую задачу, а новое *страничное преобразование* требуется лишь при выходе программы за пределы загруженной в оперативную память страницы.

При ***сегментном преобразовании адреса*** после первого считывания *дескриптора* из таблицы дескрипторов, расположенной в оперативной памяти (например, после изменения состояния сегментного регистра при переключении на новую задачу), он запоминается в программно-недоступных ("теневых") регистрах микропроцессора ( [рис. 16.4](https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=2#image.16.4)). При последующих обращениях к данному сегменту используется *дескриптор* из "теневого" регистра без обращения к ОП. Поэтому на его вызов требуется минимальное время. Так как состояние сегментных регистров меняется относительно редко, то такой подход приводит к значительной экономии времени при сегментном преобразовании адреса.



**Рис. 16.4.** Сохранение дескрипторов сегментов в "теневых" регистрах микропроцессора

При *страничном преобразовании* номера *виртуальной страницы* в номер *физической страницы* используется *кэш-буфер ассоциативной трансляции* ( TLB ), содержащий физические адреса 32-х наиболее активно используемых страниц ( [рис. 16.5](https://www.intuit.ru/studies/courses/60/60/lecture/1796?page=2#image.16.5)) и расположенный непосредственно в микропроцессоре.



**Рис. 16.5.** Формат буфера ассоциативной трансляции адреса страницы

Номер виртуальной страницы представляет собой старшие 20 разрядов линейного адреса, полученного при сегментном преобразовании ( А31 - А12 ). По младшим разрядам ( А14 - А12 ) этого номера выбирается блок в буфере ассоциативной трансляции. Содержимое поля тэгов каждой из четырех строк этого блока ассоциативным образом (одновременно) сравнивается с разрядами ( А31 - А15 ) линейного адреса. Если значения для одной из строк выбранного блока совпали, значит, номер этой *виртуальной страницы* уже преобразовывался в номер *физической страницы* и результат этого преобразования находится в найденной строке TLB. Если сравнение не было успешным, то преобразование номера *виртуальной страницы* в номер физической проходит обычным образом через обращения к каталогу таблиц страниц и к таблице страниц, а полученное значение заносится в TLB. При этом в поле тэгов заносятся старшие 17 разрядов линейного адреса этой страницы ( A31-A15 ). Если нет свободной строки в блоке, определяемом разрядами А14 - А12 линейного адреса, то из блока вытесняется строка, информация в которой дольше всего не использовалась (механизм LRU ).

## **5. Защита памяти в мультипрограммных ЭВМ. Назначение. Способы защиты. Защита отдельных ячеек памяти. Метод граничных регистров. Метод ключей защиты памяти.**

При мультипрограммном режиме работы ЭВМ в ее памяти одновременно могут находиться несколько независимых программ. Поэтому необходимы специальные меры по предотвращению или ограничению обращений одной программы к областям памяти, используемым другими программами. Программы могут также содержать ошибки, которые, если этому не воспрепятствовать, приводят к искажению информации, принадлежащей другим программам. Последствия таких ошибок особенно опасны, если разрушению подвергнутся программы операционной системы. Другими словами, надо исключить воздействие программы пользователя на работу программ других пользователей и программ операционной системы. Следует защищать и сами программы от находящихся в них возможных ошибок.

**Назначение защиты:**

Таким образом, средства *защиты памяти* должны предотвращать неразрешенное взаимодействие пользователей друг с другом, несанкционированный доступ пользователей к данным, повреждение программ и данных из-за ошибок в программах, намеренные попытки разрушить *целостность системы*, использование информации в памяти не в соответствии с ее функциональным назначением. Чтобы воспрепятствовать разрушению одних программ другими, достаточно защитить область памяти данной программы от попыток записи в нее со стороны других программ, а в некоторых случаях и своей программы ( *защита от записи* ), при этом допускается обращение других программ к этой области памяти для считывания данных.

В других случаях, например при ограничениях на *доступ* к информации, хранящейся в системе, необходимо запрещать другим программам любое обращение к некоторой области памяти как на *запись*, так и на считывание. Такая *защита от записи* и *считывания* помогает в отладке программы, при этом осуществляется *контроль* каждого случая обращения за область памяти своей программы.

Для облегчения отладки программ желательно выявлять и такие характерные ошибки в программах, как попытки использования данных вместо команд или команд вместо данных в собственной программе, хотя эти ошибки могут и не разрушать информацию (несоответствие функционального использования информации).

Если нарушается *защита памяти*, *исполнение* программы приостанавливается и вырабатывается *запрос прерывания* по нарушению *защиты памяти*.

Защита от вторжения программ в чужие области памяти может быть организована различными методами. Но при любом подходе реализация защиты не должна заметно снижать *производительность* компьютера и требовать слишком больших аппаратурных затрат.

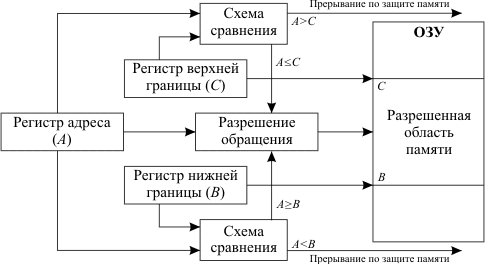
**Способы защиты:**

Методы защиты базируются на некоторых классических подходах, которые получили свое развитие в архитектуре современных ЭВМ. К таким методам можно отнести защиту отдельных ячеек, *метод граничных регистров*, *метод ключей защиты* .

**Защита отдельных ячеек памяти** организуется в ЭВМ, предназначенных для работы в системах управления, где необходимо обеспечить возможность отладки новых программ без нарушения функционирования находящихся в памяти рабочих программ, управляющих технологическим процессом. Это может быть достигнуто выделением в каждой ячейке памяти специального "разряда защиты". Установка этого разряда в "1" запрещает производить *запись* в данную ячейку, что обеспечивает сохранение рабочих программ. Недостаток такого подхода - большая *избыточность* в *кодировании информации* из-за излишне мелкого уровня защищаемого объекта (*ячейка*).

В системах с мультипрограммной обработкой целесообразно организовывать защиту на уровне блоков памяти, выделяемых программам, а не отдельных ячеек.

**Метод граничных регистров** ([рис. 17.1](https://www.intuit.ru/studies/courses/60/60/lecture/1798?page=1#image.17.1)) заключается во введении двух граничных регистров, указывающих верхнюю и нижнюю границы области памяти, куда *программа* имеет *право доступа*.



**Рис. 17.1.** Защита памяти методом граничных регистров

При каждом обращении к памяти проверяется, находится ли используемый *адрес* в установленных границах. При выходе за границы обращение к памяти не производится, а формируется *запрос прерывания*, передающий управление операционной системе. Содержание граничных регистров устанавливается операционной системой при загрузке программы в *память*.

Модификация этого *метода* заключается в том, что один *регистр* используется для указания адреса начала защищаемой области, а другой содержит длину этой области.

*Метод граничных регистров*, обладая несомненной простотой реализации, имеет и определенные недостатки. Основным из них является то, что этот *метод* поддерживает работу лишь с непрерывными областями памяти.

**Метод ключей защиты**, в отличие от предыдущего, позволяет реализовать *доступ* программы к областям памяти, организованным в виде отдельных модулей, не представляющих собой единый *массив*.

