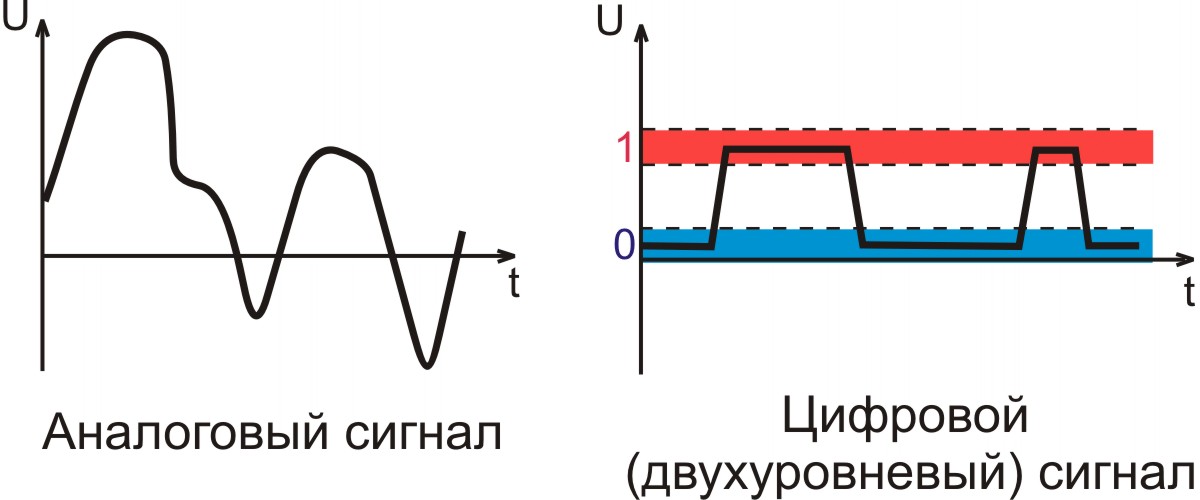
**Вопрос 1. *Базовые определения. Аналоговый и цифровой сигналы. Электронное устройство. Искажение сигналов.***

Сигнал — любая физическая величина (температура, давление воздуха, интенсивность света, сила тока и т.д.), изменяющаяся со временем.

Электрический сигнал — электрическая величина (например, напряжение, ток, мощность), изменяющаяся со временем.

Аналоговый сигнал — может принимать любые значения в определенных пределах. Устройства, работающие с аналоговыми сигналами, — аналоговые устройства. Аналоговый сигнал изменяется аналогично физической величине, т. е. непрерывно.

Цифровой сигнал — может принимать только два значения. Причём разрешены некоторые отклонения от этих значений. Устройства, работающие с цифровыми сигналами, — цифровые устройства.



**ЭЛЕКТРОННОЕ УСТРОЙСТВО**



[Электронные](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0) устройства — электронные [приборы](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%B1%D0%BE%D1%80) и [устройства](https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE), принцип действия которых основан на взаимодействии [заряженных частиц](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D1%80%D1%8F%D0%B6%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%87%D0%B0%D1%81%D1%82%D0%B8%D1%86%D0%B0) с [электромагнитными полями](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BC%D0%B0%D0%B3%D0%BD%D0%B8%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5) и используется для преобразования [электромагнитной энергии](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BC%D0%B0%D0%B3%D0%BD%D0%B8%D1%82%D0%BD%D0%B0%D1%8F_%D1%8D%D0%BD%D0%B5%D1%80%D0%B3%D0%B8%D1%8F) (например для передачи, обработки и хранения [информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F)).

Электронные устройства делятся на два класса: аналоговые и цифровые. Аналоговые устройства работают с непрерывно изменяющимися сигналами, а цифровые устройства – с сигналами в цифровой форме, т.е. в форме дискретных импульсов, по сути, с информацией, представленной посредством двоичного кода.

Для аналоговых устройств характерно непрерывное изменение сигнала в соответствии с физическим процессом, который он описывает. По сути, такой сигнал является непрерывной функцией с неограниченным числом значений в различные моменты времени.

Цифровые электронные устройства работают с дискретными сигналами. Как правило, такой цифровой сигнал состоит из последовательности импульсов, значений в которой всего два – «Ложь» или «Истина» (0 или 1). В целом цифровые устройства могут быть реализованы на различных элементных базах: на [электромагнитных реле](http://electricalschool.info/naladka/193-jelektromagnitnye-rele-upravlenija.html), на транзисторах, на оптоэлектронных элементах, или на микросхемах.

Главным образом, современные цифровые схемы строятся из [логических элементов](http://electricalschool.info/electronica/1152-logicheskie-ustrojjstva.html), и могут быть связанны между собой посредством триггеров и счетчиков. Они нашли широкое применение в системах автоматизации и робототехнике, измерительных приборах, а также в системах радио и телекоммуникации.

Цифровой сигнал устойчив к помехам, его легко обрабатывать и записывать, а также передавать без искажений, что и дает электронным устройствам на этой основе неоспоримое преимущество перед аналоговыми устройствами.

К цифровым устройствам относятся: триггер, логический элемент, счетчик, компаратор, генератор тактовых импульсов, дешифратор, шифратор, мультиплексор, демультиплексор, сумматор, полусумматор, регистр, арифметическо-логическое устройство, микропроцессор, микрокомпьютер, микроконтроллер, память и т.д.

**ИСКАЖЕНИЯ СИГНАЛОВ**

Искажения сигнала — изменение [сигнала](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB_(%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)), вызванное несовпадением [идеальных](https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5) и реальных характеристик системы его [обработки](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D0%BE%D0%B2) и [передачи](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8).

**ПРИЧИНЫ ИСКАЖЕНИЯ СИГНАЛОВ:**

1.Несовершенство характеристик элементов аппаратуры;

2.Шумы (слабые хаотические сигналы, вырабатываемые любым электронным компонентом);

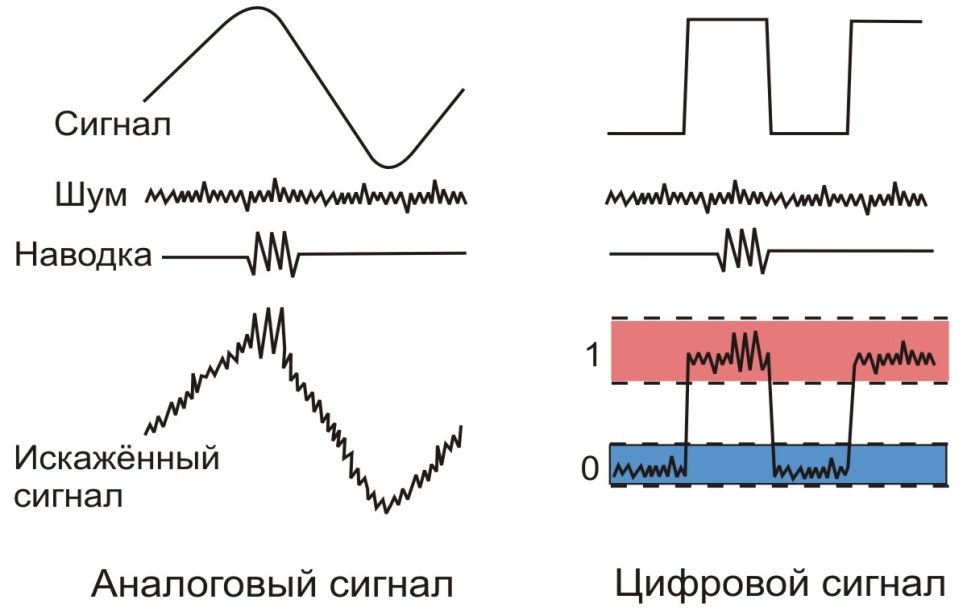
3.Наводки, помехи (сигналы, вызываемые внешними электромагнитными полями — радиопередача, трансформаторы, взаимовлияние цепей и т.д.);

4.Старение элементов — изменение характеристик со временем;

5.Внешние физические воздействия: температура, влажность, давление, вибрация и т.д.

6.Паразитные эффекты (утечки, ёмкости, индуктивности, сопротивления).

Искажения сигналов шумами и наводками



**Вопрос 2. Преимущества и недостатки цифровых сигналов. Виды цифровых сигналов.**

**ПРЕИМУЩЕСТВА ЦИФРОВЫХ СИГНАЛОВ:**

1.Более сложная и многоступенчатая обработка, чем в случае аналоговых сигналов;

**2.**Длительное хранение без потерь с возможностью многократного копирования без искажений;

**3.**Качественная передача на большие расстояния без искажений;

**4.**Цифровые устройства проще отлаживать, они меньше подвержены старению;

**5.**Поведение цифровых устройств всегда можно точно рассчитать и предсказать;

**6.**Цифровые устройства проще проектировать, отлаживать, тестировать.

**НЕДОСТАТКИ ЦИФРОВЫХ СИГНАЛОВ:**

**1.**Принципиально меньшее предельное быстродействие цифровых устройств по сравнению с аналоговыми;

**2.**Информационная ёмкость цифрового сигнала гораздо меньше, чем аналогового, поэтому для замены одного аналогового сигнала требуется несколько цифровых сигналов (от 4 до16) — код;

**3.**Для связи с реальным миром требуются преобразователи аналоговых сигналов в цифровые (на входе, АЦП) и цифровых сигналов в аналоговые (на выходе, ЦАП);

**4.**При простом алгоритме обработки цифровые устройства гораздо сложнее аналоговых.

**ВИДЫ ЦИФРОВЫХ СИГНАЛОВ**

* **Одиночные цифровые сигналы:**

- Разрешающие/запрещающие сигналы;

- Сигнализирующие сигналы (флаги);

- Синхронизирующие сигналы (определяющие момент времени выполнения операции).

* **Сгруппированные (шинные) цифровые сигналы (коды):**

- Коды выборок аналоговых сигналов;

- Коды адресации устройств (выбора нужного устройства);

- Коды команд (инструкций);

- Коды данных.

**Вопрос 3. Понятие двоичной логики. Типы цифровых устройств. Типы организации связей.**

**Код** — двоичное число, а также метод представления двоичных чисел;

**Разрядность кода** - количество двоичных разрядов кода (210 = 1 024, 220 = 1 048 576; 230 = 1 073 741 824);

**Бит** — один разряд двоичного числа (от англ. binary digit);

**Байт** — восемь двоичных разрядов (битов) — принимает 28 значений: от 0 до 255;

**Тетрада** (полубайт, ниббл) — четыре двоичных разряда, половина байта — принимает 24 значений: от 0 до 15;

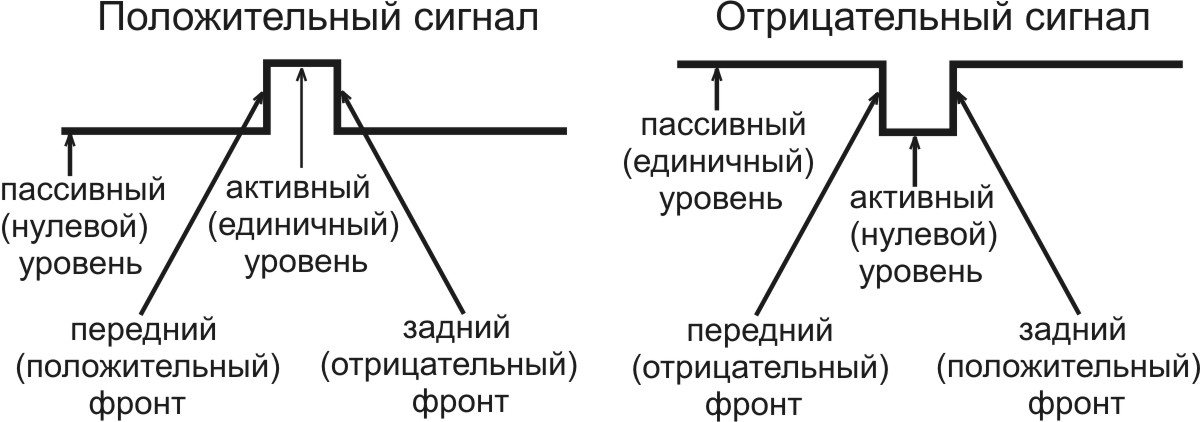
**Слово** — код, состоящий из нескольких байтов (чаще всего 2 байта — 16 разрядов, 4 байта — 32 разряда, 8 байт — 64 разряда);

**ТИПЫ ЛОГИКИ:**

**Положительная логика** — логической единице соответствует высокий уровень напряжения, логическому нулю — низкий уровень напряжения;

**Отрицательная логика** — логической единице соответствует низкий уровень напряжения, логическому нулю — высокий уровень напряжения.

Типы логики относятся к шинам (кодам). Одиночные сигналы (импульсы) называются положительными (единичными) или отрицательными (нулевыми).

**Элементы цифрового сигнала**

**ТИПЫ ЦИФРОВЫХ УСТРОЙСТВ**

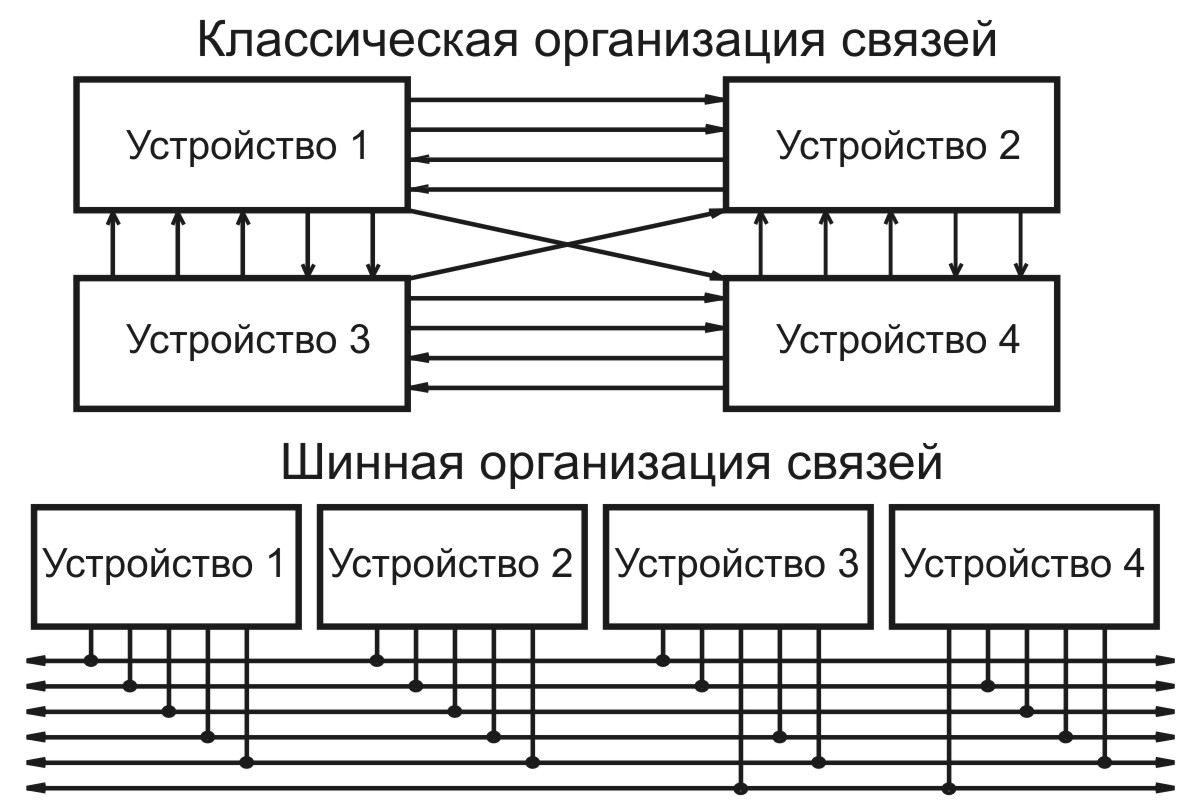
**Устройства с жёсткой логикой работы** (выходные сигналы в каждый момент жёстко определяются входными сигналами и это соответствие не может быть изменено);

**Устройства с программируемым алгоритмом работы** (соответствие выходных сигналов входным сигналам может быть изменено программой — набором управляющих кодов).

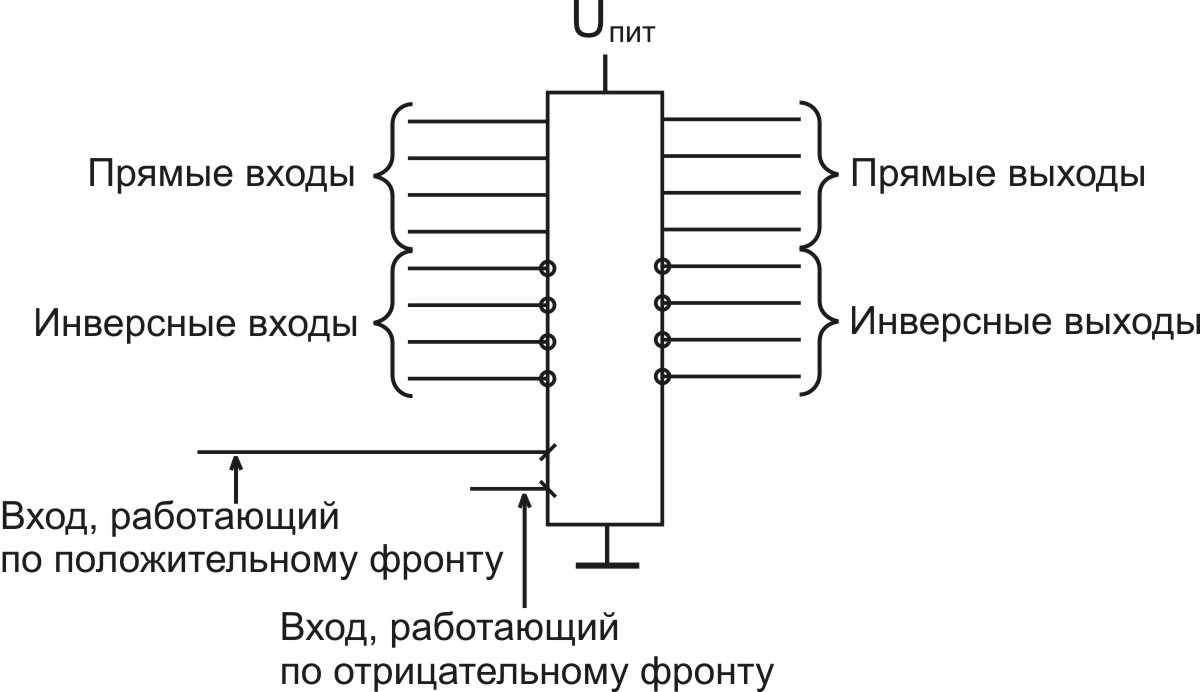
Устройства с жёсткой логикой быстрее, проще для простых функций, сложнее в разработке.

Устройства с программируемой логикой медленнее, проще для сложных функций, проще в разработке.

**ТИПЫ ОРГАНИЗАЦИИ СВЯЗЕЙ**

****

**Вопрос 4. Цифровые элементы, узлы, микросхемы. Инвертор (элемент НЕ).**

****

**Цифровым узлом** называется функциональная часть цифрового устройства, состоящая из цифровых элементов и выполняющая операции над n-разрядными двоичными кодами.

Цифровые микросхемы представляют собой устройства, которые преобразовывают и обрабатывают изменяющиеся по закону дискретной сигналы. Они являются основой в построении вычислительных цифровых устройств, цифровых механизмов измерительного оборудования, приборов автоматического управления и т. д.

В зависимости от функционального назначения цифровые микросхемы делят на подгруппы:

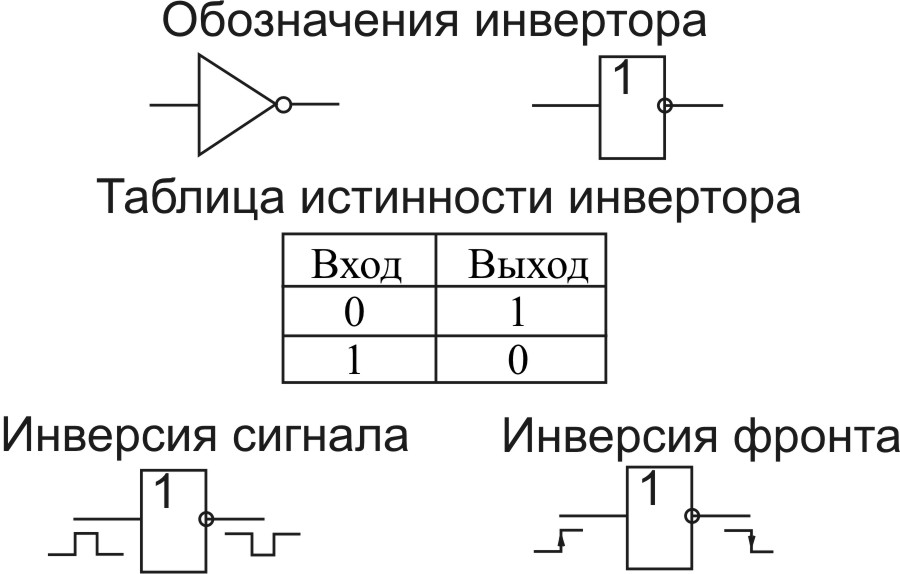
• Триггеры;

• Цифровые логические микросхемы;

• Механизмы дискретных и арифметических элементов и др.

Цифровые микросхемы предназначены для преобразования и обработки сигналов, изменяющихся по закону дискретной, например двоичной, функции. Они применяются для построения цифровых вычислительных машин, а также цифровых узлов измерительных приборов, аппаратуры автоматического управления, связи и т. д.

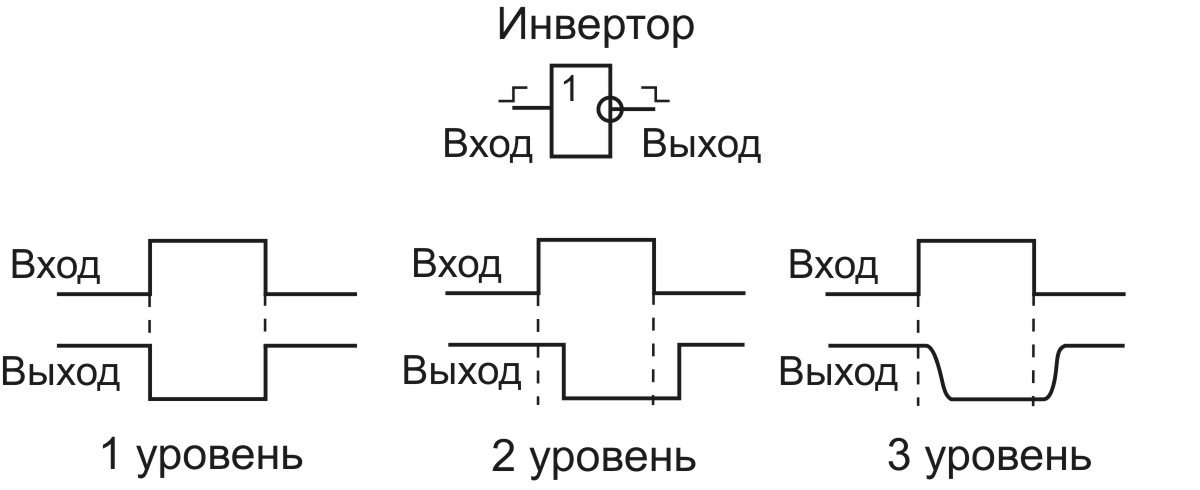
Простейшим логическим элементом является **ИНВЕРТОР**, который просто изменяет значение входного сигнала на прямо противоположное значение.



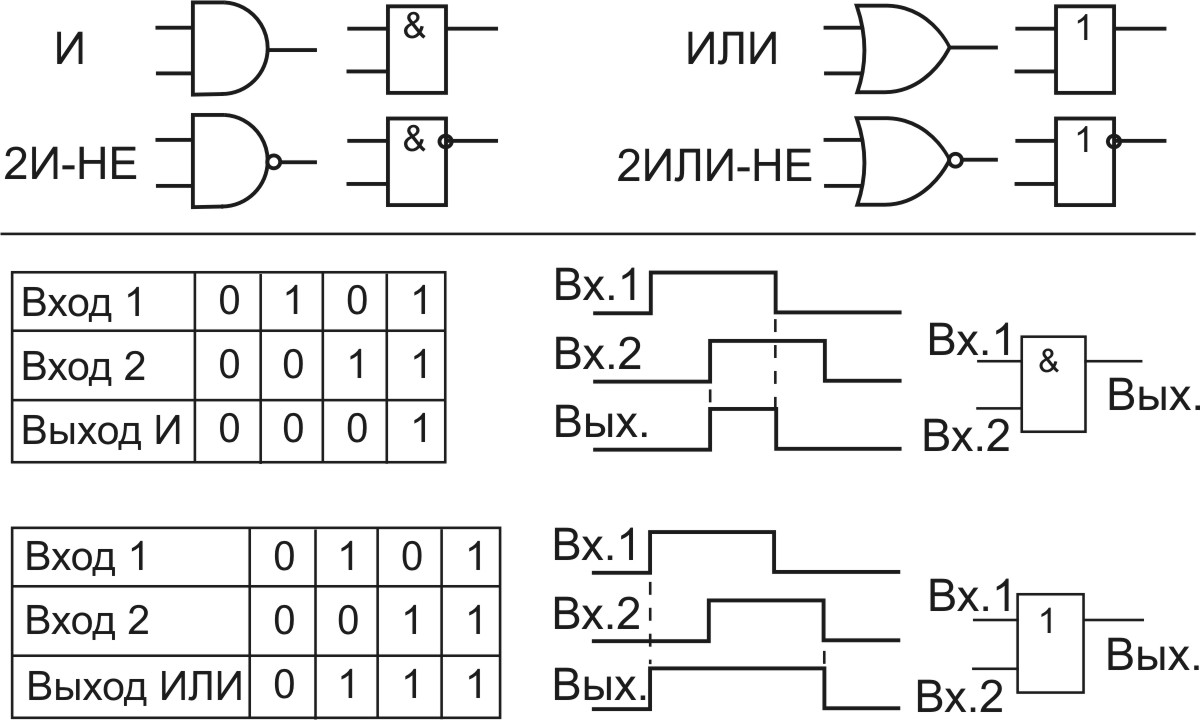
**Вопрос 5. Три модели цифровых устройств. Элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ.**

1. Логическая модель.
2. Модель с временными задержками.
3. Модель с учетом электрических эффектов (или электрическая модель).

Первая, простейшая модель оказывается достаточной примерно в 20% всех случаев. Она применима для всех цифровых схем, работающих с низкой скоростью, в которых быстродействие не принципиально. Привлечение второй модели, учитывающей задержки срабатывания логических элементов, позволяет охватить около 80% всех возможных схем. Ее применение необходимо для всех быстродействующих устройств и в случае одновременного изменения нескольких входных сигналов. Наконец, добавление третьей модели, учитывающей входные и выходные токи, входные и выходные сопротивления и емкости элементов, позволяет проектировать практически 100% цифровых схем. В первую очередь эту третью модель надо применять при объединении нескольких входов и выходов, при передаче сигналов на большие расстояния и при нетрадиционном включении логических элементов (с переводом их в аналоговый, в линейный режим).

****

В первой, логической, модели считается, что элемент срабатывает мгновенно, любое изменение уровня входного сигнала сразу же, без всякой задержки приводит к изменению уровня выходного сигнала. Во второй модели выходной сигнал изменяется с некоторой задержкой относительно входного. Наконец, в третьей модели выходной сигнал не только задерживается по сравнению с входным, но и его изменение происходит не мгновенно, процесс смены уровней сигнала (или, как говорят, фронт сигнала) имеет конечную дли­тельность. Кроме того, третья модель учитывает изменение уровней логических сигналов.



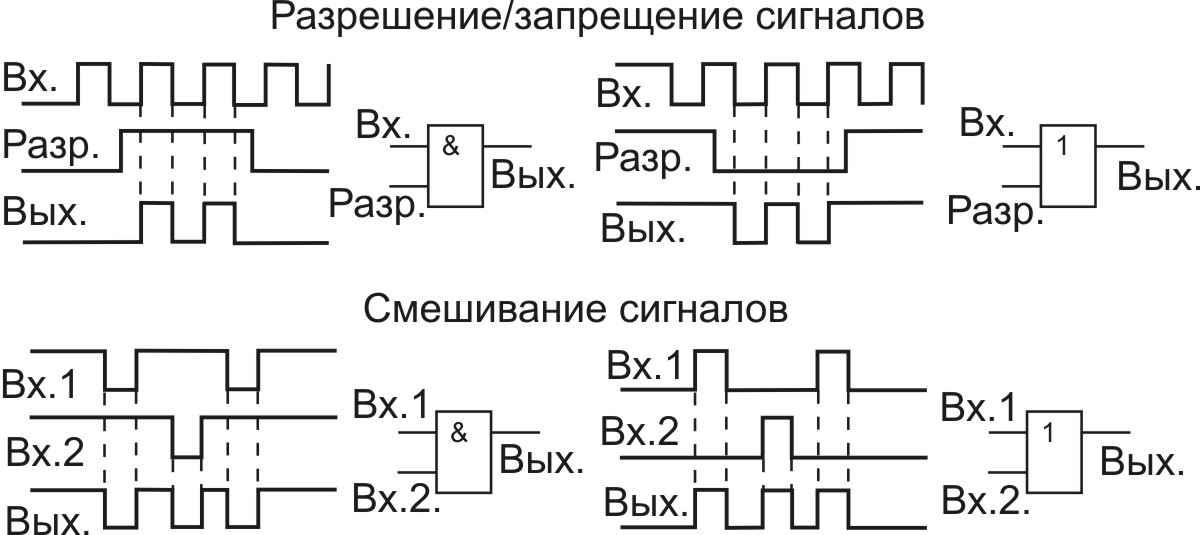
**И**-**НЕ:** элемент имеет ту же функцию что и элемент И, но только сигнал на выходе инвертируется.



