Лекции по архитектуре ЭВМ

§1 Введение

Развитие Вычислительной Техники (ВТ) обусловлено успехами в 3-х областях:

1. В технологии производства, как элементарной базы ВТ, так и самих машин в целом.
2. В принципах организации ВМ (успехи в развитии архитектуры).
3. В разработке математического и программного обеспечения.

Любая ВМ должна рассматриваться, как некоторый программно – аппаратный комплекс, обеспечивающий реализацию некоторого класса алгоритмов над информацией.

В процессе работы ВМ все ее компоненты каким-то образом взаимодействуют между собой. Причем уровни рассмотрения этого взаимодействия могут быть различными:

1) Низший уровень: на уровне электрических импульсов.
2) Высший уровень: взаимодействие узлов ВМ на уровне программных модулей (1 и 2 рассматривать не будем).
3) Функциональный уровень каждого отдельного узла: функция и их реализация программно – аппаратными средствами (под этим и понимается понятие “Архитектура”).

Опр. Под Архитектурой понимается совокупность свойств и характер ВМ, рассматриваемая с точки зрения пользователя.

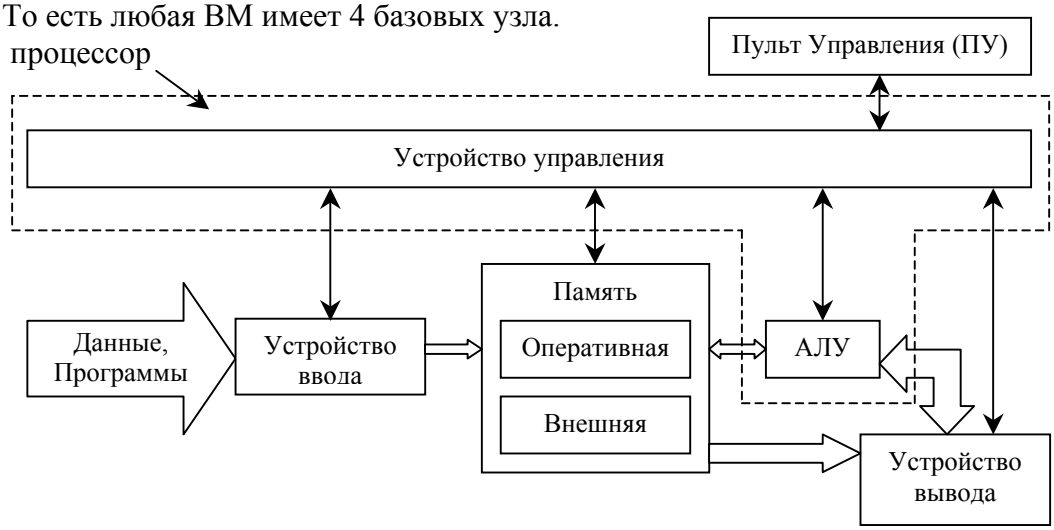
Обобщенная структура ЭВМ.

Принцип действия обычной ВМ можно считать копией обычного процесса вычислений (например, с помощью калькулятора).

Этапы вычислений:

1. Определение и задание порядка вычислений.
2. Задание исходных данных.
3. Выполнение вычислений (для получения промежуточных результатов)
4. Получение конечного результата.

То есть любая ВМ имеет 4 базовых узла.



В основе функционирования любой ВМ лежат два фундаментальных понятия в вычислительной технике.

1. понятие алгоритма.
2. принцип программного управления.

Опр. Алгоритм – некоторая однозначно определенная последовательность действий, состоящая из формально заданных операций над исходными данными, приводящая к решению за конечное число шагов.

Свойства алгоритмов:

1. дискретность алгоритма (действия выполняются по шагам, а сама информация дискретна)

2. детерменированность (сколько бы раз один и тот же алгоритм не реализовывался для одних и тех же данных результат один и тот же)
3. массовость (алгоритм “решает задачу” для различных исходных данных из допустимого множества и дает всегда правильный результат)

Опр. Программа – описание алгоритма на каком-либо языке.

Принцип программного управления (ППУ) впервые был сформулирован Венгерским математиком и физиком Джоном фон Нейманом, при участии Гольцтайна и Берца в 1946 году.

ППУ включает в себя несколько архитектурно – функциональных принципов.

1. Любой алгоритм представляется в виде некоторой последовательности управляющих слов – команд. Каждая отдельная команда определяет простой (единичный) шаг преобразования информации.
2. Принцип условного перехода. В процессе вычислений в зависимости от полученных промежуточных результатов возможен автоматический переход на тот или иной участок программы.
3. Принцип хранимой программы. Команды в ЭВМ представляются в такой же кодируемой форме, как и любые данные и хранятся в таком оперативном запоминающем устройстве (ОЗУ). Это значит, что если рассматривать содержимое памяти, то без какой-то команды невозможно различить данные и команды. Следовательно, любые команды можно принципиально обрабатывать как данные (информация в ЭВМ отличается не представлением, а способом ее использования).
4. Принцип двоичного кодирования.
5. Принцип иерархии запоминающих устройств (ЗУ).

§2 История развития ВТ. Поколения ЭВМ.

История ВТ отсчитывается с опубликования работы Джона фон Неймана. Впервые возможность построения цифровой ВМ была доказана английским математиком Тьюрингом в 1936 году. Он показал, что любой алгоритм реализуется с помощью его дискретного автомата, который был назван машиной Тьюринга. Независимо это же доказал Пост (машина Поста).

Физически первая цифровая ВМ была сконструирована в 1935 году фирмой Белл (США). Такого же вида машина была сконструирована для специальных задач под руководством К. Цунзе (1941, Германия). Попытка построения универсальной ЭВМ была предпринята Айтнетом (США). Она получила название “Марк-1”. Спроектирована и изготовлена в Гарвардском университете.

Характеристики ВМ (работали с 23 разрядными десятичными цифрами):

1. Программа вводилась покамандно с перфоленты.
2. Сложение 2-х чисел 0.3 секунды
3. Умножение 2-х чисел 6 секунд
4. Деление 2-х чисел 11 секунд.

Релейная основа была ненадежна. Для ЭВМ были разработаны специальные реле. На которых была разработана ВМ “Марк-2”.

Реальный отсчет ВТ ведется с перехода от реле к триггерам. Триггер был изобретен в 1918 году в России Бонч-бруевичем.

Первая ЭВМ, разработанная на электронных компонентах, изготовлена в 1942 году (“Эниак”). Серийный выпуск в 1945-1946 годах. Разработана в Пенсельванском университете под руководством Маушли и Энкера. В 1943 году под руководством Тьюринга была разработана ЭВМ “Колос”. После рассекречивания архивов в 70-х годах оказалось, что первая ЭВМ была разработана в 1939 году выходцем из Германии Антоносовым, которая получила название “АВС”.

Первое поколение ЭВМ.

Ламповые ЭВМ, промышленный выпуск начат в начале 50-х годов.

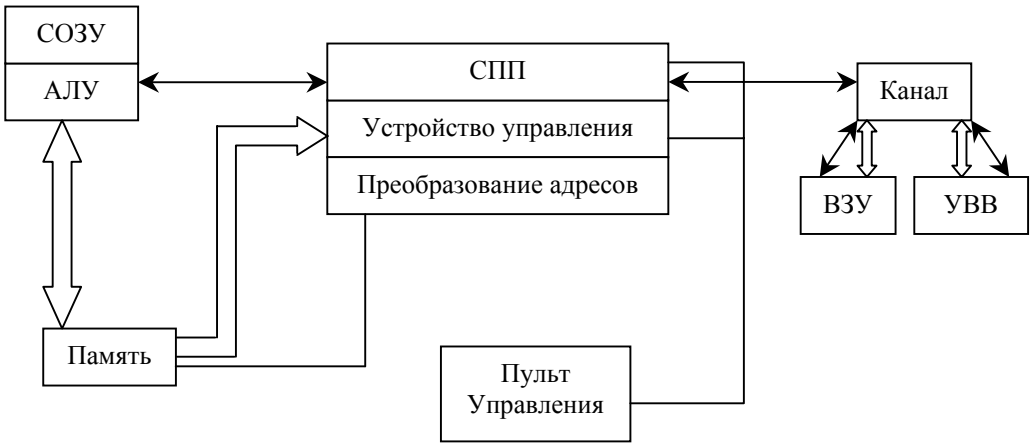
В нашей стране началом выпуска можно считать начало 50-х годов “МЭСМ”. Разработана под руководством Лебедева. В 1952-1953 годах на этой основе, под руководством Мельникова и Бурцева была разработана “БЭСМ-1” (Большая электронная счетная машина). А на ее основе был произведен серийный выпуск машины “БЭСМ-2”. В это же время в США выпускают машину “Эдвак”. Технические характеристики машины “БЭСМ-2” были гораздо выше. Это было связано с тем, что в “БЭСМ-2”, использовались два совершенно новых принципа: конвейеризации и стека. Для “БЭСМ-2”, быстродействие АЛУ составляло порядка 10000 операций в секунду.

В 1953 году была разработана машина “Стрела” под руководством Василевского. А так же в Московском Энергетическом институте под руководством академика Брука были разработаны ЭВМ получившие название “М”. В Минске был создан завод по производству ЭВМ, серийное производство машин “Минск”. В городе Пензе было создано ОКБ (отдел конструкторского бюро) под руководством академика Рамеева, где разработали и выпускали серийно ЭВМ “Урал”.

Структура ЭВМ первого поколения полностью соответствовали машине фон Неймана. Технические характеристики машин были значительно ниже характеристик современных ПК. Программирование велось в машинных кодах. Емкость ОЗУ – 2 тысячи слов. Ввод информации с перфоленты и кинопленки.

Второе поколение ЭВМ.

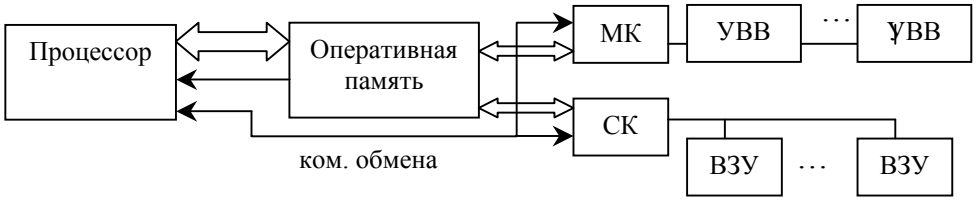
Связывают с переходом от ламповых к транзисторным ЭВМ. Транзисторы позволяли обеспечить большую надежность, быстродействие и меньшее энергопотребление (среднее время отказа около 100 часов, тогда как на машинах первого поколение около 10 часов, энергоемкость на два порядка ниже, по сравнению с машинами первого поколения). Переход к печатному монтажу также улучшило надежность.



Начинается бурное развитие математического и программного обеспечения. Высшая точка: создание алгоритмических языков (Fortran, ALGOL). Создаются простейшие компиляторы и интерпретаторы. Становится нецелесообразна работа пользователя у пульта управления. Основным режимом становится работа через операторов. Появляются многопрограммные ЭВМ. Многопрограммность достигается за счет программной обработки. Для работы в пакетном режиме создаются первые мониторы и supervisor’ы. Вследствие чего происходит резкое увеличение использование ЭВМ второго поколения.

Третье поколение ЭВМ.

В конце 60-х годов появляются первые машины третьего поколения. Переход к третьему поколению ЭВМ связывают с серьезными архитектурными изменениями. Изменение технической базы связано с переходом на интегральную схематехнику. Правда степень интеграции была небольшой. Вследствие чего произошло заметное увеличение надежности. В машинах третьего поколения формируется концепция канала, начинается работа с распараллеливанием процессора, появляется микропрограммное управление, иерархируется память, впервые вводится понятие агрегатирования.



МК – мультиплетный канал (медленные устройства)
СК – селекторный канал (высокоскоростные устройства)
Канал является основным структурным элементом.
В структуре процессора и оперативной памяти появляются специальные устройства, которые организуют адресные механизмы (обеспечивающие адресацию, перемещение программы в памяти, взаимную защиту). В процессоре появляется несколько АЛУ (целочисленные, с плавающей арифметикой, для работы с адресами). Правда, эти устройства параллельно не работают, но для выполнения той или иной обработки выбирается определенное АЛУ. В памяти четко выделяется основная память, к которой процессор обращается непосредственно, и массовая память, емкость которой значительно больше емкости основной памяти, но непосредственно процессору она недоступна. Тем более данные с внешних устройств непосредственно недоступны процессору. Так как память иерархична, то создаются механизмы для управления памятью. Развивается и внутренняя память процессора (создаются предпосылки кэширования). В конце третьего поколения ЭВМ появляется концепция управления виртуальной памяти, развиваются внешние устройства и терминальное оборудование. Самое **главное** в тот период: унификация ЭВМ по конструктивно – технологическим параметрам. ЭВМ третьего поколения начинают выпускаться сериями или семействами, совместимыми моделями.