*Память* в логическом отношении делится на одинаковые блоки, например, страницы. Каждому блоку памяти ставится в соответствие код, называемый ключом защиты памяти, а каждой программе, принимающей участие в мультипрограммной обработке, присваивается код ключа программы. *Доступ* программы к данному блоку памяти для чтения и записи разрешен, если ключи совпадают (то есть данный *блок памяти* относится к данной программе) или один из них имеет код 0 (код 0 присваивается программам операционной системы и блокам памяти, к которым имеют *доступ* все программы: общие данные, совместно используемые подпрограммы и т. п.). Коды ключей защиты блоков памяти и ключей программ устанавливаются операционной системой.

В ключе *защиты памяти* предусматривается дополнительный разряд режима защиты. Защита действует только при попытке записи в блок, если в этом разряде стоит 0, и при любом обращении к блоку, если стоит 1. Коды ключей *защиты памяти* хранятся в специальной памяти ключей защиты, более быстродействующей, чем оперативная *память*.

Функционирование этого механизма *защиты памяти* поясняется схемой на [рис. 17.2](https://www.intuit.ru/studies/courses/60/60/lecture/1798?page=1#image.17.2).



**Рис. 17.2.** Защита памяти методом ключей защиты

При обращении к памяти *группа* старших разрядов адреса *ОЗУ*, соответствующая номеру блока, к которому производится обращение, используется как *адрес* для выборки из памяти ключей защиты кода ключа защиты, присвоенного операционной системой данному блоку. Схема анализа сравнивает *ключ* защиты блока памяти и *ключ* программы, находящийся в регистре слова состояния программы (*ССП*), и вырабатывает сигнал "Обращение разрешено" или сигнал "*Прерывание* по защите памяти". При этом учитываются значения режима обращения к *ОЗУ* (*запись* или считывание), указываемого триггером режима обращения ТгРО, и режима защиты, установленного в разряде режима обращения (РРО) ключа *защиты памяти*.

## **6. Организация защиты памяти в персональной ЭВМ. Защита при управлении памятью. Защита по привилегиям.**

Защита памяти в персональной ЭВМ делится на защиту при управлении памятью и защиту по привилегиям.

Средства защиты при управлении памятью осуществляют проверку превышения эффективным *адресом* длины сегмента, прав доступа к сегменту на запись или только на чтение, функционального назначения сегмента. Первый механизм базируется на *методе граничных регистров*. При этом начальные адреса того или иного сегмента программы устанавливаются операционной системой. Для каждого сегмента фиксируется его *длина*. При попытке обращения по относительному адресу, превышающему длину сегмента, вырабатывается сигнал нарушения *защиты памяти*.

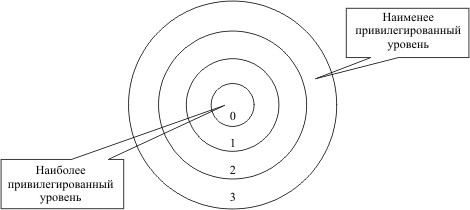
При проверке функционального назначения сегмента определяются операции, которые можно проводить над находящимися в нем данными. Так, сегмент, представляющий собой стек программы, должен допускать обращение как на запись, так и на чтение информации. К сегменту, содержащему программу, можно обращаться только на исполнение. Любое обращение на запись или чтение данных из этого сегмента будет воспринято как ошибочное. Здесь наблюдается некоторый отход от принципов Неймана в построении ЭВМ, в которых утверждается, что любая информация, находящаяся в ЗУ, функционально не разделяется на программу и данные, а ее идентификация проводится лишь на стадии применения данной информации. Очевидно, что такое развитие вызвано во многом появлением мультипрограммирования и необходимостью более внимательного рассмотрения вопросов защиты информации.

**Защита по привилегиям** фиксирует более тонкие ошибки, связанные, в основном, с разграничением прав доступа к той или иной информации.

В какой-то степени *защиту по привилегиям* можно сравнить с классическим *методом ключей защиты* памяти. Различным объектам (программам, сегментам памяти, запросам на обращение к памяти и к внешним устройствам), которые должны быть распознаны процессором, присваивается идентификатор, называемый ***уровнем привилегий*** . *Процессор* постоянно контролирует, имеет ли текущая *программа* достаточные привилегии, чтобы выполнять некоторые команды, выполнять команды ввода-вывода на том или ином внешнем устройстве, обращаться к данным других программ, вызывать другие программы.

На аппаратном уровне в процессоре различаются 4 *уровня привилегий*. Наиболее привилегированными являются программы на *уровне* 0.

Число программ, которые могут выполняться на более высоком *уровне привилегий*, уменьшается от *уровня* 3 к *уровню* 0. Программы *уровня* 0 действуют как *ядро* операционной системы. Поэтому *уровни привилегий* обычно изображаются в виде четырех *колец защиты* ([рис. 17.3](https://www.intuit.ru/studies/courses/60/60/lecture/1798?page=2#image.17.3)).



**Рис. 17.3.** "Кольца защиты"

Как правило, распределение программ по кольцам защиты имеет следующий вид:

уровень 0 - *ядро* ОС, обеспечивающее инициализацию работы, *управление доступом* к памяти, защиту и ряд других жизненно важных функций, нарушение которых полностью выводит из строя *процессор*;

уровень 1 - основная часть программ ОС (утилиты);

уровень 2 - служебные программы ОС (драйверы, *СУБД*, специализированные подсистемы программирования и др.);

уровень 3 - прикладные программы пользователя.

Конкретная *операционная система* не обязательно должна поддерживать все четыре *уровня привилегий*. Так, ОС **UNIX** работает с двумя *кольцами защиты*: *супервизор* ( *уровень* 0) и *пользователь* ( *уровни* 1,2,3). *Операционная система* **OS/2** поддерживает три *уровня*: код ОС работает в *кольце* 0, специальные процедуры для обращения к устройствам ввода-вывода действуют в *кольце* 1, а прикладные программы выполняются в *кольце* 3.

Простую незащищенную систему можно целиком реализовать в одном *кольце* 0 (в других *кольцах* это сделать невозможно, так как некоторые команды доступны только на этом *уровне* ).

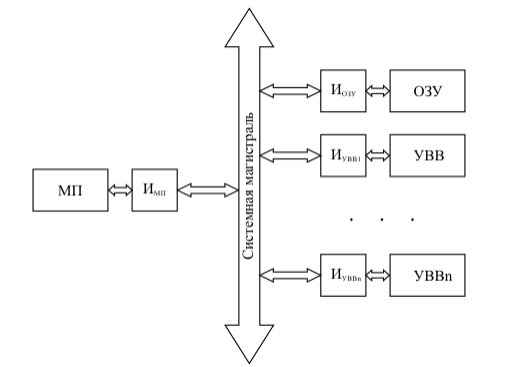
*Уровень привилегий* сегмента определяется полем *DPL* *уровня привилегий* его дескриптора. *Уровень привилегий* запроса к сегменту определяется *уровнем привилегий* *RPL*, закодированном в селекторе. Обращение к сегменту разрешается только тогда, когда *уровень привилегий* сегмента не выше *уровня запроса*. Обращение к программам, находящимся на более высоком *уровне привилегий* (например, к утилитам операционной системы из программ пользователя), возможно с помощью специальных аппаратных механизмов - шлюзов вызова.

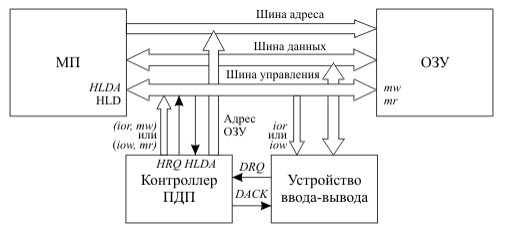
При страничном преобразовании адреса применяется простой двухуровневый механизм защиты: пользователь ( уровень 3) / супервизор ( уровни 0,1,2), указываемый в поле U/S соответствующей таблицы страниц.