**ИЛИ-НЕ:** тот элемент тоже имеет такую функцию, как и элемент ИЛИ.



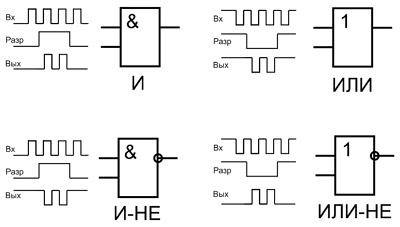
**Вопрос 6. Разрешение/запрещение и смешивание сигналов.**

****

**Схема разрешения/запрещения**

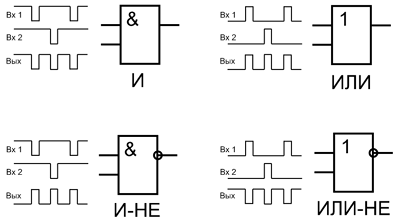
Например, применение логического элемента 2И в качестве управляющего можно описать следующим образом. Один из входов считают управляющим, а второй информационным, тогда при лог. 1 на управляющем входе, сигнал с информационного входа проходит на выход без ограничения, но если на управляющем входе низкий логический уровень, то прохождение сигнала с входа на выход отсутствует. Очень часто логические элементы в таком качестве используют для работы на мультиплексированную или двунаправленную линию.

Точно также в качестве элементов **разрешении/запрещения** используются и другие элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ. Применение того или иного элемента обуславливается уровнем управляющего сигнала, инверсии (или её отсутствия) входного сигнала. Ниже показаны схемы использования логических элементов в качестве разрешающих/запрещающих прохождение сигнала.

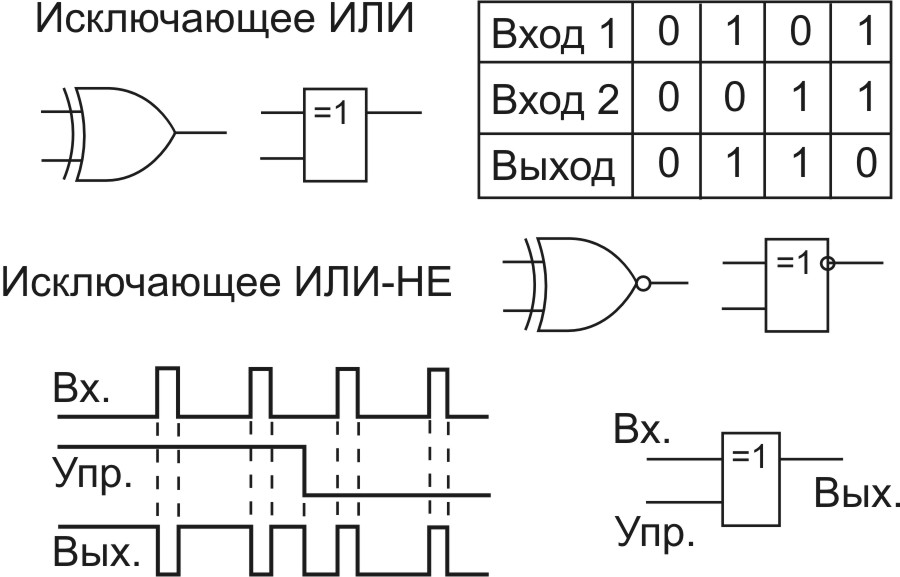


**Схема смешивания сигналов**

Довольно часто требуется реализовать **смешивание сигналов**, когда выходной сигнал должен появляться при приходе сигналов на любой вход логического элемента. Например, использую элемент 2ИЛИ можно реализовать смешивание двух сигналов без инверсии, то есть сигналы, которые приходят на первый и на второй вход, будут отображаться в выходном сигнале. Ниже показаны схемы использования логических элементов в качестве смешивающих с различными уровнями.



**Вопрос 7. Элемент Исключающее ИЛИ. Три типа выходов.**

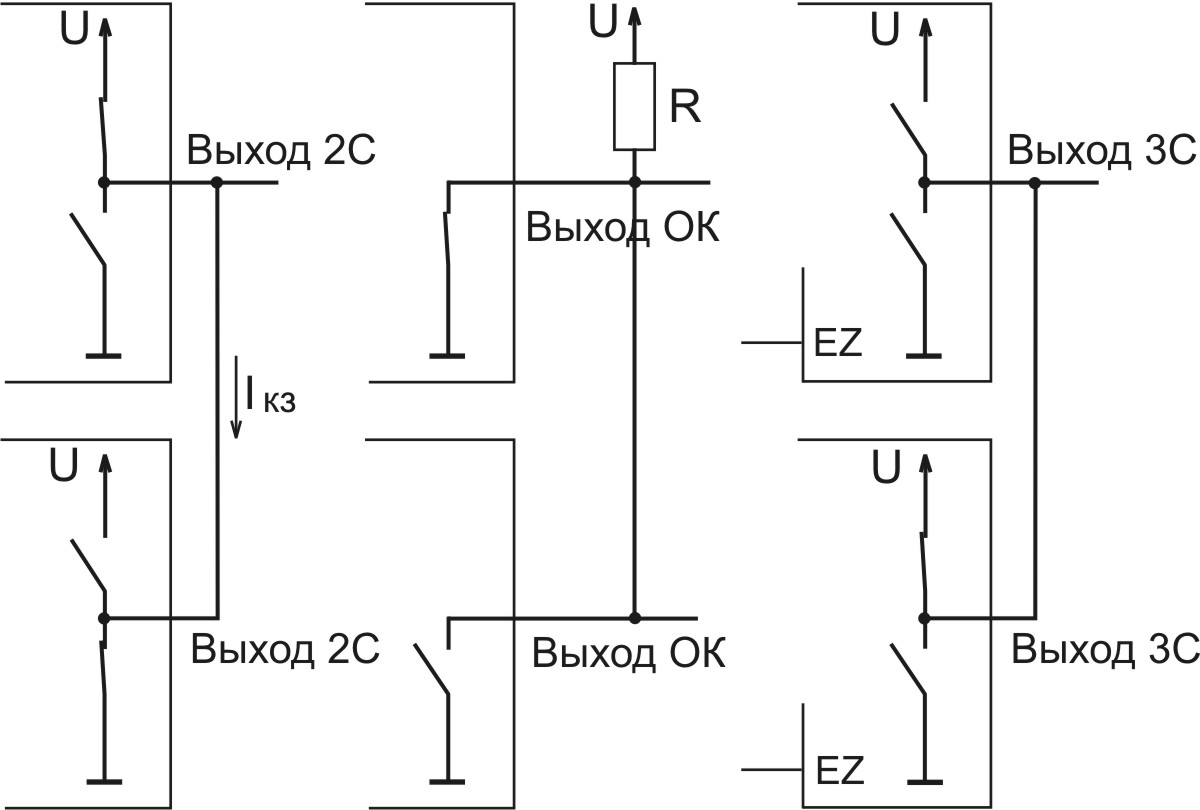
****

На практике наиболее часто используют двухвходовые элементы «исключающее ИЛИ. По простому, суть данного элемента сводится к следующему, сигнал на выходе появляется только в том случае, когда логические уровни на входах не одинаковые.

**ТРИ ТИПА ВЫХОДОВ**

****

**Вопрос 8. Соединение выходов разных типов. Буферы.**

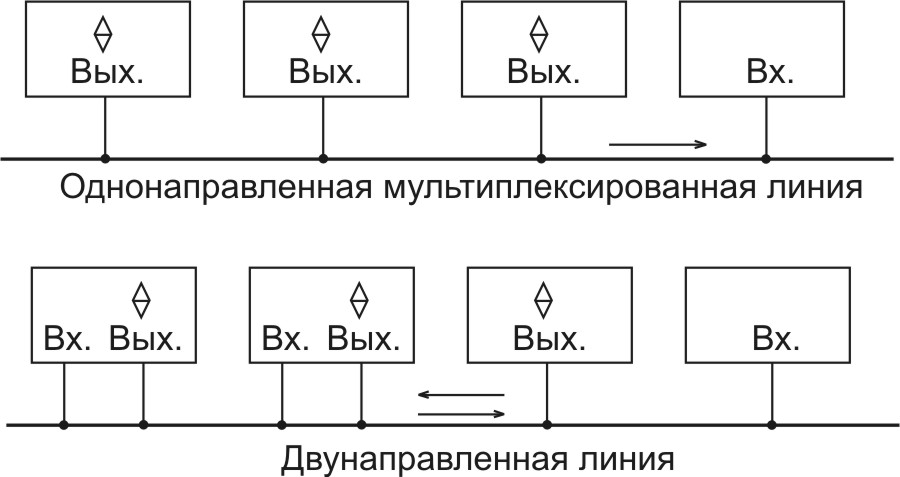
****

**Буфер** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) buffer)— это область [памяти](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C), используемая для временного хранения данных при вводе или выводе. Задача буфера осуществить задержку.



Повторители и буферы отличаются от инверторов прежде всего тем, что они не инвертируют сигнал (правда, существуют и инвертирующие буферы). Во-первых, они выполняют функцию увеличения нагрузочной способности сигнала, то есть позволяют подавать один сигнал на много входов. Для этого имеются буферы с повышенным выходным током и выходом 2С, например, ЛП16 (шесть буферных повторителей). Во-вторых, большинство буферов имеют выход ОК или 3С, что позволяет использовать их для получения двунаправленных линий или для мультиплексирования сигналов.

**ПРИМЕНЕНИЕ БУФЕРОВ**



Под двунаправленными линиями понимаются такие линии (провода), сигналы по которым могут распространяться в двух противоположных направлениях. В отличие от однонаправленных линий, которые идут от одного выхода к одному или нескольким входам, к двунаправленной линии могут одновременно подключаться несколько выходов и несколько входов.

**Вопрос 9. Комбинационные устройства. Дешифраторы. Шифраторы.**

Строятся на основе логических элементов (НЕ, И, ИЛИ, а также буферов);

Состояние выходов однозначно определяется состояниями входов в данный момент (нет внутренней памяти) — статические устройства;

Служат для преобразования кодов, пересылки сигналов, выполнения арифметических операций над кодами и т.д.;

Примеры: дешифраторы, шифраторы, мультиплексоры, преобразователи кодов, компараторы кодов, сумматоры, умножители и т.д.

**ДЕШИФРАТОРЫ**

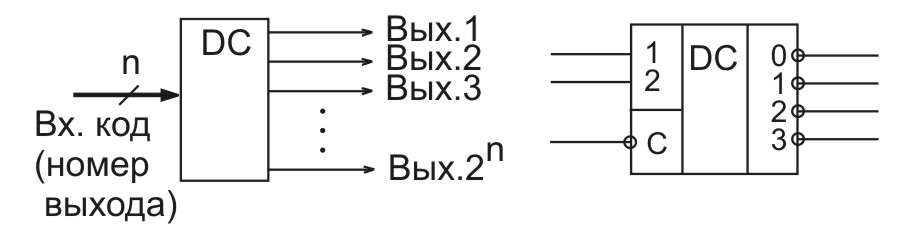
**Дешифратором**называется комбинационный узел, предназначенный для преобразования входного n-разрядного двоичного кода X(2)=xn-1xn-2…x1 х0 в выходной m-разрядный унитарный код *У = ут-1ут-2...yi…y1n y0,* где nчисло входов, а *т* = *2n -* число выходов.

Сущность работы дешифратора сводится к тому, что логическая «1», должна появиться на выходе дешифратора с номером, равному десятичному эквиваленту двоичного кода *Х(2)*, например: в случае дешифратора с 3 входами и 8 выходами (n = 3 и m= 8) при x2x1x0 = 101(2) унитарный код y7y6y5y4y3y2y1y0 примет вид 00100000, т. е. только у5= 1.

**Дешифратор** (декодер) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) decoder) в цифровой электронике — [комбинационная схема](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%85%D0%B5%D0%BC%D0%B0), преобразующая n-разрядный [двоичный](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), [троичный](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) или k‑ичный код в ‑ичный одноединичный код, где — основание [системы счисления](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F).

**Дешифраторы (ДШ)** — это комбинационные схемы с n входами и m = 2n выходами. Единичный сигнал, формирующийся на одном из m выходов, однозначно соответствует комбинации входных сигналов.

Дешифраторы широко используются в ЭВМ для выбора информации по определенному адресу, для расшифровки кода операции и др.



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Входы** | | | **Выходы** | | | |
| **С** | **2** | **1** | **0** | **1** | **2** | **3** |
| **1** | **Х** | **Х** | **1** | **1** | **1** | **1** |
| **0** | **0** | **0** | **0** | **1** | **1** | **1** |
| **0** | **0** | **1** | **1** | **0** | **1** | **1** |
| **0** | **1** | **0** | **1** | **1** | **0** | **1** |
| **0** | **1** | **1** | **1** | **1** | **1** | **0** |

**ШИФРАТОР**

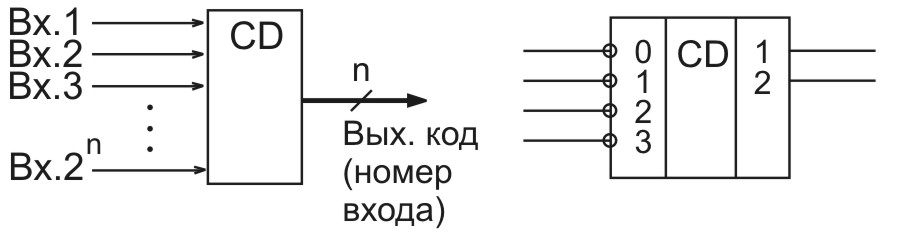
**Шифратором**называется комбинационный узел цифровой техники, предназначенный для преобразования входного унитарного m-разрядного кода Y(2) = ym-1ym-2…y1y0 в выходной n-разрядный двоичный код X(2) = xn-1xn-2…x1x0.

Унитарнымназывают код, в котором только один из *т* разрядов равен логической единице ("1"). Остальные *т-1* разрядов равны логическому нулю ("0"). Каждому разряду унитарного кода присваивается десятичный индекс, возрастающий справа налево от 0 до *т-1.* Например, при *т* = 8 нумерация разрядов примет вид У(2) = y7y6y5y4y3y2y1y0.

[**Шифратор**](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0)) — [логическое устройство](https://ru.wikipedia.org/w/index.php?title=%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE&action=edit&redlink=1), выполняющее преобразование позиционного кода в n-разрядный двоичный код. Таким образом, шифратор — это комбинационное устройство, реализующее обратную [дешифратору](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%88%D0%B8%D1%84%D1%80%D0%B0%D1%82%D0%BE%D1%80) функцию.

**Шифратор** — программа, кодек для кодирования (шифрования информации).

**Шифратор (ШР)** решает задачу, обратную схемам ДШ, т. е. по номеру входного сигнала формирует однозначную комбинацию выходных сигналов. Номер входного сигнала определяется присутствием логической единицы на соответствующем входе (только одном).

****

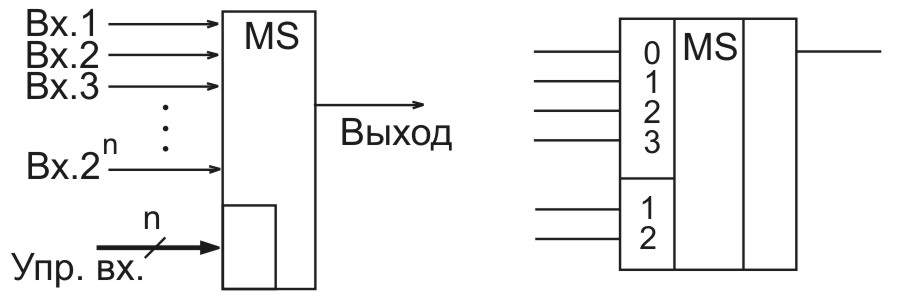
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Входы** | | | | **Выходы** | |
| **0** | **1** | **2** | **3** | **2** | **1** |
| **0** | **1** | **1** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **0** | **1** |
| **1** | **1** | **0** | **1** | **1** | **0** |
| **1** | **1** | **1** | **0** | **1** | **1** |

**Вопрос 10. Мультиплексоры. Компараторы кодов.**

**Мультиплексор** — устройство, имеющее несколько сигнальных входов, один или более управляющих входов и один выход. Мультиплексор позволяет передавать сигнал с одного из входов на выход; при этом выбор желаемого входа осуществляется подачей соответствующей комбинации управляющих сигналов.

**Мультиплексор (МS)** – это функциональный узел, осуществляющий подключение (коммутацию) одного из нескольких входов к выходу у. На выход такого устройства передаѐтся логический уровень того информационного разряда, номер которого в двоичном коде задан на адресных входах х1 и х2.

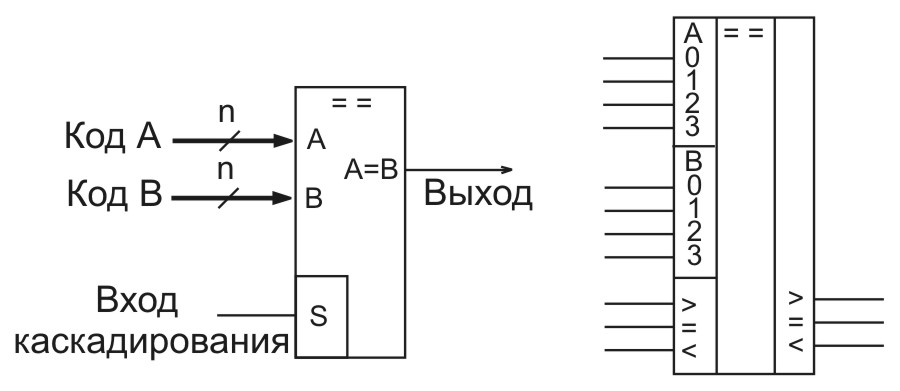
**Мультиплексором** называют узел цифровой техники, позволяющий осуществить подключение одного из входных каналов Di(i=0, 1,..., m-1) к выходному каналу F под воздействием адресующего сигнала в виде n-разрядного двоичного кода хn-1хn-2... x1х0.



|  |  |  |
| --- | --- | --- |
| Входы | | Выход |
| 2 | 1 |
| 0 | 0 | Вход 0 |
| 0 | 1 | Вход 1 |
| 1 | 0 | Вход 2 |
| 1 | 1 | Вход 3 |

**Цифровые компараторы** выполняют сравнение двух чисел, заданных в двоичном коде. Они могут определять равенство двух двоичных чисел А и В с одинаковым количеством разрядов либо вид неравенства А>В или А<В. Цифровые компараторы имеют три выхода.

**Микросхемы компараторов кодов** применяются для сравнения двух входных кодов и выдачи на выходы сигналов о результатах этого сравнения (о равенстве или неравенстве кодов). На схемах компараторы кодов обозначаются двумя символами равенства: "==".

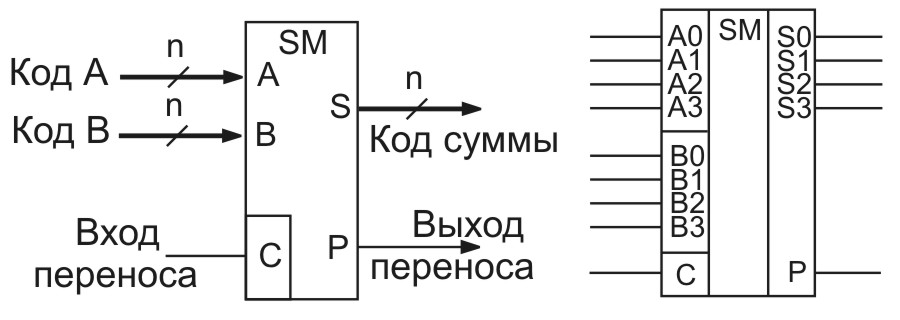


|  |  |  |
| --- | --- | --- |
| Входы | | Выход  A = B |
| А, В | S |
| A ≠ B | X | 0 |
| A = B | 1 | 1 |
| A = B | 0 | 0 |

**Вопрос 11. Сумматоры. Кварцевые генераторы импульсов.**

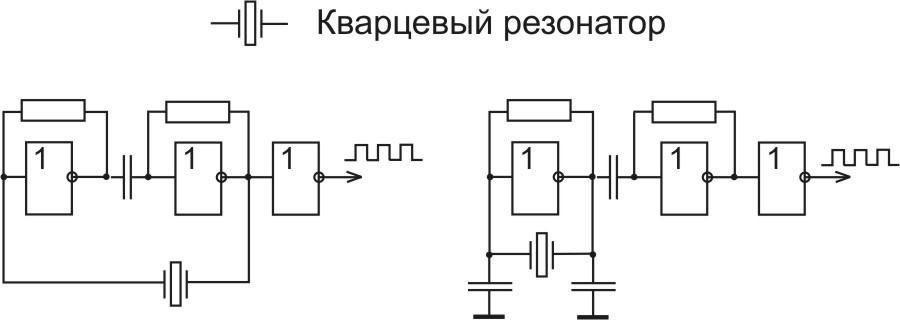
**Сумматоры** – это комбинационные устройства, предназначенные для

сложения чисел.



|  |  |
| --- | --- |
| Вход переноса С | Выходной код S |
| 0 | А+В (P=1 при переносе) |
| 1 | А+В+1 (P=1 при переносе) |

**КВАРЦЕВЫЕ ГЕНЕРАТОРЫ ИМПУЛЬСОВ**



**Резонатор** – это система, которая способна совершать колебания с максимальной амплитудой, то есть резонировать, при воздействии внешней силы определенной частоты и формы. Получается, кварцевый резонатор в электронике, а в народе просто "кварц", – это радиоэлемент, который способен резонировать, если на него подать переменный ток определенной частоты и формы.

Кварц является диэлектриком. А что будет если тонкий диэлектрик разместить между двумя металлическими пластинами? Получится [конденсатор](http://www.ruselectronic.com/news/kondjensatory/)!

**Вопрос 12.Особенности устройств с внутренней памятью.**

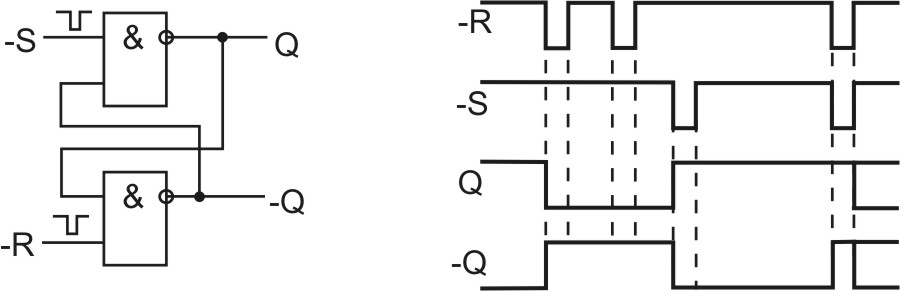
* Строятся на основе логических элементов (НЕ, И, ИЛИ, буферы);
* Состояния выходных сигналов определяются не только текущими входными сигналами, но и предшествующей историей (память);
* Сохраняют информацию во внутренней памяти до тех пор, пока есть питание, при выключении питания информация пропадает;
* При включении питания информация во внутренней памяти не определена (может быть любой);
* Примеры устройств: триггеры, регистры, счётчики, оперативная память (ОЗУ).

**Вопрос 13.Простейший триггер на двух элементах 2И-НЕ. D-триггер.**

**Триггеры** — это устройства с двумя состояниями. Они предназначены для запоминания двоичной информации. Использование триггеров позволяет реализовывать устройства оперативной памяти (то есть памяти, информация в которой хранится только на время вычислений). Однако это не единственная их область применения. Триггеры широко используются для построения цифровых устройств с памятью, таких как [счётчики](http://digteh.ru/digital/counter.php), [преобразователи последовательного кода в параллельный,](http://digteh.ru/digital/Reg.php#PoslReg) последовательные порты или цифровые линии задержки, применяемые в составе цифровых фильтров.

Аббревиатура RS означает, что триггер переключается из одного состояния в другое (0 или 1 на выходе) при подаче на его входы кратковременных импульсов. Вход **R** - reset (сброс) предназначен для установки триггера в исходное состояние, когда на выходе присутствует 0. Вход **S** - set (установка) получив кратковременный импульс, устанавливает триггер в состояние 1 на выходе. В простейшем случае такой триггер состоит из двух двухвходовых элементов ИЛИ и двух элементов НЕ.

* **R** – сброс (Reset),
* **S** – установка (Set),
* **Q** и -**Q** – прямой и инверсный выходы триггера.



**D-триггер**

Часто, в реальных электронных приборах требуются триггеры, которые изменяют своё состояние при подаче импульсов на один вход. В этом случае триггер работает по следующей схеме: первый импульс устанавливает триггер в 1, следующий импульс устанавливает триггер в 0, следующий импульс устанавливает в 1 и так далее. Очевидно, что частота повторения импульсов на выходе триггера в два раза ниже частоты на входе, то есть, такой триггер можно использовать для деления частоты пополам.

В RS-триггерах для записи логического нуля и логической единицы требуются разные входы, что не всегда удобно. При записи и хранении данных один бит может принимать значение, как нуля, так и единицы. Для его передачи достаточно одного провода. Как мы уже видели ранее, сигналы установки и сброса триггера не могут появляться одновременно, поэтому можно объединить эти входы при помощи инвертора, как показано на рисунке 1.

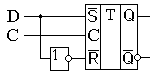
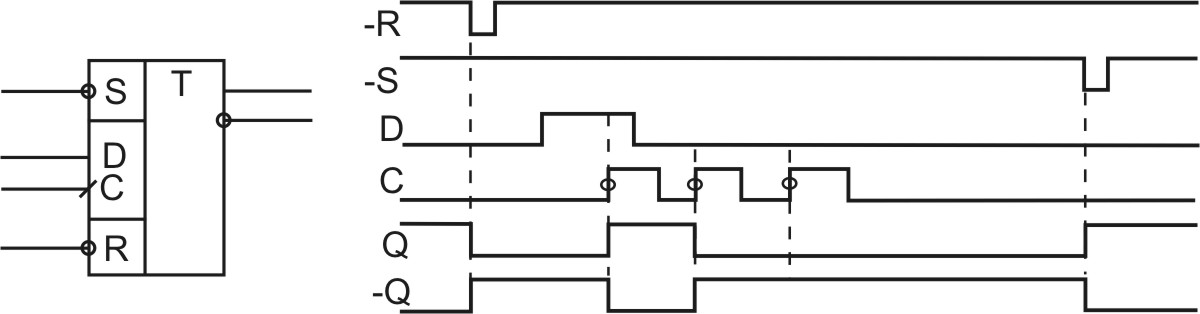


Рисунок 1. Принципиальная схема D триггера (защелки)

Такой триггер получил название [**D-триггер**](http://digteh.ru/digital/D_trigg/). Название происходит от английского слова delay — задержка. Конкретное значение задержки определяется частотой следования импульсов синхронизации. Условно-графическое обозначение D триггера на принципиальных схемах приведено на рисунке 2.



Рисунок 2. Условно-графическое обозначение D триггера (защелки)



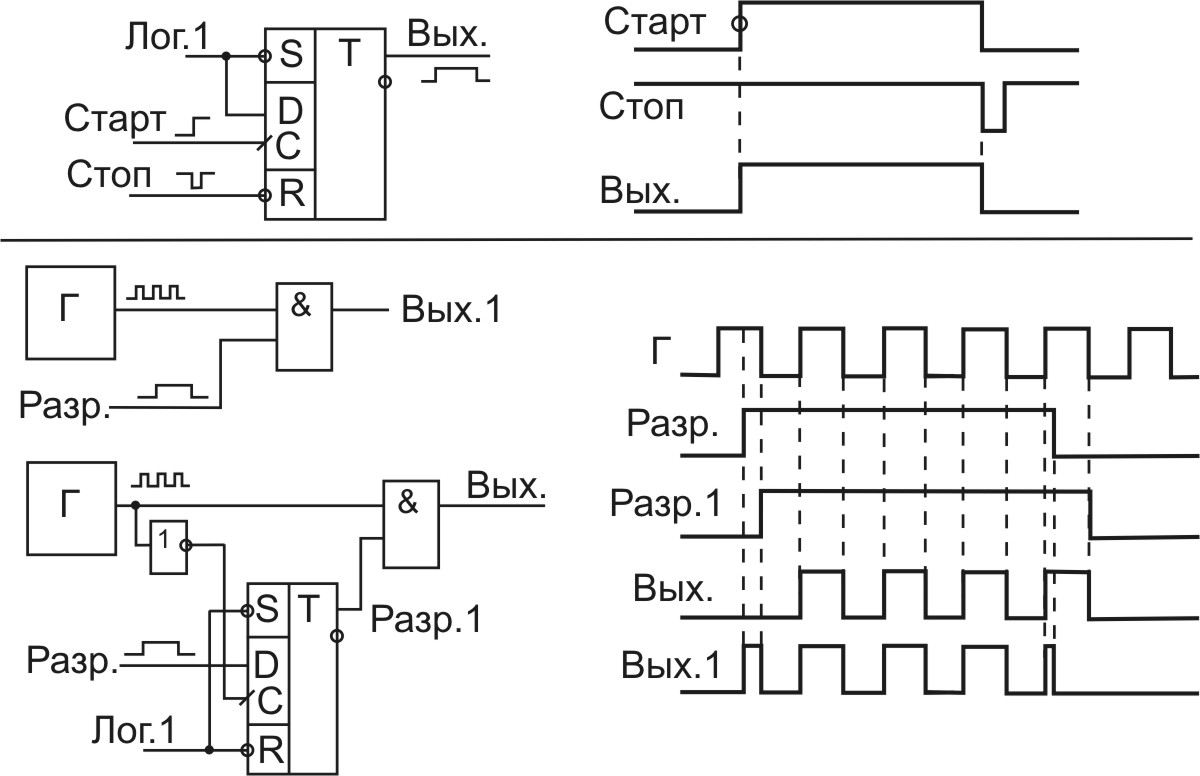
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Входы | | | | Выходы | |
| -S | -R | C | D | Q | -Q |
| 0 | 1 | X | X | 1 | 0 |
| 1 | 0 | X | X | 0 | 1 |
| 0 | 0 | X | X | Не определено | |
| 1 | 1 | 0🡪1 | 1 | 1 | 0 |
| 1 | 1 | 0🡪1 | 0 | 0 | 1 |
| 1 | 1 | 0 | X | Не меняется | |
| 1 | 1 | 1 | X | Не меняется | |
| 1 | 1 | 1🡪0 | X | Не меняется | |

**Вопрос 14.Применение триггера: флаг и синхронизация.**

Основное применение триггеры находят в тех случаях, когда надо сформировать сигнал, длительность которого соответствует длительности какой-то выполняемой операции, какого-то продолжительного процесса в схеме. Выходной сигнал триггера при этом может разрешать этот самый процесс, а может информировать остальные узлы устройства о том, что процесс идет (или, как говорят, служить флагом процесса). Например, в схеме в начале процесса (операции) по сигналу "Старт" триггер перебрасывается в единицу, а в конце процесса (операции) по сигналу "Стоп" — обратно в нуль.