Дальнейшее развитие математического и программного обеспечения приводит к созданию пакетных программ для решения типовых задач, проблемно – ориентированных программных языков (для решения задач отдельной категории) и впервые создаются уникальные программные комплексы, – операционные системы (разработаны IBM).

Четвертое поколение ЭВМ.

В конце 70-х кодов появляются первые ЭВМ четвертого поколения. Связано с переходом на интегральные схемы средней и большой степени интеграции.

Характерные свойства ЭВМ четвертого поколения:
1. Мультипроцессорность
2. Параллельно – последовательная обработка
3. Языки высокого уровня
4. Появляются первые сети ЭВМ

Технические характеристики ЭВМ четвертого поколения:
1. Средняя задержка сигнала 0.7 нс./вентиль (вентиль – типовая схема)
2. Впервые основная память – полупроводниковая. Время выработки данного из такой памяти 100-150 нс. Емкость 10^{12} –10^{13} символов.
3. Впервые применяется аппаратная реализация оперативной системы
4. Модульное построение стало применяться и для программных средств

Основная внимание машин четвертого поколения было направлено на сервис (улучшение общения ЭВМ и человека).

Пятое поколение ЭВМ.

В конце 80-х годов появляются первые ЭВМ пятого поколения.

Пятое поколение ЭВМ связывают с переходом к микропроцессорам. С точки зрения структурного построения характерна максимальная децентрализация управления. С точки зрения программного и математического обеспечения – переход на работу в программных средах и оболочках. Производительность

10^8 - 10^9 операций в секунду. Для пятого и шестого поколения характерны многопроцессорные структуры созданные на упрощенных микропроцессорах, которых очень много (решающие поля или среды). Создаются ЭВМ ориентированные на языки высокого уровня.

В этот период существуют две диаметрально противоположных тенденции:

1. Персонификация ресурсов
2. Коллективизация ресурсов (коллективный доступ – сети)

§3 Классификация и основные характеристики ЭВМ.

Характеристики:

1. Операционные ресурсы ЭВМ – это (грубо говоря) перечень возможностей ЭВМ. Сюда включаются:
    1. Способы представления информации в ЭВМ
    2. Система команд ЭВМ
    3. Способы адресации

    Операционные ресурсы ЭВМ напрямую связаны с аппаратными средствами, которые характеризуют степень приспособленности ЭВМ для решения тех или иных задач.
2. Емкость памяти (внешняя и основная) Основная память, какой бы большой она не была, всегда ограничена. Внешняя память не ограничена. Для характеристики компьютера используют емкость основной памяти. Использование памяти идет многобайтно, следовательно, доступ измеряется в байтах (максимальная память 4Гб). Внешняя память – суммарная емкость всех накопительных устройств. Следовательно, необходимо использовать косвенную характеристику – количество накопителей подключаемых к ЭВМ. В современных компьютерах есть также и сверхоперативная память (cashe), ее объем – один из важнейших параметров влияющих на время решения задачи.
3. Быстродействие ЭВМ характеризует скорость обработки информации компьютером (число операций в секунду (V), время выполнения ($\tau=1/v$)). Но для различных операций эти показатели различны, следовательно, реальная характеристика – номинальное быстродействие ($V_н$)– количество коротких операций в единицу времени (обычно берут операцию “+”, а операнды хранятся во внутренних регистрах процессора (R-R)). Иногда также используют в качестве характеристики быстродействия – цикл обращения к основной памяти, а также эффективное быстродействие ($V_ф$) $V_ф=1/\sum p_i\tau_i$ p_i – вероятность выполнения i-ой операции. По содержанию производительность ЭВМ – это среднее число операций в единицу времени.

    Производительность ЭВМ зависит от:
    1. Быстродействия процессора
    2. Класса решаемых задач
    3. Порядка прохождения задачи через ЭВМ

    Для оценки числового выражения эффективности ЭВМ используют смеси команд.

    Для научно-технических расчетов используют “Смесь Гибсона”

Вид команды	Весовой коэффициент
“+”,”-“ фикс. зпт.	33
“*” – фикс. зпт.	0.6
“/” – фикс. зпт.	0.2
“+”,”-“ плав. зпт	7.3
“*” - плав. зпт.	4.0
“/” - плав. зпт.	1.6
Логические операции	1.7
Безусловный переход	17.5
Условное ветвление	6.5

$P=\sum K_з/\sum K_з\tau_з$ – для n задач.

4. Надежность ЭВМ. Надежность – свойство ЭВМ выполнять возложенные на нее функции в течение заданного промежутка времени, необходимого для решения поставленной задачи. В процессе функционирования ЭВМ возникают отказы, связанные с неисправностью отдельных элементов либо соединений между ними.

   По характеру проявлений отказы могут быть:

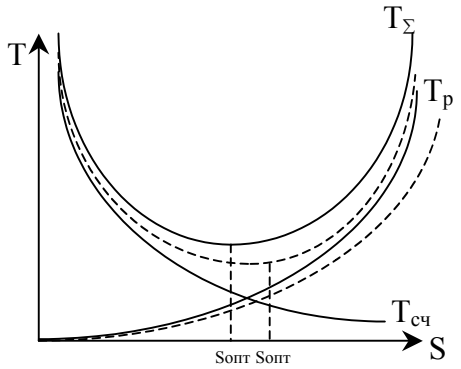
   1. Внезапный отказ (механическое разрушение элементов)
   2. Постепенный отказ (деградация параметров ЭВМ)

   С точки зрения математического подхода – отказы это случайное событие. Используется самая простейшая математическая модель – "Простейший поток отказов". Если поток отказов простейший, то в качестве характеристики надежности используется величина интенсивности потоков отказа. $\lambda=1/T_р$ $T_р$ – среднее время безотказной работы между двумя очередными отказами. Если ЭВМ можно ремонтировать, то после находки отказа – работоспособность компьютера восстанавливается ($T_в$ – среднее время восстановления) $T_в$ – фактически время, которое происходит от момента обнаружения отказа до полного восстановления работоспособности. Для компьютеров с простейшим потоком отказов в качестве показателя используют показатель готовности: $К_г = T_р/(T_р+T_в)$ который характеризует вероятность того, что в данный момент времени компьютер готов к решению требуемой задачи.
5. Показатель стоимости – суммарная стоимость всего оборудования, входящего в состав ЭВМ. Если возрастает количество оборудования ЭВМ, то в конечном итоге, будет расти не только стоимость, но будет расти и ее производительность. Путем статистического анализа была выведена связь между стоимостью и производительностью. Впервые это было установлено Найтом и получило название "Закон Гроша". ($V=kS^2$)

   k – константа определяется эмпирически

   Вывод, если не менять технологическую базу компьютеров, то:

   1. При росте стоимости ЭВМ растет количество оборудования и, следовательно, снижается скорость решения задачи.
   2. При росте стоимости ЭВМ растет объем оборудования и, следовательно, увеличивается время ремонта.



------ при улучшении технологической базы

$T_Σ=T_{сч}+T_р$ т.е. для данного уровня технологии всегда есть некоторая оптимальная стоимость, которая дает лучшие технические характеристики.

Классификация ЭВМ.

ЭВМ классифицируются по:

1. Назначению. Обычно выделяют ЭВМ общего применения и ЭВМ ориентированные на вполне определенный класс задач.
2. Производительности: ЭВМ подразделяются по величине производительности.
3. Режимам работы:

   а) однопрограммные ЭВМ

   б) мультипрограммные ЭВМ (Эти ЭВМ должны иметь большую оперативную память, средства управления временем, ввода-вывода, средства позволяющие исключить влияния программ друг на друга)

в) ЭВМ для построения много машинных и многопроцессорных вычислительных систем (дополнительно к мультипрограммным ЭВМ должны реализовывать функции взаимного обмена между ЭВМ)

г) ЭВМ для работы в системах реального времени. (Говоря о машинах реального времени наиболее очевиден пример, когда ЭВМ управляет техническим объектом (автопилот). К ним предъявляют требования быстродействия и способность получать массу сигналов от внешних источников)

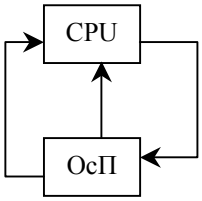
4. Способ структурной организации.

Для увеличения скорости ЭВМ в ее состав включают несколько процессоров. Различают:

а) Однопроцессорные ЭВМ

б) Мультипроцессорные ЭВМ (можно также выделить квазипроцессорные ЭВМ), состоят как из однотипных, так и из разнотипных процессоров (неоднородные ЭВМ). Основная цель мультипроцессирования – получение сверх высокой производительности вычислительных систем (ВС). Как правило, такие системы содержат несколько десятков, сотен или тысяч сравнительно простых процессоров, и их число позволяет увеличивать производительность. Принципиально такие системы ориентируются на большой круг задач, которые допускают эффективное распараллеливание вычисление на регулярную структуру (связи между процессорами, как правило, фиксированы). Вообще-то не каждая задача достаточно хорошо распараллеливается на заданную ВС. ВС с параллельной обработкой также классифицируются. В качестве такой классификации выступает классификация по Флину. В ее основе лежит способ организации параллелизма ВС (множественность). Этот параллелизм определяется как максимальное число одновременных команд или операндов, которые находятся на одинаковой или какой-то определенной стадии выполнения. Согласно Флину существует 4 разновидности ВС:

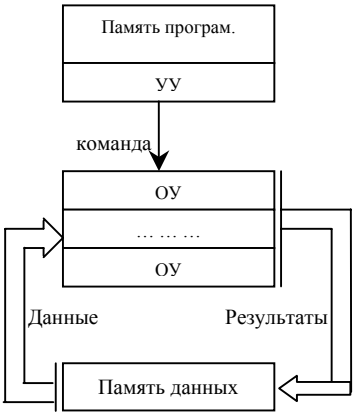
1. ОКОД (SISD, одиночный поток команд одиночный поток данных). Такое структурное построение характерно для классических машин фон Неймана.



Функционирование в виде линейного процессора.
ОУ, ОсП (основная память), УУ

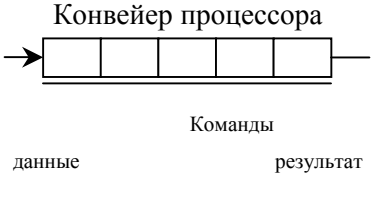
Линейная организация вычислительного процесса обуславливает весьма низкую эффективность аппаратных средств (велик коэффициент простоя) Для повышения работы такой структуры применяются методы локального параллелизма – совмещенная или опережающая выборка команд, расслоение памяти, но, как правило, это требует дополнительных аппаратных затрат.

2. ОКМД (SIMD, одиночный поток команд множественный поток данных). Для данной ВС обычный поток команд воздействует на несколько процессорных блоков одновременно, которые обрабатывают различные данные по одной команде. Память в такой ВС является разделенной.



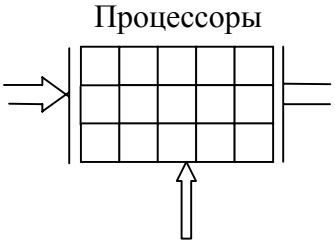
Первоначально типовыми представителями таких ВС были супер-ЭВМ (ILLIAC IV, STARAN, PEPE, ПС-300). ВС с такой структурной организацией направлены на решение задач с естественным параллелизмом. В современных ЭВМ это реализовано в Pentium MMX.

3. МКОД (MISD, множественный поток команд одиночный поток данных). Эту ВС обычно рассматривают как результат идей локального параллелизма. Иначе их называют конвейерные ВС. Операционная часть Вс является регулярной и представляет собой цепочку последовательно (линейно) соединенных процессорных блоков, которые образуют конвейер процессора.



Данный конкретный блок является специализированным и выполняет вполне определенную часть команды. Впервые такую ВС разработал академик Лебедев.

4. МКМД (MIMD, множественный поток команд множественный поток данных) – общий случай мультипроцессорной системы. В общем случае связи между элементарными процессорами являются перестраевыми.

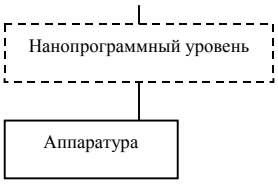


Такая ВС позволяет повысить не только производительность, но и надежность. Как правило отказ одного процессора не приводит к выходу из строя всей системы. При такой организации ВС возникают сложности взаимодействия управления, при решение одной задачи.
Иногда MIMD называют «моделью коллектива вычислителей»

§4 Уровни организации ЭВМ.

Основой функционирования любой ЭВМ является ее способность выполнять заданные действия. Аппаратные средства любой ЭВМ способны выполнять только ограниченный набор сравнительно простых команд. Эти примитивные команды примитивные команды составляют так называемый машинный язык машины. Говоря о сложности аппаратуры компьютера, машинные команды целесообразно делать как можно проще, но примитивность большинства машинных команд делают их использование неудобным и трудным. Вследствие чего разработчики вводят другой набор команд более удобный для человеческого общения (языки более высокого уровня).