При сегментно-страничном преобразовании адреса сначала проверяются привилегии при доступе к сегменту, а затем - при доступе к странице. Это дает возможность установить более высокую степень защиты отдельных страниц сегмента.

## 7. Ввод-вывод информации в ЭВМ. Проблемы организации ввода вывода и пути их решения.

Любая ЭВМ представляет собой сложную систему, включающую в себя большое количество различных устройств. Связь устройств ЭВМ между собой осуществляется с помощью сопряжений, которые в вычислительной технике называются интерфейсами.  
  
**Интерфейс** - это совокупность программных и аппаратных средств, предназначенных для передачи информации между компонентами ЭВМ и включающих в себя электронные схемы, линии, шины и сигналы адресов, данных и управления, алгоритмы передачи сигналов и правила интерпретации сигналов устройствами.  
  
Интерфейсы характеризуются следующими параметрами:  
  
· пропускная способность - количество информации, которая может быть передана через интерфейс в единицу времени;  
  
· максимальная частота передачи информационных сигналов через интерфейс ;  
  
· максимально допустимое расстояние между соединяемыми устройствами;  
  
· общее число проводов (линий) в интерфейсе ;  
  
· информационная ширина интерфейса - число бит или байт данных, передаваемых параллельно через интерфейс.  
  
К динамическим параметрам интерфейса относится время передачи отдельного слова и блока данных с учетом продолжительности процедур подготовки и завершения передачи.  
  
Разработка систем ввода-вывода требует решения целого ряда проблем, среди которых выделим следующие:  
  
· необходимо обеспечить возможность реализации ЭВМ с переменным составом оборудования, в первую очередь, с различным набором устройств ввода-вывода, с тем, чтобы пользователь мог выбирать конфигурацию машины в соответствии с ее назначением, легко добавлять новые устройства и отключать те, в использовании которых он не нуждается;  
  
· для эффективного и высокопроизводительного использования оборудования компьютера следует реализовать параллельную во времени работу процессора над вычислительной частью программы и выполнение периферийными устройствами процедур ввода-вывода;  
  
· необходимо упростить для пользователя и стандартизовать программирование операций ввода-вывода, обеспечить независимость программирования ввода-вывода от особенностей того или иного периферийного устройства;  
  
· в ЭВМ должно быть обеспечено автоматическое распознавание и реакция процессора на многообразие ситуаций, возникающих в УВВ (готовность устройства, отсутствие носителя, различные нарушения нормальной работы и др.).  
  
Главным направлением решения указанных проблем является **магистрально-модульный способ** построения ЭВМ рис. 18.1: все устройства, составляющие компьютер, включая и микропроцессор, организуются в виде модулей, которые соединяются между собой общей магистралью. Обмен информацией по магистрали удовлетворяет требованиям некоторого общего интерфейса, установленного для магистрали данного типа. Каждый модуль подключается к магистрали посредством специальных интерфейсных схем ( Иi ).

  
  
Рис. 18.1. Магистрально-модульный принцип построения ЭВМ  
  
На интерфейсные схемы модулей возлагаются следующие задачи:  
  
· обеспечение функциональной и электрической совместимости сигналов и протоколов обмена модуля и системной магистрали;  
  
· преобразование внутреннего формата данных модуля в формат данных системной магистрали и обратно;  
  
· обеспечение восприятия единых команд обмена информацией и преобразование их в последовательность внутренних управляющих сигналов.  
  
Эти интерфейсные схемы могут быть достаточно сложными и по своим возможностям соответствовать универсальным микропроцессорам. Такие схемы принято называть контроллерами.  
  
**Контроллеры** обладают высокой степенью автономности, что позволяет обеспечить параллельную во времени работу периферийных устройств и выполнение программы обработки данных микропроцессором.  
  
Недостатком магистрально-модульного способа организации ЭВМ является невозможность одновременного взаимодействия более двух модулей, что ставит ограничение на производительность компьютера. Поэтому этот способ, в основном, используется в ЭВМ, к характеристикам которых не предъявляется очень высоких требований, например в персональных ЭВМ.  
  
В ЭВМ используются **два основных способа организации передачи данных** между памятью и периферийными устройствами: программно-управляемая передача и прямой доступ к памяти (ПДП).  
  
Программно-управляемая передача данных осуществляется при непосредственном участии и под управлением процессора. Например, при пересылке блока данных из периферийного устройства в оперативную память процессор должен выполнить следующую последовательность шагов:  
  
1. сформировать начальный адрес области обмена ОП;  
  
2. занести длину передаваемого массива данных в один из внутренних регистров, который будет играть роль счетчика;  
  
3. выдать команду чтения информации из УВВ; при этом на шину адреса из МП выдается адрес УВВ, на шину управления - сигнал чтения данных из УВВ, а считанные данные заносятся во внутренний регистр МП;  
  
4. выдать команду записи информации в ОП; при этом на шину адреса из МП выдается адрес ячейки оперативной памяти, на шину управления - сигнал записи данных в ОП, а на шину данных выставляются данные из регистра МП, в который они были помещены при чтении из УВВ;  
  
5. модифицировать регистр, содержащий адрес оперативной памяти;  
  
6. уменьшить счетчик длины массива на длину переданных данных;  
  
7. если переданы не все данные, то повторить шаги 3-6, в противном случае закончить обмен.  
  
Как видно, программно-управляемый обмен ведет к нерациональному использованию мощности микропроцессора, который вынужден выполнять большое количество относительно простых операций, приостанавливая работу над основной программой. При этом действия, связанные с обращением к оперативной памяти и к периферийному устройству, обычно требуют удлиненного цикла работы микропроцессора из-за их более медленной по сравнению с микропроцессором работы, что приводит к еще более существенным потерям производительности ЭВМ.  
  
Альтернативой программно-управляемому обмену служит прямой доступ к памяти - способ быстродействующего подключения внешнего устройства, при котором оно обращается к оперативной памяти, не прерывая работы процессора [ 3 ] . Такой обмен происходит под управлением отдельного устройства - контроллера прямого доступа к памяти (КПДП) . Структура ЭВМ, имеющей в своем составе КПДП, представлена на рис. 18.2.

  
  
Рис. 18.2. Обмен данными в режиме прямого доступа к памяти  
  
Перед началом работы контроллер ПДП необходимо инициализировать: занести начальный адрес области ОП, с которой производится обмен, и длину передаваемого массива данных. В дальнейшем по сигналу запроса прямого доступа контроллер фактически выполняет все те действия, которые обеспечивал микропроцессор при программно-управляемой передаче.  
  
Последовательность действий КПДП при запросе на прямой доступ к памяти со стороны устройства ввода-вывода следующая:  
  
1. Принять запрос на ПДП (сигнал DRQ ) от УВВ.  
  
2. Сформировать запрос к МП на захват шин (сигнал HRQ ).  
  
3. Принять сигнал от МП ( HLDA ), подтверждающий факт перевода микропроцессором своих шин в третье состояние.  
  
4. Сформировать сигнал, сообщающий устройству ввода-вывода о начале выполнения циклов прямого доступа к памяти ( DACK ).  
  
5. Сформировать на шине адреса компьютера адрес ячейки памяти, предназначенной для обмена.  
  