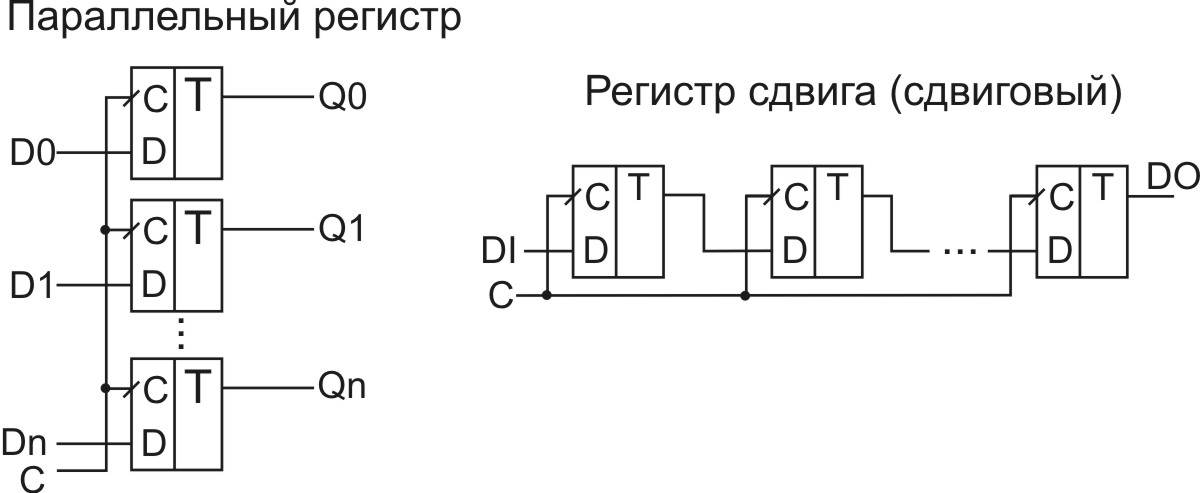
Вторая важнейшая область применения триггеров — **синхронизация сигналов.**

Например, триггер позволяет наиболее просто избавиться от паразитных коротких импульсов на выходах комбинационных схем, возникающих при почти одновременном изменении нескольких входных сигналов. Для синхронизации в данном случае необходимо иметь синхросигнал (синхропереход), сопровождающий входные информационные сигналы (входной код) и задержанный относительно момента изменения этих сигналов на время tз, большее задержки комбинационной схемы. При подаче этого синхроимпульса на вход С триггера, а выходного сигнала комбинационной микросхемы (Вых. 1) на вход D триггера на выходе триггера получаем сигнал (Вых. 2), полностью свободный от паразитных импульсов.

****

**Вопрос 15.Построение регистров из триггеров. Виды параллельных регистров.**

**ПОСТРОЕНИЕ РЕГИСТРОВ ИЗ ТРИГГЕРОВ**

****

Параллельный регистр — для хранения кодов.

Регистр сдвига — для преобразования параллельного кода в последовательный и обратно.

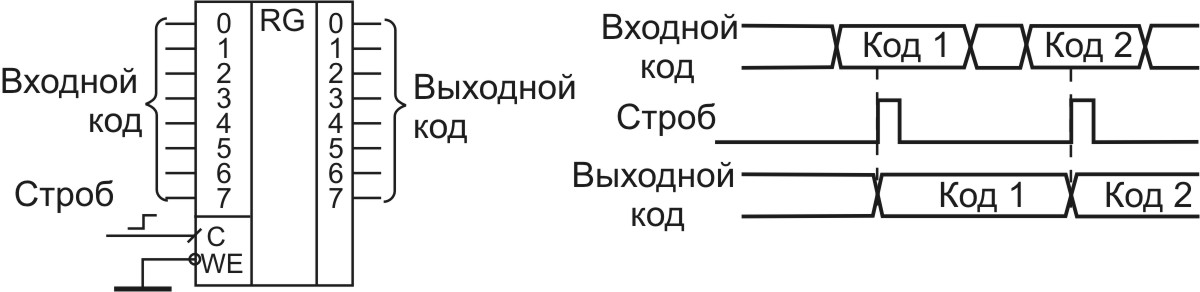
**ВИДЫ ПАРАЛЛЕЛЬНЫХ РЕГИСТРОВ**

1.Регистры, срабатывающие по фронту управляющего сигнала (тактируемые регистры). Изменение состояния — по фронту сигнала С. До прихода следующего фронта — хранение.

2.Регистры, срабатывающие по уровню управляющего сигнала (регистры-защёлки). Если сигнал С=1, то выходные сигналы повторяют входные. Если сигнал С=0, то запоминание и хранение входных сигналов.

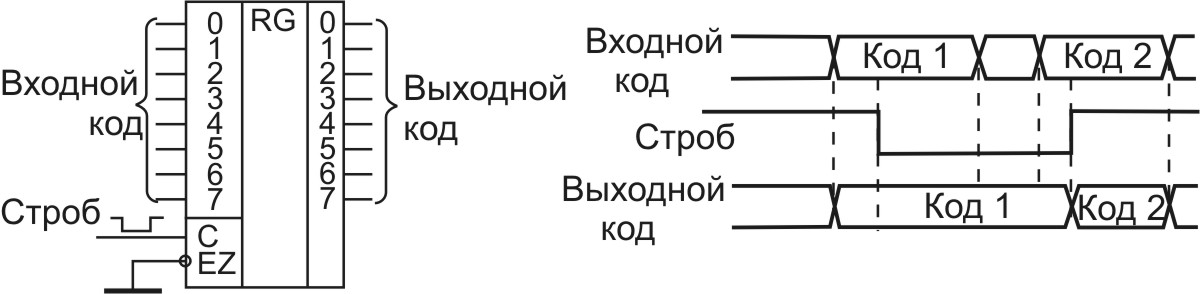


**ТАКТИРУЕМЫЙ РЕГИСТР**

****

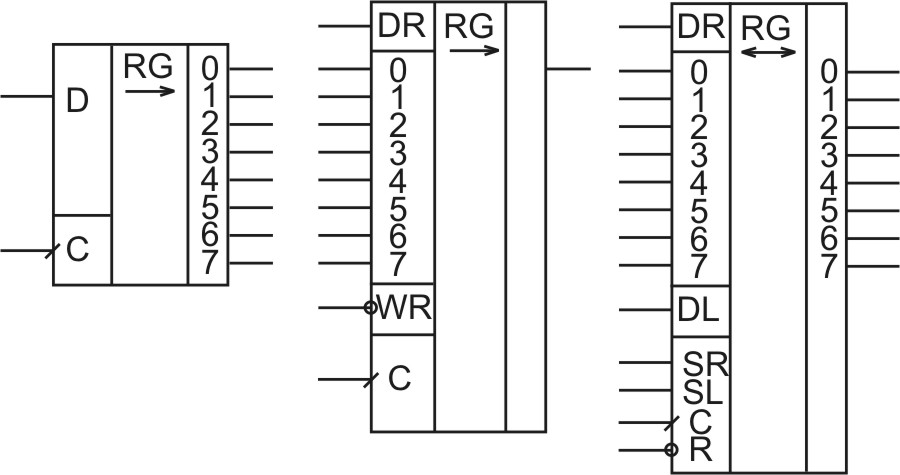
|  |  |  |  |
| --- | --- | --- | --- |
| **Входы** | | | **Выходы** |
| **-WE** | **C** | **D** | **Q** |
| **0** | **0🡪1** | **0** | **0** |
| **0** | **0🡪1** | **1** | **1** |
| **0** | **0** | **X** | **Не меняется** |
| **0** | **1** | **X** | **Не меняется** |
| **1** | **X** | **X** | **Не меняется** |

**РЕГИСТР-ЗАЩЁЛКА**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Входы** | | | **Выходы** |
| **-EZ** | **C** | **D** | **Q** |
| **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **0** |
| **0** | **0** | **X** | **Не меняется** |
| **1** | **X** | **X** | **Z-состояние** |

**ТИПЫ РЕГИСТРОВ СДВИГА**

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Входы** | | **Выходы** | | | |
| **C** | **D** | **Q0** | **Q1** | **…** | **Q7** |
| **0** | **X** | **Не меняется** | | | |
| **1** | **X** | **Не меняется** | | | |
| **0🡪1** | **0** | **0** | **Q0** | **…** | **Q6** |
| **0🡪1** | **1** | **1** | **Q0** | **…** | **Q6** |

**Вопрос 16.Последовательная передача данных. Построение счетчика из триггеров.**

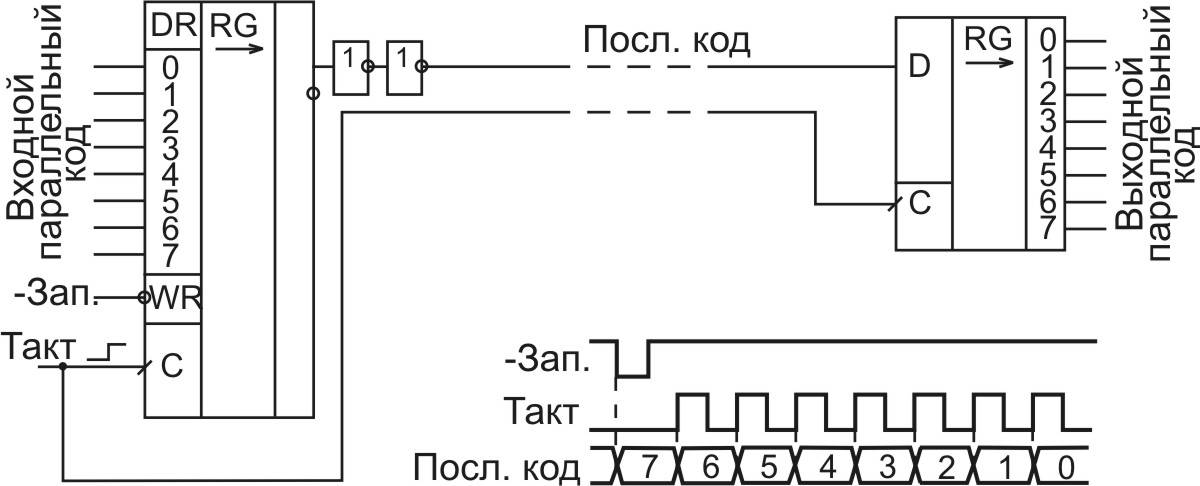
**Под последовательной передачей данных** понимают процесс передачи данных по одному [биту](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) за один промежуток времени, последовательно один за одним по одному коммуникационному каналу или компьютерной шине, в отличие от параллельной передачи данных, при которой несколько бит пересылаются одновременно по линии связи из нескольких параллельных каналов. Последовательная передача всегда используется при связи на дальние расстояния и в большинстве компьютерных сетей, так как стоимость кабеля и трудности синхронизации делают параллельную передачу данных неэффективной. При передаче данных на короткие расстояния последовательные компьютерные шины также используются всё чаще, так как и здесь недостатки параллельных шин перевешивают их преимущества в простоте.

Для передачи информации на большие расстояния, например, при объединении компьютеров в сети, используется **последовательный способ передачи.** Возможны два режима последовательной передачи: **синхронный** и **асинхронный.**

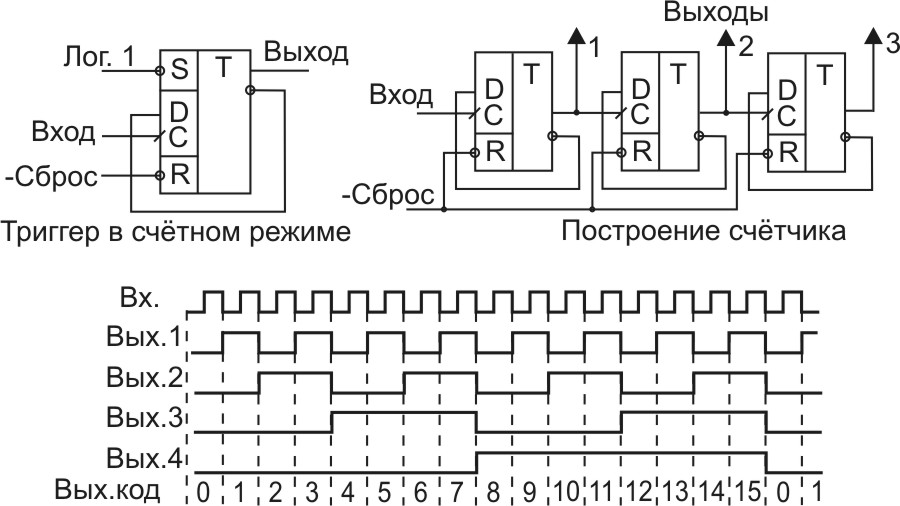
При **синхронной передаче** каждый передаваемый бит сопровождается импульсом синхронизации, информирующим приемник о наличии в линии информационного бита. Синхронизирующие импульсы управляют приемом информации. Следовательно, между передатчиком и приемником должны быть протянуты минимум три провода: один – для передачи данных, второй – для передачи синхроимпульсов, третий – общий заземленный. Если же расстояние между источником и приемником составляет несколько метров, то каждый из сигналов (информационный и синхронизирующий) приходится посылать по экранированному кабелю, что значительно увеличивает стоимость линии связи.

**Асинхронный способ** передачи не требует синхронизации действий приемника и передатчика; по этой причине для связи достаточно линии из двух проводников, причем, оказывается возможным использование даже телефонных линий.

Передача производится машинными словами (информационными битами), дополненными несколькими служебными.

****

**ПОСТРОЕНИЕ СЧЁТЧИКА ИЗ ТРИГГЕРОВ**

****

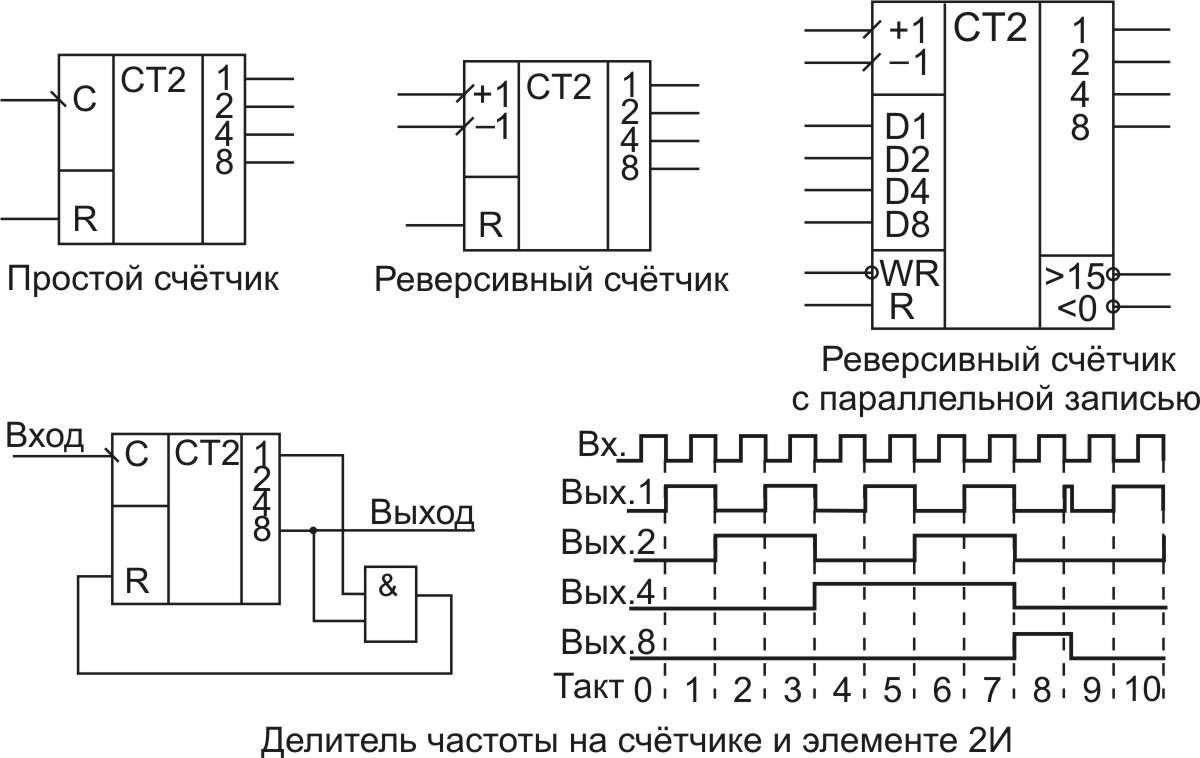
На базе счетных триггеров можно построить цифровое устройство, получившее название электронного счетчика. Электронные счетчики (далее, просто счетчики) позволяют вести подсчет электрических импульсов, количество которых (поступивших на вход счетчика) представляется, обычно, в параллельном коде.

**Вопрос 17.Функции и типы счетчиков.**

**ФУНКЦИИ СЧЁТЧИКОВ**

* Счёт входных импульсов;
* Деление частоты входного сигнала;
* Формирование сигналов заданной длительности;
* Формирование последовательностей сигналов;
* Измерение временных интервалов;
* Часы (таймер);
* Синтез (формирование) частоты;
* Измерение частоты входного сигнала;
* Последовательный перебор кодов (например, адресов памяти);
* Последовательный перебор каналов (входных и выходных) — с дешифратором или мультиплексором.

**ТИПЫ СЧЁТЧИКОВ**

****

**Вопрос 18.Типы памяти. Основные понятия памяти. Запись и чтение оперативной памяти.**

**ТИПЫ ПАМЯТИ**

1.Постоянная память (ПЗУ, ROM) — энергонезависимая, хранит записанную информацию постоянно;

2.Программируемая постоянная память (ППЗУ, PROM) — информация перезаписывается ограниченное число раз, энергонезависимая.

3.Оперативная память (ОЗУ, RAM) — информация перезаписывается неограниченное число раз, хранится при включённом питании:

- Статическая оперативная память — не требует регенерации для хранения;

- Динамическая оперативная память — необходима регенерация для хранения.

**ОСНОВНЫЕ ПОНЯТИЯ ПАМЯТИ**

**Ячейка памяти** — элемент, хранящий информацию (например, триггер, регистр);

**Адрес памяти** — код номера ячейки памяти;

**Разрядность памяти** — разрядность каждой ячейки;

**Организация памяти** — объём и разрядность памяти: 1К х 16, 16М х 8, 1Г х 1.

**Запись памяти** — обновление содержимого ячейки памяти, определяемой адресом;

**Чтение памяти** — вывод содержимого ячейки памяти, определяемой адресом.

**Регенерация** — необходимое регулярное освежение информации в динамической памяти.

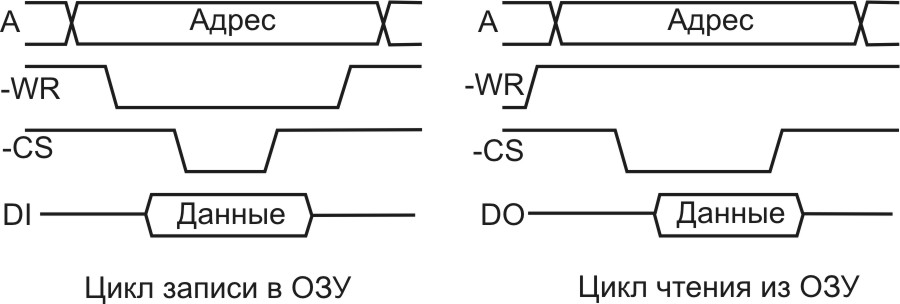
**ОБОЗНАЧЕНИЯ ПАМЯТИ**

****

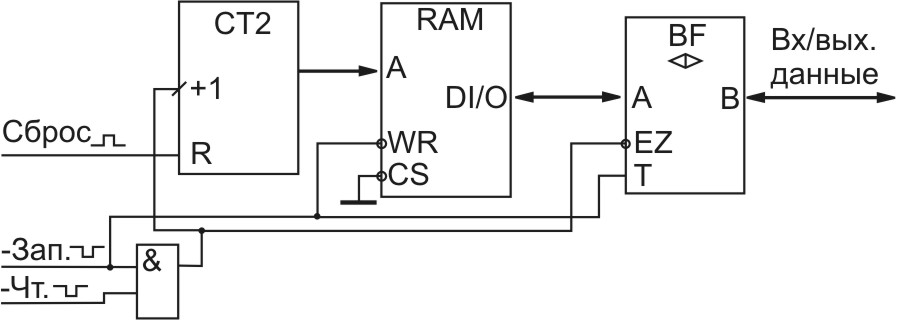
**ЗАПИСЬ И ЧТЕНИЕ ОПЕРАТИВНОЙ ПАМЯТИ**

Для описания характеристик быстродействия оперативной памяти в пакетном режиме применяются так называемые циклы чтения/записи (или временная схема пакета). Эти числа относятся к количеству тактов процессора для каждого доступа при чтении. Дело в том, что при обращении к памяти на считывание или запись первого машинного слова расходуется больше тактов, чем на обращение к трем последующим словам. Так, для асинхронной SRAM (обеспечивает быстродействие от 22 до 20 нс, при частоте шины центрального процессора от 50 до 66 МГц) чтение одного слова выполняется за 3 такта, запись - за 4 такта, чтение нескольких слов определяется последовательностью 3-2-2-2 такта (что означает, что чтение 2-го элемента данных занимает 3 такта ЦП, включая 2 такта ожидания, а чтение последующих - по 2 временных такта), а запись - 4-3-3-3.

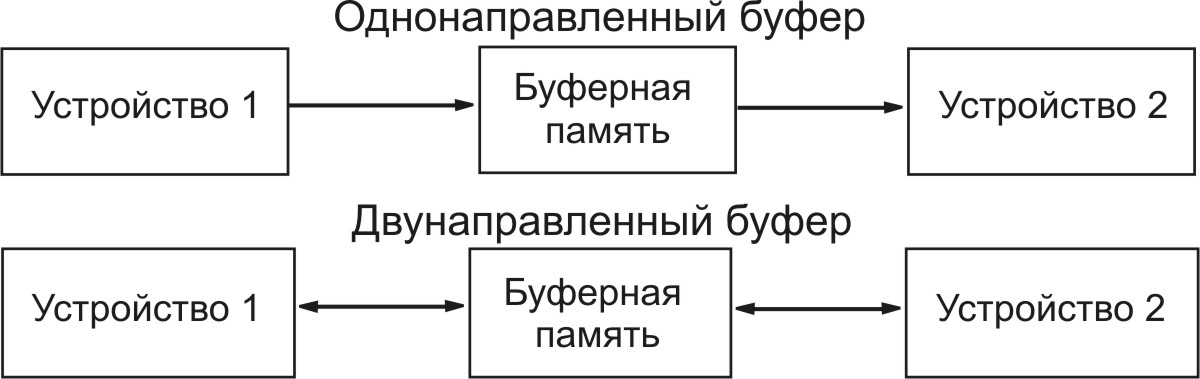
Применение кэширования особенно эффективно, когда доступ к данным осуществляется преимущественно в последовательном порядке. Тогда после первого запроса на чтение данных, расположенных в медленной (кэшируемой) памяти, можно заранее (упреждающее чтение) выполнить чтение следующих блоков данных в кэш-память для того, чтобы при следующем запросе на чтение данных почти мгновенно выдать их из кэш-памяти.

****

**ХРАНЕНИЕ МАССИВА ДАННЫХ В ОЗУ**

****

**ОЗУ как информационный буфер**

****

1.FIFO ─ чтение в том же порядке, что и запись;

2.LIFO — чтение в порядке, противоположном записи.

**Применение буферной памяти**

1.Обеспечение независимой работы двух устройств, обменивающихся информацией через буфер;

2.Согласование скоростей обмена различных устройств;

3.Постепенное накопление информации перед передачей одним массивом;

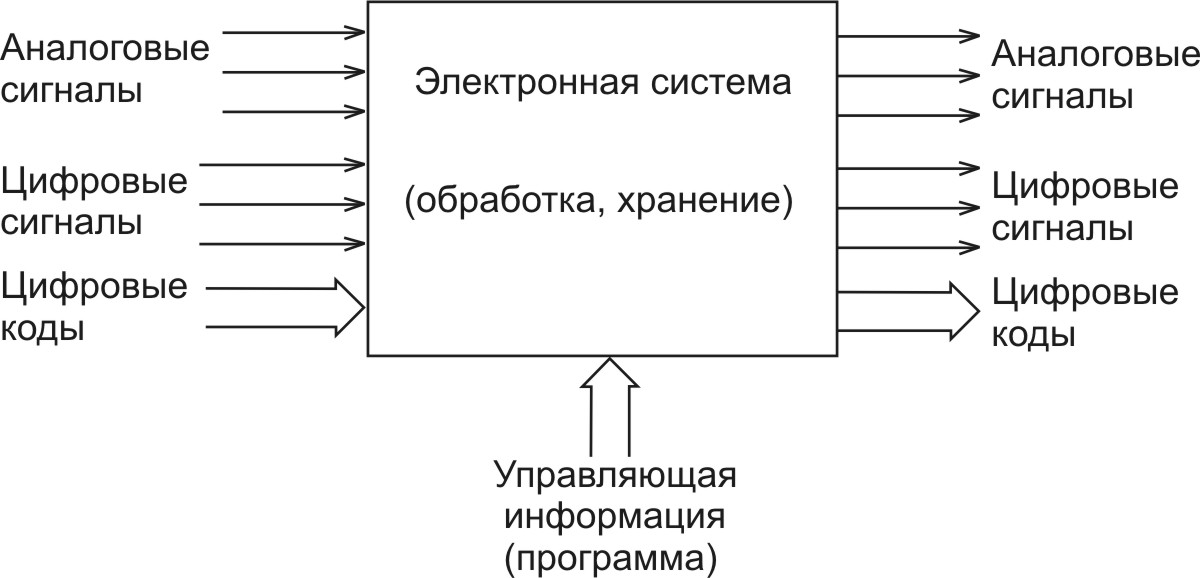
4.Выборочное чтение информации, переданной одним массивом;

5.Передача информации пакетами со стандартным обрамлением (управляющая информация).

**Вопрос 19.Микропроцессорная система. Информационные потоки в микропроцессорной системе.**

**МИКРОПРОЦЕССОРНАЯ СИСТЕМА**

**Микропроцессорная система** представляет собой функционально законченное изделие, состоящее из одного или нескольких устройств, главным образом из [микропроцессора](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) и/или [микроконтроллера](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80).

****

**Особенности микропроцессорных систем**

1.Гибкая логика работы — меняется в зависимости от задачи;

2.Универсальность — может решать очень много задач;

3.Простота проектирования аппаратуры — единообразие схемотехнических решений;

4.Простота отладки — единообразие системы связей и протоколов обмена;

5.Аппаратурная избыточность, особенно для простых задач;

6.Ниже быстродействие, чем у устройств с жёсткой логикой;

7.Необходимость разработки и отладки программного обеспечения.

**Процессор** — обработчик и вычислитель, выполняющий все операции над кодами и сигналами;

**Программа** — набор управляющих кодов (команд), определяющих логику работы системы;

**Команда** — управляющий код, указывающий процессору, что ему надо делать в данный момент;

**Шина** (магистраль, канал) — линии связи, объединяющие устройства микропроцессорной системы;

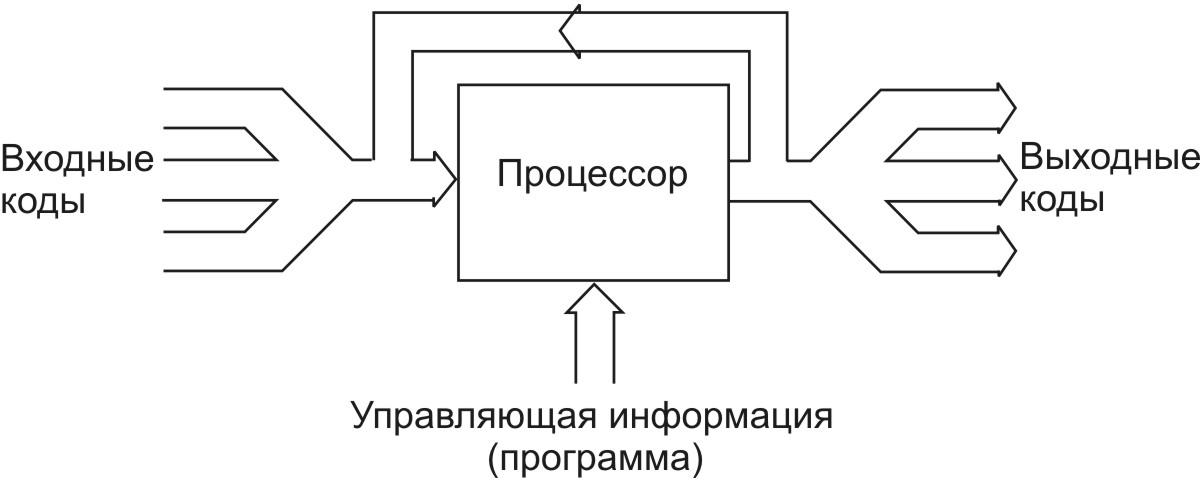
**Интерфейс** (сопряжение) — соглашение об обмене информацией, а также технические средства для реализации этого обмена.

**ИНФОРМАЦИОННЫЕ ПОТОКИ В МИКРОПРОЦЕССОРНОЙ СИСТЕМЕ**

С логической точки зрения в микропроцессорной системе возможны три информационных потока: процессор – память, процессор – устройства ввода-вывода (УВВ) и УВВ – память.

Процессор может принимать информацию по ШД (режимы чтения памяти или ввода) либо выдавать информацию на ШД (режимы записи в память или вывода). В этих режимах системные шины адреса и данных находятся в распоряжении процессора. В режиме прямого доступа в память (ПДП) системные шины адреса и данных поступают в распоряжение внешнего УВВ, которое обменивается данными с памятью, минуя процессор. Режим ПДП обычно реализуется специальным устройством – контроллером ПДП.