Рисунок показывает, что между языками программирования и существующей виртуальной машиной существует тесная связь, но в общем случае она является нелинейной.
Пользователь, который работает на каком-то уровне в принципе Может и не знать процессы, происходящие на других уровнях организации, но для составления эффективных программ, необходимо знание



более низких языков программирования. Большинство современных ЭВМ включают 6-7 уровней виртуализации. Нижние уровни, начиная с машинного более консервативны к изменениям.

Многие коммерческие ЭВМ принципиально могут не иметь уровня нанопрограммирования, но и даже микропрограммирования (по крайней мере, на уровне пользователя). В современных ЭВМ машинные команды, как правило, интерпретируются с помощью микропрограмм. Уровень ОС обычно является смешанным уровнем, т.к. большинство супервизорных команд являются командами машинного уровня. В состав уровня ОС дополнительно включаются команды, которые фактически являются некоторыми типовыми программами машинного уровня (команды ввода-вывода, переключения между программами). Во многих ЭВМ были варианты, когда отдельные программы ОС непосредственно интерпретируются микропрограммами. В современных ЭВМ прослеживается тенденция все боле тесной связи уровня ОС с микропрограммным уровнем. Можно также найти массу команд уровня ОС, реализованных, на уровне ассемблирования. Простые пользователи, как правило, ограничиваются уровнем изучения начиная с машинного. Нижние уровни необходимы для разработчиков.

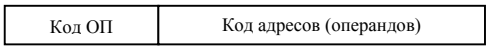
Глава 1
Машинный уровень организации

Форматы команд.

Любая команда ЭВМ представляет собой некоторую упорядоченною последовательность битов, которая определяет:

1. Операцию, инициируемую этой командой.
2. Адреса операндов участвующих в этой операции.

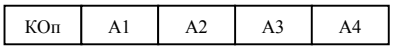
Поэтому в большинстве ЭВМ команда имеет операционно-адресную систему.



В операционной части с точки зрения машинного представления записывается код операции. А в адресной части задается код адресов операндов. Он содержит информацию не только об адресах операндов и результата операции, но и об адресе следующей команды.

Под **форматом команды** понимается состав, назначение и расположение отдельных полей команды.

Развитие структуры происходит лишь вследствие уменьшения поля адреса (SISC процессоры). С появлением RISC процессоров произошел возврат к расширенной адресности системы команд. Говоря о базовом уровне, речь идет о SISC процессорах. Первоначально система команд имела следующий формат:



$A_3 := (A_1) * (A_2)$ СчАК:=A_4 СчАК – счетчик адреса команд.

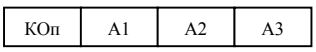
$*$ - операция

(x) – содержимое адреса x

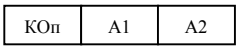
В каждой команде задается адрес следующей команды (это самый универсальный вариант), но длина такой команды велика. При такой системе команд их можно располагать в любом порядке, но обычно стараются расположить последовательно. Такой порядок называется **естественным**, но при естественном расположении необходимость в поле A_4 отпадает. Следовательно, формат приобрел вид:

$A_3 := (A_1) * (A_2)$

СчАК := (СчАК)+1



Даже переход к такому формату слабо уменьшает длину команды. Большинство вычислений имеет рекуррентную схему вычислений, следовательно, систему команд можно сделать двух адресной.

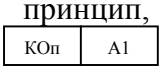


$(A_1) := (A_1)*(A_2)$
СчАК := (СчАК)+1

Такая схема вычислений является основной для современных ЭВМ.

Но развитие шло и в направление развития процессора и его внутренней памяти, это привело к тому, что в процессорах появились специальные регистры, которые всегда используются в вычислениях. АСС := (АСС)$*$Y

Если использовать аккумуляторный принцип, то для системы команд достаточно использовать один адрес.



Основная масса команд SISC процессора является одно и двух адресная. В некоторых случаях команды можно не адресовать, если команда задает операцию над фиксированными адресами (например, действие над аккумуляторами), в формате таких команд адресное поле отсутствует. Команда называется **безадресной** или **нульадресной**. Качественный сравнительный анализ показывает:

1. Короткие команды предпочтительнее длинных, так как занимают меньше памяти. Любая память ЭВМ характеризуется скоростью передачи (пропускной способностью). Если физическое быстродействие фиксировано, то количество выбираемых из памяти команд, обратно пропорционально длине команды. А значит скорость работы процессора для коротких команд больше, чем для более длинных. Короткие команды увеличивают производительность процессора. Для большинства современных ЭВМ процессор работает быстрее памяти, следовательно, чтобы обеспечить высокую производительность, нужно увеличить количество команд извлекаемых из памяти за один цикл обращения.
2. Форматы команд должны обеспечивать достаточное пространство для задания всех операций (если система компьютера включает в себя N операций n_{min}=] $\log_2 N$ [- округление в большую сторону)
3. Длина команды должна быть кратна длине базовой структурной информационной единице (т.е. либо команда должна занимать целое число байт или слов, или в одном слове должно находится целое число команд). Если в одном слове не целое число команд, то либо память недоиспользуется, либо усложняются процедуры выбора команд. Кроме того, длина команды должна выбираться с учетом длины кодов символов обрабатываемых данных.
4. Длина адресного поля команды очень тесно связана с организацией памяти компьютера, а также с размером адресного пространства памяти, которую можно непосредственно адресовать в памяти компьютера. Если память включает в себя M адресных элементов, тогда минимальная длина адресного поля: m_{min}=]$\log_2 M$[. Если исходить от фиксированной длины адресного поля, то емкость памяти будет зависеть от того, какова минимально адресуемая единица информации.

Имеется два способа решения задач при ограниченном командном слове:

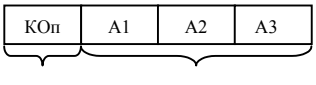
1. Применением команд переменной длины.
2. Расширением кода операций.

Для задания часто используемых операций, как правило, используют короткий код операций, а в сочетании с коротким адресным полем получаем короткие команды, а значит:

1. Хорошее использование памяти
2. Максимально возможная скорость обработки

Для задания других менее часто используемых операций и действий, используются более длинные поля кодов (может быть без изменения длины команды).

Пусть имеется двух байтная команда:

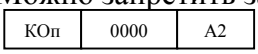


$(A_3)*(A_2) \Rightarrow A_1$

0,5б 1,5б

Как можно не изменяя длины команды изменить длину адресного поля?

Можно запретить запись в ячейку с адресом 0000. A_1 – адрес приемника.

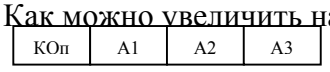


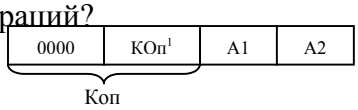
Формат адресной части изменился, и адресное пространство возросло до 256 элементов. Но большинство команд относящихся к SISC процессорам имеет переменную длину команд:

0000 – обозначает что A_1 и A_2 возросли в два раза.

Если требуется еще больше расширить адресное пространство, то можно, используя, принцип переменной длины команды, определить, что 0000 означает, что длина адресного поля, например, стала 5 байт.

Как можно увеличить набор операций?





Например, это означает, что КОп, занимает целый байт. Следовательно, стало 31 команд (15 трех адресных и 16 двухадресных команд). Так можно продолжать и дальше.

Адресация данных и команд.

Любая машинная команда – машинный код, который определяет:

1. Операцию
2. Указывает на данные

В адресной части команды хранится адресный код. В большинстве случаев фактическое обращение к данным происходит по физическому (исполнительному) адресу. Обычно физический адрес не совпадает с адресным полем команды, но зависит от него. В общем случае происходит преобразование из адресного кода в физический код – режим (способ) адресации.

Способы адресации являются одним из основных архитектурных признаков. В настоящее время известно более двух десятков различных способ адресации и их модификации. Все известные способы адресации данных разделены на две большие группы.

1. Прямые
2. Не прямые

При прямых способах адресации либо накопительный адрес операнда, либо сам операнд, находятся непосредственно по адресному коду без всякого преобразования. Не прямые способы требуют выполнения процедур формирования физического адреса по адресному коду, для этого в ЭВМ встраивается специальный адресный механизм.

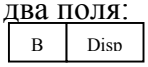
Прямые способы адресации.

1. Неявная адресация. В таких командах явного адресного поля нет (нуль адресная команда). Операнд задается кодом операции. Обычно такой способ адресации используется для фиксированных программно доступных регистров процессора (аккумуляторный принцип).
2. Непосредственная адресация. В адресном поле фактически указывается не адресный код, а сам операнд. Этот способ не требует дополнительных обращений к памяти за операндами, но адресное поле должно иметь длину операнда. Обычно применяется для задания констант вычисления.
3. Абсолютная (прямая) адресация – характеризуется тем, что в адресном поле, задается полный адрес памяти, где хранится операнд. В этом случае. длина адресного поля и емкость оперативной памяти связаны между собой соотношением. $m =]\log_2 E_m[$ Если Em очень большая, то длина адресного поля в команде большая. Способ абсолютной адресации данных – тормоз в развитии применения компьютеров. Он не позволяет загружать данные в любое место памяти. В современных условиях применяются в ограниченном количестве (при загрузке драйверов).

Все современные ЭВМ используют массу способов непрямой адресации. Они позволяют обеспечить мобильность программных средств.

Непрямые способы адресации:

1. **Базирование** (относительная адресация). Процедура формирования исполнительного адреса: $A_{исп} = A_{база} +$ <смещение>. Для реализации этого способа в ЭВМ выделяются специальные ячейки, которые выполняют функции базовых регистров. В общем случае в ЭВМ может быть несколько базовых регистров. Тогда адресные код включает в себя два поля:



B – адрес регистра базирования

Disp – смещение.

В общем случае исполнительный адрес формируется соотношением:

$$A_{исп} = \begin{cases} (B)+Disp \\ Disp \end{cases}$$

Смещение, которое задается, может быть длины адреса, но может быть и короче. С точки зрения длины команды короткое смещение предпочтительнее.

Базирование, как способ адресации требует наличие сумматора в адресном устройстве. Вследствие чего очень часто в адресных механизмах операцию суммирования сводят к операции конкатенации (присоединительная адресация). Правда в этом случае как правило накладываются определенные ограничения на регистры базирования. Связано это с тем, что базовый разряд определяет только старшие разряды исполнительного адреса, младшие разряды всегда равны нулю.

$A_{исп} = (B).Disp$

Время вычисления адреса значительно улучшается. Любая схема базирования обеспечивает перемещение программ по памяти ЭВМ. Для ее исполнения в области памяти необходимо только установить базовый адрес. Относительная адресация облегчает задачу программирования различными программистами. В простейшем случае в компьютере регистр базирования единственен, следовательно, в адресном коде отсутствует адресация. На каждом шаге выполнения программы, как минимум необходимо два регистра базирования. Один для задания базового адреса программы, второй для базирования массивов данных. Минимальное число регистров базирования один, но реально нужно четыре-восемь.

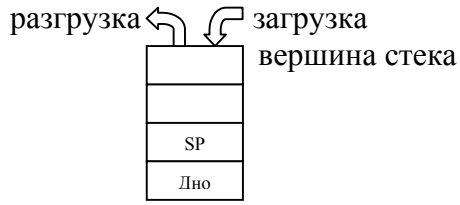
2. **Косвенная адресация.** При косвенной адресации адресный код в команде содержит не адрес самого операнда, а содержит адрес памяти, где хранится адрес операнда. Можно сказать, что **адресный код** – это адрес адреса. В самом простом варианте исполнительный адрес при косвенной адресации имеет следующий вид: $A_{ипс} = (M[A_k])$ M – адрес памяти A_k – содержимое. В общем случае может использоваться много ступенчатая косвенная адресация. $A_k \rightarrow A_{k1} \rightarrow A_{k2} \rightarrow ... \rightarrow A_{исп} \rightarrow$операнд. Количество обращений к памяти для получения операнда, характеризует глубину косвенной адресации. Минимальное число обращений для получения операнда – два. Для получения исполнительного адреса не требуется арифметическая обработка. Фактически нахождение операнда – процедура целенаправленного поиска. Косвенная адресация позволяет без изменения самой команды обрабатывать ей различные данные, так как фактически другие операнды находятся в памяти, а не в команде, значит можно обрабатывать линейные структуры данных. Косвенная адресация позволяет загружать программу в произвольное место памяти.

   Недостатки косвенной адресации: Если указатели косвенного адреса указывают на исполнительный адрес в памяти, то резко замедляется скорость данной адресации. Обычно используют разновидность косвенной адресации, когда указатель косвенного адреса – это адрес регистра процессора (укороченная адресация).