6. Выработать сигналы, обеспечивающие управление обменом ( IOR, MW для передачи данных из УВВ в оперативную память и IOW, MR для передачи данных из оперативной памяти в УВВ).  
  
7. Уменьшить значение в счетчике данных на длину переданных данных.  
  
8. Проверить условие окончания сеанса прямого доступа (обнуление счетчика данных или снятие сигнала запроса на ПДП ). Если условие окончания не выполнено, то изменить адрес в регистре текущего адреса на длину переданных данных и повторить шаги 5-8.  
  
Прямой доступ к памяти позволяет осуществлять параллельно во времени выполнение процессором программы и обмен данными между периферийным устройством и оперативной памятью.  
  
Обычно программно-управляемый обмен используется в ЭВМ для операций ввода-вывода отдельных байт (слов), которые выполняются быстрее, чем при ПДП, так как исключаются потери времени на инициализацию контроллера ПДП, а в качестве основного способа осуществления операций ввода-вывода используют ПДП. Например, в стандартной конфигурации персональной ЭВМ обмен между накопителями на магнитных дисках и оперативной памятью происходит в режиме прямого доступа.  
  
Как отмечалось выше, обычно компьютер строится по магистрально-модульному принципу. При этом все составляющие его устройства объединяются общей шиной, по которой между ними происходит обмен данными, адресами, а также управляющими сигналами. В качестве примера перечислим основные линии, составляющие одну из распространенных системных магистралей - шину ISA:  
  
A0-A23 - шина адреса;  
  
D0-D15 - двунаправленная шина данных, допускает обмен как байтами, так и словами (2 байта);  
  
CLK - шинный тактовый сигнал, синхронизирует работу процессора, ОП и УВВ;  
  
MR - управляющий сигнал чтения из ОП;  
  
MW - управляющий сигнал записи в ОП;  
  
IOR - управляющий сигнал чтения из УВВ;  
  
IOW - управляющий сигнал записи в УВВ;  
  
IRQi - запрос прерывания от i -го источника;  
  
DRQi - запрос прямого доступа к памяти по i -му каналу контроллера ПДП ;  
  
DACKi - разрешение прямого доступа к памяти i -му каналу контроллера ПДП ;  
  
AEN - сигнал занятости шин обменом в режиме ПДП,  
  
READY - сигнал готовности УВВ к обмену.  
  
Магистраль обеспечивает подключение до семи внешних устройств, работающих в режиме прямого доступа к памяти, и до 11 запросов прерываний от УВВ. Еще четыре запроса прерываний зарезервированы за устройствами, входящими в состав стандартной конфигурации ЭВМ, и на магистраль не выведены.  
  
Хотя шина ISA имеет небольшую информационную ширину и в настоящее время используется для подключения только относительно медленных периферийных устройств, ее состав позволяет в определенной мере проследить взаимосвязь различных рассмотренных ранее устройств, составляющих ЭВМ.  
  
Организация ЭВМ на основе общей шины является сдерживающим фактором для повышения производительности компьютера. Следует отметить, что даже при использовании прямого доступа к памяти процессор полностью не освобождается от управления операциями ввода-вывода. Он обеспечивает инициализацию контроллера ПДП, а также взаимодействует с ним по некоторым управляющим линиям. Более того, во время операции передачи данных интерфейс оказывается занятым, а связь процессора с оперативной памятью - блокированной.  
  
Это существенно сказывается на эффективности работы ЭВМ, особенно в тех случаях, когда в вычислительной системе используется большое количество высокоскоростных внешних устройств. Для решения этой проблемы в состав высокопроизводительных компьютеров иногда включают специализированные процессоры ввода-вывода, способные полностью разгрузить основной процессор от управления операциями обмена с внешними устройствами.  
  
(Главная задача, которая рассматривает подсистема ввода-вывода это организация взаимодействия ЭВМ и внешних устройств.  
  
ЭВМ связывается с внешними устройствами посредством 3 шин: Шина Адреса, Шина Данных, Шина Устройства. Вместе они организуют системную магистраль.  
  
1. Скорость обмена данными с внешними устройствами значительно ниже пропускной способности ЦП и основной подмены, поэтому нерационально загружать высокоскоростную магистраль медленным процессом обмена с внешним устройством.  
  
2. Широкая номенкулатура внешних устройств с различными методами управления, которыми не стоит нагружать ЦП.  
  
3. Внешние устройства могут иметь другие форматы представления данных, отличных от ЭВМ.  
  
В связи с этими тремя причинами, внешние устройства невозможно подключить к шине напрямую. Поэтому используется спец модуль МВВ (модуль ввода-вывода).  
  
Две основные задачи, которые решает МВВ:  
  
1. Взаимодействие с ЦП и ОП через системную магистраль  
  
2. Взаимодействие с устройствами через специальные линии данных.)

## 8. Организация обмена информацией между оперативной памятью и внешними устройствами. Программно-управляемая передача данных Режим прямого доступа к памяти. Контроллер прямого доступа к памяти.

В МПС используются два основных способа организации передачи данных между памятью и периферийными устройствами: программно управляемая передача и прямой доступ к памяти ПДП ( Direct Memory Access - DMA ).

Программно управляемая передача данных осуществляется при непосредственном участии и под  
управлением процессора. Например, при пересылке блока данных из внешнего устройства в  
оперативную память процессор должен выполнить следующую последовательность шагов:

1. сформировать начальный адрес области обмена ОП;

2. занести длину передаваемого массива данных в один из своих внутренних регистров, который будет играть роль счетчика;

3. выдать команду чтения информации из ВУ; при этом на шину адреса из МП выдается адрес ВУ, на шину управления - сигнал чтения данных из ВУ, а считанные данные заносятся во внутренний регистр МП;

4. выдать команду записи информации в ОП; при этом на шину адреса из МП выдается адрес ячейки оперативной памяти, на шину управления - сигнал записи данных в ОП, а на шину данных выставляются данные из регистра МП, в который они были помещены при чтении из ВУ;

5. модифицировать регистр, содержащий адрес оперативной памяти;

6. уменьшить счетчик длины массива на длину переданных данных;

7. если переданы не все данные, то повторить шаги 3-6, в противном случае закончить обмен.

Как мы видим, программно управляемый обмен ведет к нерациональному использованию мощности микропроцессора, который вынужден выполнять большое количество относительно простых операций, приостанавливая работу над основной программой. При этом действия, связанные с обращением к оперативной памяти и к внешнему устройству, обычно требуют удлиненного цикла работы микропроцессора, что приводит к еще более существенным потерям производительности.

Альтернативой программно управляемому обмену служит прямой доступ к памяти - способ быстродействующего подключения внешнего устройства, при котором оно обращается к оперативной памяти, не прерывая работы процессора. Такой обмен происходит под управлением отдельного устройства - контроллера прямого доступа к памяти (КПДП).

Схема включения КПДП в состав микропроцессорной системы представлена на рис. 8.4

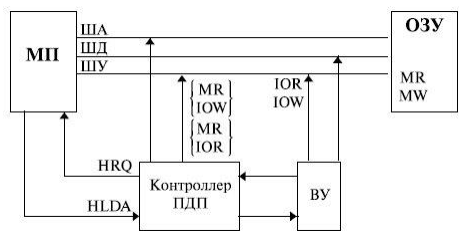


Рис. 8.4. Структура МПС с контроллером прямого доступа к памяти

Перед началом работы контроллер ПДП необходимо инициализировать: занести начальный адрес области ОП, с которой производится обмен, и длину передаваемого массива данных. В дальнейшем по сигналу запроса прямого доступа контроллер фактически выполняет все те действия, которые обеспечивал микропроцессор при программно управляемой передаче.