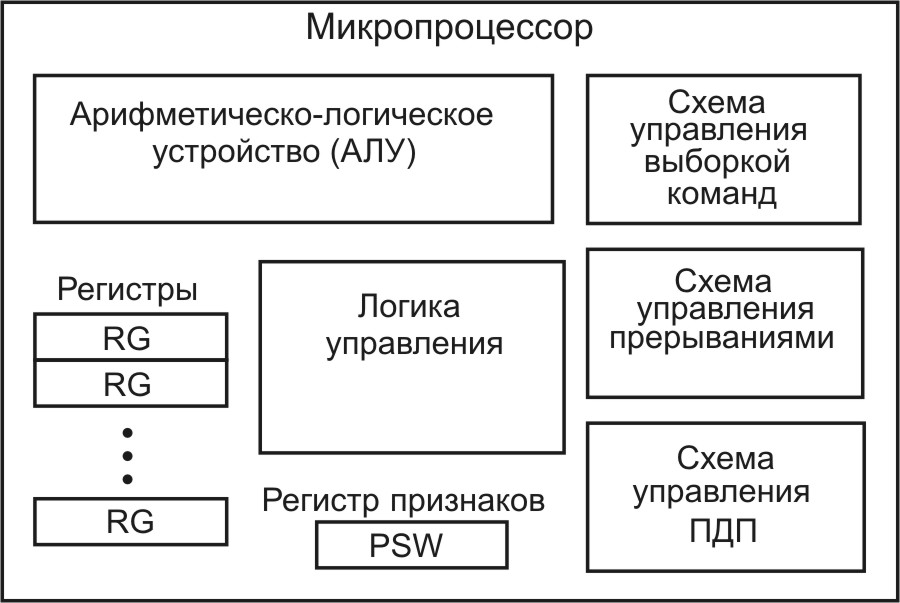
При физической реализации любого из указанных информационных потоков используются одни и те же физические шины адреса и данных, однако каждый тип обмена поддерживается различными управляющими сигналами. Путем комбинации информации (адрес + сигналы управления) процессор или контроллер ПДП задает физическое устройство (вплоть до ячейки памяти), к которому производится обращение. Таким образом, логически мы можем рассматривать в вычислительной системе три информационных потока, которые физически реализуются на одних и тех же информационных шинах.

****

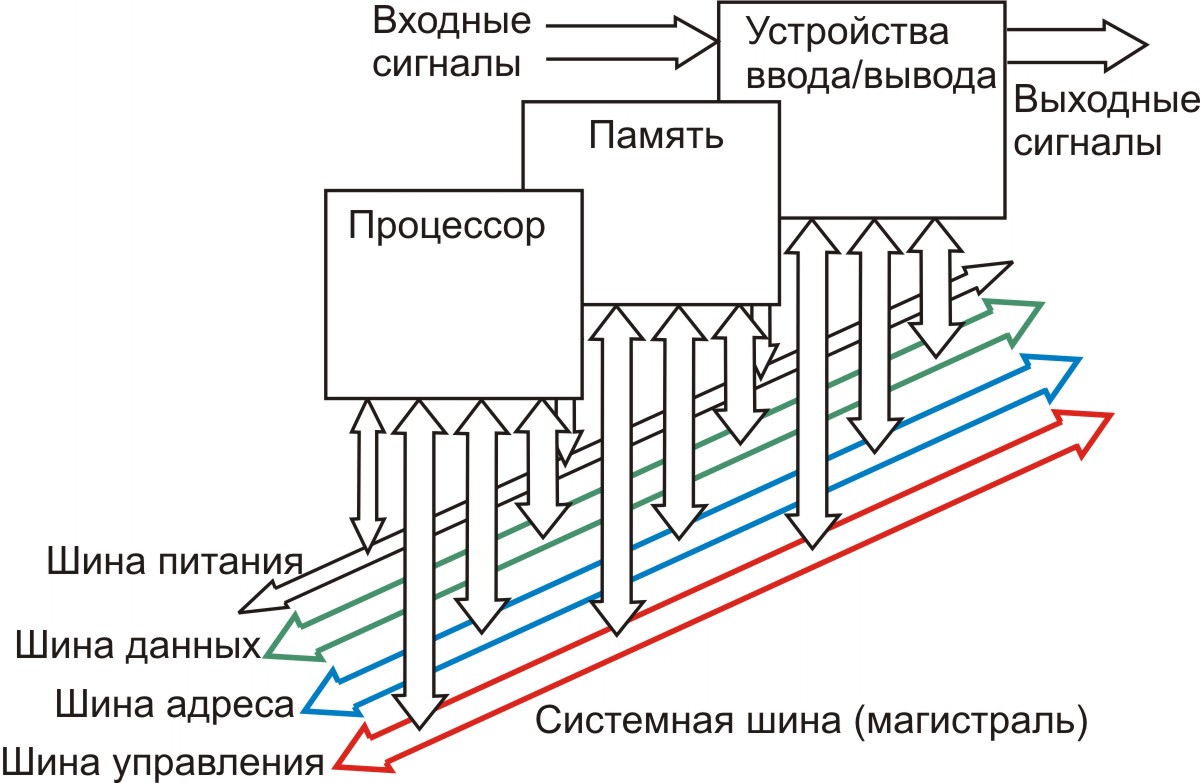
**Вопрос 20. Структура простейшего микропроцессора. Структура микропроцессорной системы.**

**СТРУКТУРА ПРОСТЕЙШЕГО МИКРОПРОЦЕССОРА**

Для выполнения команд в структуру процессора входят внутренние регистры, арифметико-логическое устройство (АЛУ, ALU — Arithmetic Logic Unit), мультиплексоры, буферы, регистры и другие узлы. Работа всех узлов синхронизируется общим внешним тактовым сигналом процессора. То есть процессор представляет собой довольно сложное цифровое устройство.

****

**СТРУКТУРА МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ**

****

**УСТРОЙСТВА МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ**

* Процессор — обработчик, выполняет пересылку и обработку информации (арифметическую, логическую) в соответствии с программой; управляет выборкой команд;
* Память — оперативная (RAM) и постоянная (ROM) — хранит данные и программы. Оперативная — для временного хранения данных и программ, постоянная — для постоянного хранения, главное — для программы начального запуска при включении питания.
* Устройства ввода/вывода (УВВ, I/O — Input/Output) — для обеспечения связи микропроцессорной системы с внешними устройствами и с пользователем (внешние интерфейсы и пользовательский интерфейс). Они же помогают процессору в пересылке данных и в реагировании на внешние события.

**ШИНЫ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ**

**Шина адреса** (Address Bus) — для пересылки кода адреса (индивидуального номера устройства, участвующего в обмене в данный момент).

**Шина данных** (Data Bus) — для пересылки данных между устройствами. Двунаправленная шина, состоит из нескольких байтов (1, 2, 4, 8);

**Шина управления** (Control Bus) — для пересылки отдельных управляющих сигналов: тактовых, стробирующих, подтверждающих, инициирующих и т.д.;

**Шина питания** (Power Bus) — для подведения к устройствам напряжений питания (положительных, отрицательных, общего провода).

**Вопрос 21.Циклы обмена в микропроцессорной системе. Программный обмен информацией.**

**ФАЗЫ ЦИКЛА ОБМЕНА**

1.Адресная фаза: процессор (задатчик, Master) выставляет адрес УВВ (или ячейки памяти), к которому хочет обратиться (исполнитель, Slave);

2.Фаза данных:

*2.1.Цикл записи*: процессор выставляет данные, предназначенные для записи, и выдаёт строб записи. Исполнитель принимает данные от процессора.

*2.2.Цикл чтения*: процессор выдаёт строб чтения. Исполнитель выставляет данные для передачи процессору. Процессор принимает данные от исполнителя.

3.Фаза подтверждения (не обязательна): исполнитель выдаёт процессору сигнал подтверждения выполнения операции

**ЦИКЛЫ ОБМЕНА В МИКРОПРОЦЕССОРНОЙ СИСТЕМЕ**

1.Программные циклы обмена

*1.2.*Чтение (ввод, выборка) команды из памяти (оперативной или постоянной);

*1.3.*Чтение (ввод) данных из памяти;

*1.4.*Запись (вывод) данных в память;

*1.5.*Приём (чтение, ввод) данных из устройства ввода/вывода;

*1.6.*Передача (запись, вывод) данных в устройство ввода/вывода;

2.Циклы обмена по прерываниям (Interrupts);

3.Циклы обмена по прямому доступу к памяти (ПДП, DMA – Direct Memory Access);

4.Циклы обмена при захвате шины.

**ПРОГРАММНЫЙ ОБМЕН ИНФОРМАЦИЕЙ**

Программный обмен информацией является основным в микропроцессорной системе.

- Он предусмотрен всегда, без него невозможны другие режимы обмена.

- В этом режиме только процессор управляет системной магистралью.

- Все операции (циклы) обмена информацией в данном случае инициируются только процессором, все они выполняются строго в порядке, предписанном исполняемой программой.

В этом режиме процессор

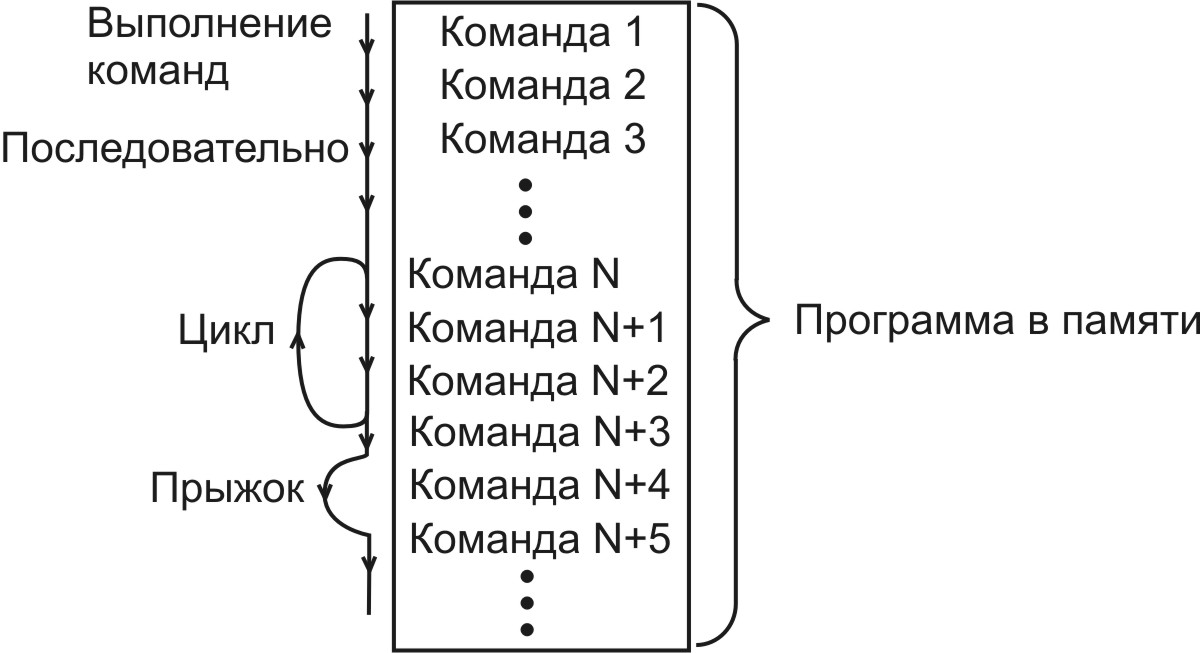
- выбирает из памяти коды команд и исполняет их,

- читает данные из памяти или из устройства ввода/вывода,

- обрабатывает их,

- записывает данные в память или передавая их в устройство ввода/вывода.

Ни на какие внешние события, не связанные с программой, процессор не реагирует.

****

**Вопрос 22.Методы реакции на внешнее событие. Обслуживание прерывания.**

**МЕТОДЫ РЕАКЦИИ НА ВНЕШНЕЕ СОБЫТИЕ**

- С помощью периодического программного контроля факта наступления события (метод опроса флага или Polling). Самая быстрая реакция, но процессор не может заниматься ничем другим;

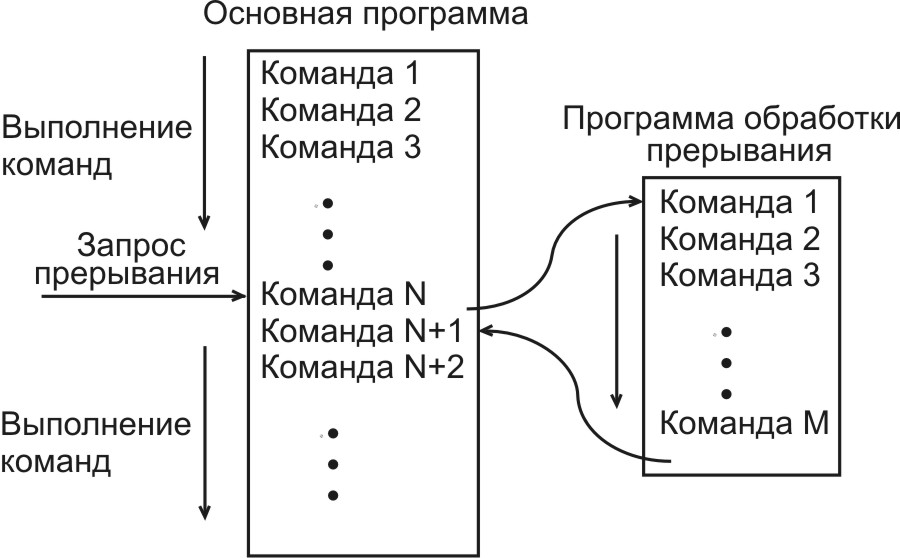
- С помощью прерывания, то есть насильственного перевода процессора с выполнения текущей программы на выполнение экстренно необходимой программы ─ программы обработки прерывания. Более медленная реакция, обмен — со скоростью процессора

- С помощью прямого доступа к памяти (ПДП), то есть без участия процессора при его отключении от системной магистрали. Медленная реакция, обмен — со скоростью контроллера ПДП (быстрее, чем процессор).

**ОБСЛУЖИВАНИЕ ПРЕРЫВАНИЯ**

Обмен по прерываниямиспользуется тогда, когда необходима реакция микропроцессорной системы на какое-то внешнее событие, на приход внешнего сигнала. В случае компьютера внешним событием может быть, например, нажатие на клавишу клавиатуры или приход по локальной сети пакета данных. Компьютер должен реагировать на это, соответственно, выводом символа на экран или же чтением и обработкой принятого по сети пакета. Процессор, получив запрос прерывания от внешнего устройства (часто называемый IRQ — Interrupt ReQuest), заканчивает выполнение текущей команды и переходит к программе обработки прерывания. Закончив выполнение программы обработки прерывания, он возвращается к прерванной программе с той точки, где его прервали

Обслуживание прерывания начинается выполнением процедур классификации ошибок как сбоев ( восстанавливающихся отказов) или невосстанавливающихся отказов. Если при повторениях программы ошибка подтверждается, то она классифицируется как отказ и осуществляется переход к локализации неисправности. Вход в процедуру локализации неисправности может осуществляться автоматически или с пульта управления оператором. В системе автоматической диагностики ЭВМ ЕС-1030 реализован метод раскрутки, который характеризуется последовательным расширением работоспособной части ЭВМ.



**Вопрос 23.Обслуживание прямого доступа к памяти (ПДП).**

Прямой доступ к памяти освобождает процессор от управления операциями ввода - вывода, позволяя осуществлять параллельно во времени выполнение процессором программы с обменом данными между периферийными устройствами (ПУ) и оперативной памятью (ОП), и производить этот обмен со скоростью, ограниченной только пропускной способностью ПУ и ОП. Таким образом, увеличивается производительность ЭВМ. Прямым доступом к памяти управляет контроллер ПДП, который выполняет следующие функции: 1) Управление инициируемой процессором или ПУ передачей данных между ПУ и ОП; 2) Задание размеров блока данных, который подлежит передаче, и области памяти, используемой при передаче; 3) Формирование адресов ячеек ОП, участвующих при передаче.

**Прямой доступ к памяти (ПДП, DMA)** — это режим, при котором обмен по системной шине идет без участия процессора. Внешнее устройство, требующее обслуживания, сигнализирует процессору, что режим ПДП необходим, в ответ на это процессор заканчивает выполнение текущей команды и отключается от всех шин, сигнализируя запросившему устройству, что обмен в режиме ПДП можно начинать.

ПДП предполагает наличие на системной шине дополнительного модуля — контроллера прямого доступа к памяти (КПДП), способного брать на себя функции ЦП по управлению системной шиной и обеспечивать прямую пересылку информации между ОП и ВУ, без участия центрального процессора. В сущности, КПДП - это и есть модуль ввода/вывода, реализующий режим прямого доступа к памяти.

Когда пересылаются большие объемы данных, требуется более эффективный способ ввода/вывода - **прямой доступ к памяти** (ПДП). ПДП предполагает наличие на системной шине дополнительного модуля — контроллера прямого доступа к памяти (КПДП), способного брать на себя функции ЦП по управлению системной шиной и обеспечивать прямую пересылку информации между ОП и ВУ, без участия центрального процессора.

Если ЦП желает прочитать или записать блок данных, он прежде всего должен поместить в КПДП информацию, характеризующую предстоящее действие. Этот процесс называется инициализацией КПДП и включает в себя занесение в контроллер следующих четырех параметров:

1.вида запроса (чтение или запись);

2.адреса устройства ввода/вывода;

3.адреса начальной ячейки блока памяти, откуда будет извлекаться или куда будет вводиться информация;

4.количества слов, подлежащих чтению или записи.

**Общий алгоритм ПДП.**

Для осуществления прямого доступа к памяти контроллер должен выполнить ряд последовательных операций:

1.принять запрос (DREQ) от устройства ввода-вывода;

2.сформировать запрос (HRQ) в процессор на захват шины;

3.принять сигнал (HLDA), подтверждающий захват шины;

4.сформировать сигнал (DACK), сообщающий устройству о начале обмена данными;

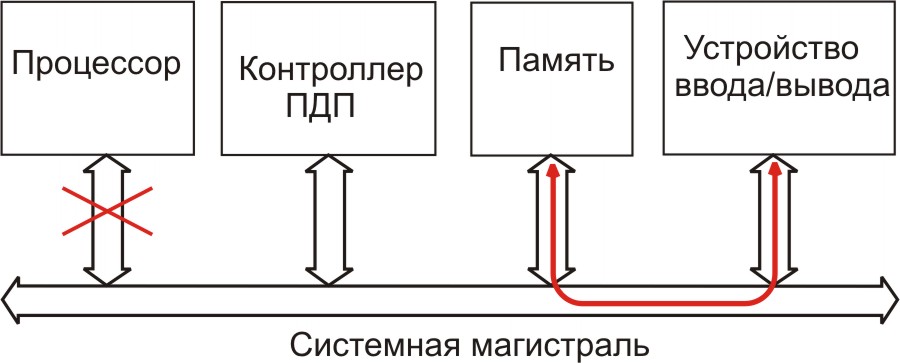
5.выдать адрес ячейки памяти, предназначенной для обмена;

6.выработать сигналы (MEMR, IOW или MEMW, IOR), обеспечивающие управление обменом;

7.по окончании цикла DMA либо повторить цикл DMA, изменив адрес, либо прекратить цикл.

****

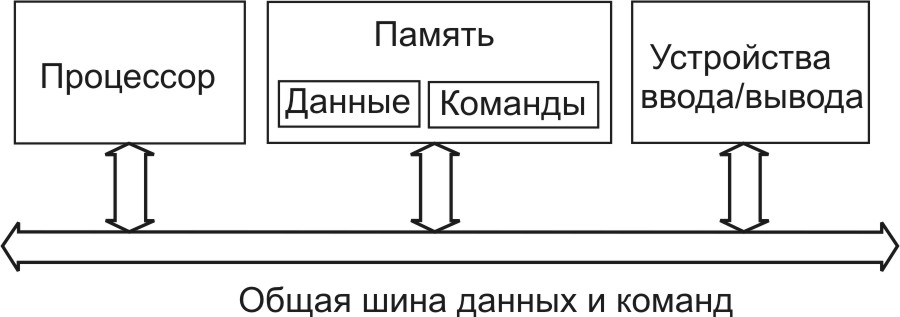
**Информационные потоки в режиме ПДП**

****

**Вопрос 24.Одношинная (принстонская) и двухшинная (гарвардская) архитектура.**

**ОДНОШИННАЯ (ПРИНСТОНСКАЯ) АРХИТЕКТУРА**

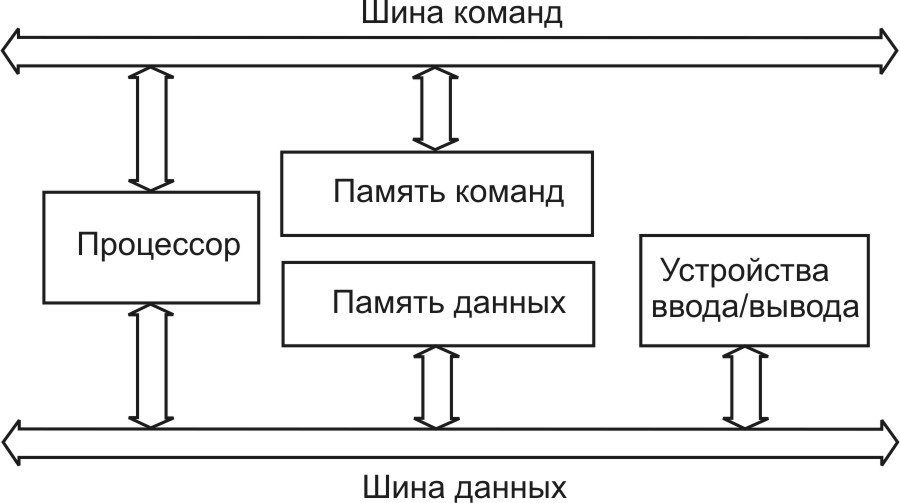
**Одношинная(принстонская) архитектура** — проще, меньше требований к процессору, более гибкое перераспределение памяти между программами и данными (память обычно большая), но медленнее (тратится время на чтение команд). Сложные универсальные системы.

****

**ДВУХШИННАЯ (ГАРВАРДСКАЯ) АРХИТЕКТУРА**

Архитектура с раздельными шинами данных и команд – это **двухшинная, или гарвардская, архитектура**. Эта архитектура предполагает наличие в системе отдельной памяти для данных и отдельной памяти для команд. Обмен процессора с каждым из двух типов памяти происходит по своей шине. Эта архитектура применяется в основном в однокристальных микроконтроллерах.

**Двухшинная(гарвардская) архитектура** — сложнее, больше требований к процессору(одновременное обслуживание двух потоков), нельзя перераспределять память (память обычно небольшая), но быстрее (команды читаются одновременно с пересылкой данных). Простые однокристальные системы — специализированные

****

1.Одношинная (принстонская) архитектура — проще, меньше требований к процессору, более гибкое перераспределение памяти между программами и данными (память обычно большая), но медленнее (тратится время на чтение команд). Сложные универсальные системы.

2.Двухшинная (гарвардская) архитектура — сложнее, больше требований к процессору(одновременное обслуживание двух потоков), нельзя перераспределять память (память обычно небольшая), но быстрее (команды читаются одновременно с пересылкой данных). Простые однокристальные системы — специализированные.

**Вопрос 25. Типы микропроцессорных систем. Схема включения процессора.**

**ТИПЫ МИКРОПРОЦЕССОРНЫХ СИСТЕМ**

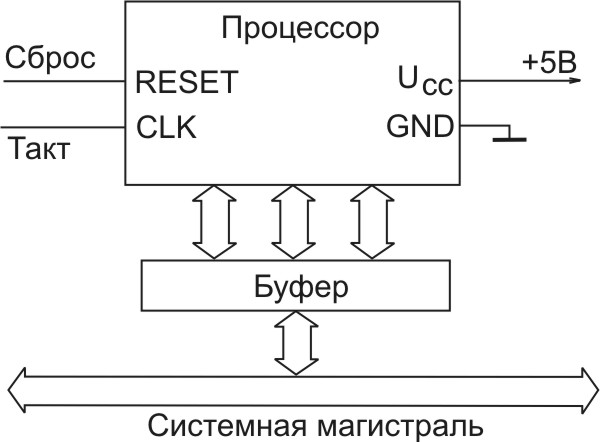
**Микроконтроллеры** — наиболее простой тип микропроцессорных систем, в которых все или большинство узлов системы выполнены в виде одной микросхемы. Узко специализированы, закрыты, шина недоступна.

**Контроллеры** — управляющие микропроцессорные системы, выполненные в виде отдельных модулей. Класс задач.

**Микрокомпьютеры** — более мощные микропроцессорные системы с развитыми средствами сопряжения с внешними устройствами. Гибко настраиваемые. Шина доступна.

**Компьютеры** (в том числе и персональные компьютеры) — самые мощные и наиболее универсальные микропроцессорные системы. Универсальные, дорогие, избыточные.

**СХЕМА ВКЛЮЧЕНИЯ ПРОЦЕССОРА**

****

**Вопрос 26. Методы ускорения работы процессора. Назначение регистров процессора.**

**МЕТОДЫ УСКОРЕНИЯ РАБОТЫ ПРОЦЕССОРА**

1.Повышение тактовой частоты — совершенствование технологии;

2.Уменьшение времени выполнения одной команды;

3.Оптимизация системы команд — уменьшение количества команд и добавление сложных команд (например, мультимедийных);

4.Распараллеливание выполнения команд: два и более арифметическо-логических устройств (АЛУ);

5.Распараллеливание процессов выборки команд и их выполнения:

*5.1.*Конвейер команд (быстрая FIFO- память);

*5.2.*Кэш-память.

**НАЗНАЧЕНИЕ РЕГИСТРОВ ПРОЦЕССОРА**

**Регистры данных** — временное хранение кодов данных.

**Регистры адресные** — коды адресов в памяти для работы с массивами информации. Могут работать как реверсивные счётчики с параллельной записью информации. Постинкремент и предекремент.

**Регистры универсальные** — могут хранить как адрес, так и данные.

**Регистр состояния процессора (PSW)** — флаги состояния.

**Регистр-счётчик команд** — хранит адрес текущей команды, параллельная запись и постинкремент.

**Регистр-указатель стека** — хранит адрес в специальной зоне памяти — стеке. Постинкремент и предекремент.

**Вопрос 27. Методы ускорения работы памяти. Принцип работы стека.**

**МЕТОДЫ УСКОРЕНИЯ РАБОТЫ ПАМЯТИ**

- Уменьшение внутренних временных задержек в памяти — совершенствование технологии;

- Использование статической оперативной памяти вместо динамической — только в небольших микропроцессорных системах (дороже) ;

- Добавление небольшой быстрой статической памяти к большой медленной динамической — кэш-память;

- Использование копии содержимого постоянной памяти в оперативной памяти;

- Оптимизация структуры модулей памяти и способов обмена с модулями памяти.

**ОСОБЫЕ ОБЛАСТИ ПАМЯТИ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ**

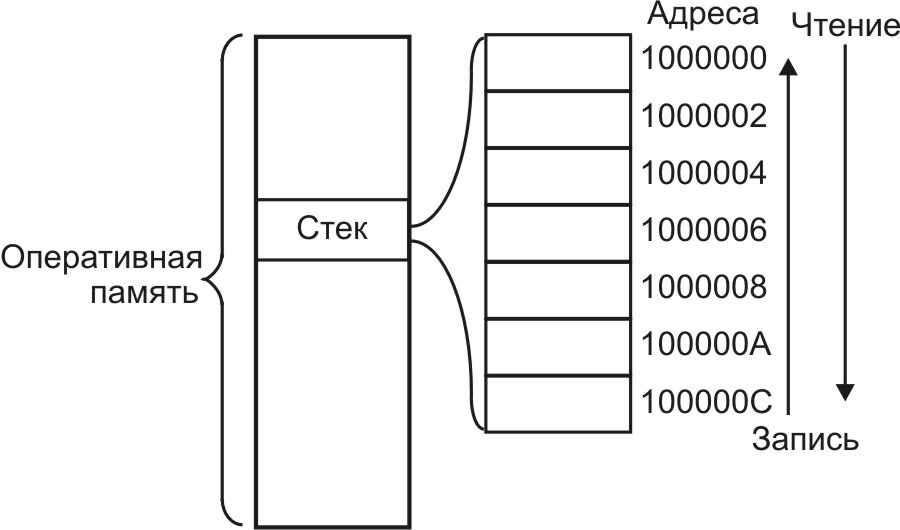
- Память программы начального запуска (ROM) — содержит программу, которая выполняется при включении питания или при подаче сигнала сброса;

- Память для стека или стек — используется для временного хранения данных в режиме LIFO. Необходима при обслуживании прерываний и при работе подпрограмм.

- Память с таблицей векторов прерываний — содержит список начальных адресов программ обработки прерываний.

- Память устройств ввода/вывода (УВВ) — даёт возможность процессору общаться с внутренней памятью УВВ как со своей собственной.

**ПРИНЦИП РАБОТЫ СТЕКА**

****

**Вопрос 28. Механизм обработки прерывания.**

**МЕХАНИЗМ ОБРАБОТКИ ПРЕРЫВАНИЯ**

Механизм прерываний реализуется аппаратно-программными средствами. Структуры систем прерывания (в зависимости от аппаратной архитектуры) могут быть самыми разными, но все они имеют одну общую особенность – прерывание непременно влечет за собой изменение порядка выполнения команд процессором.

Механизм обработки прерываний независимо от архитектуры вычислительной системы включает следующие элементы:

1. Установление факта прерывания (прием сигнала на прерывание) и идентификация прерывания (в операционных системах иногда осуществляется повторно, на шаге 4).

2. Запоминание состояния прерванного процесса. Состояние процесса определяется, прежде всего, значением счетчика команд, содержимым регистров процессора и может включать также спецификацию режима (например, режим пользовательский или привилегированный) и другую информацию.

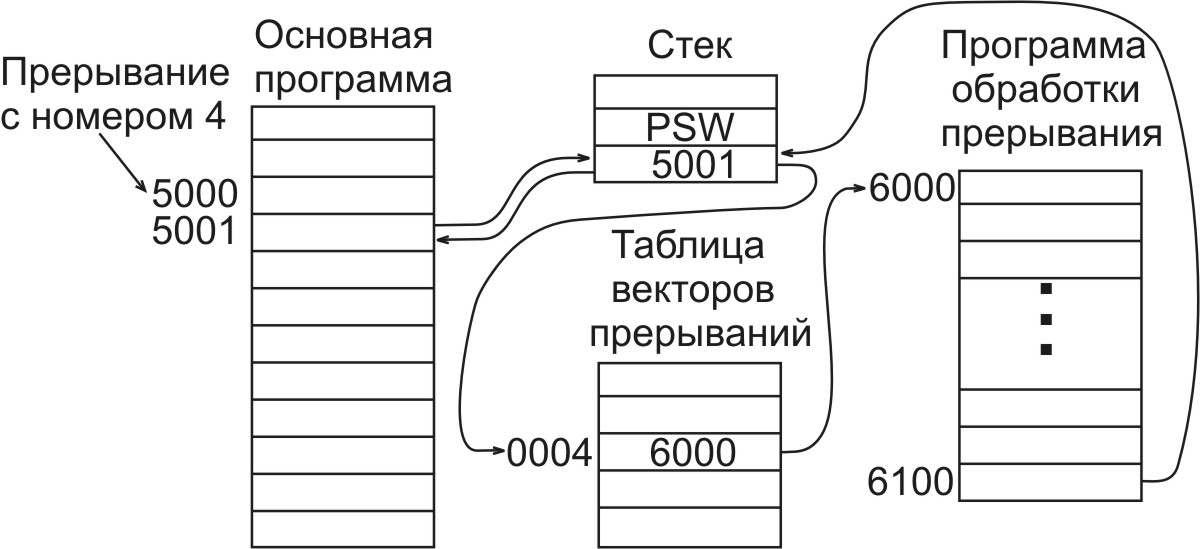
3. Управление аппаратно передаётся подпрограмме обработки прерывания.