3. **Автоинкрементная, автодекрементная (индексная) адресация.** К необходимости введения такого способа адресации приводят задачи обработки данных, хранящихся в последовательно расположенных ячейках памяти. При обработке таких данных, адрес данного меняется по правилу счета. Такая рекуррентная схема привела к появлению индексной адресации. В тех ЭВМ, где изменение указателя адреса при

обработке данных делается автоматически, там индексация называется автодекрементной или автоинкрементной. Название лишь указывает направление изменения адреса (+1, -1). Данный способ адресации значительно упрощает программирование вычислительных циклов, хотя исторически, изменение исполнительного адреса, могло производиться за счет изменения текущего адресного кода в команде. Поскольку согласно принципу фон Неймана команды и данные в памяти не различаются друг от друга, то над кодом команды можно выполнять все те же операции, что и над данными, но изменение адресного кода команд приводит к тому, что программа становится неперемещаемой. Но модификацию адресного кода не применяют (проблемы с отладкой при сбое), хотя такая возможность есть. Ее можно использовать в тех программах, которые загружаются в фиксированную область памяти.

4. **Укороченная адресация** – всевозможные способы, ориентированные на уменьшение длины команды за счет сокращения адресного кода. Для современных ЭВМ укороченная адресация привела к тому, что базовые адреса, указатели косвенного адреса, указатели индексов при индексации хранятся либо в фиксированных ячейках памяти, либо в фиксированных регистрах процессора. В последнем случае в адресном поле команды задается короткий адрес регистра. Это позволяет не только сократить длину команды, но и уменьшить количество обращений к основной памяти, так как при упорядоченной адресации указатель извлекается из регистра, что гораздо быстрее. Дополнительное обращение к памяти исключается.
5. **Стековая адресация**. При использовании стековой адресации, команды не имеют адресного поля (безадресные) для задания адресов операндов. Стековая адресация – очень эффективный способ и применяется в большинстве ЭВМ. Стек может реализовываться либо аппаратными, либо программными средствами. Рассмотрим стек, реализованный программно-аппаратно. Стек – некоторая область памяти в общем пространстве, доступ к ячейкам этой области осуществляется с помощью указателя стека.



Sp – указатель стека либо указывает адрес загруженной команды, либо первой свободной ячейки. Доступ к данным в стеке, только по очереди, начиная в вершины. Помещение данных в стек – загрузка стека. Извлечение данных – разгрузка данных. Расположение данных в стеке строго упорядочено. В любой момент времени можно либо считать верхнее данное, либо загрузить поверх него другое. При каждом обращении к стеку, указатель стека автоматически корректируется на величину, равную длине данного. Данное, извлеченное из стека, как бы в стеке теряется. Для извлечения произвольного данного из стека необходимо предварительно удалить все вышележащие данные. Такой механизм не требует адресного поля в команде, команды фактически безадресные. Для обеспечения эффективной работы стека необходимо чтобы обрабатываемые данные были структурированные. Исторически стековая адресация использовалась при конструировании трансляторов.

Преимущества стековой адресации:

1. Безадресные команды
2. Перемещаемость команд

Недостатки стековой адресации:

1. В адресном механизме аккумуляторный узел управления.
2. При использовании стека, как средства адресации команд, возникают сложности при реализации ветвлении

В современных ЭВМ стековый механизм является одним из основных при работе с подпрограммами, а также при организации прерываний.

Адресация команд.

Под адресацией команд понимается способ вычисления следующей команды.
Принудительная адресация. Адрес следующей команды в самой команды.
Недостатки:

1. Неперемещаемость программы.
2. Команды, с точки зрения длины формата, длинные.

При написании программы в адресном поле следующей команды размещается следующий адрес, что неэффективно. Программист, как правило, располагает команды последовательно, в соседних ячейках памяти. Значит адрес следующей команды вычисляется просто по адресу текущей программы. Для этого в состав процессора вводится специальный узел (программный счетчик PC, СчК, СчАК), обеспечивающий последовательную адресацию команд (естественная адресация).
Схема формирования адреса следующей команды: PC := (PC) + l_k

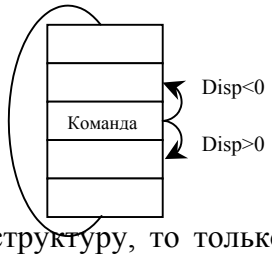
l_k – длина предшествующей команды

Однако, при осуществлении ветвления, в программах реализуются циклические участки программы выполняющие обращения к процедурам и подпрограммам, следовательно, естественный порядок выполнения команд, безусловно нарушается. Для этого используются специальные команды – команды передачи управления. Если в командах с естественным порядком нет адресного поля, то в командах передачи управления, адресное поле обязательно, в нем, в общем случае задается адресный код, на основе которого, при выполнении этой команды формируется физический адрес последующей команды.
Способы формирования исполнительных адресов команд различны:

1. **Неявная адресация.** По существу адресное поле в команде управления отсутствует. Адрес следующей команды извлекается из фиксированных ячеек или адрес фиксирован. Обычно, неявная адресация используется для фиксированных причин прерывания
2. **Абсолютная адресация.** В адресном поле команды передачи управления, указывается полный адрес следующей команды.

Более распространены непрямые способы адресации, обычно применяют три способа:

1. **Относительная адресация –** адресация относительно текущего адреса. В адресном поле команды задается относительный адрес перехода.



Если память имеет кольцевую структуру, то только сложением можно добиться перехода в любое место памяти. Но если в распоряжении пользователя достаточно много памяти, то в этом нет необходимости. Кроме того, есть множество программ, к которым пользователи доступа не имеют, поэтому реально команды с относительной адресацией имеют упорядоченное смещение. Если величина смещения меньше разрядности адреса и смещение со знаком, то для выполнения сложения требуется дополнительное расширение второго операнда. Такая относительная адресация позволяет иметь короткие команды передачи управления

2. **Косвенная адресация.** В схеме косвенной адресации, память для хранения адресов не применяется. Применяется только косвенная регистровая адресация, где хранят адрес перехода. Косвенная адресация требует короткого адресного поля. Позволяет перемещать программы в любое место памяти. Если еще допустить применение относительной адресации, то исполнительный адрес следующей команды вычисляется по схеме. PC := (reg)+disp.

3. **Стековая адресация.** Стековая адресация применяется прежде всего при работе с подпрограммами и обслуживании прерываний. Команды передачи управления со стековой адресацией фактически безадресные. Схема работы стека, как и у данных. Стек в этом случае является универсальным и используется, как для адресации данных так и для команд. Следовательно, возникают сложности распознавания команд и данных.

Типы машинных команд.

Для современных ЭВМ (с CISC процессорами) имеют от 60 до 120 базовых команд. Базовая команда – команда, которая определяет в процессоре операцию, без учета способов адресации, которые могут быть применены в данной команде, не учитывая какие РОН'ы используются при выполнении программы. Общее количество команд, которое может быть в процессорах, с учетом способов адресации и регистров 250-400. Такое большое количество команд, по замыслу разработчиков, должно сокращать длину программы пользователя, следовательно, уменьшается время решения задачи. Практика же показывает, что программист всем возможным множеством команд никогда не пользуется. Обычно пользователь ограничивается некоторым подмножеством команд, которые он четко понимает и знает. В различных программах, а также в программах для различных ЭВМ, частота появления различных команд оказывается разная. Основной характеристикой любой ЭВМ команды, является ее формат. Все команды любой ЭВМ при рассмотрении разделяют по следующим признакам.

1. Функциональное назначение.
   1.1 Команды передачи данных
   1.2 Команды обработки данных
   1.3 Команды передачи управления
   1.4 Дополнительные команды
2. Адресность команды.
   2.1 безадресные команды
   2.2 одноадресные команды
   2.3 двух адресные команды
   2.4 прочие команды
3. По способам адресации
   3.1 данных
   3.2 команд
4. Способ кодирования операций.
   4.1 Команды с фиксированным полем кода операций
   4.2 Команды с расширяющимся полем кода операций
5. По длине.
   5.1 однобайтные
   5.2 двухбайтные
   5.3 трехбайтные
   5.4 многобайтные.

Функциональное назначение команды определяет ее код операции.

Команды передачи данных – группа команд включающая в себя три подгруппы.

1. Команды передачи кодов внутри процессора. Фактически определяет операцию копирования, т.е. создание новых данных в приемнике (dst). Значение источника (src), как правило сохраняется при командах передачи данных. Правда существуют такие команды, когда источник не сохраняется (команды пересылки). Команды регистровой пересылки, которые обычно имеются в системе команд процессора, обеспечивает либо однонаправленный обмен, либо взаимный обмен. С точки зрения длины команд, самые короткие – команды пересылки, как правило двухадресные. В отдельных случаях, команды могут быть одноадресные, если существует фиксированный регистр адресной команды, не определенный мнемоникой, а машинным форматом.

2. Команды обмена процессора с памятью. Связаны с передачей данных из памяти в регистры и из регистров в память. Очень часто команды этой подгруппы имеют несколько другую мнемонику. ST (store) – передача в основную память. LD (load) – в регистры. Если позволяет мнемоника, то в принципе производится независимая адресация, как источника, так и приемника команд. Если используется команда MOV и возможности адресации достаточно большие, то с помощью команды MOV в отдельных ЭВМ возможна пересылка память – память.
3. Команды передачи кодов между процессором и периферией. Происходит передача данных между процессором и периферийным устройством. В ЭВМ используются два принципиально разных варианта для обмена с внешними устройствами:

    1. Специальные команды ввода-вывода (in, out). Такие команды применяются, если внешнее устройство имеет автономное адресное пространство памяти. В таких командах, как правило, адресуется только один операнд, другой операнд располагается в аккумуляторе.

    2. Единая команда MOV. Используется в тех случаях, если регистры внешних устройств рассматриваются как часть общего адресного пространства компьютера. Это позволяет оперировать с внешними устройствами, как с обычными ячейками памяти компьютера. Разделение внешних устройств и памяти производится на аппаратном уровне. Совмещенное адресное пространство повышает гибкость программирования работы с внешними устройствами, уменьшает набор команд, но несколько уменьшает размеры доступной основной памяти. Наличие единой команды приводит к некоторому увеличению общей длины команды. Все команды передачи данных не формируют и не изменяют значения признаков результата операций.

Команды обработки данных.

Эта группа команд самая большая и самая главная. Разделяется в зависимости от операций, которые выполняются над данными:

1. арифметические
2. логические
3. команды сдвига
4. команды обработки строк.

1. Базовые арифметические команды предназначены для задания арифметических операций над какими-то операндами. Любая арифметическая операция двухместная (пример: ADD dst,src; Схема вычислений: dst:=(dst)$*$(src) (dst)$*$(src)$\Rightarrow$dst). Если использовать аккумуляторный принцип, то ACC:=(ACC)$*$(src). Команды арифметических операций формируют практически всегда признаки результата операций. Базовой арифметической операцией является **арифметическое сложение** (сложение двоичных кодов, т.к. сложение без знаковое). Большинство ЭВМ не ограничиваются операцией сложения, имеется еще вычитание двоичных кодов (SUB dst,src). Эта операция не коммутативная. Сама по себе команда вычитания двоичных кодов обеспечивает вычитание без знаковых кодов. Но сформулировав должным образом коды можно и обрабатывать данные со знаком. Если обрабатывать многобайтные данные, то т.к. система счисления двоичная позиционная, то обработка начинается с младшего разряда. Как правило, для этих случаев имеются специальные команды (ADC dst,src; SBB dst,src; Схема вычисления dst:=(dst)*(src) $\pm$ (CY)). Если в арифметических операциях могут участвовать операнды разной длины, то предварительно короткий операнд увеличивается до длины длинного (для целочисленной арифметики), причем здесь идет выравнивание по правому краю, а расширение идет с помощью знака. Это может реализовываться автоматически, либо за счет команд расширения знака (SXT). Как правило, расширение команды происходит в фиксированном регистре, следовательно, эти команды безадресные. Если этого нет, нужно писать специальные процедуры расширения. **Операция сравнения кодов** (CMP dst,src). По содержанию это команда вычитания. Фактически схема вычисления (dst)-(src). Все

признаки результата по этой команде формируются, а результат никуда не заносится. **Однооперандные** арифметические команды. Т.к. второй операнд имеет фиксированное значение (как правило, оно =1), тогда команды INC dst; DEC dst; , а схемы вычислений dst:=(dst) $\pm$1. Эти команды очень используемы при разработке счетчиков, индексов. Не применимы для многобайтных данных, позволяют сохранить признаки для следующих ветвлений. **Команды умножения и деления.** В системах команд малых ЭВМ эти команды отсутствуют, но если в системе команд имеются эти команды, то они применяются для беззнаковых данных. Формат команды: MUL dst,src; DIV dst,src; Если взять команды ADD и SUB, то форматы результата и операндов практически совпадают, здесь же они не совпадают принципиально. Для хранения произведения обычно используются фиксированные регистры, чаще всего это аккумулятор с расширителем. Аналогично для целочисленного деления. В общем случае делимое имеет двойную длину, следовательно, как правило, приемник результата тоже фиксированный регистр (аккумулятор с расширителем). Очень часто в системах команд вводят команды умножения и деления с учетом знака (IMUL dst,src; IDIV dst,src;). Эти команды обычно ориентированны на использование базовых форматов компьютера. Для обработки многобайтных данных при умножении и делении, сначала делают декомпозицию, а затем строить умножение или деление многобайтных данных. Если машина ориентирована на научно технические расчеты, то требуются операции над данными с плавающей запятой и там они имеются. Говоря о малых ЭВМ, то, собственно говоря, команд с плавающей арифметикой нет. Но операция сложения и вычитания выполняются подпрограммами или с помощью подпроцессора. **Команды десятичной арифметики.** Основу десятичной арифметики в любых ЭВМ составляют команды двоичной арифметики. Обычно к эти командам относят команды десятичной коррекции. Обычно эти команды зависят от того, какой формат используется (упакованный или неупакованный). Не зависимо от формата, команды основаны на аккумуляторном способе (т.е. команды десятичной коррекции безадресные). В этом случае десятичная обработка состоит из двух фаз: 1) соответствующее двоичное действие 2) коррекция с учетом десятичного числа.