Последовательность действий КПДП при запросе на прямой доступ к памяти со стороны внешнего устройства следующая:

1. Принять запрос на ПДП (сигнал DRQ ) от ВУ.

2. Сформировать запрос к МП на захват шин (сигнал HRQ ).

3. Принять сигнал от МП ( HLDA ), подтверждающий факт перевода микропроцессором своих шин в третье состояние.

4. Сформировать сигнал, сообщающий устройству ввода-вывода о начале выполнения циклов прямого доступа к памяти ( DACK ).

5. Сформировать на шине адреса компьютера адрес ячейки памяти, предназначенной для обмена.

6. Выработать сигналы, обеспечивающие управление обменом ( IOR, MW для передачи данных из ВУ в оперативную память и IOW, MR для передачи данных из оперативной памяти в ВУ).

7. Уменьшить значение в счетчике данных на длину переданных данных.

8. Проверить условие окончания сеанса прямого доступа (обнуление счетчика данных или снятие сигнала запроса на ПДП). Если условие окончания не выполнено, то изменить адрес в регистре текущего адреса на длину переданных данных и повторить шаги 5-8.

Прямой доступ к памяти позволяет осуществлять обмен данными между внешним устройством и оперативной памятью параллельно с выполнением процессором программы.

Структура КПДП представлена на рис. 8.5.

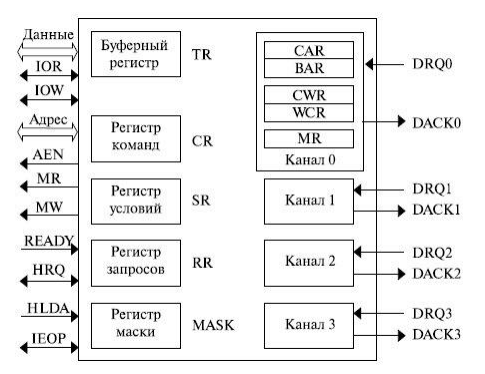


Рис. 8.5. Структура контроллера прямого доступа к памяти

Контроллер состоит из 4 идентичных каналов, позволяющих подключить к системе до четырех устройств, работающих в режиме ПДП.

В состав каждого канала входят следующие регистры:

• MR ( Mode Register ) - регистр режима. Определяет следующие параметры передачи:

o порядок изменения (увеличения или уменьшения) адреса ОП при передаче;

o возможность автоинициализации;

o режим обслуживания:

o одиночная передача (контроллер возвращает магистраль процессору после каждого цикла ПДП);

o блочная передача (контроллер владеет магистралью в течение передачи всего массива);

o по требованию (окончание передачи определяется снятием сигнала DRQ или подачей сигнала IEOP на внешний вход КПДП);

o каскадирование;

• CAR ( Current Address Register ) - регистр текущего адреса;

• BAR ( Base Address Register ) - базовый регистр адреса;

• CWR ( Current Word Register ) - текущий счетчик данных;

• WCR ( W0ord ConTRol Register ) - базовый счетчик данных.

Значения в регистрах BAR и WCR устанавливаются при инициализации и в ходе циклов ПДП не меняются. В регистры CAR и CWRв начале выполнения ПДП заносятся значения из регистров BAR и WCR соответственно. При выполнении ПДП эти регистры изменяются.

Управляющие регистры, общие для всего контроллера:

• CR ( Command Register ) - регистр команд - определяет:

o режим память-память или обычный. В режиме память-память осуществляется обмен по схеме ПДП между двумя областями ОП (только для каналов 0 и 1) с использованием буферного регистра TR ( Temporary Register );

o запрет/разрешение ПДП; o порядок изменения приоритетов каналов:

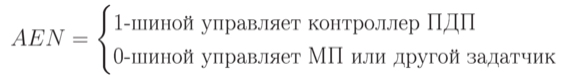
o фиксированный приоритет; o циклическое изменение приоритета после обработки запроса на ПДП по одному из каналов;

o явное указание наиболее приоритетного канала;

o уровень сигналов DRQ и DACK (настройка на активный уровень сигналов под особенности работы внешних устройств);

• SR ( Status Register ) - регистр условий: используется для фиксации признаков обмена по каждому из каналов и программных запросов на ПДП

Во избежание ложных срабатываний внешних устройств, не использующих в данный момент режим прямого доступа, контроллерПДП во время режима ПДП вырабатывает сигнал AEN, который блокирует работу остальных внешних устройств:



Для увеличения количества внешних устройств, которые могут быть подключены к микропроцессору в режиме ПДП, используется каскадное включение КПДП (рис. 8.6).

При этом ведомый КПДП подключается к одному из каналов ведущего контроллера по схеме подключения внешнего устройства, а непосредственно с микропроцессором связывается только ведущий контроллер. В стандартной конфигурации персональной ЭВМ применяются два КПДП (ведущий и ведомый), которые позволяют подключить до 7 внешних устройств, причем 2 канала закреплены за накопителями на жестком и гибком

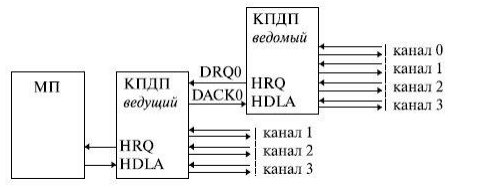


Рис. 8.6. Каскадное включение контроллеров прямого доступа к памяти

дисках. В максимальной конфигурации при использовании ведущего и 4 ведомых КПДП, подключенных к каждому из каналов ведущего, к МП можно подключить до 16 внешних устройств, работающих в режиме ПДП.

## 9. Конвейерная организация работы микропроцессора. Ступени конвейера

Выполнение каждой команды складывается из ряда последовательных этапов, суть которых не меняется от команды к команде. С целью увеличения быстродействия процессора и максимального использования всех его возможностей в современных микропроцессорах используется **конвейерный принцип обработки информации**. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в конвейере продвигается на следующую стадию обработки, выполненная команда покидает конвейер, а новая поступает в него.

В различных процессорах количество и суть этапов различаются.

Рассмотрим принципы конвейерной обработки информации на примере пятиступенчатого конвейера, в котором выполнение команды складывается из следующих этапов:

1. IF ( *INsTRuction Fetch* ) - считывание команды в процессор;

2. ID ( *INsTRuction DecodINg* ) - декодирование команды;

3. OR ( *Operand* ReadINg ) - считывание операндов;

4. EX ( ExecutINg ) - выполнение команды;

5. WB ( Write Back ) - запись результата.

|  |  |
| --- | --- |
|  |  |
|  |  |

Выполнение команд в таком конвейере представлено в табл. 9.1



Поскольку в каждом такте могут выполняться различные стадии обработки команд, длительность такта выбирается исходя из максимального времени выполнения всех стадий. Кроме того, следует учитывать, что для передачи команды с одной стадии обработки на другую требуется дополнительное время ( t), связанное с записью промежуточных результатов обработки в буферные регистры.

Пусть для выполнения отдельных стадий обработки требуются следующие затраты времени (в некоторых условных единицах):

TIF = 20, *TID* = 15, TOR = 20, TEX = 25, TWB = 20.

Тогда, предполагая, что дополнительные расходы времени составляют t = 5 единиц, получим время такта:



Оценим время выполнения одной команды и некоторой группы команд при последовательной и конвейерной  
обработке.  
При последовательной обработке время выполнения N команд составит:



Оценим время выполнения одной команды и некоторой группы команд при последовательной и конвейерной обработке.