4. Сохранение информации о прерванной программе, которую не удалось спасти на шаге 2 с помощью действий аппаратуры. В некоторых вычислительных системах предусматривается запоминание довольно большого объёма информации о состоянии прерванного процесса.

5. Обработка прерывания. Эта работа может быть выполнена той же подпрограммой, которой было передано управление на шаге 3, но в ОС чаще всего она реализуется путем последующего вызова соответствующей подпрограммы.

6. Восстановление информации, относящейся к прерванному процессу (этап, обратный шагу 4).

7. Возврат в прерванную программу.

****

Итак, главные функции механизма прерываний:

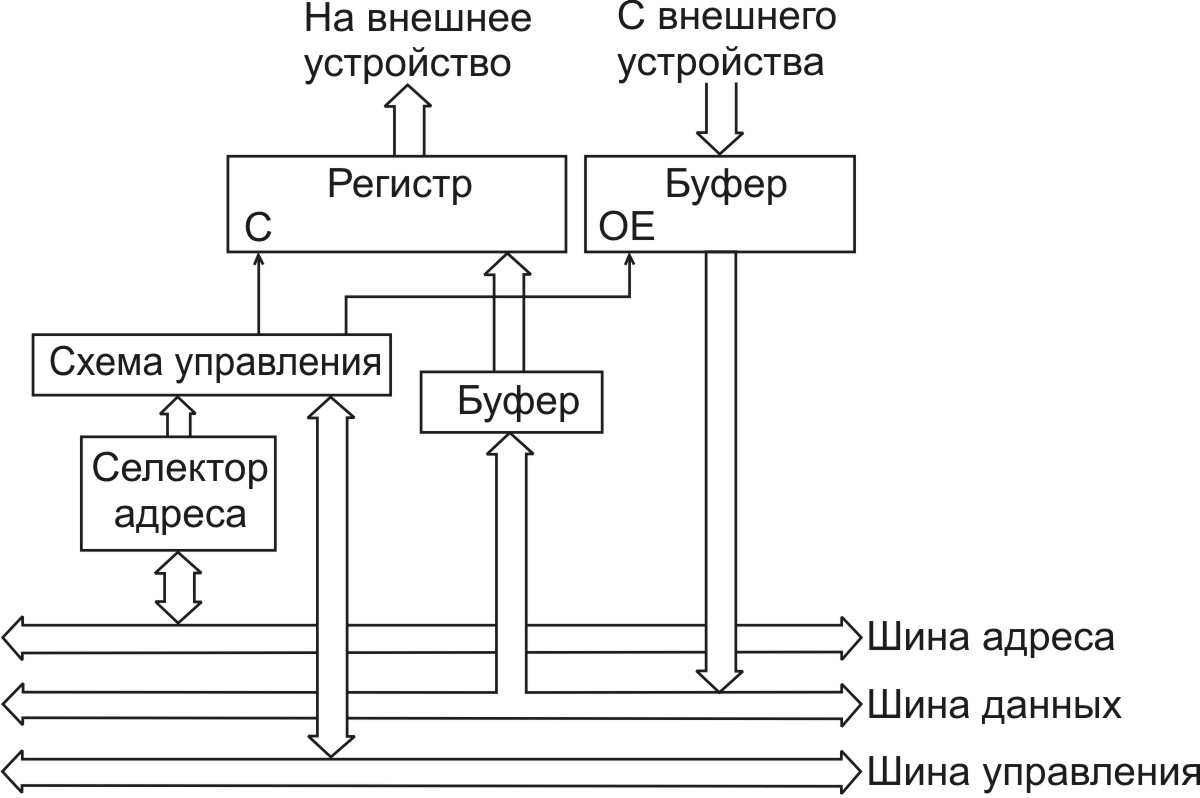
- распознавание или классификация прерываний;

- передача управления соответственно обработчику прерываний;

- корректное возвращение к прерванной программе.

**Вопрос 29. Адресные пространства памяти и устройства ввода/вывода.**

**Структура устройства ввода/вывода**



* Общее (разделённое) адресное пространство памяти и УВВ — часть адресов отводится под память, часть под УВВ. Общие стробы обмена. Процессор может обращаться к памяти и УВВ совершенно одинаково, используя те же команды — удобно. Но уменьшается адресное пространство памяти, сложнее ПДП (медленнее).
* Отдельные адресные пространства для памяти и УВВ. Разные стробы обмена для УВВ и для памяти. Специальные команды обмена с УВВ (ввод и вывод), отличные от команд обмена с памятью. Не уменьшается адресное пространство памяти, проще организовать ПДП (быстрее).

Под вводом–выводом в ЭВМ понимают передачу данных из периферийного устройства в оперативную память или наоборот. В идеале архитектура ввода–вывода должна обеспечивать обмен информацией между процессором и периферийными устройствами без снижения производительности процессора. Различают три вида архитектуры ввода–вывода (рис.3). Поскольку наиболее распространённым является второй вариант архитектуры, в дальнейшем организация ввода–вывода рассматривается применительно к ней.



Рисунок 3. Виды ввода/вывода

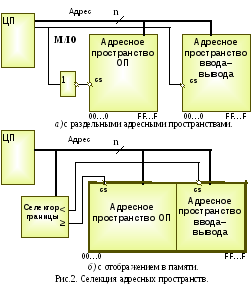
На первый взгляд между первой и второй архитектурами нет принципиальной разницы, поскольку периферийные устройства (ПУ) обмениваются данными с оперативной памятью через регистры процессора. Но в структуре с общей шиной имеется возможность организовать пересылку данных минуя процессор, напрямую, по тракту оперативная память – ПУ. Такой способ получил название примой доступ к памяти (ПДП).

## Адресное пространство ввода–вывода

Существует два способа организации адресного пространства ЭВМ – *с раздельными адресными пространствами* оперативной памяти и ввода–вывода и с одним, общим адресным пространством. Второй вариант получил название *с отображением в памяти*.

В первом случае адресные линии оперативной памяти и ввода–вывода являются общими, а разделение этих пространств адресов осуществляется например, сигналом http://www.studfiles.ru/html/4018/180/html_R56V39wAiJ.STee/img-Swpkb_.png, формируемым устройством управления. (Лучше использовать сигналы MR,MW,IOR,OIW). Принцип разделения адресных пространств показан на рис.2*а*. При обращении к ОП сигнал http://www.studfiles.ru/html/4018/180/html_R56V39wAiJ.STee/img-_GwR7c.png, а при исполнении команд ввода-вывода равен 0. Такая организация адресного пространства требует наличия в системе команд раздельных команд для обращения к ОП и по вводу-выводу. Размеры адресных пространств одинаковы и равны 2n, где n–разрядность адреса.

Во втором случае процессор имеет одно, разделённое на две неравные части адресное пространство. Меньшее по размеру пространство старших адресов используется для В/В. Размер этого пространства обычно составляют 4 – 8 Кбайт. Определение принадлежности адреса тому или иному пространству осуществляется специальной схемой – компаратором границы, как это показано на рис.2 *б*. Особенностью данного варианта является отсутствие команд вводы – вывода в системе команд процессора. Для обращения и к памяти, и к периферийным устройствам используются команды типа MOVE.

Технология ввода – вывода с отображением в памяти применяется в большинстве компьютерных систем. Некоторые процессоры для выполнения операций ввода – вывода имеют специальные команды типа IN и OUT. Вообще-то эти команды позволяют только записать или прочитать содержимое регистров периферийного устройства. Например, процессоры семейства Intel имеют специальные команды ввода – вывода и отдельное адресное пространство ввода – вывода. Такие операции называются *изолированным вводом–выводом*.

**Вопрос 30. Основные типы устройства ввода/вывода. Структура модуля памяти.**

**ОСНОВНЫЕ ТИПЫ УСТРОЙСТВ ВВОДА/ВЫВОДА**

- Устройства интерфейса пользователя (ввод — клавиатура, мышь, джойстик; вывод — дисплей, индикаторы);

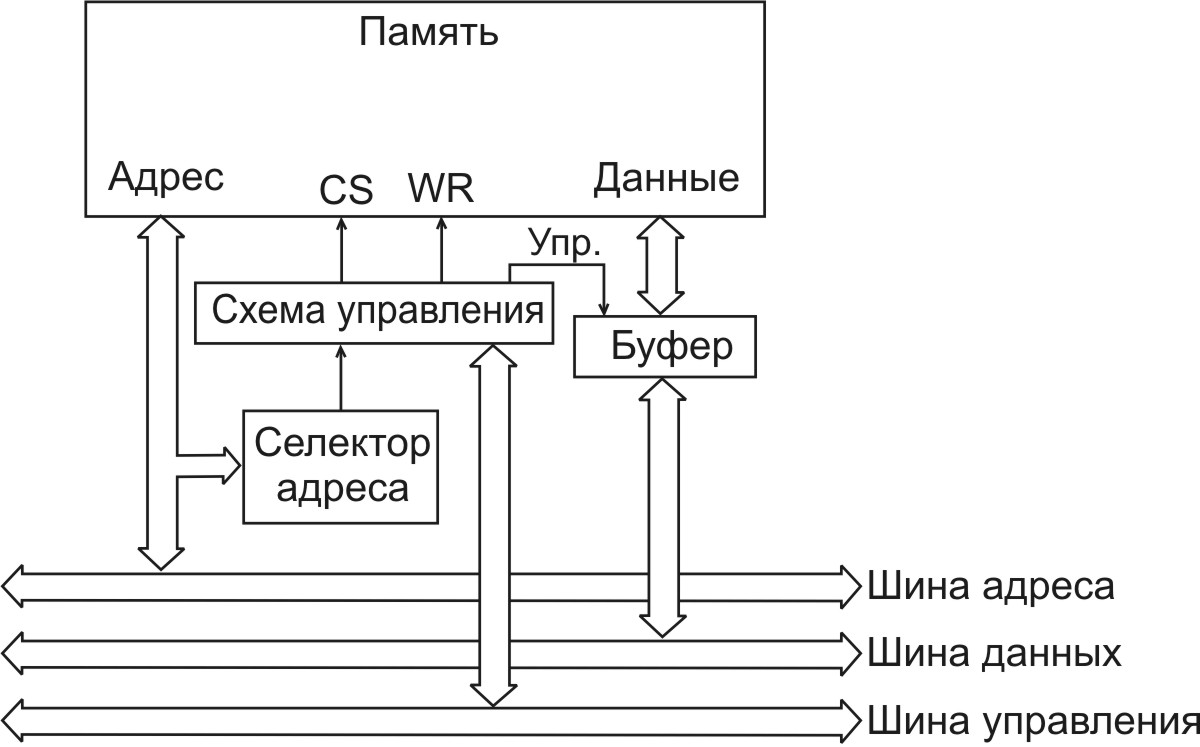
- Звуковые устройства (ввод — микрофон, линейный вход; вывод — динамик, линейный выход);

- Устройства долговременного хранения информации (диски) — в простейших системах отсутствуют;

- Таймерные устройства — могут не иметь выхода наружу, но необходимы для контроля времени (часы, интервалы);

- Контроллеры связных интерфейсов — USB, локальная сеть, Wi-Fi — для связи с удалёнными внешними устройствами и другими микропроцессорными системами.

**СТРУКТУРА МОДУЛЯ ПАМЯТИ**

****

**Вопрос 31. Термин «ассемблер» и особенности использования.**

**Ассемблер** представляет собой машинный язык в символической форме и включает набор инструкций, более понятных и удобных для человека, чем машинные коды.

Набор инструкций включает директивы, команды, макрокоманды и строки комментариев.

Исходная программа на языке ассемблера представляет собой последовательность операторов.

Операторы обычно занимают одну строку. Ассемблер воспринимает операторы в свободном формате.

**Операторы**-аргументы. **Tasm.exe** – компилятор**. Dosbox** – эмулятор.

**Директивы** – указания процессору на этапе компиляции.

**Идентификаторы** – набор букв, цифр и символов, не начинающихся с цифры. (может содержать от 1 до 31 символа. Остальные игнорируются)

Посредством идентификаторов представляются следующие объекты программы:

* Переменные
* Метки
* Имена

**Переменная** – единица программных данных, имеющая символическое имя (хранится в памяти).

Все переменные имеют 3 атрибута:

1.Сегмент, соответствующий тому сегменту, который ассемблировался, когда была определена переменная.

2.Смещение, являющееся смещением данного поля памяти относительно начала сегмента.

3.Тип, определенное число байтов, подвергающихся манипуляции при работе с переменной.

**Метка** представляет собой символическое имя для адреса ячейки памяти и предназначена для использования в качестве операнда в командах управления (перехода). Тип метки 2 байта.

Имеет 2 атрибута:

1. Сегмент
2. Смещение

EQU – директива для определения метки через другую метку.

**Именами** считаются символы, определённые директивой EQU и имеющие значением символ или число.

**Зарезервированные слова** – некоторые идентификаторы, называющиеся ключевыми словами, имеют фиксированный смысл и должны употребляться только в соответствии с:

- директивами

- инструкциями процессора

- именами регистров

- операторами выражения

**Вопрос 32. Особенности процессора Intel х86. Оперативная память.**

**Оперативная память – память к которой процессор имеет доступ по шине адреса.**

**Оперативная память состоит из отдельных ячеек, в каждой из которых хранится объем данных размером в 1 байт (до 256 значений).**

**Слова - два соединенных байта в памяти (до 65536 значений) - записывается задом наперед**

**Адрес слова – адрес младшего байта**

*Систему команд процессора х86 образуют 113 базовых команд, многие из которых допускают использование разнообразных режимов адресации.*

*По функциональному назначению выделяют следующие группы команд:*

1. *передачи данных*
2. *арифметические операции*
3. *логических операций и сдвигов*
4. *передачи управления*
5. *цепочечные*
6. *управление микропроцессором*

*Процессор – обработчик и вычислитель, выполняющий все операции над кодами и сигналами.*

**Вопрос 33. Сегментированной модель памяти**

Память для программы делится на непрерывные области памяти, называемые сегментами.

**Сегменты** - это логические элементы программы.

Сама программа может обращаться только к данным, которые находятся в этих сегментах.

Сегмент представляет собой независимый, поддерживаемый на аппаратном уровне блок памяти.

**Сегментация -** механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния.

**Замечание**. Программист может либо самостоятельно разбивать программу на фрагменты (сегменты), либо автоматизировать этот процесс и возложить его на систему программирования.

 Для микропроцессоров Intel принят особый подход к управлению памятью. Каждая программа в общем случае может состоять из любого количества сегментов, но непосредственный доступ она имеет только к 3 основным сегментам: кода, данных и стека и к дополнительным сегментам данных (всего 3).

Операционная система (! а не сама программа) размещает сегменты программы в ОП по определенным физическим адресам, а значения этих адресов записывает в определенные места, в зависимости от режима работы микропроцессора:

·       в реальном режиме адреса помещаются непосредственно в сегментные регистры (cs, ds, ss, es, gs, fs);

·       в защищенном режиме - в специальную системную дескрипторную таблицу (Элементом дескрипторной таблицы является дескриптор сегмента. Каждый сегмент имеет дескриптор сегмента -8 байт. Существует три дескрипторные таблицы. Адрес каждой таблицы записывается в специальный системный регистр).

  Для доступа к данным внутри сегмента обращение производится относительно начала сегмента линейно, т.е. начиная с 0 и заканчивая адресом, равным размеру сегмента. Этот адрес называется смещением (offset).

 Таким образом, для обращения к конкретному физическому адресу ОП необходимо определить адрес начала сегмента и смещение внутри сегмента.

Физический адрес принято записывать парой этих значений, разделенных двоеточием

                segment : offset

Например, 0040:001Ch;  000:041Ch;   0020:021Ch;  0041:000Ch.

  Каждый сегмент описывается дескриптором сегмента.

ОС строит для каждого исполняемого процесса соответствующую таблицу дескрипторов сегментов и при размещении каждого из сегментов в ОП или внешней памяти в дескрипторе отмечает его текущее местоположение (бит присутствия).

Дескриптор содержит поле адреса, с которого сегмент начинается и поле длины сегмента. Благодаря этому можно осуществлять контроль

1)     размещения сегментов без наложения друг на друга

2)     обращается ли код исполняющейся задачи за пределы текущего сегмента.

В дескрипторе содержатся также данные о правах доступа к сегменту (запрет на модификацию, можно ли его предоставлять другой задаче) Þ защита.

Достоинства:

1)     общий объем виртуальной памяти превосходит объем физической памяти

2)     возможность размещать в памяти как можно больше задач (до определенного предела) Þ увеличивает загрузку системы и более эффективно используются ресурсы системы

Недостатки:

1)     увеличивается время на доступ к искомой ячейке памяти, т.к. должны вначале прочитать дескриптор сегмента, а потом уже, используя его данные, можно вычислить физический адрес (для уменьшения этих потерь используется кэширование - дескрипторы, с которыми работа идет в данный момент размещаются в сверхоперативной памяти - в специальных регистрах процессора);

2)     фрагментация;

3)     потери памяти на размещение дескрипторных таблиц

4)     потери процессорного времени на обработку дескрипторных таблиц.

 Сегментированная модель памяти поддерживается и в реальном, и в защищенном режимах работы микропроцессора.

**Вопрос 34. Регистры общего назначения.**

**Регистр** – это память специального назначения, имеющаяся в распоряжении процессора и служащая для кратковременного хранения данных.

Регистры общего назначения предназначены для хранения операндов арифметико-логических инструкций, а также адресов или отдельных компонентов адресов ячеек памяти.

В микропроцессоре [8086](http://ru.osdev.wikia.com/wiki/8086) было восемь 16-разрядных регистров общего назначения. Все они могли выступать в качестве операндов основных арифметико - логических инструкций, но только четыре годились для целей адресации. Кроме того, каждый регистр имел свои специфические функции:

1.AX — аккумулятор. Использовался для хранения операндов в командах умножения и деления, ввода-вывода, в некоторых командах обработки строк и других операциях;

2.BX — регистр базы. Используется для хранения адреса или части адреса операнда, находящегося в памяти;

3.CX — счётчик. Содержит количество повторений строковых операций, циклов и сдвигов;

4.DX — регистр данных. Используется для косвенной адресации портов ввода-вывода, а также как «расширитель» аккумулятора в операциях удвоенной разрядности;

*5.SI — регистр адреса источника. Используется в строковых операциях, а также в качестве индексного регистра при обращении к операндам в памяти;*

*6.DI — регистр адреса приёмника. Используется в строковых операциях, а также в качестве индексного регистра при обращении к операндам в памяти;*

*7.BP — указатель кадра стека. Используется для адресации операндов, расположенных в стеке;*

*8.SP — указатель стека. Используется при выполнении операций со стеком, но не для явной адресации операндов в стеке.*

Первые четыре регистра могут делиться на две однобайтовых части каждый: AH, BH, CH и DH для старших байтов и AL, BL, CL и DL для младших байтов.

**Вопрос 35. Сегментные регистры**

В программной модели микропроцессора имеется шесть сегментных регистров: cs, ss, ds, es, gs, fs. Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых сегментами. Соответственно, такая организация памяти называется **сегментной**.

*Сегментные регистры используются при сегментной адресации памяти. При этом память делится на блоки (сегменты) различной длины. Адрес нужной ячейки памяти определяется адресом начала блока и величиной сдвига относительно него. Всего сегментных регистров четыре: для кодового сегмента, сегмента данных, сегмента стека и дополнительного сегмента*

Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены сегментные регистры. Фактически, с небольшой поправкой, как мы увидим далее, в этих регистрах содержатся адреса памяти с которых начинаются соответствующие сегменты. Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку неявно используются адреса во вполне определенных сегментных регистрах. Микропроцессор поддерживает следующие типы сегментов:

**1.Сегмент кода**. Содержит команды программы. Для доступа к этому сегменту служит регистр **cs** (code segment register) — сегментный регистр кода. Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (то есть эти команды загружаются в конвейер микропроцессора).

**2. Сегмент данных**. Содержит обрабатываемые программой данные. Для доступа к этому сегменту служит регистр **ds** (data segment register) — сегментный регистр данных, который хранит адрес сегмента данных текущей программы.

**3. Сегмент стека**. Этот сегмент представляет собой область памяти, называемую стеком. Работу со стеком микропроцессор организует по следующему принципу: *последний записанный в эту область элемент выбирается первым*. Для доступа к этому сегменту служит регистр **ss** (stack segment register) — сегментный регистр стека, содержащий адрес сегмента стека.

**4.Дополнительный сегмент данных**. Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре ds. Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре ds, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в регистрах **es, gs, fs** (extension data segment registers).

**Вопрос 36. Регистры указателей**

Регистры указателей также, как и сегментные регистры, предназначены для хранения составных частей адресов.

Как уже упоминалось, регистр IP содержит смещение в сегменте кодов для следующей команды, подлежащей исполнению. При выполнении каждой команды процессор изменяет значение в IP, поэтому этот регистр всегда указывает на следующую команду. Обычно в программах к IP не обращаются непосредственно, но его текущее значение может быть использовано в некоторых командах или директивах.

Регистр SP хранит значение смещения, которое вместе с регистром SS указывает на текущее слово в стеке.

Регистр BP обрабатывает ссылочные параметры (данные и адреса), которые передаются между программой и подпрограммой (процедурой) через стек. Также, как и другие регистры указателей, ВР содержит смещение. Процессор сочетает это смещение с адресом в SS.

Регистры индексов SI и DI используются для индексированной адресации, а также в строковых операциях.

Также как для вычисления адреса исполняемой команды процессору требуются значения, находящиеся в двух регистрах CS и IP, аналогично осуществляется доступ к данным в других сегментах.

Для доступа к сегменту данных процессор извлекает номер блока из регистра DS, а смещение из регистраВХ или индексного регистра- SI илиDI.

Следует отметить, что регистр ВХ – единственный из РОН – можно использовать в качестве указателя (индекса) для расширения адресации. С это целью он может использоваться в сочетании с регистрами SI и DI.

**SP** − регистр-указатель стека, его содержимое указывает адрес элемента на вершине стека

**BP** − регистр-указатель базы, дополнительный указатель при работе со стеком, базово-индексная адресация

**IP** − указатель команд, указывает адрес следующей команды в сегменте кода программы, при выполнении программы его содержимое изменяется автоматически

**Вопрос 37. Индексные регистры**

**Индексных регистров** два – это **индекс** источника и **индекс** приемника.

Могут использоваться для хранения временных переменных, но только тогда когда не используются по основному назначению. Хранят индексы исходных и целевых элементов массива или строки.

**SI** − индексный регистр источника, указатель адреса строки (массива) - источника из сегмента данных, базово-индексная адресация

**DI** − индексный регистр назначения, индексный регистр источника, указатель адреса строки (массива) – приемника из дополнительного сегмента, базово-индексная адресация

**Вопрос 38. Регистры флагов**

**Регистр флагов** — регистр процессора, отражающий текущее состояние процессора.

* CF (Carry Flag) – флаг переноса. Устанавливается в 1, если результат предыдущей операции не уместился в приемнике и произошел перенос из старшего бита или если требуется заем (при вычитании), иначе устанавливается в 0. Например, после сложения слова 0FFFFh и 1, если регистр, в который надо поместить результат, — слово, в него будет записано 0000h и флаг CF = 1.
* PF (Parity Flag) – флаг четности. Устанавливается в 1, если младший байт результата предыдущей команды содержит четное число бит, равных 1; устанавливается в 0, если число единичных бит нечетное. (Это не то же самое, что делимость на два. Число делится на два без остатка, если его самый младший бит равен нулю, и не делится, если он равен 1). Флаг PF был введён для совместимости с другими микропроцессорными архитектурами и по прямому назначению используется редко.
* AF (Auxiliary Carry Flag) – флаг полупереноса или вспомогательныйфлаг переноса. Устанавливается в 1, если в результате предыдущей операции произошел перенос (или заем) из третьего бита в четвертый. Этот флаг используется автоматически командами двоично-десятичной коррекции.
* ZF (Zero Flag) – флаг нуля. Устанавливается в 1, если результат предыдущей команды - ноль.
* SF (Sign Flag) – флаг знака. Этот флаг всегда равен старшему биту результата.
* TF (Trap Flag) – флаг ловушки (трассировки). Этот флаг был предусмотрен для работы отладчиков, не использующих защищенный режим. Установка его в 1 приводит к тому, что перед выполнением каждой командыгенерируется исключение #DB(прерывание INT 01h) передающее управление отладчику.
* IF (Interrupt Enable Flag) – флаг разрешенияпрерываний. Сброс этого флага в 0 приводит к тому, что процессор перестает обрабатывать прерывания от внешних устройств. Обычно его устанавливают на короткое время для выполнения критических участков кода.
* DF (Direction Flag) – флаг направления. Этот флаг контролирует поведение команд обработки строк – когда он установлен в 1, строки обрабатываются в сторону уменьшения адресов, а когда DF = 0 – наоборот.
* OF (Overflow Flag) – флаг переполнения. Этот флаг устанавливается в 1, если результат предыдущей арифметической операции над числами со знаком выходит за допустимые для них пределы. Например, если при сложении двух положительных чисел получается число со старшим битом, равным единице (то есть отрицательное) и наоборот.

**Вопрос 39. Регистровая адресация.**

**При регистровой адресации данные расположены в любом из регистров общего назначения.**

Регистровая адресация (mov AX,BX) при которой микропроцессор извлекает операнд из регистра и помещает результат в регистр.

Регистровая адресация предполагает, что операнд (входной или выходной) находится во внутреннем регистре процессора. Например, команда может состоять в том, чтобы переслать число из нулевого регистра в первый. Номера обоих регистров (0 и 1) будут определяться кодом команды пересылки.

**Вопрос 40. Прямая адресация.**

**В качестве адресного параметра адрес сегмента и смещение относительно его в начале**

**По умолчанию при передаче данных используется регистр DS (адрес сегмента данных)**

Прямая адресация - в инструкции процессора закодирован адрес операнда.

Это простейший вид адресации операнда в памяти, так как эффективный адрес содержится в самой команде и для его формирования не используется никаких дополнительных источников или регистров. Эффективный адрес берется непосредственно из поля смещения машинной команды, которое может иметь размер 8, 16, 32 бит. Это значение однозначно определяет байт, слово или двойное слово, расположенные в сегменте данных.

Прямая адресация может быть двух типов.

Относительная прямая адресация

Используется для команд условных переходов, для указания относительного адреса перехода. Относительность такого перехода заключается в том, что в поле смещения машинной команды содержится 8-, 16– или 32-битное значение, которое в результате работы команды будет складываться с содержимым регистра указателя команд ip/eip. В результате такого сложения получается адрес, по которому и осуществляется переход.

Абсолютная прямая адресация

В этом случае эффективный адрес является частью машинной команды, но формируется этот адрес только из значения поля смещения в команде. Для формирования физического адреса операнда в памяти микропроцессор складывает это поле со сдвинутым на 4 бита значением сегментного регистра. В команде ассемблера можно использовать несколько форм такой адресации.

Но такая адресация применяется редко – обычно используемым ячейкам в программе присваиваются символические имена. В процессе трансляции ассемблер вычисляет и подставляет значения смещений этих имен в формируемую им машинную команду в поле «смещение в команде». В итоге получается так, что машинная команда прямо адресует свой операнд, имея, фактически, в одном из своих полей значение эффективного адреса.

Остальные виды адресации относятся к косвенным. Слово «косвенный» в названии этих видов адресации означает то, что в самой команде может находиться лишь часть эффективного адреса, а остальные его компоненты находятся в регистрах, на которые указывают своим содержимым байт modr/m и, возможно, байт sib.

**Вопрос 41. Непосредственная адресация.**

При непосредственной адресации значение операнда является частью машинной команды. Понятно, что в этом случае операнд представляет собой константу. Примеры:

|  |
| --- |
| mov al,5  add bx,1234h  mov dx,a |

Обратите внимание, что в третьей строке в DX помещается адрес метки или переменной a, а вовсе не значение по этому адресу. Это особенность синтаксиса FASM. По сути, адрес метки тоже является числовой константой.

**Вопрос 42. Косвенная регистровая (базовая) адресация**

Адрес операнда находится в одном из регистров BX, SI или DI. Примеры:

|  |
| --- |
| add ax,[bx]  mov dl,[si] |

Размер операнда в памяти здесь определяется размером первого операнда. Так как AX — 16-разрядный регистр, то из памяти берётся слово по адресу в BX. Так как DL — 8-разрядный регистр, то из памяти берётся байт по адресу в SI. Это правило верно и для других режимов адресации.

**Вопрос 43. Косвенная базовая индексная адресация.**

Адрес операнда вычисляется как сумма содержимого одного из базовых регистров BX или BP и одного из индексных регистров SI или DI. Примеры:

|  |
| --- |
| mov ax,[bp+si]  add ax,[bx+di] |

Например, в одном из регистров может находиться адрес начала массива в памяти, а в другом — смещение какого-то элемента относительно начала.

**Вопрос 44. Понятие прерывания. Классификация.**

Прерывание – это событие, при котором происходит приостановка основной программы и переход на выполнение подпрограммы прерывания.

**В зависимости от источника различают прерывания:**

* **Аппаратные** (внешние) – реакция процессора на физический сигнал от некоторого устройства. Возникают в случайные моменты времени, а значит – асинхронные.
* **Программные** (внутренние) – возникает в заранее запланированный момент времени — синхронные.
* **Исключения** – разновидность программных прерываний, реакция процессора на некоторую не стандартную ситуацию возникшую во время выполнения команды.

Вызов прерывания осуществляется с помощью директивы **INT номер\_прерывания**.

