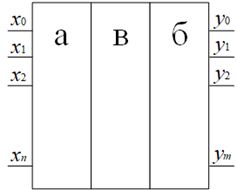
**Элементы электронных вычислительных машин**

 Элементы ЭВМ представляют собой простейшие радиотехнические детали, на основе которых строятся более крупные части ЭВМ – узлы. Общее условное графическое обозначение абстрактного элемента представлено на рисунке 3.1 и предполагает обозначение трех полей:

а) поля входов (слева). Количество входов *n*, где *n* ≥1;  
         б) поля выходов (справа). Количество выходов *m*, где *m* ≥1;  
         в) поле обозначения элемента (по середине), несущее информацию о типе элемента и возможно о исполняемых им функций.

Рисунок 3.1 – Общее обозначение элемента ЭВМ



Элементы ЭВМ бывают трех основных разновидностей:

1. Логические – реализующие какие-либо логические функции алгебры логики;
2. Запоминающие – триггеры (автомат Мура), реализующие хранение одного из двух устойчивых состояний;
3. Специальные (вспомогательные) – усилители, формирователи и генераторы и сигналов, преобразователи логических уровней, индикаторы и др.

**Основные узлы электронных вычислительных машин**

Все основные узлы ЭВМ строятся на базе элементов и как правило представляют собой совокупность нескольких логических элементов или элементов памяти.

Основные узлы ЭВМ можно разделить на:

1. Комбинационные – выходные сигналы зависят только от действующих в настоящее время входных сигналов.

2. Накапливающие – выходные сигналы зависят не только от действующих в настоящее время входных сигналов, но и от поступавших ранее.

К *комбинационным узлам* относятся:

1. Одноразрядный сумматор – узел, выполняющий операцию поразрядного сложения над одним двоичным разрядом. Входами такого сумматора являются: разряд первого слагаемого *a*, разряд второго слагаемого *b*, признак переноса из младшего разряда *p*; а выходами являются: сумма в текущем разряде *S*, признак переноса в старший разряд *P*;

2. Многоразрядный сумматор – узел, построенный на базе одноразрядных сумматоров с подключением соответствующих выходов к соответствующим входам («*p»* к «*P»*) и выполняющий сложение многоразрядных двоичных чисел. Многоразрядные сумматоры бывают с последовательным и со сквозным переносом;

3. Сумматор по модулю 2 – узел, реализующий поразрядное сложение по модулю 2 (логическую операцию исключающее ИЛИ).

4. Шифратор (кодер) – узел, преобразующий сигнал на одном из *n* входов в комбинацию сигналов на выходах *m* и используются в случае, когда необходимо передавать большое количество данных, при небольшом количестве линий связи. При этом количество входов *n* и выходов *m* связано соотношением 2*m* ≥ *n*. Также имеет вход синхронизации.

5. Дешифратор (декодер) – узел, преобразующее комбинацию сигналов на входах *n* в сигнал на одном из выходов *m*, т. е. с помощью *n* входов можно задавать выход, на который будет подаваться единичный сигнал. Количество входов *n* и выходов *m* связано соотношением 2*n* ≥ *m*. Также имеет вход синхронизации. Дешифратор применяется для построения мультиплексора и демультиплексора.

6. Мультиплексор – узел, обеспечивающий подключение одного из 2*n*+ *n* своих входов (*x*0, *x*1, …, *x*2*n*−1, *s*0, *s*1, …, *sn*−1) к единственному выходу *m*, на который подается значение на входе *xi*, где *i* — число, которое кодируется входами *s*0, *s*1, …, *sn*−1. Также имеет вход синхронизации. Используется при мультиплексированной передаче данных в шинных архитектурах связей (см. подраздел 3.3).

7. Демультиплексор – узел, обеспечивающий логическое подключение одного входа к одному из нескольких выходов, т. е. обратную мультиплексору функцию. Также имеет вход синхронизации.

*Накапливающие узлы* строятся на базе триггеров, каждый из которых предназначен для хранения отдельного разряда.