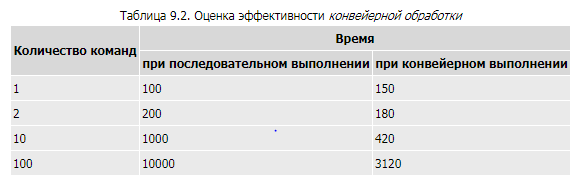
При последовательной обработке время выполнения N команд составит:



Примеры длительности выполнения некоторого количества команд при последовательной и конвейерной обработке приведены в [табл. 9.2.](https://www.intuit.ru/studies/courses/604/460/lecture/10337?page=1#table.9.2)

Очевидно, что при достаточно длительной работе конвейера его быстродействие будет существенно превышать быстродействие, достигаемое при последовательной обработке команд. Это увеличение будет тем больше, чем меньше длительность такта конвейера и чем больше количество выполненных за рассматриваемый период команд. Сокращение длительности такта может достигаться разбиением выполнения команды на большое число этапов, каждый из которых включает в себя относительно простые операции и поэтому будет выполняться за более короткий промежуток времени. Так, если в

микропроцессоре Pentium длина конвейера составляла 5 ступеней (при максимальной тактовой частоте 200 МГц), то в процессорах Pentium 4 на ядре Northwood длина конвейера составляла 20 ступеней, а на ядре Prescott она увеличена до 31 ступени при максимальной тактовой частоте 3,8 ГГц.

  
Значительное преимущество конвейерной обработки перед последовательной имеет место в идеальном конвейере, в котором отсутствуют конфликты и все команды выполняются друг за другом в установившемся режиме, то есть без перезагрузки конвейера. Наличие конфликтов в конвейере и его перезагрузки снижают реальную производительность конвейера по сравнению с идеальным случаем.

## 10. Типы конфликтов в конвейере и методы уменьшения их влияния на снижение производительности микропроцессора.

Значительное преимущество конвейерной обработки перед последовательной имеет место в идеальном конвейере, в котором отсутствуют конфликты и все команды выполняются друг за другом без перезагрузки конвейера. Наличие конфликтов снижает реальную

производительность конвейера по сравнению с идеальным случаем.

Конфликты - это такие ситуации в конвейерной обработке, которые препятствуют выполнению очередной команды в предназначенном для нее такте.

Конфликты делятся на три группы:

• структурные,

• по управлению,

• по данным.

Структурные конфликты возникают в том случае, когда аппаратные средства процессора не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.

Причины структурных конфликтов.

1. Не полностью конвейерная структура процессора, при которой некоторые ступени отдельных команд выполняются более одного такта.

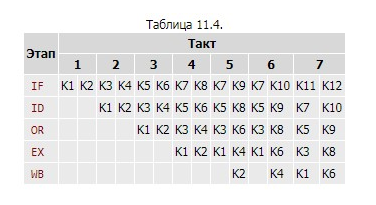
Пусть этап выполнения команды i+1 занимает 3 такта. Тогда диаграмма работы конвейера будет иметь вид, представленный в табл. 11.3.



Одним из типичных примеров служит конфликт из-за доступа к запоминающим устройствам. Из [табл. 11.1](https://www.intuit.ru/studies/courses/60/60/lecture/1786?page=1#table.11.1) видно, что в случае, когда операнды и команды находятся в одном запоминающем устройстве, начиная с такта 3, работу конвейера придется постоянно приостанавливать, поскольку различные команды в одном и том же такте обращаются к памяти на считывание команды, выборку операнда, запись результата.

Борьба с конфликтами такого рода проводится путем увеличения количества однотипных функциональных устройств, которые могут одновременно выполнять одни и те же или схожие функции. Например, в современных микропроцессорах обычно разделяют кэш-память для хранения команд и кэшпамять данных, а также используют многопортовую схему доступа к регистровой памяти, при которой к регистрам можно одновременно обращаться по одному каналу для записи, а по другому - для считывания информации. Конфликты из-за исполнительных устройств обычно сглаживаются введением в состав микропроцессора дополнительных блоков. Так, в микропроцессоре Pentium-4 предусмотрено 4 АЛУ для обработки целочисленных данных. Процессоры, имеющие в своем составе более одного конвейера, называются суперскалярными.

Недостатком суперскалярных микропроцессоров является необходимость синхронного продвижения команд в каждом из конвейеров. В [табл. 11.4](https://www.intuit.ru/studies/courses/60/60/lecture/1786?page=2#table.11.4) представлена последовательность выполнения команд в микропроцессоре, имеющем два конвейера, при условии, что команде К1 требуется 3 такта на этапе EX.



При этом команды будут завершаться в последовательности К2-К4-К1-К6-...

Следовательно, для обеспечения правильной работы суперскалярного микропроцессора при возникновении затора в одном из конвейеров должны приостанавливать свою работу и другие. В противном случае может нарушиться исходный порядок завершения команд программы. Но такие приостановки существенно снижают быстродействие процессора. Разрешение этой ситуации состоит в том, чтобы дать возможность выполняться командам в одном конвейере вне зависимости от ситуации в других конвейерах. Это приводит к неупорядоченному выполнению команд. При этом команды, стоящие в программе позже, могут завершиться ранее команд, стоящих впереди. Аппаратные средства микропроцессора должны гарантировать, что результаты выполненных команд будут записаны в приемник в том порядке, в котором команды записаны в программе. Для этого в микропроцессоре результаты этапа выполнения команды обычно сохраняются в специальном буфере восстановления последовательности команд. Запись результата очередной команды из этого буфера в приемник результата проводится лишь после того, как выполнены все предшествующие команды и записаны их результаты.

**Конфликты по управлению** возникают при конвейеризации команд переходов и других команд, изменяющих значение счетчика команд.

Суть конфликтов этой группы наиболее удобно проиллюстрировать на примере команд условного перехода. Пусть в программе, представленной в [табл. 11.1,](https://www.intuit.ru/studies/courses/60/60/lecture/1786?page=1#table.11.1) команда i+1 является командой условного перехода, формирующей адрес следующей команды в зависимости от результата выполнения команды i. Команда i завершит свое выполнение в такте 5. В то же время команда условного перехода уже в такте 3 должна прочитать необходимые ей признаки, чтобы правильно сформировать адрес следующей команды. Если конвейер имеет большую глубину (например, 20 ступеней), то промежуток времени между формированием признака результата и тактом, где он анализируется, может быть еще большим. В инженерных задачах примерно каждая шестая команда является командой условного перехода, поэтому приостановки конвейера при выполнении команд переходов до определения истинного направления перехода существенно скажутся на производительности процессора.

Наиболее эффективным методом снижения потерь от конфликтов по управлению служит предсказание переходов. Суть данного метода заключается в том, что при выполнении команды условного перехода специальный блок микропроцессора определяет наиболее вероятное направление перехода, не дожидаясь формирования признаков, на основании анализа которых этот переход реализуется. Процессор начинает выбирать из памяти и выполнять команды по предсказанной ветви программы (так называемое исполнение по предположению, или

"спекулятивное" исполнение). Однако так как направление перехода может быть предсказано неверно, то получаемые результаты с целью обеспечения возможности их аннулирования не записываются в память или регистры (то есть для них не выполняется этап WB ), а накапливаются в специальном буфере результатов.