**Вопрос 45. Внешние прерывания.**

**Аппаратные** (внешние) – реакция процессора на физический сигнал от некоторого устройства. Возникают в случайные моменты времени, а значит – асинхронные.

**Пример:**Это может быть сигнал по прошествию определенного промежутка времени, нажатие клавиши, переход принтера в состояние готовности.

**Вопрос 46. Внутренние прерывания.**

**Программные** (внутренние) – возникает в заранее запланированный момент времени — синхронные.

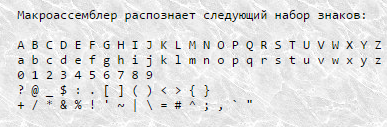
Пример:  
При выполнении операции деления на ноль или при попытке выполнить несуществующую команду. За каждым из таких прерываний закреплен определенный вектор, номер которого известен процессору. Например, за делением на 0 закреплен вектор 0, а за неправильной командой - вектор 6. Если процессор сталкивается с одной из таких ситуаций, он выполняет описанную выше процедуру прерывания, используя закрепленный за этой ситуацией вектор прерывания.

**Вопрос 47. Таблица векторов прерываний.**

|  |  |
| --- | --- |
| **Номер** | **Описание** |
| 0 | Ошибка деления. Вызывается автоматически после выполнения команд DIV или IDIV, если в результате деления происходит переполнение (например, при делении на 0). Обычно при обработке этого прерывания MS-DOS выводит сообщение об ошибке и останавливает выполнение программы. При этом для процессора i8086 адрес возврата указывает на команду, следующую после команды деления, а для процессора i80286 и более старших моделей - на первый байт команды, вызвавшей прерывание |
| 1 | Прерывание пошагового режима. Вырабатывается после выполнения каждой машинной команды, если в слове флагов установлен бит пошаговой трассировки TF. Используется для отладки программ. Это прерывание не вырабатывается после пересылки данных в сегментные регистры командами MOV и POP |
| 2 | Аппаратное немаскируемое прерывание. Это прерывание может использоваться по-разному в разных машинах. Обычно оно вырабатывается при ошибке четности в оперативной памяти и при запросе прерывания от сопроцессора |
| 3 | Прерывание для трассировки. Генерируется при выполнении однобайтовой машинной команды с кодом CCh и обычно используется отладчиками для установки точки прерывания |
| 4 | Переполнение. Генерируется машинной командой INTO , если установлен флаг переполнения OF. Если флаг не установлен, команда INTO выполняется как NOP. Это прерывание используется для обработки ошибок при выполнении арифметических операций |
| 5 | Печать копии экрана. Генерируется, если пользователь нажал клавишу <PrtSc>. В программах MS-DOS обычно используется для печати образа экрана. Для процессора i80286 и более старших моделей генерируется при выполнении машинной команды BOUND, если проверяемое значение вышло за пределы заданного диапазона |
| 6 | Неопределенный код операции или длина команды больше 10 байт |
| 7 | Особый случай отсутствия арифметического сопроцессора |
| 8 | IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду |
| 9 | IRQ1 - прерывание от клавиатуры. Генерируется, когда пользователь нажимает и отжимает клавиши. Используется для чтения данных из клавиатуры |
| A | IRQ2 - используется для каскадирования аппаратных прерываний |
| B | IRQ3 - прерывание асинхронного порта COM2 |
| C | IRQ4 - прерывание асинхронного порта COM1 |
| D | IRQ5 - прерывание от контроллера жесткого диска (только для компьютеров IBM PC/XT) |
| E | IRQ6 - прерывание генерируется контроллером НГМД после завершения операции ввода/вывода |
| F | IRQ7 - прерывание от параллельного адаптера. Генерируется, когда подключенный к адаптеру принтер готов к выполнению очередной операции. Обычно не используется |
| 10 | Обслуживание видеоадаптера |
| 11 | Определение конфигурации устройств в системе |
| 12 | Определение размера оперативной памяти |
| 13 | Обслуживание дисковой системы |
| 14 | Работа с асинхронным последовательным адаптером |
| 15 | Расширенный сервис |
| 16 | Обслуживание клавиатуры |
| 17 | Обслуживание принтера |
| 18 | Запуск BASIC в ПЗУ, если он есть |
| 19 | Перезагрузка операционной системы |
| 1A | Обслуживание часов |
| 1B | Обработчик прерывания, возникающего, если пользователь нажал комбинацию клавиш <Ctrl+Break> |
| 1C | Программное прерывание, вызывается 18,2 раза в секунду обработчиком аппаратного прерывания таймера |
| 1D | Адрес видеотаблицы для контроллера видеоадаптера 6845 |
| 1E | Указатель на таблицу параметров дискеты |
| 1F | Указатель на графическую таблицу для символов с кодами ASCII 128-255 |
| 20-5F | Используется MS-DOS или зарезервировано для MS-DOS |
| 60-67 | Прерывания, зарезервированные для программ пользователя |
| 68-6F | Не используются |
| 70 | IRQ8 - прерывание от часов реального времени |
| 71 | IRQ9 - прерывание от контроллера EGA |
| 72 | IRQ10 - зарезервировано |
| 73 | IRQ11 - зарезервировано |
| 74 | IRQ12 - зарезервировано |
| 75 | IRQ13 - прерывание от арифметического сопроцессора |
| 76 | IRQ14 - прерывание от контроллера жесткого диска |
| 77 | IRQ15 - зарезервировано |
| 78 - 7F | Не используются |
| 80-85 | Зарезервировано для BASIC |
| 86-F0 | Используются интерпретатором BASIC |
| F1-FF | Не используются |

**Вопрос 48. Элементы языка Ассемблер. Алфавит, идентификаторы.**

Алфавит:



Идентификаторы:

Конструкции языка ассемблера формируются из идентификаторов и ограничителей. Идентификатор  представляет  собой  набор букв, цифр и символов   \_  ,  ?,  $  или  @,  не  начинающийся  с  цифры.

Идентификатор должен полностью размещаться на одной строке и может содержать от 1 до  31  символа  (точнее,  используются только    первые   31   символ   идентификатора,   остальные игнорируются) . Следует заметить, что  символ  \_  ,  хотя  и допускается    в    качестве    компоненты   идентификатора, ассемблером игнорируется. Поэтому, например,  идентификаторы A\_B и AB совпадают.  Друг от друга идентификаторы  отделяются  пробелом или ограничителем,  которым  считается  любой   недопустимый   в идентификаторе символ.

**Вопрос 49. Элементы языка Ассемблер. Переменные, имена, метки.**

Посредством  идентификаторов  представляются следующие объекты программы:

               1. Переменные.

               2. Метки.

               3. Имена.

**Переменные**    идентифицируют   хранящиеся   в   памяти данные. Все переменные имеют три атрибута:

               1. СЕГМЕНТ,  соответствующий  тому  сегменту,  который

                  ассемблировался, когда была определена переменная.

               2. СМЕЩЕНИЕ,  являющееся смещением данного поля памяти

                  относительно начала сегмента.

               3. ТИП,  определяющий  число  байтов,   подвергающихся

                  манипуляциям при работе с переменной.

**Метка** является   частным  случаем  переменной,  когда известно, что определяемая ею память содержит машинный  код. На  нее  можно ссылаться посредством скачков (переходов) или вызовов. Метка имеет два атрибута: СЕГМЕНТ и СМЕЩЕНИЕ. Метка также  может  быть  определена через другую метку директивой EQU.

**Именами** считаются символы, определенные директивой EQU и имеющие значением  символ  или  число.  Значения  имен  не фиксированы  в  процессе  ассемблирования, но при выполнении программы именам соответствуют константы. С переменными дело обстоит   наоборот   (если   значением   переменной  считать содержимое идентифицируемой ею области памяти).

Примеры имен:

         REGS   EQU   AX

         DIGIT  EQU   18

**Вопрос 50. Элементы языка Ассемблер. Зарезервированные слова.**

Некоторые   идентификаторы,    называемые    ключевыми словами, имеют фиксированный смысл  и  должны  употребляться только в соответствии с этим. Ключевыми словами являются:

1. Директивы ассемблера.
2. Инструкции процессора.
3. Имена регистров.
4. Операторы выражений.

**Вопрос 51. Элементы языка Ассемблер. Типы данных.**

В этом разделе описываются типы и формы  представления данных,   которые  могут  быть  использованы  в  выражениях, директивах и инструкциях языка ассемблера.

***Целые числа.***

Синтаксис:

цифры

               цифрыB

               цифрыQ

               цифрыO

               цифрыD

               цифрыH

Латинский символ, который может кодироваться на  обоих регистрах, задает основание счисления числа: B - двоичное, Q и O  -  8-ричное,  D  -  десятичное,  H  -  16-ричное. 16-ричные  числа  не  должны  начинаться  с  буквенных 16-ричных  цифр  (например, вместо некорректного ABh следует употреблять  0ABh).  16-ричные  цифры  от  A  до   F   могут кодироваться на обоих регистрах.

Первая  форма  целого  числа  использует  умалчиваемое основание, которое может устанавливаться директивой  .RADIX. Если директива .RADIX не используется, полагается D.

***Действительные числа.***

Синтаксис:

цифрыR

[[+|-]]цифры.цифры[[E[[+|-]]цифры]]

Первый    формат    задает    16-ричное   кодированное

представление содержимого памяти, содержащей  действительное число.     Действительное     число    может    кодироваться последовательностью из 8, 12 или 20 16-ричных цифр (возможно добавление  предшествующих  0,  т.к.  первой   должна   быть

десятичная   цифра).  Каждое  число  имеет  знак,  зависимую экспоненту и мантиссу, которые хранятся как поля бит в самом числе.  Точный  размер и смысл каждого бита зависит от числа

бит в действительном числе.   Кодированные действительные числа могут использоваться в директивах DD, DQ и DT. Число 16-ричных цифр должно быть 8 для  DD,  16  для DQ и 20 для DT (с предшествующим 0 - 9, 17 или 21 соответственно).  Вторым  форматом  задается  действительное   число   с плавающей точкой. Цифры должны быть десятичными. Такие числа могут использоваться в директивах DD, DQ и DT.

***Десятичные числа.***

Синтаксис:

[[+|-]]цифры

Этот вид соответствует упакованному формату десятичных чисел,  который включает в себя 1 знаковый байт и 9 значимых байтов.  Каждый  значимый  байт содержит 2 десятичные цифры.

Допустимо кодирование до 18 включительно цифр.  Старший  бит знакового байта имеет значение 0 для положительных чисел и 1 - для отрицательных.

Упакованные   десятичные    числа    имеют    то    же представление,  что  и  целые  десятичные числа (п.2.3.1) за исключением того, что они могут иметь знак и  использоваться только в директиве DT.

***Знаковые и строковые константы.***

Синтаксис:

'знаки'

"знаки"

Знаковая  константа  состоит  из одного знака алфавита языка.  Строковая  константа  включает  в  себя  2 или более знака.  В  отличие  от  других  компонент   языка,   строковые константы чувствительны к регистру.

Символы  '  и  "  в теле константы должны кодироваться дважды.

***Предложения.***

Синтаксис:

[[имя]] мнемоника [[операнды]][[;комментарии]]

Предложения определяют структуру и функции  программы. Они  могут  начинаться  с любой позиции и содержать не более 128 символов.

Предложение не может содержать внутри себя  комбинации символов   CARRIAGE-RETURN/LINE-FEED,  т.е.  не  может  быть продолжено на другую строку. Однако, все предложения,  кроме последнего, должны заканчиваться этой комбинацией символов.

Все  предложения языка ассемблера делятся на директивы ассемблера и инструкции процессора. Директивы   ассемблера   действуют   лишь   в   период компиляции  программы  и  позволяют   устанавливать   режимы компиляции,   задавать   структуру   сегментации  программы, определять  содержимое  полей  данных,   управлять   печатью листинга программы, а также обеспечивают условную компиляцию и  некоторые другие функции. В результате обработки директив макроассемблером объектный код не генерируется.

***Комментарии.***

Синтаксис:  
;комментарий

Часть строки исходного текста после символа ; (если он не  является элементом знаковой константы или строки знаков) считается комментарием и ассемблером игнорируется. Комментарии вносятся в программу как поясняющий  текст и  могут  содержать  любые  знаки  до  ближайшей  комбинации символов CARRIAGE-RETURN/LINE-FEED, т.е. до конца строки.  Для создания комментариев, занимающих несколько строк, может быть использована директива COMMENT.

**Вопрос 52. Синтаксис предложений, команд и директив Ассемблера.**

Предложения:

       Синтаксис:

               [[имя]] мнемоника [[операнды]][[;комментарии]]  
Команды:

       Синтаксис:

               [метка:][префикс] мнемоника [[операнд1, операнд 2]][[;комментарии]]

Директивы:

       Синтаксис:

               [[имя]] директива [[операнды]][[;комментарии]]

**Вопрос 53. Трансляция, компоновка и отладка программы**

Процесс подготовки и отладки программы на языке ассемблера включает этапы подготовки файла с исходным текстом, его трансляции и компоновки и, наконец, отладки программы с помощью специальной программы интерактивного отладчика

**Трансляция** исходного текста программы состоит в преобразовании предложений исходного языка в коды машинных команд и выполняется с помощью транслятора с языка ассемблера (т. е. с помощью программы ассемблера).

**Компоновка** объектного файла выполняется с помощью программы компоновщика (редактора связей). Эта программа получила такое название потому, что ее основное назначение - подсоединение к файлу с основной программой файлов с подпрограммами и настройка связей между ними.

**Отладка** готовой программы может выполняться разными методами, выбор которых определяется структурой и функциями отлаживаемой программы.

**Вопрос 54. Директивы определения моделей памяти.**

Модели памяти задаются директивой .MODEL

.model модель,язык,модификатор

где модель — одно из следующих слов:

TINY — код, данные и стек размещаются в одном и том же сегменте размером до 64 Кб. Эта модель памяти чаще всего используется при написании на ассемблере небольших программ;

SMALL — код размещается в одном сегменте, а данные и стек — в другом (для их описания могут применяться разные сегменты, но объединенные в одну группу). Эту модель памяти также удобно использовать для создания программ на ассемблере;

COMPACT — код размещается в одном сегменте, а для хранения данных могут использоваться несколько сегментов, так что для обращения к данным требуется указывать сегмент и смещение (данные дальнего типа);

MEDIUM — код размещается в нескольких сегментах, а все данные — в одном, поэтому для доступа к данным используется только смещение, а вызовы подпрограмм применяют команды дальнего вызова процедуры;

LARGE и HUGE — и код, и данные могут занимать несколько сегментов;

FLAT — то же, что и TINY, но используются 32-битные сегменты, так что максимальный размер сегмента, содержащего и данные, и код, и стек, — 4 Мб.

**Вопрос 55. Директивы задания набора допустимых команд.**

.8086 — используется по умолчанию. Разрешены только команды 8086;

.186 — разрешены команды 80186;

.286 и .286c — разрешены непривилегированные команды 80286;

.286p — разрешены все команды 80286;

.386 и .386c — разрешены непривилегированные команды 80386;

.386p — разрешены все команды 80386;

.486 и .486c — разрешены непривилегированные команды 80486;

.486p — разрешены все команды 80486;

.586 и .586c — разрешены непривилегированные команды P5 (Pentium);

.586p — разрешены все команды P5 (Pentium);

.686 — разрешены непривилегированные команды P6 (Pentium Pro, Pentium II);

.686p — разрешены все команды P6 (Pentium Pro, Pentium II);

.8087 — разрешены команды NPX 8087;

.287 — разрешены команды NPX 80287;

.387 — разрешены команды NPX 80387;

.487 — разрешены команды FPU 80486;

.587 — разрешены команды FPU 80586;

.MMX — разрешены команды IA MMX;

.K3D — разрешены команды AMD 3D.

**Вопрос 56. Глобальные объявления.**

public    язык метка...       ;  Для TASM и MASM.  
public   метка язык...      ; Для WASM.  
Метка, объявленная директивой PUBLIC, становится доступной для других модулей программы. Так, можно объявлять имена процедур, переменные и кон­станты, определенные директивой EQU. Необязательный операнд языка (С, PASCAL, BASIC, FORTRAN, SYSCALL или STDCALL) указывает, что метка будет вызываться из модуля, написанного на соответствующем языке, и при не­обходимости изменяет ее (например, добавляет \_ перед первым символом метки).

**Вопрос 57.**  Условное ассемблирование дает возможность при трансляции обходить тот или иной участок программы. Существует три вида условного ассемблирования.

а)

IF выражение

...

ENDIF

б) IF выражение

...

ELSE

...

ENDIF

в) IF выражение 1

...

ELSEIF выражение 2

...

ELSEIF выражение 3

...

ELSE

...

ENDIF

Условие считается не выполненным, если выражение принимает значение 0 и выполненным, если выражение отлично от нуля.

2. Ассемблеры MASM и TASM поддерживают также несколько условных специальных директив, назовем некоторые из них.

а) IFE выражение

...

ELSEIFE

...

ENDIFE

б) Операторы IF1 и IF2 проверяют первый и второй проход при ассемблировании.

в) Оператор IFDEF - проверяет, определено ли в программе символическое имя, IFDEFN - обратный оператор. И другие IF операторы. Они имеются в любом справочнике по ассемблеру.

г) Имеется целый набор директив, начинающихся с .ERR. Например, .ERRE выражение - вызовет прекращение трансляции и сообщение об ошибке, если выражение станет равным 0.

Условное ассемблирование понадобится нам в конце главы для написания программы, транслируемой как в MASM, так и TASM.

**Вопрос 58. Основные группы команд.**

1. Команды передачи данных

dst – получатель

src – источник

mem адрес памяти

reg – регистр процессора

sreg – сегментный регистр

data – непосредственные данные

MOV – команда пересылки данных (копирование)

1) одним из операндов должен бать обязательно регистр

2) нельзя пересылать из одной ячейки памяти в другую

3) нельзя содержимое одного сегментного регистра переслать в другой сегментный регистр (нужно использовать РОНы или STACK)

4) нельзя использовать сегментный регистр CS в качестве приемника (потеряются коды)

5) ОПЕРАНДЫ ДОЛЖНЫ СОАПАДАТЬ ПО ДЛИНЕ

Команды ввода/вывода из портов: IN AL,42H

OUT AX,71H

1) LEA AX,mem (аналогично mov AX,OFFSET(mem))

2) Push, Pop

2. Команды арифметической обработки целых чисел.

1) Команды сложения:

а) ADD – сложение mem/reg1, reg2/mem (1\_операнд+2\_операнд=1\_операнд)

dst:=dst+src;

б) INC – увеличение на 1

2) Команды вычитания:

а) DEC – уменьшение на 1

б) SUB dst,src (dst:=dst-src)

в) SBB dst,src – вычитание с переносом

г) NEG – изменяет знак на противоположный

3) Команды сравнения:

а) CMP dst,src

б) SUB dst,src (сравнение осуществляется путем вычитания источника из приемника (воздействует на флаги: CF, OF, SF, ZF, AF, PF)

4) Умножние:

а) MUL src – команда умножения для беззнаковых чисел (выполняет умножение операнда на регистр AX)

б) IMUL src – для знаковых чисел (результат может изменять знаки)

5) Деление:

а) DIV <делитель> (делимое должно находиться в AX!; частное попадает в AL, а остаток в AH)

б) IDIV – деление со знаком (целочисленное)

3. Логические операции (выполняются побитно).

AND, OR, XOR, NOT, TEST dst,src

**Вопрос 59.** Одно из самых мощных языковых средств ассемблера — макроопределения. Макроопределением (или макросом) называется участок программы, которому присвоено имя и который ассемблируется всякий раз, когда ассемблер встречает это имя в тексте программы. Макрос начинается директивой MACRO и заканчивается ENDM. Например: пусть описано макроопределение hex2ascii, переводящее шестнадцатеричное число, находящееся в регистре AL, в ASCII-код соответствующей шестнадцатеричной цифры:

hex2ascii macro

cmp al,10

sbb al,69h

das

endm

Теперь в программе можно использовать слово hex2ascii, как если бы это было имя команды, и ассемблер заменит каждое такое слово на три команды, содержащиеся в макроопределении. Разумеется, можно оформить этот же участок кода в виде процедуры и вызывать его командой CALL — если процедура вызывается больше одного раза, этот вариант программы займет меньше места, но вариант с макроопределением станет выполняться быстрее, так как в нем не будет лишних команд CALL и RET. Однако скорость выполнения — не главное преимущество макросов. В отличие от процедур макроопределения могут вызываться с параметрами, следовательно, в зависимости от ситуации, включаемый код будет немного различаться.

Теперь можно использовать S\_MOV вместо команды MOV для того, чтобы скопировать значение из одного сегментного регистра в другой.

Следующее важное средство, использующееся в макроопределениях, — директивы условного ассемблирования. Например: напишем макрос, выполняющий умножение регистра AX на число, причем, если множитель — степень двойки, то умножение будет выполняться более быстрой командой сдвига влево.

fast\_mul macro number

if number eq 2

shl ax,1 ; Умножение на 2

elseif number eq 4

shl ax,2 ; Умножение на 4

elseif number eq 8

shl ax,3 ; Умножение на 8

... ; Аналогично вплоть до:

elseif number eq 32768

shl ax,15 ; Умножение на 32768

else

mov dx,number ; Умножение на число, не являющееся

mul dx ; степенью двойки.

endif

endm

Можно, конечно, усложнить этот макрос, применяя особые свойства команды LEA и ее комбинации, сдвиги и сложения, однако в нынешнем виде он чрезмерно громоздкий. Проблема решается с помощью третьего средства, постоянно использующегося в макросах, — блоков повторений.

**Вопрос 60. Переменные и константы**

Инициализированные переменные

Для определения переменных в Ассемблере используются соответствующие директивы, которые резервируют в памяти определенное пространство и записывают туда значение. Понятие «тип данных» здесь применимо очень условно. В большинстве случаев интерпретация значения той или иной области памяти зависит от программиста. Вот эти директивы:

DB - (1 байт )

DW - (2 байта) - слово

DD - (4 байта) – двойное слово

DQ - (8 байт) – четверное слово

DT - (10 байт)

Они могут использоваться различными способами (правда, DQ и DT не допускают в качестве операндов числовые или строковые константы):

p1 db 0x55 ; просто байт 0x55

p2 db 0x55,0x56,0x57 ; последовательно 3 байта

p3 db 'a',0x55 ; символьная константа

p4 db 'hello',13,10,'$' ; это строковая константа

p5 dw 0x1234 ; 0x34 0x12

p6 dw 'a' ; 0x41 0x00 (это просто число)

p7 dw 'ab' ; 0x41 0x42 (символьная константа)

p8 dw 'abc' ; 0x41 0x42 0x43 0x00 (строка)

p9 dd 0x12345678 ; 0x78 0x56 0x34 0x12

p10 dd 1.234567e20 ; константа с плавающей точкой

p11 dq 1.234567e20 ; двойной точности

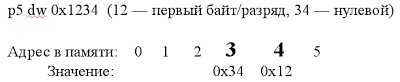
p12 dt 1.234567e20 ; расширенной точности

В программах на Ассемблере переменные и константы (инициализированные переменные) описываются в отдельной секции .data. Формат записи:

<метка\имя переменной> <директива> <значение>

Метка - сущность времени написания исходного кода, и в бинарном коде она заменяется адресом памяти, на который указывает. Для задач отладки метки могут сохраняться в соответствующей секции исполняемого файла.

Согласно принципа «младший байт по младшему адресу» (little-endian) данные длиной более одного байта непосредственно в памяти «переворачиваются». Например, пусть двухбайтовая переменная p5 размещена в памяти в ячейках с номерами 3 и 4 (p5 указывает на ячейку 3), таким образом, что нулевой байт этой переменной запишется в ячейку 3, а первый - в 4:

[](http://2.bp.blogspot.com/--z40dtZK38g/UDjn8j8lJ5I/AAAAAAAAAOY/admd-8k4IlY/s1600/07little_e.png)

Неинициализированные переменные

Кроме создания инициализированных переменных, NASM позволяет резервировать области памяти без первоначальной инициализации (часто это большие области памяти, например, предназначенные для загрузки файлов). Такие данные размещаются в специальной секции .bss. Директивы, которые используются для определения неинициализиванных переменных:

RESB

RESW

RESD

RESQ

REST

Каждая директива принимает один операнд, являющийся числом резервируемых байт, слов, двойных слов и т.д: например:

buffer: resb 64 ; резервирование 64 байт

wordvar: resw 1 ; резервирование слова

realarray resq 10 ; массив из 10 чисел с плавающей точкой

Определение констант

Директива EQU вводит символ для указанного константного значения: если используется EQU, в этой строке кода должна присутствовать метка. Смысл EQU — связать имя метки со значением ее (только) операнда. Данное определение абсолютно и не может быть позднее изменено. Например:

message db 'Привет!'

msglen equ $-message

определяет msglen как константу. Значение msglen не может быть позднее переопределено. Символ $ - это обозначение текущей ячейки памяти.

Типы констант

NASM знает четыре различных типа констант: числовые, символьные, строковые и с плавающей точкой.

Числовая константа — это просто число. NASM позволят определять числа в различных системах счисления и различными способами: вы можете использовать суффиксы H, Q и B для шестнадцатеричных, восьмеричных и двоичных чисел соответственно; можете использовать для шестнадцатеричных чисел префикс 0х в стиле С, а также префикс $ в стиле Borland Pascal. Однако имейте в виду, что префикс $ может быть также префиксом идентификаторов, поэтому первой цифрой шестнадцатеричного числа при использовании этого префикса должна быть обязательно цифра, а не буква.

mov ax,100 ; десятичная

mov ax,0a2h ; шестнадцатеричная

mov ax,$0a2 ; снова hex: нужен 0

mov ax,0xa2 ; опять hex

mov ax,777q ; восьмеричная

mov ax,10010011b ; двоичная

Символьная константа содержит от одного до четырех символов, заключенных в одиночные или двойные кавычки. Тип кавычек для NASM несущественен, поэтому если используются одинарные кавычки, двойные могут выступать в роли символа и, соответственно, наоборот. Символьная константа, содержащая более одного символа, будет загружаться в обратном порядке следования байт: если вы пишете

mov eax,'abcd'

сгенерированной константой будет не 0x61626364, а 0x6463626.

Строковые константы допустимы только в некоторых псевдо-инструкциях, а именно в семействе DB и инструкции INCBIN. Строковые константы похожи на символьные, только длиннее. Они обрабатываются как сцепленные друг с другом символьные константы. Так, например, следующие строки кода эквивалентны.

db 'hello' ; строковая константа

db 'h','e','l','l','o' ; эквивалент из символьных констант

Следующие строки также эквивалентны:

dd 'ninechars' ; строковая константа в двойное слово

dd 'nine','char','s' ; три двойных слова

db 'ninechars',0,0,0 ; и действительно похоже

Константы с плавающей точкой допустимы только в качестве аргументов DD, DQ и DT. Выражаются они традиционно: цифры, затем точка, затем возможно цифры после точки, и наконец, необязательная Е с последующей степенью. Точка обязательна, т.к. dd 1 NASM воспримет как объявление целой константы, в то время как dd 1.0 будет воспринята им правильно.

dd 1.2 ; "простое" число

dq 1.e10 ; 10,000,000,000

dq 1.e+10 ; синоним 1.e10

dq 1.e-10 ; 0.000 000 000 1

dt 3.141592653589793238462 ; число pi

В процессе компиляции NASM не может проводить вычисления над константами с плавающей точкой (это сделано с целью переносимости).

**Вопрос 61. Команды передачи данных разделяют на 4 подгруппы:**

- общие команды передачи данных;

- стековые команды;

- команды ввода-вывода;

- команды передачи цепочек байт или слов.

Команды передачи данных не модифицируют состояния флажков. Исключение составляют команды POPF и SAHF, прямо воздействующие на регистр флажков.

Далее будут использоваться следующие обозначения: dst  - получатель,

src        -        источник,

mem     -    адрес памяти (смещение), заданный любым методом адресации,

reg        -             регистр общего назначения,

sreg      -             сегментный регистр,

aс         -             регистр-аккумулятор (AL или AX),

data      -             непосредственные данные.

**Вопрос 62. XCHG     (eXCHanGe)  Обмен**

Для двунаправленной пересылки данных применяют команду xchg. Для этой операции можно, конечно, применить последовательность из нескольких команд mov, но из-за того, что операция обмена используется довольно часто, разработчики системы команд микропроцессора посчитали нужным ввести отдельную команду обмена xchg.

Схема команды: xchg операнд\_1,операнд\_2

Назначение: обмен двух значений между регистрами или между регистрами и памятью.

Алгоритм работы:  обмен содержимого операнд\_1 и операнд\_2.

Применение:

Команду xchg можно использовать для выполнения операции обмена двух операндов с целью изменения порядка следования байт, слов, двойных слов или их временного сохранения в регистре или памяти. Альтернативой является использование для этой цели стека.

!! операнды должны иметь один тип.

!! Не допускается (как и для всех команд ассемблера) обменивать между собой содержимое двух ячеек памяти.

Пример 1

        xchg    ax,bx                        ;обменять содержимое регистров ax и bx

        xchg    ax,word ptr [si]        ;обменять содержимое регистра ax

                                                   ;и слова в памяти по адресу в [si]

Пример 2

;поменять порядок следования байт в слове

ch1     label   byte

        dw      0f85ch

...

        mov     al,ch1

        xchg    ch1+1,al

        mov     ch1,al

**Вопрос 63. КОМАНДЫ ВВОДА-ВЫВОДА**

Команда ввода IN и команда вывода OUT допускают работу как с байтами, так и со словами. Команда IN загружает данные из задан­ного порта в аккумулятор, а команда OUT выполняет передачу из аккумулятора в порт. Для портов ввода-вывода в диапазоне 00-FF можно использовать прямую укороченную адресацию, а остальные порты в диапазоне 100-FFFF можно адресовать только косвенно через регистр DX.

Формат команд:

IN ac,port                              OUT port,ac                              (прямая укороченная адресация)

IN ac,DX                              OUT DX,ac                              (косвенная адресация)

**Вопрос 64. КОМАНДЫ СЛОЖЕНИЯ**

Команда ADD

Команда ADD позволяет производить сложение 8- или 16-битовых двоичных чисел в режиме регистр-регистр, регистр-память и память­регистр, причем адресация памяти осуществляется в любом допусти­мом режиме. Общее представление команды имеет вид

ADD dst, src

т.е. первый  операнд  складывается со вторым и результат операции

замещает первый операнд.