2. Команды логических операций. Логические команды в системе команд ЭВМ играют не только вспомогательную роль, но в логических задачах могут быть основными операторами обработки. Для этого система логических операций в ЭВМ должна быть функционально полной. Как правило, а систему команд закладываются избыточные логические операции. Операции выполняются побитно и одновременно справа на лево. Команды логической обработки одно и двух операндные. Одноместную операцию реализует отрицание: NOT dst; Схема dst:= $\neg$(dst). Двухоперандные логические операции реализуют: &, $\cup$, $\oplus$. AND dst,src; OR dst,src; XOR dst,src; dst:=(dst)$*$(src);. Чаще всего логические операции применяются для решения трех задачек.

1. Маскирование и выделение определенных разрядов операндов. Реализуется с помощью операции конъюнкции (&) с заданной маской
2. Формирование требуемых значений в требуемых битах (с помощью $\cup$).
3. Инвертирование определенных битов (с помощью $\oplus$)

Логические операции избирательно действуют на флаги, т.е. часть флагов после выполнения операции не изменяются (OVR:=0; CY:=0), некоторые могут иметь неопределенное значение (AF:=?). В системе команд логических операций еще существуют команды (TEST dst,src; Схема (dst)&(src)). В следствии выполнения этой операции, результат никуда не записывается, но формируются все флаги.

3. Команды сдвигов. Все реализуемые команды сдвигов могуи быть разделены по признакам:

1. вид сдвига (арифметический или логический)
2. направление сдвига
3. характер сдвига (простой или циклический)
4. по количеству разрядов, на которое сдвигается операнд после выполнения операции.

Формат команды содержит как минимум три поля.

КОп	Адрес операнда	Направление и число сдвигов

Опр. Под логическим сдвигом понимается сдвиг числового кода операнда без учета его числового эквивалента.

Опр. Арифметический сдвиг – сдвиг числового кода операнда с учетом его числового эквивалента.

В том случае если для представления числовых значений не используются специальным образом формируемые коды (т.е. данные беззнаковые), то различий между логическим и арифметическими сдвигами нет. Если же для представления числовых значений с учетом знака, применяются специальные коды (прямой, обратный, дополнительный), то арифметический сдвиг принципиально отличается от процедуры логического сдвига.

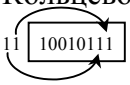
Пример:

1. Логический сдвиг.

E5 = 11100101 ACC:=L2(ACC) результат: ←(11) [100101] ←(00)

Точная запись: ACC:=L2(ACC(5:0)).00 Правый сдвиг аналогично.

Кольцевой сдвиг на 2 бита: ACC:=L2(ACC(5:0)).ACC(7:6)



2. Арифметический сдвиг.

Сдвиг числа влево и вправо соответствует делению или умножению числа на основание СС.

11100101 двоичный код числа –27.

При арифметическом сдвиге влево [1] 00101 [00]

сдвиге вправо: [1] (11) 11001 в дополнительном коде.

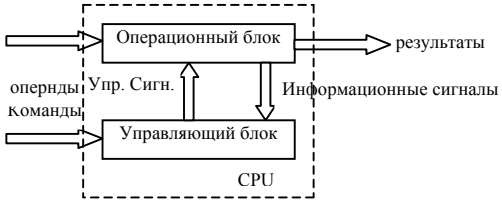
Глава 2.

Микропрограммный уровень организации ЭВМ.

§1 Принцип микропрограммного управления.

В общем случае между программными и аппаратными средствами четкие границы отсутствуют.

В большинстве современных ЭВМ непосредственная связь между аппаратурой и программными средствами осуществляется через микропрограммный уровень. Любая машинная команда исполняется аппаратурой не непосредственно, а путем их интерпретации в соответственную последовательность более простых действий. А значит всегда существует задача программирования машинных команд из более простых действий – микропрограммирование. Впервые этот термин был введен в 1953 году специалистом по ВТ Уилксом. Но это было применимо только к аппаратным средствам. Примерно в середине 60-х годов, усилиями разработчиков IBM, идеи Уилкса превратились в принцип организации ВМ. Микропрограммирование обеспечило переход к модульному построению ЭВМ. Развивая идеи микропрограммирования, Глушков показал, что в любом устройстве обработки информации функционально можно выделить операционный автомат и управляющий автомат. На этом уровне структура любой информации:



Управляющий блок выдает последовательность сигналов, которые обеспечивают выполнение данной команды. Информационные сигналы зависят не только от исходных значений обрабатываемых данных, но и от результатов получаемых в процессе обработки.

Порядок функционирования устройства базируется на следующих положениях:

1) Любая машинная команда рассматривается, как некоторое сложное действие, которое состоит из последовательности элементарных действий над словами информации – микроопераций.
2) Порядок следования микроопераций зависит не только от значений преобразуемых слов, но также от их информационных сигналов, вырабатываемых операционным автоматом. Примерами таких сигналов могут быть признаки результата операции, значения отдельных битов данных и т.п.
3) Процесс выполнения машиной команды описывается в виде некоторого алгоритма в терминах микроопераций и логических условий. Описание информационных сигналов – микропрограмма.
4) Микропрограмма служит не только для обработки данных, но и обеспечивает управление работой всего устройства в целом – принцип микропрограммного управления.

Операционный блок обеспечивает выполнение определенного набора микроопераций и вычисление необходимых логических условий.

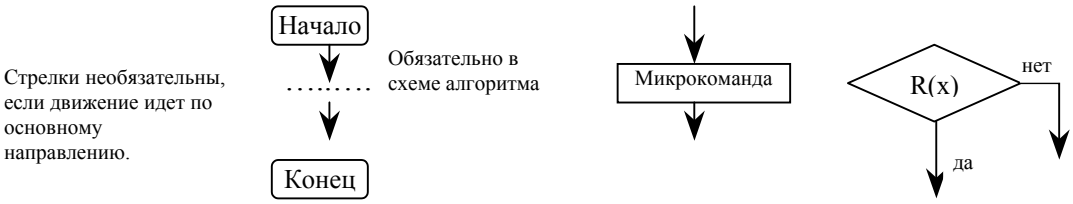
Управляющий автомат, согласно заданной машинной команде, генерирует необходимую последовательность сигналов, инициирующих соответствующие микрооперации, согласно микропрограмме и значениями логических условий, формируемых операционным в ходе обработки по микропрограмме. Таким образом, микропрограммы выступают, с одной стороны, в роли закона по которому выполняется обработка, с другой стороны, закон, по которому работает управляющий блок.

§2 Описание функциональных микропрограмм.

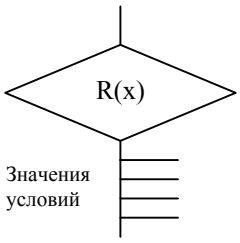
Существуют языки микропрограммирования, которые обеспечивают описание законов функционирования микропроцессора. Все функциональные блоки рассматриваются на уровне регистров. Языки микропрограммирования являются сильно ориентируемыми на конкретную структуру обработки информации. Основным оператором языка является микрокоманда, которая состоит из нескольких функционально совместимых микроопераций, т.е. порядок выполнения операций не влияет на конечный результат.

Совместимость микроопераций зависит не только от действий, но и от структуры операционного автомата. С точки зрения языка для описания объектов используются конструкция идентификатор. Эти идентификаторы используются с конкретным числом битов того или иного данного. Отдельные поля регистров могут иметь собственное обозначение (идентификатор). Любая микрооперация, как некоторый акт преобразования данных, записывается в виде оператора :=. Сами по себе операции по преобразованию могут быть арифметическими, логическими, функциональными. Многофункциональное действие может быть использовано только в случае, если в операционном автомате есть соответствующий аппаратурный элемент. Основной единицей информации является слово. Микропрограммы пишутся с точки зрения слов. Любая микрокоманда выполняется за 1 такт машинного (автоматного) времени. Микрокоманда записывается всегда в 1 строку. Сама микрокоманда записывается в виде отдельных микрооператоров. Для того, чтобы организовать ветвления, используются условные микрооператоры – if, как полный так и укороченный вариант, вплоть до операторов условного присваивания := (условие).

Для наглядности представления микропрограмм используется графическое представление – схема алгоритма микропрограммы. В основе вычерчивания схем лежат общие требования, которые определяются стандартами группы ГОСТ 19… - группа единой системы программной документации. При графическом представлении учитывается, что в микропрограмме количество различных типов вершин в схеме алгоритмов резко ограниченно.



Множественное ветвление



Конкретный номер микрооперации, который реализуется в операционном автомате и номер логических условий, которые вычисляется в различных автоматах, различны. Но этот номер должен удовлетворять требованиям алгоритмической полноты и эффективности.

Полнота – свойства набора операций и логических условий, которые позволяют построить микропрограмму для любой выполняемой операции устройствам машинной команды.

Эффективность предполагает, что набор микрооперации и логических условий позволяет строить наиболее быстродействующий алгоритм машинных команд при заданных ограничениях на аппаратные средства.

§3 Набор микроопераций и микроэлементов.

Операционный блок процессора представляет собой некоторую композицию так называемых композиционных элементов. Операционный элемент имеет аппаратную реализацию в виде «отдельной» части. Обеспечивает хранение слов, выполнение микроопераций над словами и их полями, а также вычисление логических условий. Все элементы в операционном автомате (ОП) соединяются между собой с помощью шин, которые обеспечивают передачу слов с выхода одного операционного элемента на вход другого. Все микрооперации разделяются на семь групп:

1. МО установки (A:=0) G:=N
2. МО передачи (A:=B)
3. МО инвертирования (A:=¬A)
4. МО сдвига (A:=L1(A).0) · – конкатенация.
5. МО счета (C:=C±1)
6. МО сложения (C:=A ± B)
7. МО поразрядные, логические &,∪,⊕ (C:= A⊕B)

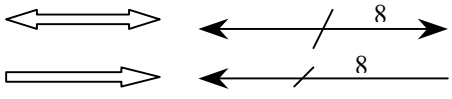
Набор микроопераций и микроэлементов.

В соответствии с выполняемыми МО операционные элементы процессора разделяются на шины, регистры, счетчики, сумматоры (вычитатели), логические устройства, сдвигатели, преобразователи и формирователи кодов, комбинированные операционные элементы.

1°. Шина.

Для передачи 1 бита данных требуется 1 сигнальная линия (1 проверка). Совокупность всех цепей, которые обеспечивают одновременную предачу всех битов слова, называется **шиной**. Шина реализует МО передачи. Шина, по которой передается только прямое или только инверсное значение, называется однофазной. Если по шине одновременно передается и прямое и инверсное значение, то шина называется парафазной. Если по сигнальным линиям информация может передаваться только в одном направлении, то такая сигнальная линия

называется однонапрвленной. Если информация может передаваться в том и другом направлении, то такая шина – двунаправленная.
Передача информации происходит по сигналу, который имитирует передачу.



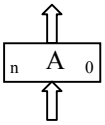
Двунаправленные шины (передают 8 бит)

Однонаправленные шины

Ширина шины – количество одновременно передаваемых данных.

2°. Регистры.

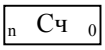
Регистр – совокупность запоминающих элементов, которые предназначены для приема, хранения и выдачи информации.



Вход и выход шины.

С помощью регистра выполняются МО установки, инвертирования, сдвига на заданное число разрядов. Любой регистр может подразделяться на подрегистры, которые соответствуют определенным полям слова. Над подрегистрами могут выполняться автономные операции. Выдача информации из регистра производится без потери в источнике.

3°. Счетчики.



Обеспечивает МО счета (прямого или обратного)

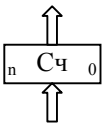
Сч:=Сч±const const = 2^k – степень двойки.

Суммирующие счетчики (прямое)

Вычитающие счетчики (обратное)

Реверсирующие счетчики (то и другое направление)

Счетчики строятся с использованием запоминающих элементов (регистров). Кроме МО счета счетчика можно реализовать все МО, которые характерны для регистров, то есть прием, выдачу, сдвиг кода, хранение.



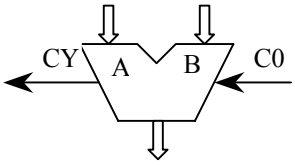
4°. Сумматоры.

Операционный элемент, который реализует МО:

1) C:=A+B
2) C:=C+B – рекуррентная схема.