Если после формирования анализируемых признаков оказалось, что направление перехода выбрано верно, все полученные результаты переписываются из буфера по месту назначения, а выполнение программы продолжается в обычном порядке. Если направление перехода предсказано неверно, то буфер результатов очищается. Также очищается и конвейер, содержащий команды, находящиеся на разных этапах обработки, следующие за командой условного перехода. При этом аннулируются результаты всех уже выполненных этапов этих команд. Конвейер начинает загружаться с первой команды другой ветви программы. Так как конвейерная обработка эффективна при большом числе последовательно выполненных команд, то перезагрузка конвейера приводит к значительным потерям производительности. Поэтому вопросам эффективного предсказания направления ветвления разработчики всех микропроцессоров уделяют большое внимание.

Методы предсказания переходов делятся на статические и динамические. При использовании статических методов до выполнения программы для каждой команды условного перехода указывается направление наиболее вероятного ветвления. Это указание делается или программистом с помощью специальных средств, имеющихся в некоторых языках программирования, по опыту выполнения аналогичных программ либо результатам тестового выполнения программы, или программойкомпилятором по заложенным в ней алгоритмам.

Методы динамического прогнозирования учитывают направления переходов, реализовывавшиеся этой командой при выполнении программы. Например, подсчитывается количество переходов, выполненных ранее по тому или иному направлению, и на основании этого определяется направление перехода при следующем выполнении данной команды.

В современных микропроцессорах вероятность правильного предсказания направления переходов достигает 90-95 %.

**Конфликты по данным** возникают в случаях, когда выполнение одной команды зависит от результата выполнения предыдущей команды.

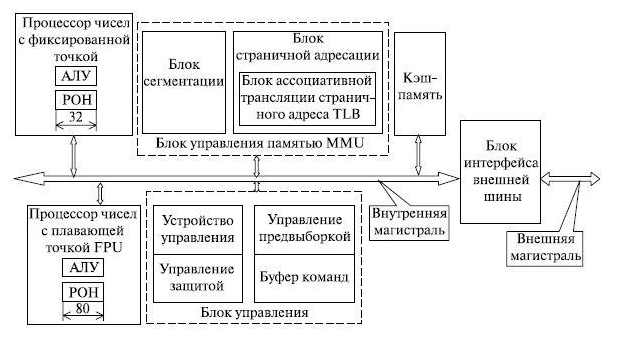
При обсуждении этих конфликтов будем предполагать, что команда i предшествует команде j.

## 11. Структура 32-разрядного микропроцессора

Рассмотрение архитектуры IA-32 начнем с микропроцессора i486. В нем впервые появились те блоки, которых не было на кристалле первого 32-разрядного микропроцессора i386, - кэш-память и процессор обработки чисел с плавающей точкой. Именно его архитектуру можно рассматривать как базовую для IA-32. Структура микропроцессора i486 представлена на [рис. 1.3.](https://www.intuit.ru/studies/courses/604/460/lecture/10321?page=3#image.1.3)

Рассмотрим состав и назначение основных блоков этого микропроцессора.

**Процессор обработки чисел с фиксированной точкой** содержит 32-разрядное АЛУ и блок **регистров общего назначения**. АЛУпредназначено для обработки двоичных чисел длиной 1, 2 или 4 байта без знака или со знаком, а также двоично-десятичных чисел, не превышающих 99. Двоичные числа со знаком представляются в дополнительном коде. Блок регистров общего назначения содержит восемь 32-разрядных регистров, часть из которых допускает 16- и 8-разрядное обращение.



**Рис. 1.3.** Структура универсального микропроцессора

**Процессор обработки чисел с плавающей точкой** состоит из 80-разрядного АЛУ, блока из восьми 80разрядных регистров общего назначения, а также управляющих регистров. Главным образом он предназначен для обработки чисел с плавающей точкой, но также используется для обработки целых чисел со знаком длиной 8 байт и двоично-десятичных чисел величиной от 100 до 99…9 (18 цифр). На первых этапах развития SIMD-обработки регистры **FPU** использовались для хранения операндов, представленных в новых форматах.

**Блок управления памятью** (Memory Management Unit - MMU) состоит из двух основных блоков в соответствии с организацией памяти.

В общем случае память в микропроцессоре делится на сегменты, которые, в свою очередь, делятся на страницы. В соответствии с этим, MMU содержит блок сегментации (или блок сегментного преобразования адреса) и блок страничного преобразования, в состав которого входит так называемый **буфер ассоциативной трансляции адресов стра- ниц** (TLB).

**Кэш-память** представляет собой промежуточную ступень между оперативной памятью и регистрами микропроцессора и предназначена для хранения наиболее часто используемой информации.

В состав **блока управления** входят:

• собственно устройство управления, то есть та классическая схема, которая под действием кода команды вырабатывает набор управляющих сигналов, поступающих на разные узлы как самого микропроцессора, так и на блок интерфейса внешней шины;

• управление защитой памяти: обеспечивает аппаратную защиту программ и данных при управлении памятью и по привилегиям;

• блок управления предвыборкой команд: реализует опережающее заполнение буфера команд, представляющего собой некоторую буферную память. Буфер команд имеет емкость 32 байта и заполняется командами из следующих ячеек памяти команд по мере своего освобождения. Этим обеспечивается ускорение обработки микропроцессором следующей команды. Данный блок подвергался, пожалуй, наиболее существенным переработкам по мере развития архитектуры IA-32 - причина в широком последующем использовании конвейерной организации работы МП и связанной с этим необходимости постоянного совершенствования блока предсказания адреса следующей команды.

**Блок интерфейса внешней шины** осуществляет электрическое согласование параметров внутренней магистрали с сигналами внешних магистралей, формирование необходимых сигналов на внешнюю магистраль и прием сигналов извне. Внешняя магистральмикропроцессора состоит из шины

адреса, шины данных и сигналов управления:

• шина данных имеет ширину 32 разряда;

• 32-разрядный адрес передается по 34-разрядной шине А31...А2+(B3,B2,B1,B0). Чтобы с минимальными потерями согласовывать 32-разрядную шину данных с передачей данных меньшей разрядности, младшие разряды адреса (А1 и А0) передаются в дешифрированном виде (B3, B2, B1, B0). Они показывают, какие байты из 32-разрядной шины данных в данный момент реально востребованы: 1 байт, 2 младших байта, 2 старших байта либо все 32 разряда данных;

• шина управления - 32-разрядная. По ней передаются сигналы записи и чтения содержимого оперативной памяти и внешних устройств, сигналы запросов прерываний, прямого доступа к памяти и т. д.

Особый интерес представляют три режима работы микропроцессора: реальный, защищенный и режим виртуального МП i8086. В **реальном режиме** обеспечивается совместимость на уровне объектных кодов с микропроцессором i8086 и микропроцессором i286, работающем в реальном режиме. В этом

режиме архитектура 32-разрядного микропроцессора почти полностью идентична архитектуре 16-разрядного МП. Для программиста же он вообще представляется как МП i8086, выполняющий написанные программы с большей скоростью и обладающий расширенной системой команд и регистрами. Благодаря этим качествам фирма Intel сохранила прежних клиентов, которые хотели модернизировать свои системы, не отказываясь от имевшегося задела в области программного обеспечения, и привлекла тех, кому изначально требовалась высокая скорость обработки информации.

Одно из основных ограничений реального режима было связано с предельной емкостью адресуемой памяти, равной 1 Мбайт. От него свободен **защищенный режим**, позволяющий воспользоваться всеми преимуществами архитектуры нового МП. Размер адресного пространства в этом случае увеличивается до 4 Гбайт, а общий объем поддерживаемого адресного пространства - до 64 терабайт (1 Тбайт = 240 байт). МП, работающие в защищенном режиме, обладают более высоким быстродействием и возможностями организации истинной многозадачности.