Триггеры бывают различных видов:

1. *RS*-триггер. Имеет два входа *qR* (reset, сброс) и *qS* (set, установка) и два выхода *Ǫ* и *Ǭ* (обратное значение выхода *Ǫ*, т. е. инверсия разряда). Для установки триггера в значение «0» (*Ǫ* = 0, *Ǭ* =1) необходимо подать единичный сигнал на вход *qR*, а для установки триггера в значение «1» (*Ǫ* = 1, *Ǭ* = 0) необходимо подать единичный сигнал на вход *qS*. Комбинация из двух единичных сигналов на обоих входах запрещена;

2. *T*-триггер. Имеет один вход *qT*и два выхода *Ǫ* и *Ǭ*. Единичный сигнал, поданный на вход *qT* переводит выходы *T*-триггер в противоположное состояние;

3. *JK*-триггер. Имеет два входа *qJ* и *qK* и два выхода *Ǫ* и *Ǭ*. Аналогичен *RS*-триггеру, только комбинация из двух единичных сигналов на обоих входах не запрещена и переводит выходы *JK*-триггер в противоположное состояние (т. е. аналогично *T*-триггеру);

4. *D*-триггер. Имеет один вход *qD* (digital, цифра)и два выхода *Ǫ* и *Ǭ*. Для установки триггера в значение «0» (*Ǫ* = 0, *Ǭ* =1) необходимо подать нулевой сигнал на вход *qD*, а для установки триггера в значение «1» (*Ǫ* = 1, *Ǭ* = 0) необходимо подать единичный сигнал на вход *qD*. Т. е. триггер хранит значение, поданное на вход.

Зависимость хранимого триггерами значения от сигналов на входах наглядно показана в таблице 3.1.  


*К накапливающим узлам* относятся:

1. Регистр – узел, предназначенный для хранения многоразрядных значений (отдельного числа).

*Регистр* – группа триггеров, количество которых соответствует количеству разрядов в хранимом регистром двоичном числе.

*Разрядность регистра* определяется архитектурой ЭВМ.

*Машинное слово* – двоичное число, хранимое в регистре.

Для упрощения доступа к регистрам они имеют уникальные номера. Этот уникальный номер называется *адресом регистра*.

Совокупность всех регистров – это *оперативная память*.

2. Счетчик – узел, предназначенный для хранения как правило многоразрядных значений, который по каждому сигналу изменяет (увеличивает на единицу) хранимый код числа.

Счетчики могут работать в двух режимах: в режиме счетчика событий и в режиме таймера.

В режиме счетчика происходит увеличение на единицу хранимого значения по каждому единичному сигналу, поступающему на вход счетчика по факту наступления какого-либо события.

В режиме таймера счетчик считает тактовые импульсы процессора или импульсную последовательность с выхода управляемого делителя частоты (например, для счета времени).

Существует множество различных счетчиков/таймеров, каждый из которых в ЭВМ выполняет свою функцию, например:

* счетчик команд – служит для хранения адреса ячейки оперативной памяти (по сути регистра) текущей команды, выполняемой процессором;
* сторожевой таймер – предназначен для защиты микропроцессорной системы (МПС) от зависания программы. При запуске на выполнение программы сторожевой таймер начинает считать тактовые импульсы. Сторожевой таймер сбрасывается на программном уровне, т. е. при зависании программы на аппаратном уровне еще продолжится счет тактов и при достижении счетчиком максимального кода (в зависимости от разрядности) генерируется сигнал внутреннего сброса (внутреннее прерывание), перезагружающий МПС.

**Принципы построения электронных вычислительных машин**

 В 1945 году Джоном фон Нейманом были сформулированы четыре основополагающих принципа построения ЭВМ, которые позже реализовались на практике и такие машины получили название фон Неймановских.

К этим принципам относятся:

***1) Принцип двоичного кодирования***

Вся информация, как данные, так и команды, кодируются двоичными цифрами «0» и «1». Каждый тип информации представляется двоичной последовательностью и имеет свой формат, который составляется из участков разной длины. Каждый участок последовательности битов в формате, имеет определенный смысл и называется полем. В числовой информации обычно выделяют поле знака и поле модуля числа. В формате команды обычно выделяют два основных поля: поле кода операции и поле адресов. Код операции представляет собой указание, какая операция должна быть выполнена, и задается с помощью *r*-разрядной двоичной комбинации.