Формат команды:

ADD mem/reg1,mem/reg2

ADD mem/reg,data

Команда ADC

Команда ADC выполняет сложение с переносом: в отличие от команды ADD в операции сложения участвует флажок CF, значение которого прибавляется к младшему биту результата сложения операн­дов.

Формат команды:

ADC mem/reg1,mem/reg2

ADC mem/reg,data

Команда INC

Команда INC позволяет увеличить на 1 содержимое любого обще­го регистра или ячейки памяти.

Формат команды:

INC mem/reg

**КОМАНДЫ ВЫЧИТАНИЯ**

Команда SUB позволяет производить вычитание 8- или 16-битных двоичных чисел. Общее представление команды имеет вид

SUB dst, src

т.е. второй операнд вычитается из первого  и  результат  операции

замещает первый операнд.

Формат команды:

SUB mem/reg1,mem/reg2

SUB mem/reg,data

Команда SBB

Команда SBB выполняет вычитание с переносом: в отличие от команды SUB в операции вычитания участвует флажок CF, значение которого вычитается из младшего бита результата вычитания операн­дов.

Формат команды:

SBB mem/reg1,mem/reg2

SBB mem/reg,data

Команда DEC

Команда DEC позволяет уменьшить на 1 содержимое любого обще­го регистра или ячейки памяти.

Формат команды:

DEC mem/reg

Команда NEG

Команда NEG изменяет знак числа, находящегося в регистре или ячейке памяти, на противоположный.

Формат команды:

NEG mem/reg

**Вопрос 65. КОМАНДЫ УМНОЖЕНИЯ**

Микропроцессор 8086 имеет две команды умножения: для беззна­ковых и для знаковых двоичных чисел. Умножение десятичных чисел требует использования специальных команд коррекции, которые будут рассматриваться позднее.

Команда MUL

Команда умножения беззнаковых целых чисел MUL выполняет умножение адресуемого операнда на содержимое аккумулятора. Общее представление команды имеет вид

MUL src

При операции над байтами функции аккумулятора выполняет регистр AL, а 16-битный результат операции помещается в регистр AX. При операции над словами функции аккумулятора выполняет регистр AX, а произведение длиной 32 бита формируется в регистрах DX (старшее слово) и AX (младшее слово).

Формат команды:

MUL reg

MUL mem

Команда IMUL

Команда IMUL аналогична команде MUL, но сомножители и произ­ведение интерпретируются как знаковые двоичные числа в дополни­тельном коде.

Формат команды:

IMUL reg

IMUL mem

**КОМАНДЫ ДЕЛЕНИЯ**

Микропроцессор 8086 имеет две команды деления: для беззнако­вых и для знаковых двоичных чисел. Деление десятичных чисел также требует использования специальных команд коррекции.

Команда DIV

Команда деления беззнаковых чисел DIV производит деление содержимого аккумулятора и его расширения на содержимое адресуе­мого операнда.

При делении 16-битного делимого на 8-битный делитель делимое помещают в регистр AX. В результате выполнения операции частное формируется в регистре AL, а остаток - в AH.

При делении 32-битного делимого на 16-битный делитель стар­шая часть делимого помещается в регистр DX, а младшая - в AX. В результате выполнения операции частное формируется в регистре AX, а остаток - в DX.

При делении на 0 автоматически происходит прерывание и пере­ход к специальной программе обработки.

Формат команды:

DIV reg

DIV mem

Команда IDIV

Команда IDIV аналогична команде DIV, но делимое, делитель и частное интерпретируются как знаковые двоичные числа в дополни­тельном коде.

Формат команды:

IDIV reg

IDIV mem

**Вопрос 66.Команды преобразования**

Команда преобразования байта в слово CBW расширяет знак содержимого регистра AL в регистр AH. Команда преобразования сло­ва в двойное слово CWD передает знак содержимого регистра AX во все биты регистра DX.

Команды преобразования не влияют на состояния флагов.

Форматы команд:

CBW

CWD

**Вопрос 67. КОМАНДЫ ЛОГИЧЕСКИХ ОПЕРАЦИЙ**

Логические операции представлены командами NOT (инверсия), AND (конъюнкция), OR (дизъюнкция), XOR (исключающее ИЛИ) и коман­дой TEST, которая выполняет конъюнкцию операндов, но не изменяет их значений. Все логические операции являются поразрядными, т.е. выполняются независимо для всех бит операндов.

Бинарные команды AND, OR, XOR и TEST воздействуют на флажки OF, SF, ZF, PF и CF. Унарная операция NOT не влияет на состояние флажков.

Форматы команд:

AND mem/reg1,mem/reg2

AND mem/reg,data

OR mem/reg1,mem/reg2

OR mem/reg,data

XOR mem/reg1,mem/reg2

XOR mem/reg,data

TEST mem/reg1,mem/reg2

TEST mem/reg,data

NOT mem/reg

**Вопрос 68. Логические команды**

and операнд\_1,операнд\_2 — операция логического умножения.

Команда выполняет поразрядно логическую операцию И (конъюнкцию) над битами операндов операнд\_1 и операнд\_2. Результат записывается на место операнд\_1.

or операнд\_1,операнд\_2 — операция логического сложения.

Команда выполняет поразрядно логическую операцию ИЛИ (дизъюнкцию) над битами операндов операнд\_1 и операнд\_2. Результат записывается на место операнд\_1.

xor операнд\_1,операнд\_2 — операция логического исключающего сложения.

Команда выполняет поразрядно логическую операцию исключающего ИЛИ над битами операндов операнд\_1 и операнд\_2. Результат записывается на место операнд\_1.

test операнд\_1,операнд\_2 — операция “проверить” (способом логического умножения).

Команда выполняет поразрядно логическую операцию И над битами операндов операнд\_1 и операнд\_2. Состояние операндов остается прежним, изменяются только флаги zf, sf, и pf, что дает возможность анализировать состояние отдельных битов операнда без изменения их состояния.

not операнд — операция логического отрицания.

Команда выполняет поразрядное инвертирование (замену значения на обратное) каждого бита операнда. Результат записывается на место операнда.

С помощью логических команд возможно выделение отдельных битов в операнде с целью их установки, сброса, инвертирования или просто проверки на определенное значение.

Для организации подобной работы с битами операнд\_2 обычно играет роль маски. С помощью установленных в 1 битов этой маски и определяются нужные для конкретной операции биты операнд\_1.

**Вопрос 69. Команда логического исключения сложения**

xor операнд\_1,операнд\_2 — операция логического исключающего сложения.

Команда выполняет поразрядно логическую операцию исключающего ИЛИ над битами операндов операнд\_1 и операнд\_2. Результат записывается на место операнд\_1.

**Вопрос 70. Логический сдвиг операнда влево/вправо**

SHL операнд, количество\_сдвигов

SHR операнд, количество\_сдвигов

SHL и SHR сдвигают биты операнда (регистр/память) влево или вправо соответственно на один разряд и изменяют флаг переноса cf. При логическом сдвиге все биты равноправны, а освободившиеся биты заполняются нулями. Указанное действие повторяется количество раз, равное значению второго операнда.

Пример:

; al = 01011011 (двоичное)

shr al, 3

Это означает: сдвиг всех битов регистра al на 3 разряда вправо. Так что al станет 00001011. Биты слева заполняются нулями, а биты справа выдвигаются. Последний выдвинутый бит, становится значением флага переноса cf.

Команда shl такая же, как и shr, но сдвигает влево.

; bl = 11100101 (двоичное)

shl bl, 2

После выполнения команды регистр bl будет равен 10010100 (двоичное). Два последних бита заполнились нулями, флаг переноса установлен, потому, что последний выдвинутый слева бит был равен 1

**Арифметический сдвиг влево/вправо**

SAL операнд, количество\_сдвигов

SAR операнд, количество\_сдвигов

Команда SAR — сдвигает биты операнда (регистр/память) вправо на один разряд, значение последнего вытолкнутого бита попадает в флаг переноса, а освободившиеся биты заполняются знаковым битом.

Пример:

; al = 10100110

sar al, 3

; al = 11110100

sar al, 2

; al = 11111101

; bl = 00100110

sar bl, 3

; bl = 00000010

Команда SAL — сдвигает биты операнда (регистр/память) влево на один разряд, значение последнего вытолкнутого бита попадает в флаг переноса, а освободившиеся биты заполняются нулями, при этом знаковый бит не двигается.

Пример:

; al = 10100110

sal al, 3

; al = 10110000

sal al, 4

; al = 10000000

**Команды циклического сдвига**

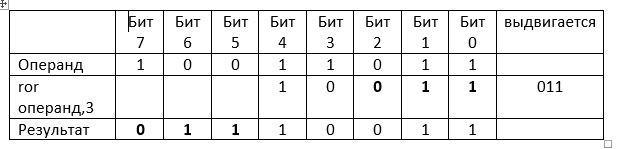
rol операнд, количество\_сдвигов

ror операнд, количество\_сдвигов

rcl операнд, количество\_сдвигов

rcr операнд, количество\_сдвигов

Циклический сдвиг напоминает смещение, выдвигаемые биты, снова вдвигаются с другой стороны:

Пример: команды ror (циклический сдвиг вправо)  


Из рисунка выше, биты вращаются, то есть каждый бит, который выталкивается снова вставляется с другой стороны. Флаг переноса cf содержит значение последнего выдвинутого бита.

ROL и ROR сдвигают все биты операнда влево(для ROL) или вправо(для ROR) на один разряд, при этом старший(для ROL) или младший(для ROR) бит операнда вдвигается в операнд справа(для ROL) или слева(для ROR) и становится значением младшего(для ROL) или старшего(для ROR) бита операнда; одновременно выдвигаемый бит становится значением флага переноса cf. Указанные действия повторяются количество раз, равное значению второго операнда.

RCL и RCR сдвигают все биты операнда влево (для RCL) или вправо (для RCR) на один разряд, при этом старший(для RCL) или младший(для RCR) бит становится значением флага переноса cf; одновременно старое значение флага переноса cf вдвигается в операнд справа(для RCL) или слева(для RCR) и становится значением младшего(для RCL) или старшего(для RCR) бита операнда. Указанные действия повторяются количество раз, равное значению второго операнда.

### Вопрос 71. Команды безусловного перехода.

Безусловный переход — это переход, который выполняется всегда. Безусловный переход осуществляется с помощью команды JMP. У этой команды один операнд, который может быть непосредственным адресом (меткой), регистром или ячейкой памяти, содержащей адрес.

Существуют также «дальние» переходы — между сегментами.

Примеры безусловных переходов:

**jmp** metka ;Переход на метку

**jmp** bx ;Переход по адресу в BX

**jmp** word[bx] ;Переход по адресу, содержащемуся в памяти по адресу в BX

Команды безусловного перехода JMP идентичны командам CALL по их возможностям адресации. Однако существует дополнительная команда перехода, указывающая однобайтовое смещение для близкого относительного перехода (команда короткого перехода). Соответствующей ей команды CALL не существует, так как вызовы подпрограмм, расположенных поблизости, происходят очень редко. Команды переходов используют те же методы генерации адреса, что и команды вызова.

Сделаем здесь замечание об оптимизации кода и о том, как работает ассемблер. По мере того, как ассемблер делает первый переход по тексту программы и назначает адреса командам, он должен решить, использовать двух- или трехбайтовую разновидность команды JMP. Если это переход назад, т.е. на место, уже известное ассемблеру, он может определить правильное смещение; тем самым ассемблер знает, находится ли переход в диапазоне короткого смещения. Однако, если переход делается вперед, на метку, о которой ассемблер еще не знает, он должен предположить, что метка находится далее, чем 128 байт от текущего места. Затем ассемблер порождает длинную форму команды перехода. Худший случай ассемблер обязан выбирать потому, что потом уже не может возвратиться назад и увеличить размер команды. Затем ассемблер заместит трехбайтовую команду перехода двухбайтовой командой JMP и однобайтовой командой NOP, если обнаружит, что переход делается ближе 128 байт от текущего места. Так как такой переход выполняется несколько быстрее, время выполнения в этом случае сокращается, но объектный код остается больше необходимого.

Если программисту заранее известно, что переход вперед делается на место, лежащее в диапазоне 128 байт от текущего места, он может об этом сообщить ассемблеру с помощью следующей строки:

**JMP SHORT LABEL**

Атрибут SHORT заставляет ассемблер сформировать короткую форму SHORT команды перехода, даже если он еще не встречал метку. Если же программист сделал ошибку и переход в действительности не может быть коротким, ассемблер выдает сообщение об ошибке.

### Вопрос 72. Команды условного перехода.

Для корректной обработки данных в ассемблере нужна разветвленность программы. Это достигается использованием условных переходов. Условный переход это такая команда процессору, при которой в зависимости от состояния регистра флагов производится передача управления по некоторому адресу иначе говоря прыжок. Этот адрес может быть ближним или дальним.

Прыжок считается ближним, если адрес, на который делается прыжок, находится не дальше чем 128 байт назад и 127 байт вперёд от следующей команды. Дальний прыжок это прыжок дальше, чем на [-128,127] байт. Прыгать просто так в любом месте программы бессмысленно. Обычно перед командой прыжка идёт команда сравнения, которая изменяет регистр флагов в зависимости от результата. Команд сравнения две: cmp и test. Наиболее универсальная cmp. **CMP** eax, ebx

Команда Cmp производит сравнение двух операндов. Она сравнивает два значения (регистр, память, непосредственное значение) и устанавливает флаг нуля Z (zeroflag) если они равны.

Пример самой распространённой команды условного перехода jz:

**CMP** eax, 42h ; Если регистр eax равен 42

**JZ** metka2 ; то управление передастся первой команде после метки metka2

Условные переходы могут делать только ближний прыжок, а команда jmp может делать и короткий и дальний прыжки.

Существует много команд для различных условных переходов. Также для некоторых команд есть синонимы (например, JZ и JE — это одно и то же). Для наглядности все команды условных переходов приведены в таблице:

|  |  |  |
| --- | --- | --- |
| **Команда** | **Переход, если** | **Условие перехода** |
| JZ/JE | нуль или равно | ZF=1 |
| JNZ/JNE | не нуль или не равно | ZF=0 |
| JC/JNAE/JB | есть переполнение/не выше и не равно/ниже | CF=1 |
| JNC/JAE/JNB | нет переполнения/выше или равно/не ниже | CF=0 |
| JP | число единичных бит чётное | PF=1 |
| JNP | число единичных бит нечётное | PF=0 |
| JS | знак равен 1 | SF=1 |
| JNS | знак равен 0 | SF=0 |
| JO | есть переполнение | OF=1 |
| JNO | нет переполнения | OF=0 |
| JA/JNBE | выше/не ниже и не равно | CF=0 и ZF=0 |
| JNA/JBE | не выше/ниже или равно | CF=1 или ZF=1 |
| JG/JNLE | больше/не меньше и не равно | ZF=0 и SF=OF |
| JGE/JNL | больше или равно/не меньше | SF=OF |
| JL/JNGE | меньше/не больше и не равно | SF≠OF |
| JLE/JNG | меньше или равно/не больше | ZF=1 или SF≠OF |
| JCXZ | содержимое CX равно нулю | CX=0 |

У всех этих команд один операнд — имя метки для перехода. Обратите внимание, что некоторые команды применяются для беззнаковых чисел, а другие — для чисел со знаком. Сравнения «выше» и «ниже» относятся к беззнаковым числам, а «больше» и «меньше» — к числам со знаком. Для беззнаковых чисел признаком переполнения будет флаг CF, а соответствующими командами перехода JC и JNC. Для чисел со знаком о переполнении можно судить по состоянию флага OF, поэтому им соответствуют команды перехода JO и JNO. Команды переходов не изменяют значения флагов.

### Вопрос 73. Команды управления циклом.

Для организации цикла предназначена команда LOOP. У этой команды один операнд — имя метки, на которую осуществляется переход. В качестве счётчика цикла используется регистр CX. Команда LOOP выполняет декремент CX, а затем проверяет его значение. Если содержимое CX не равно нулю, то осуществляется переход на метку, иначе управление переходит к следующей после LOOP команде.

Содержимое CX интерпретируется командой как число без знака. В CX нужно помещать число, равное требуемому количеству повторений цикла. Понятно, что максимально может быть 65535 повторений. Ещё одно ограничение связано с дальность перехода. Метка должна находиться в диапазоне -127…+128 байт от команды LOOP (если это не так, FASM сообщит об ошибке).

**Пример простого цикла.** В качестве примера я приведу простую программу, которая будет печатать все буквы английского алфавита. ASCII-коды этих символов расположены последовательно, поэтому можно выводить их в цикле. Для вывода символа на экран используется функция DOS 02h (выводимый байт должен находиться в регистре DL).

press **db** 13,10,'Press any key...$'

**mov** **ah**,02h *;Для вызова функции DOS 02h - вывод символа*

**mov** **dl**,'A' *;Первый выводимый символ*

**mov** **cx**,26 *;Счётчик повторений цикла*

metka:

**int** 21h *;Обращение к функции DOS*

**inc** **dl** *;Следующий символ*

**loop** metka *;Команда цикла*

**mov** **ah**,09h *;Функция DOS 09h - вывод строки*

**mov** **dx**,press *;В DX адрес строки*

**int** 21h *;Обращение к функции DOS*

**mov** **ah**,08h *;Функция DOS 08h - ввод символа без эха*

**int** 21h *;Обращение к функции DOS*

**mov** **ax**,4C00h *;\*

**int** 21h *;/ Завершение программы*

Команды «int 21h» и «inc dl» (строки 8 и 9) будут выполняться в цикле 26 раз. Для того, чтобы программа не закрылась сразу, используется функция DOS 08h — ввод символа с клавиатуры без эха, то есть вводимый символ не отображается. Перед этим выводится предложение нажать любую кнопку (но Reset лучше не нажимать). Для примера адрес строки объявлен с помощью метки. Символы с кодами 13 и 10 обозначают переход на следующую строку (символ 13(0Dh) называется CR — Carriage Return — возврат каретки, а символ 10(0Ah) LF — Line Feed — перевод строки . Эти символы унаследованы со времён древних телетайпов, когда текст печатался, как на печатной машинке).

**Результат**: ABCDEFGHIJKLMNOPQRSTUVWXYZ

**Вложенные циклы**

Иногда требуется организовать вложенный цикл, то есть цикл внутри другого цикла. В этом случае необходимо сохранить значение CX перед началом вложенного цикла и восстановить после его завершения (перед командой LOOP внешнего цикла). Сохранить значение можно в другой регистр, во временную переменную или в стек. Следующая программа выводит все доступные ASCII-символы в виде таблицы 16×16. Значение счётчика внешнего цикла сохраняется в регистре BX.

press **db** 13,10,'Press any key...$'

**mov** **ah**,02h *;Для вызова функции DOS 02h - вывод символа*

**sub** **dl**,**dl** *;Первый выводимый символ*

**mov** **cx**,16 *;Счётчик внешнего цикла (по строкам)*

lp1:

**mov** **bx**,**cx** *;Сохраняем счётчик в BX*

**mov** **cx**,16 *;Счётчик внутреннего цикла (по столбцам)*

lp2:

**int** 21h *;Обращение к функции DOS*

**inc** **dl** *;Следующий символ*

**loop** lp2 *;Команда внутреннего цикла*

**mov** **dh**,**dl** *;Сохраняем значение DL в DH*

**mov** **dl**,13 *;\*

**int** 21h *; \*

**mov** **dl**,10 *; / Переход на следующую строку*

**int** 21h *;/*

**mov** **dl**,**dh** *;Восстанавливаем значение DL*

**mov** **cx**,**bx** *;Восстанавливаем значение счётчика*

**loop** lp1 *;Команда внешнего цикла*

**mov** **ah**,09h *;Функция DOS 09h - вывод строки*

**mov** **dx**,press *;В DX адрес строки*

**int** 21h *;Обращение к функции DOS*

**mov** **ah**,08h *;Функция DOS 08h - ввод символа без эха*

**int** 21h *;Обращение к функции DOS*

**mov** **ax**,4C00h *;\*

**int** 21h *;/ Завершение программы*

### Вопрос 74. Команды прерываний.

Прерывание означает временное прекращение основного процесса вычислений для выполнения некоторых запланированных или незапланированных действий, вызванных работой устройств или программы.

Свое название прерывания получили из-за того, что при их выполнении программа передает управление в DOS или BIOS для совершения определенного действия и затем возвращает управление в прерванную программу. Необходимость прерываний обусловлена двумя основными причинами:

1) Преднамеренный запрос таких действий, как операции ввода-вывода на различные устройства

2) Непредвиденные программные ошибки (например, переполнение при делении)

В зависимости от источника различают прерывания:

***Аппаратные*** (внешние) – реакция процессора на физический сигнал от некоторого устройства. Возникают в случайные моменты времени, а значит – асинхронные

***Программные*** (внутренние) – возникает в заранее запланированный момент времени — синхронные

***Исключения*** – разновидность программных прерываний, реакция процессора на некоторую не стандартную ситуацию возникшую во время выполнения команды;

Вектор прерываний – адрес процедуры обработки прерываний. Адреса размещаются в специальной области памяти доступной для всех подпрограмм. Вектор прерывания одержит 4 байта: старшее слово содержит сегментную составляющую адреса процедуры обработки исключения, младшее — смещение.

Вызов прерывания осуществляется с помощью директивы INT номер\_прерывания.

00h – 1Fh – прерывания BIOS

20h – 3Fh – прерывания DOS

40h – 5Fh – зарезервировано

60h – 7Fh – прерывания пользователя

80h – FFh – прерывания Бейсика

BIOS

Система **BIOS** находится в **ROM** (ПЗУ) и управляет всеми прерываниями в системе. Начальный адрес **ROM - FFFF0h**. Напомним, что ее основное назначение (помимо размещения BIOS) - поддержка процедур начальной загрузки: выполнение различных проверок, загрузка данных с системного диска и т.д. Остановимся более подробно на порядке загрузки компьютера...

При включении компьютера процессор устанавливает в регистре **CS** значение **FFFFh**, а в регистре **IP - 0000h**. Таким образом, первая выполняемая команда находится по адресу **FFFFh:0000h** или **FFFF0h**, что является точкой входа в BIOS.   
BIOS проверяет различные порты компьютера для определения и инициализации подключенных устройств. Затем он создает в начале памяти по адресу **00000h** таблицу векторов прерываний, которая содержит адреса обработчиков прерываний, и выполняет две операции **INT 11h** (запрос списка присоединенного оборудования) и **INT 12h** (запрос размера физической памяти).

Операнд в команде прерывания, например, **10h** в команде **INT 10h**, содержит тип прерывания, являющийся идентификатором запроса. Для каждого прерывания система содержит адрес в таблице векторов прерываний.   
Так как в таблице имеется 256 четырехбайтовых элементов, то она занимает первые 1024 байта памяти (256\*4=1024, от **00000h до 003FFh**). Каждый элемент таблицы определяет адрес подпрограммы обработки указанного типа прерывания и содержит значения кодового сегмента и смещение, которые при прерывании устанавливаются в регистры **CS** и **IP** соответственно.

Прерывание заносит в стек содержимое флагового регистра, регистра **CS** и регистра **IP**. Например, для прерывания **10h** адрес элемента таблицы равен **0040h** (0010h\*0004h=0040h).   
Операция выделяет четырехбайтовый элемент по адресу **0040h** и заносит два байта в регистр **IP** и два байта в регистр **СS**. Адрес, который получается в регистровой паре **CS:IP**, представляет собой адрес начала подпрограммы в области BIOS, которая получает управление. Возврат из этой подпрограммы осуществляется командой **IRET** (Interrupt Return), которая восстанавливает флаги и регистры **CS** и **IP** из стека и передает управление на команду, следующую за выполненной командой прерывания.

DOS

Все прерывания делятся на две большие группы: **BIOS** и **DOS-прерывания**. Рассмотрим более подробно выполнение DOS-прерываний...   
Как было отмечено, при загрузке DOS в памяти компьютера располагаются два файла: IO.SYS и MSDOS.SYS, которые обеспечивают, в том числе, выполнение DOS-прерываний. Заметим, что т.к. модули DOS обеспечивают большое количество разных дополнительных проверок, то операции-DOS более просты в использовании и менее машиннозависимы, чем их BIOS аналоги.

Модуль **IO.SYS** обеспечивает интерфейс с BIOS низкого уровня. Эта программа выполняет управление вводом-выводом при чтении данных из внешних устройств в память и записи из памяти на внешние устройства.   
Модуль **MSDOS.SYS** содержит средства управления файлами и ряд сервисных функций, таких как блокирование и деблокирование записей.

Порядок выполнения DOS-прерывания следующий... Когда пользовательская программа выдает запрос на прерывание (например, INT 21H), то в модуль MSDOS.SYS через регистры передается определенная информация. Затем модуль MSDOS.SYS транслирует эту информацию в один или несколько вызовов IO.SYS, который, в свою очередь, вызывает соответствующую подпрограмму BIOS.

**Пример установки курсора в любую позицию экрана**

Команда **INT 10h** включает в себя установку курсора в любую позицию и очистку экрана. Ниже приведен пример установки курсора на 5-ю строку и 12-й столбец:

**MOV  AH,2**   ;Запрос на установку курсора  
**MOV  BH,0**   ;Экран 0.  
**MOV  DH,05**  ;Строка 05.  
**MOV  DL,12**  ;Столбец 12.  
**INT  10h**    ;Передача управления в BIOS.

Значение **02** в регистре **AH** указывает команде **INT 10H** на выполнение операции установки курсора. Значения строки и столбца должны быть в регистре **DX**, а номер экрана (или страницы) - в регистре **BH** (обычно 0). Содержимое других регистров несущественно.

Все функции DOS вызываются с помощью прерывания 21h (в десятичной нотации 33). Первая версия DOS содержала 42 функции. Во второй к ним добавлено еще 33 функции, которые сохраняются во всех последующих версиях. Выбор конкретной функции осуществляется путем записи соответствующего номера в регистр AH.

#### Функция 02: вывод одного символа на экран

Для вывода одного символа на экран ПК используется

функция 02 прерывания 21h:

mov DL, <код выводимого символа>

mov AH, 2

int 21h

#### Функция 9: вывести строку на экран дисплея

Для вывода на экран строки (последовательности символов) можно, конечно, использовать функцию 02, однако сделать это можно и за один прием с помощью функции 09 прерывания 21h:

DS:DX := начальный адрес строки

mov AH, 9

int 21h.

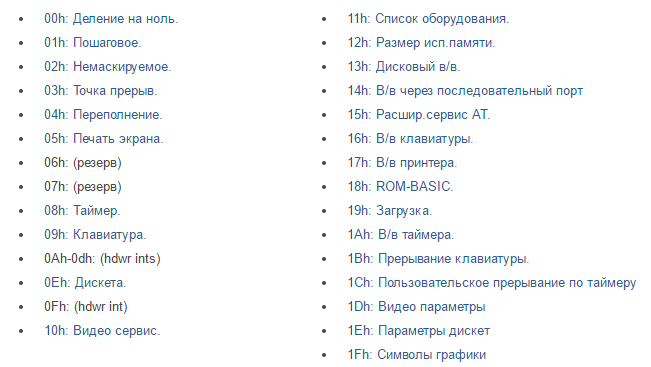
#### Функция 4Ch: завершение программы

Завершив все свои действия, программа обязана вернуть управление операционной системе, чтобы пользователь мог продолжить работу на ПК. Такой возврат реализуется функцией 4Ch прерывания 21h, которую помещают в конце программы:

mov AL, <код завершения>

mov AH, 4Ch

int 21h

Прерывания биоса



### Вопрос 75. Цепочечные команды.

Кроме привычного всем понятия массивов в ассемблере существует структура называемая цепочкой. Цепочка - непрерывная последовательность байт, слов или двойных слов, обрабатываемая как единое целое. Основное отличие цепочек от массивов состоит в способе доступа к элементам: для массивов - произвольный доступ, для цепочек - только последовательный (от начала цепочки к концу или от конца к началу).

Цепочечные команды - команды для обработки цепочек. Особенностью всех цепочечных команд (кроме обработки очередного элемента цепочки) является автоматическое продвижение к следующему элементу цепочки.

**Цепочечные команды:**

|  |  |  |
| --- | --- | --- |
| **Название** | **Команды** | **Действие** |
| пересылка цепочки | movs <адр. приемника>, <адр. источника> movsb, movsw, mowsd | копирует один элемент цепочки из операнда источника в операнд приемник |
| сравнение цепочек | cmps <адр. приемника>, <адр. источника> cmpsb, cmpsw, cmpsd | сравнивает элементы цепочек из операнда источника и операнда приемника |
| сканирование цепочки | scas <адр. приемника> scasb, scasw, scasd | сканирует цепочку приёмник на присутствие некоторого элемента (задаётся в регистре аккумуляторе) |
| загрузка элемента из цепочки | lods <адр. источника> lodsb, lodsw, lodsd | загрузить элемент из цепочки источника в регистр аккумулятор |
| сохранение элемента в цепочке | stos <адр. приемника> stosb, stosw, stosd | восстановить элемент из регистра аккумулятора в цепочку |
| получение элемента цепочки из порта ввода/вывода | ins <адр. приемника>, <номер порта> insb, insw, insd | загрузить элемент в цепочку приемник из указанного порта ввода/вывода |
| вывод элементов цепочки в порт ввода/вывода | outs <номер порта>, <адр. источника> outbs, outws, outds | переслать элемент из цепочки источника в указанный порт ввода/вывода |

**MOVS** адрес\_приемника, адрес\_источника — переслать цепочку (MOVe String);  
**MOVSB** — переслать цепочку байтов; **MOVSW** — цепочку слов; **MOVSD** — цепочку двойных слов.

**CMPS** адрес\_приемника, адрес\_источника — сравнить строки (CoMPare String);  
**CMPSB/CMPSW**/**CMPSD**  — сравнить строку байтов/слов/ двойных слов.

**SCAS адрес\_приемника** — сканировать цепочку (SCAning String);  
**SCASB**/**SCASW**/**SCASD** — цепочку байтов/слов/двойных слов.  
**LODS** адрес\_источника — загрузить элемент из цепочки (LOaD String) в регистр- аккумулятор AL/AX/EAX;  
**LODSB**/**LODSW/LODSD** — загрузить байт из цепочки в регистр AL/ слово АХ/двойное слово ЕАХ.

**STOS адрес\_приемника** — сохранить в цепочке элемент (STOre String) из регистра-аккумулятора AL/AX/EAX;  
**STOSB**/**STOSW**/**STOSD** — сохранить в цепочке байт из регистра AL/слово АХ/двойное слово ЕАХ.