Различают комбинационные и накапливающие сумматоры, которые реализуют соответственно (1) и (2).

Комбинационный сумматор.



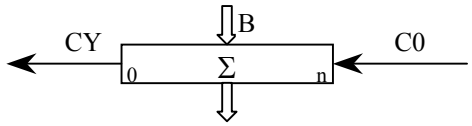
На самом деле сумматор обрабатывает три операнда: A, B – слова одной разрядности, а третий операнд однобитовый.

CY.C(n:0) = A(n:0) + B(n:0)+C0

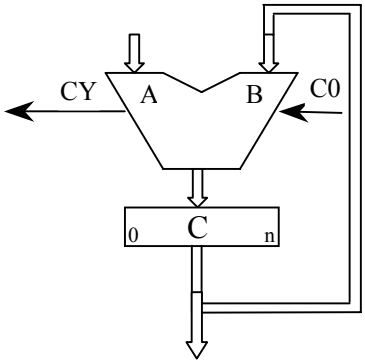
Обычно используют комбинационный сумматор, т.к. написание упрощенное.

Комбинационный сумматор – техническое устройство (схема), в которой результат присутствует на выходе до тех пор, пока на входе присутствуют входные сигналы.

Накапливающие сумматоры строятся на основе регистров, поэтому результат суммирования может быть считан из соответствующего выхода регистра в любой момент времени до новой МО.

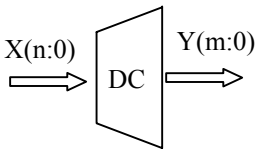


Также может быть реализован с помощью комбинационного сумматора.



5°. Преобразователи кодов.

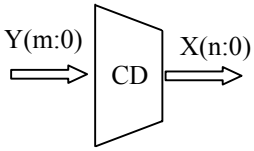
Обеспечивают перекодировку значений, т.е. преобразование из одного кода в другой. Самые распространенные, преобразуют двоичный позиционный код в унитарный двоичный код – дешифрация. (Знать унитарный и позиционный бинарный код).



Разрядности входных и выходных слов не равны.

Если $m = 2^{n+1} - 1$, то такой дешифратор называется полным.

В противном случае – неполным. Эти операционные элементы используются для организации связи между устройствами, в том числе, с памятью ЭВМ. Обратное преобразование (из унитарного кода в позиционный код) реализуется с помощью шифратора.



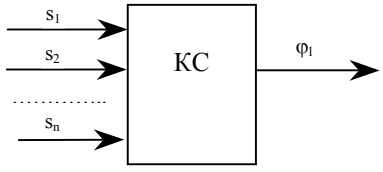
Более сложные преобразования (из бинарного кода в десятичный) требуют более сложных преобразователей.

6°. Вычисление значений логических условий.

Эти функции операционным элементом в виде булевой функции.

$\varphi_l(s_1,s_2,\ldots,s_n)$ s_i – некоторые слова, поля слов или отдельные биты слов.

Функция вычисления комбинационной схемой:



Наиболее распространенными схемами для вычисления логических условий являются схемы, которые вычисляют отношение 2-х слов A и B: $\varphi_l(A,B)$

7°. Комбинированные операционные элементы.

Комбинированные операционные элементы обеспечивают реализацию нескольких разнотипных МО. Комбинированные операционные элементы строятся на основе регистров, имеющих входную и выходную логику.

§4 Каноническое построение операционных автоматов.

Структура операционного автомата (состав и связь) зависит от алгоритмов выполнения машинных команд. Если структура уже задана, то один и тот же алгоритм может быть реализован по разному. Большинство операционных частей процессоров современных ЭВМ строится по некоторым устоявшимся схемам. Эти схемы называются каноническими. Одним из вариантов канонической схемы является универсальная каноническая структура операционного автомата с общими МО. Она достаточно эффективна с позиций технологического процесса, т.к. обладает высокой однотипностью узлов.

В соответствии с канонической структурой, в которой машинная операция описывается в виде набора многодоступных функциональных преобразований данных.

$C = g(f(A),B))$

g – оператор сдвига.

f – оператор бинарной арифметико-логических операций

h – оператор формирования инверсных кодов (дополнительного, обратного)

A,B,C – слова внутри памяти операционного автомата процессора.

Логические условия в этом операционном автомате представляются в виде булевой функции $\varphi(z)$ от некоторых слов памяти операционного автомата.

Каноническая структура напоминает структуру операционного модуля, для примера ВС1, если убрать регистр Q (см. рисунок на практике).

Такое каноническое построение операционного автомата заранее определяет какие МО оказываются совместимыми, т.е. структура автомата определяет состав операционной части микрокоманды. Структура определяет также последовательность выполнения МО операционной части микрокоманды. В большинстве современных процессоров в основе построения операционной части базируется на схеме с общими МО.

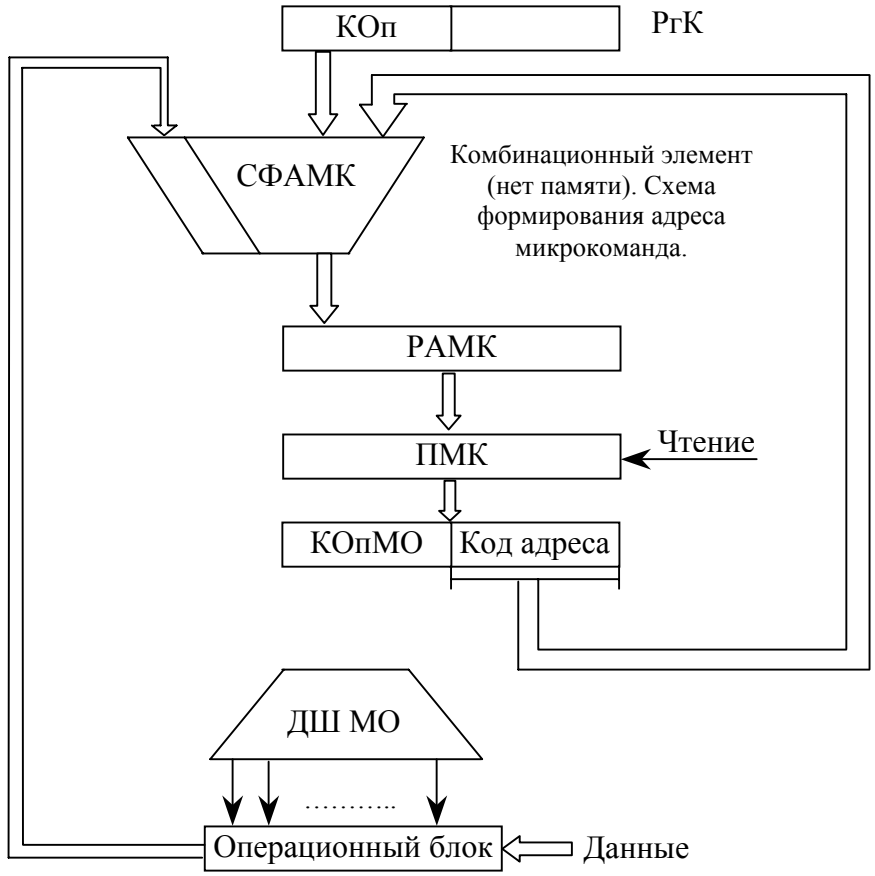
§5 Структурное построение и функционирование микропрограммных устройств управления.

Управление работой операционного автомата по микропрограмме реализуются с помощью микропрограммного устройства управления.

1. Управление по схемной (жесткой) логике (микропрограмма реализуется в виде схемы).
2. Управление по хранимой микропрограмме.

Управление по схемной логике реализуется на основе принципа Фон-Неймана, на микропрограммном уровне. Микрокоманды имеют операционно-адресную структуру. ВУУ – типовой модуль микропрограммного устройства управления. Любая микропрограмма состоит из микрокоманд. Любой следующий шаг начинается после предыдущего. Управление работой обеспечивает микропрограмма УУ. Два варианта: управление по хранимой микропрограмме, управление по схемной логике (Intel). Управляющие слова имеют операционно-адресную структуру.

Коды МО	Код адреса МК



Адрес формируется в СФАМК, в РАМК – хранение адреса. Выбирается начальная МК, поле в регистре МК дешифруется (в ДШ МО), операционным автоматом, регистром МК – формируется адрес следующей МК.

Микропрограммы УУ различаются по:

1) Типу памяти МП
   - ПЗУ – статическая МП
   - ОЗУ – динамическая МП
2) способу кодирования МО
   a) прямое
   b) косвенное
3) технической реализации – тактность использования МК

Способы кодирования МО:

1) горизонтальное микропрограммирование
2) вертикальное микропрограммирование
3) смешанное микропрограммирование

$Y_m = \{y_0,\ldots y_m\}$ – номер МО

1) **Горизонтальное микропрограммирование**. Любому сигналу ставится 1 бит в операционной части МК (для примера 1 – выполняется 0 – нет)

V_1	V_2	…	V_{m-1}	V_m

В любой МК может выполняться любая МО

Достоинства:

1) Если нужен ввод новой МК, то никаких схемотехнических изменений нет.

2) Добавление новой МО к существенному увеличению схемы УУ не приводит
3) Нет схемы дешифрации

Недостатки:

1) Сложность (длинна операционной части МК определяется количеством МО, следовательно, емкость памяти МК увеличивается)
2) Сложность доступа
3) Из операционной части полезную информацию несет 10-15%

Горизонтальное микропрограммирование применяется в простых устройствах обработки с малым количеством МО.

3) **Вертикальное микропрограммирование.** Любой сигнал управления задается значением всего кода операционной части.

Достоинства:

1) $n_{оч} =]\log_2(M+1)[$ (длина операционной части)
2) появление новых МК к принципиальным трудностям не приводит
3) емкость памяти МК достаточно небольшая.

Недостатки:

1) нужна дешифрация
2) две одинаковых МО в одной МК не могут быть, следовательно, количество МК возрастает.

3) **Смешанное микропрограммирование.** Весь набор микроопераций разбивается на некоторое количество подмножеств, причем необязательно, чтобы подмножества были непересекающимися, тогда операционная часть МК будет состоять из нескольких подмножеств.

Длина любого поля $n_{iоч} =]\log_2(M_i+1)[$

Длина операционной части $n_{оч} = \sum n_{iоч}$

Внутри подмножества любая МО кодируется вертикальным способом, может реализовываться способ горизонтально – вертикального программирования.

Адресная часть.

Так как МК хранится в памяти, то любая МК идентифицируется своим адресом в памяти. Все способы должны исключать арифметику. Используется естественная адресация. Для этого в состав УУ должен вводится счетчик адреса МК. Это эффективно работает только для линейных участков МП, но как правило сложные МП ветвящаяся, следовательно, без принудительной адресации обойтись нельзя.

Варианты адресации:

- адресная часть

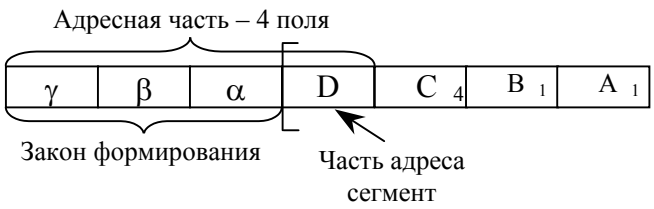
iif (условие, A0, A1)

- ветвление на 2 направления

Но для МП характерно, что ветвление происходит на несколько направлений, но таким способ это будет занимать длительный промежуток времени.

Для МП был придуман специальный механизм множественного ветвления – формирование адресов МК.

Суть данного метода (на примере):



Сам адрес МК содержит 4 поля, поля C, B, A, формируются по γ,β,α.

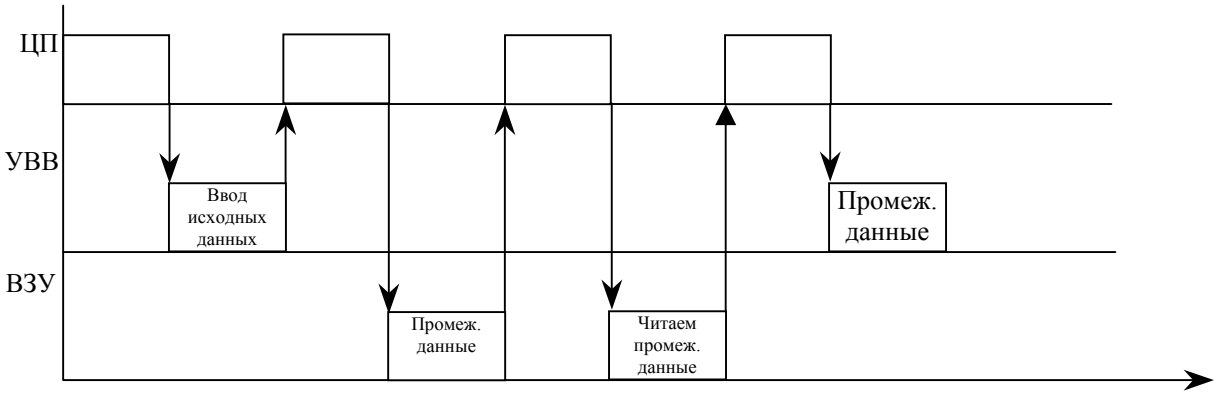
Пусть поля A – 1 бит, B – 1 бит ,C – 4 бит.