Вид адресной части и число составляющих ее адресов зависят от типа команды: в командах преобразования данных адресное поле содержит адреса операндов и результата; в командах изменения порядка вычислений – адрес следующей команды программы; в командах ввода/вывода – номер устройства;

***2) Принцип программного управления***

Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы, состоящей из последовательности управляющих слов-команд. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются в естественной последовательности, то есть в порядке их следования в программе.

***3) Принцип однородности памяти***

Команды и данные могут храниться в одной и той же памяти. Циклически изменяя адресную часть команды, обеспечивается обращение к последовательным элементам массива данных. Это носит название модификации команд.

Другой подход принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе трансляции программы – перевода исходного кода с языка высокого уровня на машинный язык конкретной ЭВМ.

***4) Принцип адресности***

Основная память (ОП) состоит из пронумерованных ячеек, доступных процессору в произвольный момент времени. Двоичные коды команд называются словами и хранятся в ячейках памяти, для доступа к которым используются номера соответствующих ячеек – адреса. Команды и данные должны располагаться в ОП, так, что каждое слово хранится в отдельной ячейке, определяемой адресом и, соседние ячейки памяти имеют следующие по порядку адреса. Доступ к любым ячейкам ОП может производиться в произвольной последовательности (память с произвольным доступом), т. е. реализуется возможность условного перехода в процессе выполнения программы.

Адресность памяти определяет использование условного перехода.

Возможность условного перехода в процессе выполнения программы позволяет строить программы различной конструкции (не только линейные). Таким образом, порядок выполнения команд однозначно задается программой.

*Следствие введения фон Неймановских принципов:*

* программа перестала быть постоянной частью машины.
* программу стало возможно легко изменить.

**Классификация архитектур электронных вычислительных машин**

 Практически любая микропроцессорная система (МПС) имеет в своем составе следующие устройства:

1. *Процессор* – устройство, предназначенное для выполнения машинных команд, управления вычислительным процессом и его выполнением по программе, и координации работы всех других устройств МПС;

2. *Память* (в общем виде) – физическое устройство хранения данных;

3. *Устройства ввода-вывода* – устройства, предназначенные для взаимодействия МПС с внешней средой, в том числе с пользователем (человеком) и реализующее ввод данных в МПС и/или вывод данных пользователю или другой МПС или устройству.

*Архитектура* – это принцип построения чего-либо.

С точки зрения МПС, в частности ЭВМ, существует множество классификаций архитектур, таких как:

* архитектура связи устройств ЭВМ;
* архитектура ЭВМ по способам расположения команд и данных;
* архитектура ЭВМ по месту хранения операндов и др.

*Архитектура связи устройств ЭВМ*

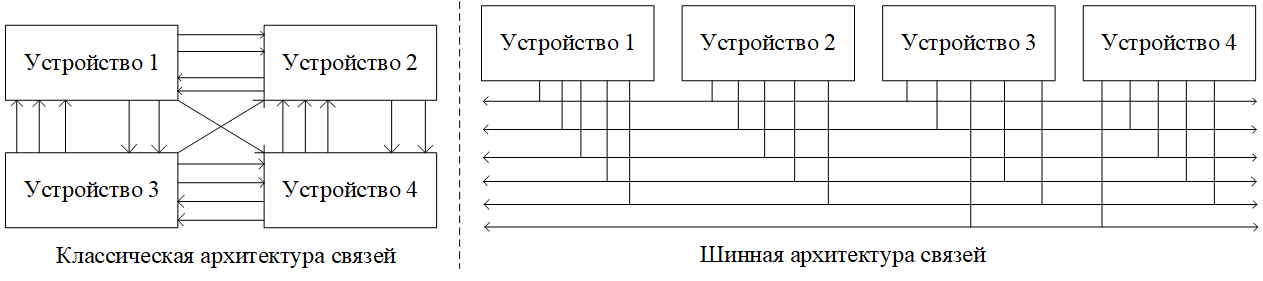
Различают классическую структуру связей (архаичная, с момента построения ЭВМ и до 1960-х гг.) и шинную (современная) (рисунок 3.2).  