Наконец, **режим виртуального МП** открывает возможность одновременного исполнения программ, написанных для МП i8086, i286 и i386.

Поскольку емкость памяти, адресуемой микропроцессором, не ограничена значением 1 Мбайт, этот режим позволяет формировать несколько виртуальных сред i8086.

# История эвм

## 1. Основные этапы развития вычислительной техники до 40-х г.г. ХХ века. Машина Бэббиджа. Электромеханическая счетная машина Холерита. Компьютеры инженера Цузе. Вычислительная машина Mark 1.

Идея использования программного управления для построения устройства, автоматически выполняющего арифметические вычисления, была впервые высказана английским математиком Ч.Бэббиджем еще в 1833г. Однако его попытки построить механическое вычислительное устройство с программным управлением не увенчались успехом.

Холерит в 1886г сконструировал счетную машину,в 1890 на ней обрабатывалась перепись населения.(США)

В [1938 году](https://ru.wikipedia.org/wiki/1938_%D0%B3%D0%BE%D0%B4) появилась первая действующая разработка Цузе [Z1](https://ru.wikipedia.org/wiki/Z1_(%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0)). Это был двоичный механический вычислитель с электрическим приводом и ограниченной возможностью [программирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) при помощи [клавиатуры](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D0%B2%D0%B8%D0%B0%D1%82%D1%83%D1%80%D0%B0). Результат вычислений в десятичной системе отображался на ламповой панели. В отличие от Z1, новая машина Z2 считывала [инструкции](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) перфорированной 35-миллиметровой киноплёнки. Она тоже была демонстрационной моделью и не использовалась для практических целей, в [1941 году](https://ru.wikipedia.org/wiki/1941_%D0%B3%D0%BE%D0%B4) Цузе создал уже более совершенную модель — [Z3](https://ru.wikipedia.org/wiki/Z3), которую сегодня многие считают первым реально действовавшим программируемым компьютером. Впрочем, программируемость этого двоичного вычислителя, собранного, как и предыдущая модель, на основе телефонных реле, также была ограниченнойТем не менее, Z3 первым среди вычислительных машин Цузе получил практическое применение и использовался для расчётов параметров [стреловидных крыльев](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B5%D0%BB%D0%BE%D0%B2%D0%B8%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%BA%D1%80%D1%8B%D0%BB%D0%B0) самолёта и расчётов для [управляемых ракет](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D0%B5%D0%BC%D0%B0%D1%8F_%D1%80%D0%B0%D0%BA%D0%B5%D1%82%D0%B0) немецким [Исследовательским институтом аэродинамики](https://ru.wikipedia.org/w/index.php?title=%D0%98%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82_%D0%B0%D1%8D%D1%80%D0%BE%D0%B4%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D0%BA%D0%B8&action=edit&redlink=1).

Первой работающей универсальной автоматически управляемой ВМ считается расчетно-механическая машина "Марк - 1" ( США, 1944г. ). Простои машины составляли большую часть времени. Столь же низкая производительность оказалась и у машины "Марк - 2", построенной на реле улучшенной конструкции.

Электронные лампы стали элементной базой ВМ первого поколения. Основная схема – симметричный триггер был создан в 1918г. советским ученым Бонч-Бруевичем М.А. В 1919г. аналогичная схема была разработана также американскими учеными Икклзом и Джорданом.

## 2.История создания, и характеристики первой полностью электронной вычислительной машины ENIAC. ЭВМ UNIVAC.

Эвм 1-ого поколения. Эниак (eniac)

Начиная с 1943 года группа специалистов под руководством Говарда Эйкена, Дж. Моучли и П. Эккерта в США начала конструировать вычислительную машину на основе электронных ламп, а не на электромагнитных реле. Эта машина была названа ENIAC (Electronic Numeral Integrator And Computer) и работала она в тысячу раз быстрее, чем «Марк-1». ENIAC содержал 18 тысяч вакуумных ламп, занимал площадь 915 метров, весил 30 тонн и потреблял мощность 150 киловатт. ENIAC имел и существенный недостаток – управление им осуществлялось с помощью коммутационной панели, у него отсутствовала память, и для того чтобы задать программу приходилось в течение нескольких часов или даже дней подсоединять нужным образом провода. Худшим из всех недостатков была ужасающая ненадежность компьютера, так как за день работы успевало выйти из строя около десятка вакуумных ламп.

## 3.Начало развития вычислительной техники в СССР

Первые проекты отечественных ЭВМ были предложены С.А. Лебедевым, Б.И. Рамеевым в 1948г. В 1949-51гг. по проекту С.А. Лебедева была построена МЭСМ ( малая электронно-счетная машина ). К ЭВМ 1-го поколения относится и БЭСМ-1 (большая электронно-счетная машина ), разработка которой под руководством С.А. Лебедева была закончена в 1952г., она содержала 5 тыс. ламп, работала без сбоев в течение 10 часов. Быстродействие достигало 10 тыс. операций в секунду. Почти одновременно проектировалась ЭВМ "Стрела" под руководством Ю.Я. Базилевского, в 1953г. она была запущена в производство. Позже появилась ЭВМ "Урал - 1", положившая начало большой серии машин "Урал", разработанных и внедренных в производство под руководством Б.И. Рамеева. В 1958г. запущена в серийное производство ЭВМ первого поколения М – 20 ( быстродействие до 20 тыс. операций/с ).

## 4.ЭВМ МИФИ

1967-1962.В машине "МИФИ" использовалась 16-ричная двоично-кодированная система представления чисел с плавающей десятичной точкой. Это представление значительно сокращало время выполнения операций выравнивания порядков и нормализации мантисс при выполнении арифметических операций.

Разрядная сетка числа состояла из 42-х разрядов: один разряд - знак порядка, три разряда - код порядка, один разряд -знак числа, остальные 37 разрядов - мантисса числа. Для представления (хранения) отрицательных порядков принят дополнительный код, а положительных порядков и мантисс независимо от знака - прямой. Последнее было сделано для упрощения операций умножения и деления.

Арифметическое устройство (АУ) машины по принципу выполнения операций являлось последовательно-параллельным. Прием исходных данных и выдача результата производились последовательно, выполнение самой операции - параллельно. Этот выбор определился тем, что первым вариантом оперативной памяти являлся магнитный барабан. АУ включало три регистра и сумматор.

Система команд содержала 66 команд. Использовалось два типа адресации: трехадресная адресация с возможностью модификации и одноадресная. Одноадресная система позволяла работать в режиме с накапливающим сумматором а АУ, а также выполнять команды в групповом режиме (повторять команды определенное количество раз).

Разрядная сетка команды также содержала 42 разряда. Среди них: 3 разряда признаков (для автоматического изменения адреса с помощью модификатора), 6 разрядов кода операции, по 11 разрядов на адрес в трехадресной команде или по 13 разрядов для адреса в одноадресной команде. В последнем случае в одном слове размещались 2 одноадресных команды.

Арифметические и логические операции, выполнявшиеся в АУ (в одноадресных и в трехадресных командах):

|  |
| --- |
| сложение,  вычитание,  вычитание модулей,  умножение,  деление,  логическое сложение,  логическое умножение,  сравнение,  сложение по всей разрядной сетке,  вычитание по всей разрядной сетке,  присвоение знака числа по данному,  выделение целой части,  сложение порядков,  вычитание порядков,  логический сдвиг. |