Поле D можем рассматривать как начальный адрес – сегмент.
Поле C – зона содержит 4 адреса.
Из всего множества условий выделяется некоторое количество номеров логических условий (сигналов), которые кодируются с помощью 4 разрядов кода.
$X_j = \{x_1,\ldots,x_m\}$ $x_1 = x_1x_2x_3x_4$ $x_2 = x_3x_5x_6x_8$
Таких условий должно быть 16. Все остальные условия – однобитовые, которые кодируются либо в поле β, либо в α. Вычисляется значение условия в γ, если 0 – поле C 0, если не 0, то в C подставляется значение логического условия (например $x_3x_5x_6x_8$ 1001)
В поле B и A подставляется код условия, записанный в β. Одна МК поддерживает 64 разветвления. Сложность заключается в том, что все адреса рядом, МК не может находится там же, но время вычисления соответствует комбинационной схеме.

Глава 3.
Системный уровень организации ЭВМ.

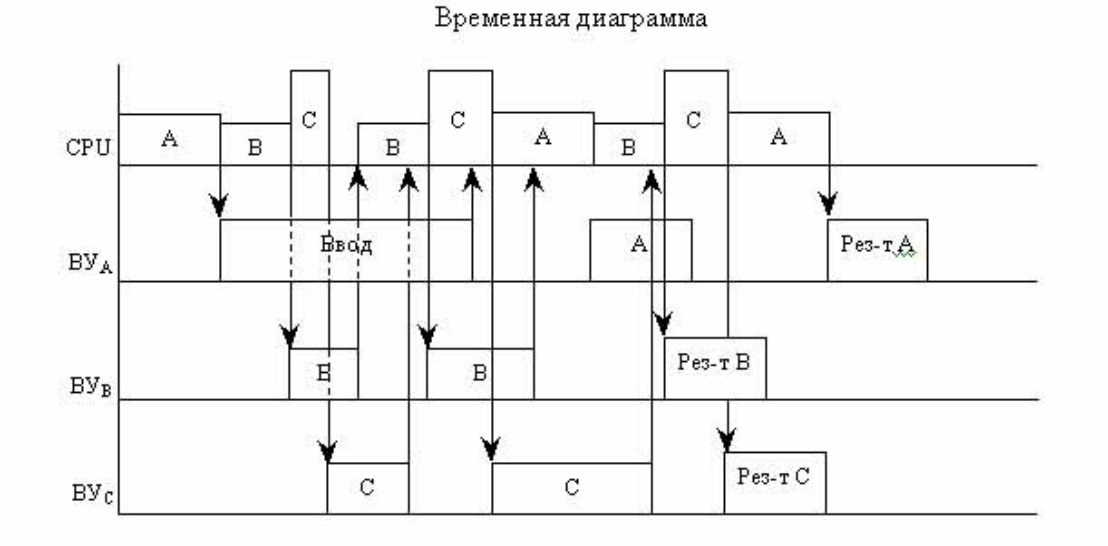
Программный режим работы.

Однозадачный режим работы характеризуется тем, что в основной памяти ВМ находится 1 задание, которое является активным (обрабатывается), следовательно, все ресурсы ВМ отданы в распоряжение задания до момента окончания. Исторически любой пользователь был монополистом по отношению к ресурсам ЭВМ. Это характерно для программирования на машинном языке. Но есть серьезные ограничения, связанные с пропускной способностью компьютера. Рассмотрим гипотетическую задачу и процесс работы компьютера при ее решении (задача уже загружена в основную память компьютера и инициирована для своего исполнения). Выделим основные ресурсы: ЦП(основная память), устройство ввода-вывода, внешняя память (накопители). Однопрограммный режим: в любой момент времени работает только одно устройство. В данном случае процессор самое «узкое» место. Чем быстрее процессор тем ниже коэффициент его загрузки. То есть процессор такого компьютера фактически большее время находится в режиме простоя., чем в режиме решения задач.



То есть производительность ВМ (количество решенных задач в единицу времени) слабо зависит от характеристик процессора. Повышение производительности достигается за счет распараллеливания процесса.
Методы распараллеливания процесса:
1) Аппаратно-алгоритмические методы – приводят к конвейеру вычислений, совмещению работы различных устройств.
2) Мультипрограммирование.

В память ЭВМ загружается сразу несколько задач, которые инициируются по мере освобождения требуемых ресурсов ВМ с определенной очередностью. Суть мультипрограммного режима рассмотрим не примере трех задач. Будем предполагать, что каждой задаче выделены свои ресурсы внешних устройств и памяти.



A, B, C – порядок решения задач.
(Все задания в памяти)
Время решения любой задачи не уменьшается, но за один промежуток времени решается больше задач, т.е. количество решаемых задач за единицу времени увеличивается, следовательно, производительность ВМ увеличивается. За счет организации процесса можно увеличить производительность ЭВМ.
Известно два различных способа организации мультипрограммного режима работы:
1. пакетная обработка
2. разделение времени

При **пакетной обработке** обычно формируется пакет задач, который находится во внешней памяти. Выбор задачи на решение производится автоматически без вмешательства пользователя. Пакет рассчитан на длительное время работы компьютера. Такой режим применяется при решении технических задач (нечувствительных ко времени)
Режим разделения времени ориентирован на многопользовательский режим работы. Любому пользователю ОС выделяет некоторый фиксированный квант процессорного времени. Если программа за этот квант не решилась до конца, то она принудительно прерывается для исполнения, и управление отдается следующей задаче, а прерванная программа ставится в очередь ожидания. Применяются различные правила распределения ресурсов процессора между задачами. С одной стороны размер кванта должен быть маленьким, так как задача не должна простаивать при завершении ее, но переключение с одной задачи на другую требует расходов. Удельный вес накладных расходов должен увеличиваться, но квант нельзя сделать очень большим, следовательно, увеличивается время простоя. Квант зависит от характеристики задач. На практике величина кванта до 100 миллисекунд. Количество «одновременно» выполняемых программ – коэффициент мультипрограммирования.

Организация прерывания процессора.

В процессе выполнения любой программы, как внутри ЭВМ, так и во внешней среде (машины используемые для управления некоторыми объектами), могут возникать некоторые события, которые требуют немедленной, определенной для этого события реакции, как со стороны ЭВМ, так и со стороны программы. ВМ временно прекращают выполнение текущей программы, переходит к выполнению некоторой другой программы, специально предусмотренной для этого случая, а по завершении специальной программы ЭВМ

возвращается к выполнению исходной программы. Такой процесс перехода к специальным программам и обратно носит названия **прерывание**, причем исходная программа называется прерванной, а специальная программа – прерывающая. Различного рода сигналы, которые информируют ЭВМ о наступлении события, носят название запроса прерывания. Любое прерывание текущей программы заключается в том, что.

1) Процессор прекращает ее выполнение
2) Запоминает информацию, нужную для продолжения выполнения программы с точки прерывания.
3) Переходит к выполнению специальной программы.
4) После исполнения специальной программы, управление возвращается исходной программе.

Запросы на прерывание могут возникнуть внутри компьютера (сбой в некотором устройстве, переполнение разрядной сетки, попытка деления на нуль, требование прерываний ввода-вывода и т.п.). Несмотря на программу источник, временной момент появления прерывания определить не возможно. То есть запросы прерывания асинхронны по отношению к программе. Моменты возникновения других событий можно заранее предсказать. При многократном использовании программы эти события будут возникать в одно время (операции обмена, переполнения) – синхронные события. Самые сложные события асинхронные. Для организации правильной работы основной и прерывающей программы нужно управление. Эта координация реализуется системами прерывания. Основными функциями являются следующие:

1) временной останов выполняющей программы и выбор запроса на обслуживание
2) запоминание состояния прерванной программы
3) инициирование программ – обработчиков прерывания
4) обслуживание – выполнение прерывающей программы
5) восстановление состояния прерывающей программы и возврат к выполнению исходной программы

Эти функции также называют последовательностью обработки любого запроса на прерывание.

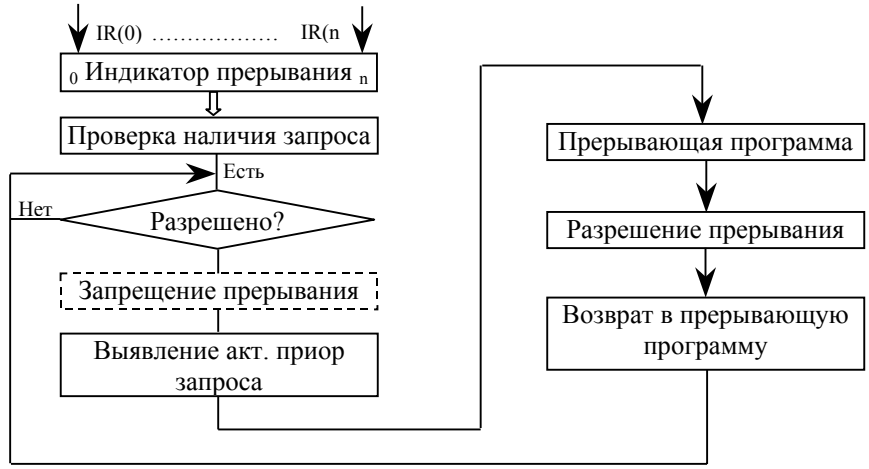
Временная диаграмма:

Любой запрос на прерывание формируется по соответствующей причине. Обычно: IR [0:n] – регистр прерывания. От момента выработки запроса до момента прерывания существует некоторое время, время реакции системы прерывания. Любой запрос фиксируется в фиксированном бите слова регистра прерывания. Время реакции в общем случае не постоянно. Оно зависит не только от того, как выпускаются допустимые моменты прерывания, но и от того, Сколько программ с более высоким приоритетом, чем прерывание ждет своего обслуживание в очереди. В ЭВМ используются различные способы определения допустимого момента прерывания. Самый простой способ заключается в том, что в формате команды ЭВМ вводится специальный бит – признак разрешения прерывания. Таким образом программист, составляя программу, может управлять разрешением прерывания. Это позволяет минимизировать объем информации, который запасается при переходе к прерывающей программе, что уменьшает общее время обработки прерывания, но само время реакции оказывается достаточно большим. Более распространенный способ предполагает, что прерывание возможно после окончания любой текущей команды. Но в этом случае нужно сохранять содержание всех программно доступных регистров. Это уменьшает время реакции, но увеличивает накладные расходы. В этом случае время реакции системы прерывания не превышает длительности выполнения самой длинной команды. Существует еще третий вариант – для машин, работающих в реальном времени. В таких ЭВМ прерывание может допускаться на любом шаге выполнения команды. Это характерно для компьютеров, имеющих микропрограммный уровень. Время реакции сводится к длительности одного такта. Но объем запоминаемой информации требует значительных временных затрат.

Выбор запроса на обслуживание – вторая часть задачи. Запросы поступают в любой момент времени, следовательно, к моменту, когда может быть разрешено прерывание в регистре запроса может находится несколько запросов (по крайней мере два). Возникает задача выбора запроса на обслуживание. Любой запрос в компьютере снабжается своим приоритетом. Обычно в качестве приоритета берут 0,1,…n. Степень важности обработчика пропорционален номеру (0 – самый высокий приоритет). Порядок выбора запросов определяется дисциплинами обслуживания. По способу исполнения приоритетов различаются дисциплины с абсолютными и относительными приоритетами. Если к моменту разрешения прерывания выбранный запрос оказывается выше приоритета текущей программы, то возможно два варианта действия:

1) текущая программа прерывается в тот момент, когда выбран приоритетный запрос (дисциплины с абсолютным приоритетом)
2) текущая программа не прерывается, продолжается до момента естественного прерывания (дисциплины с относительным приоритетом)

Независимо от вида приоритета в конкретной системе прерывания процедура обработки запроса может быть представлена:



Функции могут реализовываться как программно, так и аппаратно. Программная реализация увеличивает время обслуживания прерывания. Максимальное время – выявление активного запроса в соответствии в соответствии с приоритетом нужно построить процедуру, которая должна из n запросов выявить приоритетный. Самый простой вариант анализа запросов – последовательный просмотр запросов слева направо (схема алгоритма – самостоятельно). Последовательный перебор характерен тем, что система прерывания имеет глубину прерывания 1 (прервать прерывающую программу нельзя). Результатом выявления активного запроса является формирование системного прерывания начального адреса прерывающей программы. В процессе работы ЭВМ важность выполняемых программ не является постоянной. Между различными прерывающими программами (запросами) не всегда можно установить фиксированное распространение приоритетов, следовательно, средства прерывания должны быть программно изменяемыми.