Рисунок 3.2 – Архитектуры связей устройств ЭВМ

При классической архитектуре связей все устройства, входящие в состав ЭВМ соединены друг с другом независимо, по отдельным физическим линиям связи.

Достоинства классической архитектуры:

* независимость устройств входящих в МПС друг от друга при обмене информацией;
* относительно более высокая надежность работ МПС.

Недостатки классической архитектуры:

* множество линий связи и протоколов обмена информацией;
* громоздкость МПС из-за большого количества линий связи.

При шинной архитектуре связей все сигналы между всеми устройствами передаются по одним и тем же линиям связи, но в разное время (*мультиплексированная передача* данных). Причем передача по всем линиям связи может осуществляться в обоих направлениях (*двунаправленная передача данных*).

Достоинства шинной архитектуры:

* все устройства, подключенные к шине, имеют один протокол передачи информации (как следствие – унификация устройств входящих в МПС);
* существенная компактность системы.

Недостатки:

* поскольку все устройства подключаются к каждой линии связи параллельно, неисправность любого устройства может вывести из строя всю МПС, если неисправное устройство портит линию связи (например, выдает информационные сигналы в шину в неположенное время).
* более низкое быстродействие, обусловленное последовательной по времени передачей информации между устройствами МПС.

Несмотря на все имеющиеся недостатки шинной архитектуры, решающим фактором ее применения послужила компактность такой МПС.

Структура системной шины (называемой магистралью) представлена на рисунке 3.3.

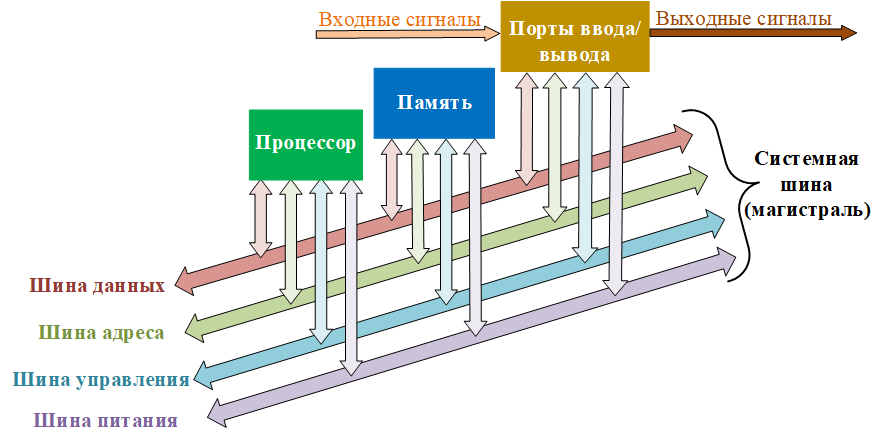
В состав системной шины (магистрали) входят шины более низкого уровня: шина данных, шина адреса, шина управления и шина питания. Каждое устройство, входящее в состав МПС подключено к каждой из этих шин параллельно.  


Рисунок 3.3 – Состав системной шины

*Шина данных (Data Bus)* – это основная шина, которая используется для передачи информационных кодов между всеми устройствами МПС. Возможна передача информации между устройствами даже без участия процессора (в режиме прямого доступа к памяти (*DMA*). Шина данных всегда двунаправленная.

*Шина адреса (Address Bus)* служит для определения адреса (номера) устройства, с которым процессор обменивается информацией в данный момент. Каждому устройству (кроме процессора), и каждой ячейке памяти в МПС присваивается собственный адрес. Шина адреса может быть однонаправленной или двунаправленной.