**INS адрес\_приемника, номер\_порта** — ввести из порта цепочку (Input String);  
**INSB**/**INSW**/**INSD** — ввести из порта цепочку байтов/слов/двойных слов

**OUTS адрес\_источника, номер\_порта** — вывести цепочку в порт ввода-вывода(Output String);  
**OUTBS/OUTWS/OUTDS**   — вывести цепочку байтов в порт ввода-вывода/слов/двойных слов

**Адресация операндов:** цепочка источник - *ds:si* цепочка приёмник - *es:di*

**Направление обработки**

1.от начала к концу   
df = 0; *si* и *di* автоматически увеличиваются   
команда cld (clear direction flag) сбрасывает флаг df

2.от конца к началу  
df = 1; *si* и *di* автоматически уменьшаются  
команда std (set direction flag) устанавливает флаг df

**Пример:** Написать программу копирования строки.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | data segment | |  |
|  |  |  | s1 db 'Тестируемая строка$' | ;строка которую будем копировать |
|  |  |  | s2 db 20 dup (' ') | ;строка куда будем копировать |
|  |  | data ends | |  |
|  |  |  |  |  |
|  |  | code segment | |  |
|  |  | start: | |  |
|  |  |  | assume cs:code, ds: data |  |
|  |  |  | mov ax, data |  |
|  |  |  | mov ds, ax | ;цепочка источник |
|  |  |  | mov es, ax | ;и цепочка приёмник в одном сегменте |
|  |  |  |  |  |
|  |  |  | Cld | ;обработка от начала к концу |
|  |  |  | lea si, s1 | ;цепочка источник |
|  |  |  | lea di, s2 | ;цепочка приёмник |
|  |  |  | mov cx, 20 | ;количество элементов для обработки |
|  |  | rep | Movsb | ;копируем строку |
|  |  |  |  |  |
|  |  |  | mov ah, 09 |  |
|  |  |  | lea dx, s2 |  |
|  |  |  | int 21h | ;выводим строку-приёмник на экран |
|  |  |  |  |  |
|  |  |  | mov ax, 4c00h |  |
|  |  |  | int 21h |  |
|  |  | code ends | |  |
|  |  | end start | |  |

### Вопрос 76. Команды установки флагов.

**Регистр флагов** – это очень важный регистр процессора, который используется при выполнении большинства команд. Регистр флагов носит название EFLAGS. Это 32-разрядный регистр. Однако старшие 16 разрядов используются при работе в защищённом режиме. К младшим 16 разрядам этого регистра можно обращаться как к отдельному регистру с именем FLAGS.

Каждый бит в регистре FLAGS является флагом.

**Флаг** – это один или несколько битов памяти, которые могут принимать двоичные значения (или комбинации значений) и характеризуют состояние какого-либо объекта.

Обычно флаг может принимать одно из двух логических значений. Поскольку в нашем случае речь идёт о бите, то каждый флаг в регистре может принимать либо значение 0, либо значение 1. Флаги устанавливаются в 1 при определённых условиях, или установка флага в 1 изменяет поведение процессора. **Рис.** Регистр флагов FLAGS:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Бит** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Флаг** | 0 | NT | IOPL | | OF | DF | IF | TF | SF | ZF | 0 | AF | 0 | PF | 1 | CF |

Флаг **установлен**, если значение соответствующего ему бита равно 1.

Флаг **сброшен**, если значение соответствующего ему бита равно 0.

**Таблица**. Описание флагов регистра FLAGS.

|  |  |  |  |
| --- | --- | --- | --- |
| **Бит** | **Обозн** | **Название** | **Описание** |
| 0 | CF | Carry Flag | **Флаг переноса**. Устанавливается в 1, если результат предыдущей операции не уместился в приёмнике и произошёл перенос из старшего бита или если требуется заём (при вычитании). Иначе установлен в 0. |
| 1 | 1 | - | Зарезервирован. |
| 2 | PF | Parity Flag | **Флаг чётности**. Устанавливается в 1, если младший байт результата предыдущей команды содержит чётное количество битов, равных 1. Если количество единиц в младшем байте нечётное, то этот флаг равен 0. |
| 3 | 0 | - | Зарезервирован. |
| 4 | AF | Auxiliary Carry Flag | **Вспомогательный флаг переноса** (или **флаг полупереноса**). Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый. Этот флаг используется автоматически командами двоично-десятичной коррекции. |
| 5 | 0 | - | Зарезервирован. |
| 6 | ZF | Zero Flag | **Флаг нуля**. Устанавливается 1, если результат предыдущей команды равен 0. |
| 7 | SF | Sign Flag | **Флаг знака**. Этот флаг всегда равен старшему биту результата. |
| 8 | TF | Trap Flag | **Флаг трассировки** (или **флаг ловушки**). Он был предусмотрен для работы отладчиков в пошаговом выполнении, которые не используют защищённый режим. Если этот флаг установить в 1, то после выполнения каждой программной команды управление временно передаётся отладчику (вызывается прерывание 1). |
| 9 | IF | Interrupt Enable Flag | **Флаг разрешения прерываний**. Если сбросить этот флаг в 0, то процессор перестанет обрабатывать прерывания от внешних устройств. Обычно его сбрасывают на короткое время для выполнения критических участков программы. |
| 10 | DF | Direction Flag | **Флаг направления**. Контролирует поведение команд обработки строк. Если установлен в 1, то строки обрабатываются в сторону уменьшения адресов, если сброшен в 0, то наоборот. |
| 11 | OF | Overflow Flag | **Флаг переполнения**. Устанавливается в 1, если результат предыдущей арифметической операции над числами со знаком выходит за допустимые для них пределы. Например, если при сложении двух положительных чисел получается число со старшим битом, равным единице, то есть отрицательное. И наоборот. |
| 12 13 | IOPL | I/O Privilege Level | Уровень приоритета ввода/вывода. |
| 14 | NT | Nested Task | Флаг вложенности задач. |
| 15 | 0 | - | Зарезервирован. |

Значения некоторых флагов можно изменять напрямую с помощью специальных команд. Однако нет инструкций, которые бы позволяли обращаться к регистру флагов как к обычному регистру по имени. Некоторые флаги устанавливаются автоматически и предназначены только для чтения.

### 

### Вопрос 77. Команды синхронизации.

**Команды синхронизации процессора**

| **Команда** | **Описание** |
| --- | --- |
| HLT | Остановка процессора до внешнего прерывания |
| LOCK | Префикс блокировки шины. Заставляет процессор сформировать сигнал LOCK# на время выполнения находящейся за префиксом команды. Этот сигнал блокирует запросы шины другими процессорами в мультипроцессорной системе |
| WAIT | Ожидание завершения команды сопроцессора. Большинство команд сопроцессора автоматически вырабатывают эту команду |
| NOP | Пустая операция |
| CPUID | Получение информации о процессоре. Возвращаемое значение зависит от параметра в EAX |

Одним из средств синхронизации процессора с внешними устройствами являются прерывания, но в его архитектуре реализованы ещё две формы синхронизации: первая относится к использованию сопроцессора, вторая - к разделению ресурсов с другими процессорами в мультипроцессорной системе.

Например, для реализации вещественной арифметики целесообразно использовать специальный арифметический сопроцессор. Он работает только во взаимодействии с основным процессором.

В частности, когда основной процессор встречает специальную команду ESC, он передает ее на исполнение сопроцессору. Команда ESC показывает, какую операцию должен выполнить сопроцессор (код операции), эту информацию основной процессор игнорирует, но команда ESC задаёт для сопроцессора операнд в памяти (или регистре). Эту информацию основной процессор использует так: он вычисляет адрес операнда, а затем считывает его или записывает в память по запросу процессора. ESC применяется вместе с командой ожидания WAIT, обеспечивающей синхронизацию процессоров; с её помощью основной процессор проверяет, когда сопроцессор заканчивает свою операцию.

Если в многопроцессорной системе возникает проблема разделения общей памяти, то для избежания использования не полностью откорректированной информации вторым процессором при проведении операций коррекции первым в системе команд микропроцессора существует префикс блокировки шины LOCK. Например, чтобы нельзя было прервать команду пересылки элементов строки можно написать: LOCK REP MOVS. Такая команда выдает сигнал LOCK на всё время выполнения команды MOVS, что запрещает другим процессорам доступ к памяти.

**Вопрос 78.** Вывод на экран средствами ОС.

Вывод на экран в текстовом режиме, Средства DOS

Одним из способов вывода текста на экран — является вызов функции DOS 09h. Это далеко не единственный способ вывода текста — DOS предоставляет для этого несколько функций.

*Функция DOS 02h* — Записать символ в STDOUT с проверкой на Ctrl-Break

|  |  |
| --- | --- |
| Ввод: | АН = 02h DL = ASCII-код символа |
| Вывод: | Никакого, согласно документации, но на самом деле: AL = код последнего записанного символа (равен DL, кроме случая, когда DL = 09h (табуляция), тогда в AL возвращается 20h). |

Эта функция при выводе на экран обрабатывает некоторые управляющие символы — вывод символа BEL (07h) приводит к звуковому сигналу, символ BS (08h) приводит к движению курсора влево на одну позицию, символ НТ (09h) заменяется на несколько пробелов, символ LF (0Ah) опускает курсор на одну позицию вниз, и CR (0Dh) приводит к переходу на начало текущей строки.

Если в ходе работы этой функции была нажата комбинация клавиш Ctrl-Break, вызывается прерывание 23h, которое по умолчанию осуществляет выход из программы.

Например, напишем программу, выводящую на экран все ASCII-символы, 16 строк по 16 символов в строке.

; dosoutl.asm

; Выводит на экран все ASCII-символы

.model tiny

.code

.stack 100h ; начало СОМ-файла

start:

mov ex,256 ; вывести 256 символов

mov dl,0 ; первый символ - с кодом 00

mov ah,2 ; номер функции DOS "вывод символа"

cloop: int 21h ; вызов DOS

inc dl ; увеличение DL на 1 - следующий символ

test dl,0Fh ; если DL не кратен 16,

jnz continue\_loop ; продолжить цикл,

push dx ; иначе: сохранить текущий символ

mov dl,0Dh ; вывести CR

int 21h

mov dl,0Ah ; вывести LF

int 21h

pop dx ; восстановить текущий символ

continue\_loop:

loop cloop ; продолжить цикл

ret ; завершение СОМ-файла

end start

Здесь с помощью команды LOOP оформляется цикл, выполняющийся 256 раз (значение регистра СХ в начале цикла). Регистр DL содержит код символа, который равен нулю в начале цикла и увеличивается каждый раз на 1 командой INC DL. Если значение DL сразу после увеличения на 1 кратно 16, оно временно сохраняется в стеке и на экран выводятся символы CR и LF, выполняющие переход на начало новой строки. Проверка выполняется командой TEST DL,0Fh — результат операции AND над DL и 0Fh будет нулем, только если младшие четыре бита DL равны нулю, что и соответствует кратности шестнадцати.

Все функции DOS вывода на экран используют устройство STDOUT, стандартный вывод. Это позволяет перенаправлять вывод программы в файл или на стандартный ввод другой программы.

*Функция DOS 06h* — Записать символ в STDOUT без проверки на Ctrl-Break

|  |  |
| --- | --- |
| Ввод: | АН = 06h DL = ASCII-код символа (кроме FFh) |
| Вывод: | Никакого, согласно документации, но на самом деле: AL = код записанного символа (копия DL) |

Эта функция не обрабатывает управляющие символы (CR, LF, HT и BS выполняют свои функции при выводе на экран, но сохраняются при перенаправлении вывода в файл) и не проверяет нажатие Ctrl-Break. Можно заменить в программе dosoutl.asm команду MOV АН,2 на MOV АН,6 и перекомпилировать этот пример, чтобы получить более полную таблицу символов.

*Функция DOS 09h* — Записать строку в STDOUT с проверкой на Ctrl-Break

|  |  |
| --- | --- |
| Ввод: | АН = 09h DS:DX = адрес строки, заканчивающейся символом $ (24h) |
| Вывод: | Никакого, согласно документации, но на самом деле: AL = 24h (код последнего символа) |

Действие этой функции полностью аналогично действию функции 02h, но выводится не один символ, а целая строка

строка, исключая завершающий ее символ '$', посылается на стандартный вывод. символы Backspace обрабатываются как в функции 02H Display Char. обычно, чтобы перейти на новую строку, включают в текст пару CR/LF (ASCII 13H и ASCII 0aH). строки, содержащие '$', можно выдать через 40H Write Handle (BX=0).

*Функция DOS 40h* — Записать в файл или устройство

|  |  |
| --- | --- |
| Ввод: | АН = 40h ВХ = 1 для STDOUT или 2 для STDERR DS:DX = адрес начала строки СХ = длина строки |
| Вывод: | CF = 0, АХ = число записанных байт |

Эта функция предназначена для записи в файл, но, если в регистр ВХ поместить число 1, функция 40h будет выводить данные на STDOUT, а если ВХ = 2 — на устройство STDERR. STDERR всегда выводит данные на экран и не перенаправляется в файлы. На этой функции основаны используемые в С функции стандартного вывода — фактически функция С fputs() просто вызывает это прерывание, помещая свой первый аргумент в ВХ, адрес строки (второй аргумент) — в DS:DX и длину — в СХ.

; Выводит на экран строку "This function can print $",

; используя вывод в STDERR, так что ее нельзя перенаправить в файл.

.model tiny

.code

.stack 100h ; начало СОМ-файла

start:

mov ah,40h ; номер функции DOS

mov bx,2 ; устройство STDERR

mov dx,offset message ; DS:DX - адрес строки

mov cx, message\_length ; CX - длина строки

int 21h

ret ; завершение СОМ-файла

message db "Эта функция может выводить знак $"

message\_length = $-message ; длина строки = текущий адрес

; минус адрес начала строки

end start

последняя функция DOS вывода на экран — недокументированное прерывание 29h.

*INT 29h*: Быстрый вывод символа на экран

|  |  |
| --- | --- |
| Ввод: | AL = ASCII-код символа |

В большинстве случаев INT 29h просто немедленно вызывает функцию BIOS «вывод символа на экран в режиме телетайпа», так что никаких преимуществ, кроме экономии байт при написании как можно более коротких программ, она не имеет.

**Вопрос 79.** Вывод на экран средствами BIOS.

Функции DOS вывода на экран позволяют перенаправлять вывод в файл, но не позволяют вывести текст в любую позицию экрана и не позволяют изменить цвет текста. DOS предполагает, что для более тонкой работы с экраном программы должны использоваться видеофункции BIOS.

BIOS (базовая система ввода-вывода) — это набор программ, расположенных в постоянной памяти компьютера, которые выполняют его загрузку сразу после включения и обеспечивают доступ к некоторым устройствам, в частности к видеоадаптеру. Все функции видеосервиса BIOS вызываются через прерывание 10h.

***Выбор видеорежима***

BIOS предоставляет возможность переключения экрана в различные текстовые и графические режимы. Режимы отличаются друг от друга разрешением (для графических) и количеством строк и столбцов (для текстовых), а также количеством возможных цветов.

*INT 10h, АН = 00* — Установить видеорежим

|  |  |
| --- | --- |
| Ввод: | AL = номер режима в младших 7 битах |
| Вывод: | Обычно никакого, но некоторые BIOS (Phoenix и AMI) помещают в AL 30Н для текстовых режимов и 20h для графических |

Вызов этой функции приводит к тому, что экран переводится в выбранный режим.

Если старший бит AL не установлен в 1, экран очищается. Номера текстовых режимов — 0, 1, 2, 3 и 7.

0 и 1 — 16-цветные режимы 40x25 (с 25 строками по 40 символов в строке),

2 и 3 — 16-цветные режимы 80x25,

7 — монохромный режим 80x25.

*INT 10h, АН = 4Fh, AL = 02* — Установить SuperVGA-видеорежим

|  |  |
| --- | --- |
| Ввод: | ВХ = номер режима в младших 13 битах |
| Вывод: | AL = 4Fh, если эта функция поддерживается АН = 0, если переключение произошло успешно АН = 1, если произошла ошибка |

Если бит 15 регистра ВХ установлен в 1, видеопамять не очищается. Текстовые режимы, которые можно вызвать с использованием этой функции: 80x60 (режим 108h), 132x25 (109h), 132x43 (10Ah), 132x50 (10Bh), 132x60 (10Ch).

Видеорежим, используемый в DOS по умолчанию, — текстовый режим 3.

***Управление положением курсора***

*INT 10h, АН = 02* — Установить положение курсора

|  |  |
| --- | --- |
| Ввод: | АН = 02 ВН = номер страницы DH = строка DL = столбец |

С помощью этой функции можно установить курсор в любую позицию экрана, и дальнейший вывод текста будет происходить из этой позиции. Отсчет номера строки и столбца ведется от верхнего левого угла экрана (символ в левой верхней позиции имеет координаты 0, 0). Номера страниц 0 – 3 (для режимов 2 и 3)и 0 – 7 (для режимов 1 и 2) соответствуют области памяти, содержимое которой в данный момент отображается на экране. Можно вывести текст в неактивную в настоящий момент страницу, а затем переключиться на нее, чтобы изображение изменилось мгновенно.

*INТ 10h, АН = 03* — Считать положение и размер курсора

|  |  |
| --- | --- |
| Ввод: | АН = 03 ВН = номер страницы |
| Вывод: | DH, DL = строка и столбец текущей позиции курсора СН, CL = первая и последняя строки курсора |

Возвращает текущее состояние курсора на выбранной странице (каждая страница использует собственный независимый курсор).

***Вывод символов на экран***

Каждый символ на экране описывается двумя байтами — ASCII-кодом символа и байтом атрибута, указывающим цвет символа и фона, а также является ли символ мигающим.

***Атрибуты символов***

|  |  |  |
| --- | --- | --- |
|  | **Обычный цвет** | **Яркий цвет** |
| 000b | черный | темно-серый |
| 001b | синий | светло-синий |
| 010b | зеленый | светло-зеленый |
| 011b | голубой | светло-голубой |
| 100b | красный | светло-красный |
| 101b | пурпурный | светло-пурпурный |
| 110b | коричневый | желтый |
| 111b | светло-серый | белый |

*INT 10h, АН = 08* — Считать символ и атрибут символа в текущей позиции курсора

|  |  |
| --- | --- |
| Ввод: | АН = 08 ВН = номер страницы |
| Вывод: | АН = атрибут символа AL = ASCII-код символа |

*INT 10h, АН = 09* — Вывести символ с заданным атрибутом на экран

|  |  |
| --- | --- |
| Ввод: | АН = 09 ВН = номер страницы AL = ASCII-код символа BL = атрибут символа СХ = число повторений символа |

С помощью этой функции можно вывести на экран любой символ, включая даже символы CR и LF, которые обычно интерпретируются как конец строки. В графических режимах СХ не должен превышать число позиций, оставшееся до правого края экрана.

Функции BIOS удобны для переключения и настройки видеорежимов, но часто оказывается, что вывод текста на экран гораздо быстрее и проще выполнять просто копированием изображения в видеопамять.

Пример, вывести на экран все 256 символов, включая даже символы перевода строки. Кроме того, для лучшей читаемости таблицы после каждого символа будет выводиться пробел.

; Выводит на экран все ASCII-символы без исключения

start:

mov ax,0003h

int 10h ; Видеорежим 3 (очистка экрана

; и установка курсора в 0, 0)

mov dx,0 ; DH и DL будут использоваться

; для хранения положения курсора.

; Начальное положение - 0,0

mov si,256 ; SI будет счетчиком цикла

mov al,0 ; Первый символ - с кодом 00h

mov ah,9 ; Номер видеофункции "вывод символа с атрибутом"

mov cx,1 ; Выводится один символ за раз

mov bl,00011111b

;атрибут символа - белый на синем

cloop:

int 10h ; Вывести символ на экран

push ax ; Сохранить текущий символ и номер функции

mov ah,2 ; Номер видеофункции 2 -

; изменить положение курсора

inc dl ; Увеличить текущий столбец на 1

int 10h ; Переместить курсор

mov ax,0920h ; АН = 09, AL = 20h (ASCII-код пробела)

int 10h ; Вывести пробел

mov ah,2 ; Номер видеофункции 2

inc dl ; Увеличить столбец на 1

int 10h ; Переместить курсор

pop ax ; Восстановить номер функции в ah

; и текущий символ в al

inc al ; Увеличить AL на 1 - следующий символ

test al,0Fh ; Если AL не кратен 16,

jnz continue\_loop

; продолжить цикл,

push ax ; иначе - сохранить номер функции

; и текущий символ

mov ah,2 ; Номер видеофункции 2

inc dh ; Увеличить номер строки на 1

mov dl,0 ; Столбец = 0

int 10h ; Установить курсор на начало следующей строки

pop ax ; Восстановить номер видеофункции

; и текущий символ

continue\_loop:

dec si ; Уменьшить SI на 1,

; если он не стал нулем - продолжить

jnz cloop ; CX используется внутри цикла,

; так что нельзя использовать команду LOOP

; для его организации

ret ; Завершение СОМ-файла

end start

Так как функция 09 выводит символ в позиции курсора, но не перемещает сам курсор, это приходится делать каждый раз специально.

### Вопрос 80. Прямая работа с видеопамятью.

Все, что изображено на мониторе — и графика, и текст, одновременно присутствует в памяти, встроенной в видеоадаптер. Для того чтобы изображение появилось на мониторе, оно должно быть записано в память видеоадаптера. Для этого отводится специальная область памяти, начинающаяся с абсолютного адреса B800h:0000h (для текстовых режимов) и заканчивающаяся на B800h:FFFFh. Все, что программы пишут в эту область памяти, немедленно пересылается в память видеоадаптера. В текстовых режимах для хранения каждого изображенного символа используются два байта: байт с ASCII-кодом символа и байт с его атрибутом, так что по адресу B800h:0000h лежит байт с кодом символа, находящимся в верхнем левом углу экрана; по адресу B800h:0001h лежит атрибут этого символа; по адресу B800h:0002h лежит код второго символа в верхней строке экрана и т.д.

Таким образом, любая программа может вывести текст на экран простой командой пересылки данных, не прибегая ни к каким специальным функциям DOS или BIOS.

**Префиксы повторения операций**:

* **REP** — повторять;
* **REPE** — повторять пока равно;
* **REPNE** — повторять пока не равно;
* **REPZ** — повторять пока ZF=0;
* **REPNZ** — повторять пока ZF=1.

Префиксы указывает количество выполнений команд строковой обработки. Количество выполнений указывается в регистре ЕСХ (или СХ, в зависимости от разрядности). После каждого выполнения команды значение в регистре уменьшается на 1.

Операции копирования из регистров в память (в скобках указан тип процессора):

* STOS приемник (8086) — запись в строку;
* STOSB (8086) — запись байта в строку;
* STOSW (8086) — запись слова в строку;
* STOSD (80386) — запись двойного слова в строку.

Данные команды записывают значения регистров AL (STOSB), AX (STOSW) или ЕАХ (STOSD) в память, начиная с ES:EDI (или ES:DI в зависимости от разрядности). При использовании команды STOS ассемблер сам определяет по типу операнда форму этой команды (STOSB, STOSW или STOSD). После выполнения команды значение в регистре EDI (DI) увеличивается на 1, 2 или 4 (значения соответствуют байту, слову или двойному слову), если флаг DF = 0, и уменьшается, если DF = 1.

Пример, выводящий 10 раз число 67, используя прямой вывод на экран.

.stack 100h ; начало СОМ-файла

start:

mov ax,0003h

int 10h

mov cx,10

mov ax,0B800h ; сегментный адрес видеопамяти

mov es,ax

mov di,150 ; координата курсора

mov eax, 1f371f36h ; копирование (адреса) данных в видеопамять

rep stosd

ret

При копировании данных в видеопамять необходимо учесть, что в архитектуре Intel при записи данных в память старший байт копируется по старшему адресу, т.е. при копировании память двойного слова 1f37h1f36h сначала копируется младший байт 36h (ASCII-код цифры 6), потом атрибут 1Fh.

**Вопрос 88.** Конвенция Pascal передачи параметров программ на Ассемблере.

В программах, написанных на языке ассемблера, используется соглашение передачи параметров stdcall. Однако по сути получение и передача параметров в языке ассемблера производится явно, без помощи транслятора.  
При связи процедуры, написанной на языке ассемблера, с языком высокого уровня, необходимо учитывать соглашение по передаче параметров.

**Конвенция Pascal** заключается в том, что параметры из программы на языке высокого уровня передаются в стеке и возвращаются в регистре АХ/ЕАХ, — это способ, принятый в языке PASCAL (а также в BASIC, FORTRAN, ADA, OBERON, MODULA2), — просто поместить параметры в стек в естественном порядке. В этом случае запись

some\_proc(a,b,c,d);

запишется как

push a  
push b  
push с  
push d  
call some\_proc@16

Процедура some\_proc, во-первых, должна очистить стек по окончании работы (например, командой ret 16) и, во-вторых, параметры, переданные ей, находятся в стеке в обратном порядке:

some\_proc proc PASCAL, а:dword, b:dword, с:dword, d:dword  
...  
ret  
some\_proc endp

Главный недостаток этого подхода — сложность создания функции с изменяемым числом параметров, аналогичных функции языка С printf. Чтобы определить число параметров, переданных printf, процедура должна сначала прочитать первый параметр, но она не знает его расположения в стеке.

**Вопрос 89.** Конвенция C передачи параметров программ на Ассемблере.

Конвенция С используется, в первую очередь, в языках С и C++, а также в PROLOG и других. Параметры помещаются в стек в обратном порядке, и, в противоположность PASCAL-конвенции, удаление параметров из стека выполняет вызывающая процедура.  
Запись some\_proc(a,b,c,d)  
будет выглядеть как

push d  
push с  
push b  
push a  
call some\_proc@16  
add esp,16 ; освободить стек

Вызванная таким образом процедура может инициализироваться так:

some\_proc proc С, а:dword, b:dword, с:dword, d:dword  
ret  
some\_proc endp

Преимущество по сравнению с PASCAL-конвенцией заключается в том, что освобождение стека от параметров в конвенции С возлагается на вызывающую процедуру, что позволяет лучше оптимизировать код программы. Например, если необходимо вызвать несколько функций, принимающих одни и те же параметры подряд, можно не заполнять стек каждый раз заново, и это — одна из причин, по которой компиляторы с языка С создают более компактный и быстрый код по сравнению с компиляторами с других языков.

**Вопрос 90.** Конвенция StdCall передачи параметров программ на Ассемблере.

**Конвенция stdcall**

Существует конвенция передачи параметров STDCALL, отличающаяся и от C, и от PASCAL-конвенций, которая применяется для всех системных функций Win32 API. Здесь параметры помещаются в стек в обратном порядке, как в С, но процедуры должны очищать стек сами, как в PASCAL.  
Еще одно отличие от С-конвенции – это быстрое или регистровое соглашение FASTCALL. В этом случае параметры в функции также передаются по возможности через регистры. Например, при вызове функции с шестью параметрами

some\_proc(a,b,с,d,e,f);

первые три параметра передаются соответственно в ЕАХ, EDX, ЕСХ, а только начиная с четвертого, параметры помещают в стек в обычном обратном порядке:

mov a, eax

mov b, edx

mov c, ecx

mov d, [ebp+8]  
mov e, [ebp+12]  
mov f, [ebp+16]

В случае если стек был задействован, освобождение его возлагается на вызываемую процедуру.  
В случае быстрого вызова транслятор Си добавляет к имени значок @ спереди, что искажает имена при обращении к ним в ассемблерном модуле.

**Вопрос 91.** Смешанные конвенции передачи параметров программ на Ассемблере.

Существует конвенция передачи параметров STDCALL, отличающаяся и от C, и от PASCAL-конвенций, которая применяется для всех системных функций Win32 API. Здесь параметры помещаются в стек в обратном порядке, как в С, но процедуры должны очищать стек сами, как в PASCAL.  
Еще одно отличие от С-конвенции – это быстрое или регистровое соглашение FASTCALL. В этом случае параметры в функции также передаются по возможности через регистры. Например, при вызове функции с шестью параметрами

some\_proc(a,b,с,d,e,f);

первые три параметра передаются соответственно в ЕАХ, EDX, ЕСХ, а только начиная с четвертого, параметры помещают в стек в обычном обратном порядке:

mov a, eax

mov b, edx

mov c, ecx

mov d, [ebp+8]  
mov e, [ebp+12]  
mov f, [ebp+16]

В случае если стек был задействован, освобождение его возлагается на вызываемую процедуру.  
В случае быстрого вызова транслятор Си добавляет к имени значок @ спереди, что искажает имена при обращении к ним в ассемблерном модуле.

**Вопрос 92.** Искажение имен процедур при передаче параметров программам на языках высокого уровня.

Компиляторы Microsoft С (а также многие компиляторы в UNIX, как мы увидим далее) изменяют названия процедур, чтобы отразить используемый способ передачи параметров. Так, к названиям всех процедур, использующих С-конвенцию, приписывается символ подчеркивания. То есть, если в С-программе записано

some\_proc();

то реально компилятор пишет

call \_some\_proc

и это означает, что, если эта процедура написана на ассемблере, она должна называться именно \_some\_proc (или использовать сложную форму записи директивы proc).

Названия процедур, использующих STDCALL искажаются еще более сложным образом: спереди к называнию процедуры добавляется символ подчеркивания, а сзади — символ @ и размер занимаемой параметрами области стека в байтах, (то есть в точности число, стоящее после команды ret в конце процедуры).

some\_proc(a:word);

превращается в

push a

call \_some\_proc@4

**Вопрос 93, 94.** Встраивание ассемблерного кода в Pascal-программу и С-программу.

Большинство компиляторов языков высокого уровня поддерживают возможность вставки ассемблерного кода в обрабатываемые ими программы. В данном разделе основное внимание уделяется ассемблерным вставкам в программах на C++, так как именно этот язык наиболее часто используется для реализации задач системного программирования.

Оператор inline языка Pascal представляет собой следующую синтаксическую конструкцию:

i n l i n e ( м а ш и н н ы е \_ к о д ы )

В скобках указывается строка машинных кодов. Для получения такой строки целесообразно написать нужный фрагмент на ассемблере, скомпилировать исполняемый модуль, а затем запустить отладчик. В окне CPU отладчика вы увидите машинные коды ваших инструкций; их нужно переписать и вставить в скобки оператора inline.

Встраивание ассемблерного кода в программы C++ производится директивой \_asm. Причем этой директивой возможна вставка как одной команды ассемблера, так и группы команд. Синтаксис:

asm команда\_ассемблера [;комментарий] asm {

команда\_ассемблера [;комментарий]

команда\_ассемблера [//комментарий или /\* комментарий \*/]

}