*Шина управления (Control Bus)* в отличие от шин адреса и данных передает отдельные управляющие сигналы. Каждый из них имеет свою функцию: некоторые сигналы служат для стробирования передаваемых или принимаемых данных (т. е. определяют моменты времени, когда информационный код выставлен на шину данных); другие управляющие сигналы используются для подтверждения приема данных; третьи для сброса всех устройств в исходное состояние; четвертые для тактирования всех устройств и т. д. Шина управления может быть однонаправленной или двунаправленной.

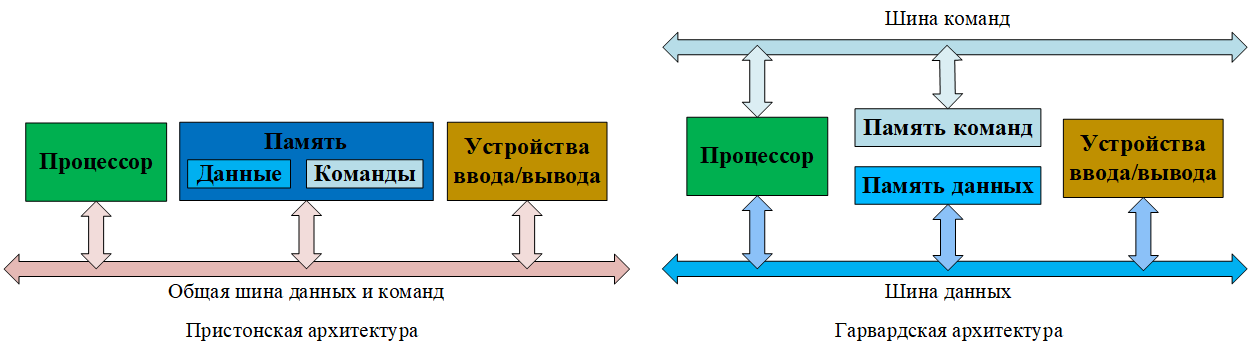
*Шина питания (Power Bus)* предназначена не для пересылки информационных сигналов, а для питания системы. Она состоит из линий питания и общего провода. В МПС может быть один источник питания или несколько источников питания (разного напряжения). Каждому напряжению питания соответствует своя линия связи. Все устройства подключены к этим линиям параллельно.

***Архитектура ЭВМ по способам расположения команд и данных***

По данной классификации существует Принстонская (одношинная, фон Неймановская) и Гарвардская (двухшинная) архитектуры, причем обе они подразумевают использование шинной архитектуры связей.

Принстонская архитектура (рисунок 3.4) реализует принцип совместного хранения команд (программы) и данных в памяти компьютера (т. е. в едином пространстве памяти), соединенной с остальными устройствами только одной шиной данных.

Гарвардская архитектура (рисунок 3.4) предполагает разделение памяти на память команд (программы) (ПК) и память данных (ПД), каждая из которых соединена с остальными устройствами отдельной шиной данных. Т. е. в состав системной шины при Гарвардской архитектуре входят две шины данных, названных по типу памяти: шина команд и шина данных.

Следует обратить внимание, что несмотря на неочевидность названия обе шины (шина данных и шина команд) являются шинами данных (data bus)!  
Рисунок 3.4 – Архитектуры ЭВМ по способам расположения команд и данных

Следует отметить, что данные архитектуры были разработаны в 1960-х года и в чистом виде давно устарели. В настоящее время на их основе разработаны более современные архитектуры, применяемые в ЭВМ, а именно:

* модифицированная гарвардская архитектура (МГА), призванная устранить существенный недостаток Гарвардской архитектуры – большое количество интерфейсных выводов (ввиду наличия двух шин данных и адреса), что приводит к увеличению размеров и соответственно стоимости кристалл, на котором выполнен микропроцессор. Поэтому МГА имеет общую шину данных и шину адреса для всех внешних данных, а внутри процессора используется шина данных, шина команд и две шины адреса. Разделение шин в МГА осуществляется при помощи раздельных управляющих сигналов: чтения, записи или выбора области памяти. МГА нашла применение в сигнальных процессорах и микроконтроллерах;
* расширенная гарвардская архитектура (РГА), предполагает использование кэш-памяти вместе с разделенными шинами данных (ПД+ПК+Кэш), что позволяет за один такт получать из разной памяти команду на исполнение и два операнда. РГА нашла применение в цифровой обработке сигналов.
* различные гибридные модификации архитектур, например процессорное ядро аппаратно является гарвардским, но программно используется как принстонское, что позволило одновременно сочетать в себе преимущества обоих систем: за один такт можно получать одновременно и команду, и операнды для ее выполнения (Гарвардская), но при этом написание программы происходит проще, как для Принстонской архитектуры. Применение: CISC-процессоры.

***Архитектура ЭВМ по месту хранения операндов***

Место хранения операндов и способ доступа к ним играет важную роль при выполнении команд процессором.

С этой точки зрения различают:

* стековую архитектуру;
* аккумуляторную архитектуру;
* регистровую архитектуру.

Использование той или иной архитектуры влияет на количество адресов в адресной части команды их длину, простоту получения операндов и в конечном счете на общую длину команды, а соответственно и скорость ее выполнения.

***Стековая архитектура***

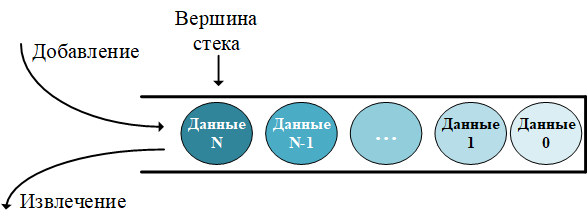
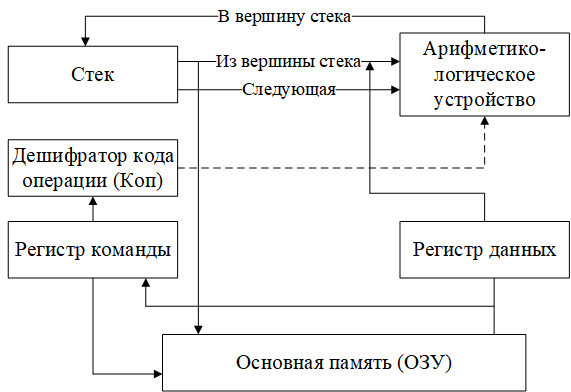
*Стек* – это отличная от оперативной памяти структура организации данных, работающая по принципу «последним вошел – первым вышел» (Last In – First Out, LIFO). Стек можно представить, как показано на рисунке 3.5 в виде трубки, запаянной с одного конца, а через второй конец, называемый вершиной стека, происходит добавление и извлечение данных.  


Рисунок 3.5 – Представление стека

 Информация может быть занесена в вершину стека из памяти или из арифметико-логического устройства (АЛУ). Для записи в стек содержимого ячейки памяти с адресом х выполняется команда *push x*, по которой информация считывается из ячейки памяти, заносится в регистр данных, а затем проталкивается в стек. Результат операции из АЛУ заносится в вершину стека автоматически.

Сохранение содержимого вершины стека в ячейке памяти с адресом х производится командой *pop х*. По этой команде содержимое верхней ячейки стека подается на шину, с которой и производится запись в ячейку *х*, после чего вся находящаяся в стеке информация проталкивается на одну позицию вверх.

Для выполнения арифметической или логической операции на вход АЛУ подается информация, считанная из двух верхних ячеек стека (при этом содержимое стека продвигается на две позиции вверх, то есть операнды из стека удаляются). Результат операции заталкивается в вершину стека или сразу же автоматически переписывается в память с помощью операции *pop х* (рисунок 3.6).  
  
Рисунок 3.6 – Стековая архитектура

***Аккумуляторная архитектура***

При такой архитектуре хранение одного из операндов арифметической или логической операции предполагается в специальном регистре процессора аккумуляторе, что предопределяет адрес нахождения этого операнда и избавляет от необходимости указывать его в команде. После выполнения операции, в аккумулятор будет записан результат ее выполнения (рисунок 3.7).

Однако изначально оба операнда хранятся в основной памяти, и до выполнения операции один из них нужно загрузить в аккумулятор. После выполнения команды обработки результат находится в аккумуляторе и, если он не является операндом для последующей команды, его также требуется сохранить в основной памяти.

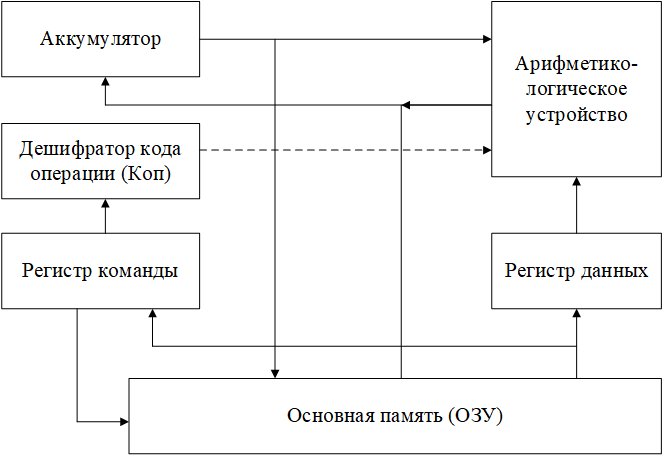
Для загрузки в аккумулятор содержимого ячейки памяти *х* предусмотрена команда загрузки *load x*. Запись содержимого аккумулятора в ячейку памяти *х* выполняется командой сохранения *store х*.  


Рисунок 3.7 – Аккумуляторная архитектура

***Регистровая архитектура***

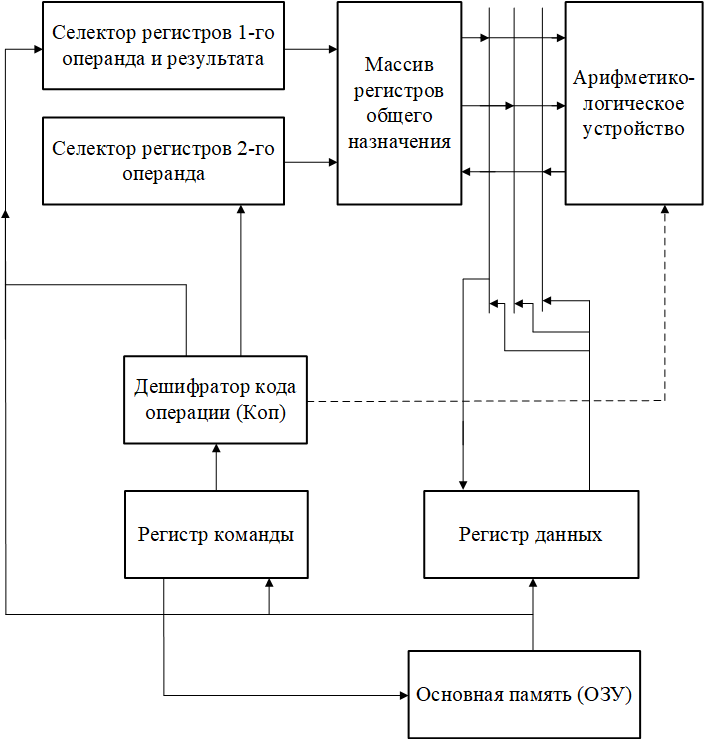
Регистровая архитектура допускает расположение операндов или в основной памяти, или в массиве регистров общего назначения (РОН). РОН можно рассматривать как явно управляемый кэш для хранения недавно использовавшихся данных (рисунок 3.8).  


Рисунок 3.8 – Регистровая архитектура

При таком размещении операндов возможны три варианта обработки:

* регистр-регистр: операнды могут находиться только в регистрах и в них же записывается и результат;
* регистр-память: один из операндов размещается в регистре, а второй в основной памяти. Результат обычно замещает один из операндов;
* память-память: оба операнда хранятся в основной памяти. Результат заносится в память.

Каждый из вариантов имеет свои достоинства и недостатки, отраженные в таблице 3.2.  