Существует два способа программного управления приоритетами прерывающих программ:

1) Любая текущая, выполняемая в данный момент программа, характеризуется некоторым кодом (приоритетом программы). В этом случае система прерывания после выявления активного запроса сравнивает приоритет этого запроса с порогом приоритета, выполняемой программы. Если приоритет выделенного запроса ниже приоритета программы, то ее прерывание не происходит, но поскольку приоритет программы устанавливается пользователем, то при этом, запросом при смене приоритета программы может быть прерывание. При этом аппаратных изменений никаких нет, все включается в процедуру выявления.
2) Маскирование. В систему прерывание кроме индикатора (программно не доступен) вводится регистр маскирования, который доступен программно (т.е. можно загружать в

него данные). Имеет аналогичную индикатору структуру. Любой запрос индикатора поставлен в соответствующий бит маски. Взаимодействие запросов прерывания и битов маскирования строятся по простой схеме:

M [i] – маска

IR [i]

Если в бите маски 1, следовательно, соответствующее прерывание разрешено для обработки (для выявления), если 0, следовательно, соответствующее прерывание замаскировано (не разрешено). Изменяя маску, можем управлять приоритетом запросов на программном уровне. Схема взаимодействия

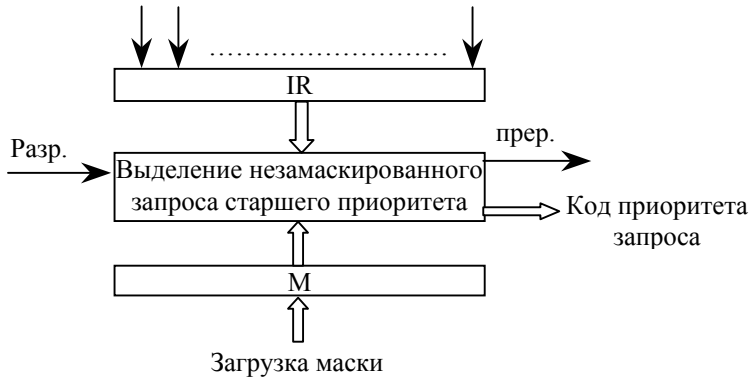


Схема усложняется, если в качестве кода запроса на прерывание применяется позиционный код номера активного запроса. Эта процедура может реализовываться либо программно, либо аппаратно. Программная реализация резко увеличивает время обработки, в следствии чего, не применяется. Аппаратная реализация с точки зрения функционирования заключается в том, что схема решает уравнение:

$$\underbrace{(\&_{j=0}^{i-1} \overline{IR[i]M[j]})}\ \underbrace{IR[i]M[i]} = 1$$

Поскольку маскирование отдано в руки пользователя, следовательно, могут возникать следующие ситуации при управлении масками:

Пусть существует 4 уровня запроса

	3	2	1	0
IR 0	0	0	0	0
IR 1	0	0	0	1
IR 2	0	0	1	1
IR 3	0	1	1	1

Фактически управления нет

Если маски прерывания программистом фиксируются

Другой вариант:

	3	2	1	0
IR 0	0	1	0	0
IR 1	0	0	0	1
IR 2	0	0	1	0
IR 3	0	1	1	1

Перераспределены приоритеты

Поскольку запросу поступают асинхронно (не связаны с работой компьютера), то в некоторый момент времени в системе прерывания поступили 4 запроса.

IR0 менее приоритетна чем IR2, переход к IR2 от IR2 переход к IR1 от IR1 переход к IR0 от IR0 переход к IR2, следовательно, зацикливание.

Вход в прерывающую программу

Основная функция – формирование начального адреса прерывающей программы. Любому запросу соответствует своя прерывающая программа. Существует три различных способа, используемых при формировании адреса:

1) **Размещение прерывающих программ по фиксированным адресам.** В некоторой постоянно распределенной области основной памяти по фиксированным адресам размещаются прерывающие программы, это размещение не меняется. Вход реализуется аппаратно, то есть адрес формируется аппаратно – это самый быстрый способ. Но у данного способа существуют серьезные ограничения:
    a) привязка к адресам
    b) количество причин прерывания должно быть достаточно малым

   Этот способ применяется при малых системах прерывания и для тех причин, которые требуют немедленной реакции.
2) вход на основании слов состояния программы (PSW). Типичная структура слова состояния программы:

Маска прерывания	Ключ защиты памяти	Код состояния CPU	Адрес команды (пр-мы)

   Схема входа в прерывающую программу. В некоторую постоянно распределенной области основной памяти формируется два массива: массив старых PSW и массив новых PSW. Любая пара слов состояния соответствует определенному запросу на прерывание. После выполнения активного запроса по соответствующему адресу в массив старых PSW загружается PSW текущей (прерываемой) программы. Характеристика программы в виде PSW записывается по определенному адресу. Из второго массива загружается новое (соответствующее прерывающей программе) PSW. Адрес записан в памяти – из PSW. Массив новых PSW всегда формируется при загрузке ОС. Массив старых PSW формируется в процессе работы. В отличие от предыдущего способа, использование PSW позволяет обслуживать и вложенные прерывания, если их приоритет выше текущей программы. Все это позволяет прерывать прерывающую программу. Недостаток: Вход в прерывающую программу требует загрузки достаточно больших слов (большого процессорного времени), следовательно, данный способ не очень быстрый. Этот вариант используется в универсальных компьютерах (для решения расчетных задач, т.е. не критичных ко времени)
3) Векторное прерывание – самый распространенный способ. Данный способ является программно-аппартным, т.е. для любого запроса (для любого выделенного запроса) аппаратно формируется адрес вектора прерывания. Чаще всего эти адреса фиксированы. Адреса векторных прерываний хранятся в системной области памяти. На основе адреса вектора из таблицы векторов прерывания извлекается начальный адрес прерывающей программы. Это приводит к тому, что код запроса может быть малобитным, но таблица векторов прерывания должна хранится в начальной области памяти. В качестве вектора прерывания используются:
    a) адрес начала прерывающей программы (применяется в PC)
    b) команда безусловного перехода к программе

   Способ a) оказывается универсальным и допускает любую глубину прерывающих программ. Если таблица векторов является загружаемой, то это дает возможность пользователю изменять прерывающие программы даже в процессе решения задачи, которые обрабатывают один и тот же запрос.

Запоминание состояния прерванной программы.

Вся запоминаемая информация делится на основную и дополнительную. Основная информация должна запоминаться всегда – адрес текущей программы, в которой произошло прерывание, состояние процессора, уровень приоритетности программы. Основная информация компонуется в слово-состояние. Основная информация запоминается аппаратно. Дополнительную информацию запоминает сам пользователь. При запоминании основной информации используются два способа:

1) Использование PSW (запоминание старого PSW – основная информация).

2) Запоминание основной информации в системном стеке, который поддерживается ОС. Использование стековых структур при входе в прерывающую программу позволяет не ограничивать глубину вложения прерываний. Ограничения только в связи с размерами стека.

Дополнительная информация с точки зрения объема различна. В каждом конкретном случае определяется самостоятельно – ресурсы процессора, которые используются при работе самой прерывающей программы.

Восстановление состояния прерванной программы.

Инвертирование тех действий, которые выполнены при запоминании.

Возврат.

Передача управления в ту точку, где произошло прерывание. Реализуется обычно аппаратно. Зависит от организации входа в прерывающую программу.

Глава 4. Организация памяти ВС.

Виды запоминающих устройств. Иерархия памяти.

Память – совокупность отдельных устройств, которые запоминают, хранят, выдают информацию. Отдельные устройства памяти называют запоминающими устройствами. Производительность вычислительных систем в значительной мере определяется составом и характеристиками отдельных запоминающих устройств, которые различают по принципу действия, техническим характеристикам, назначениям. Основные операции с памятью – процедура записи, процедура чтения (выборки). Процедуры записи и чтения также называют обращением к памяти. За одно обращение к памяти «обрабатывается» для различных устройств различные единицы данных (байт, слово, двойное слово, блок).

Основные технические характеристики памяти – емкость (Е), быстродействие (время обращения к запоминающему устройству).

В некоторых запоминающих устройствах считывание данных сопровождается их разрушением. В этом случае цикл обращения к памяти всегда должен содержать регенерацию данных (ЗУ динамического типа). Этот цикл состоит из трех шагов:

1) время от начала операции обращения до того момента, как данные станут доступны (время доступа)
2) считывание
3) регенерация

Процедура записи:

1) Время доступа
2) Время подготовки (приведение в исходное состояние поверхности магнитного диска при записи)
3) Запись

Максимальная длительность чтения-записи называется временем обращения к памяти. По физическим основам все запоминающие устройства разделяются: полупроводниковые, магнитно-оптические и т.д.

В зависимости от вида реализуемых операций память бывает двусторонней (память с любым обращением) и односторонней. Вторая сторона позволяет производить чтение-запись. Односторонняя память предназначена только для чтения или только для записи.

По способу организации доступа к данным все ЗУ разделяются:

1) ЗУ с произвольным доступом
2) ЗУ с прямым или циклическим доступом
3) ЗУ с последовательным доступом

Запоминающие устройства с произвольным доступом. Цикл обращения таких устройств не зависит от того, в каком физическом месте ЗУ находятся требуемые данные. Такой способ

доступа характерен для полупроводниковых ЗУ. Число записанных одновременно битов данных за одно обращение называют шириной выборки (доступа).

Запоминающие устройства с прямым доступом. В таких устройствах носитель информации непрерывно вращается. В результате требуемые данные доступны для чтения-записи через некоторый фиксированный промежуток времени. Такие ЗУ называют ЗУ циклического типа.

Запоминающие устройства с последовательным доступом. При последовательном доступе, прежде чем найти нужный участок ЗУ, нужно «просмотреть» либо все предыдущие участки памяти, либо предыдущий последовательно один за другим (накопитель на магнитной ленте).

Требования, которые предъявляются к емкости и быстродействию памяти, являются противоположными с точки зрения технического исполнения (т.е. если память быстрая, то емкость мала и наоборот). Поэтому в современных ЭВМ память строится в виде некоторой иерархической структуры. На разных уровнях иерархии распределяются ЗУ, которые обладают разными характеристиками. Структуру памяти можно представить в виде:

Не говоря о конкретных цифрах ЗУ, нужно отметить, что они различаются у соседних уровней, на 1,2,3 порядка. Любой нижележащий уровень имеет большую емкость и меньшее быстродействие. На любом уровне памяти может быть несколько однотипных модулей ЗУ.

Основная память обеспечивает хранение информации, которое непосредственно используется процессором (АЛУ, УУ) в ходе выполнения программы. Основная память напрямую связана с процессором, следовательно, ее характеристики самым непосредственным образом влияют на производительность ЭВМ. Быстродействие памяти меньше быстродействия процессора (7 нс – время обращения к памяти, к процессору в 5 раз меньше – 2нс). В современных ЭВМ существует сверхоперативная память – буфер между процессором и основной памятью. Она предназначена для согласования скорости работы ЗУ и процессора. КЭШ имеет небольшую емкость.

Сверхоперативная память обеспечивает временной хранение активных участков программы, активных участков данных, некоторой служебной информации для управления вычислительным процессом.

Обмен между сверхоперативной и основной памятью происходит поблочно. Приблизительно на том же уровне располагается регистр памяти компьютера. Емкость регистра памяти мала, но быстродействие самое высокое и вписано в цикл работы процессора.

Память большой емкости – оперативная память. В современных ЭВМ практически всегда отсутствует. Память большой емкости с процессором не связана. Доступ производится через основную память. Обмен между ней и основной памятью реализуется аппаратными средствами. Это приводит к тому, что увеличивается адресное пространство.

В оперативной памяти хранятся не все средства, которые необходимы для решения данной задачи, следовательно, всегда есть внешняя память. Чаще всего это дисковая память. Внешняя память (ВП) обеспечивает хранение всей информации, которая необходима в процессе решения данной задачи. ВП принципиально процессору недоступна. Обмен между ВП и ОП реализуется системами управления памяти (аппаратно-программные средства), которые пользователю недоступны (хотя можно решить такую задачу). ВП имеет емкость на несколько порядков больше, чем у ОП (время обращения к ВП исчисляется в мкс).

Архивная память. Емкость – сколько вместится. Время доступа зависит от характеристик (например, говорят о характеристиках дисковода).

В современных компьютерах существуют некоторые отдельные виды памяти: буферы различного рода устройств. Для терминала (дисплея) есть свой буфер, каналы обмена имеют свой буфер. Буфер не оказывает значительное влияние на характеристики производительности, но характеристики памяти адаптера оказывают влияние на анимацию.

Иерархия памяти в конечном итоге позволяет:

1) повысит производительность процессора в целом
2) увеличить пропускную способность памяти (среднюю скорость обмена)
3) предоставить в распоряжение пользователя практически не ограниченную память (виртуальную память)

Поскольку характеристики быстродействия различных уровней различны, то при построении памяти требуется согласование пропускных способностей каждого уровня. Достигается это за счет буферизации обмена между разными уровнями. Смысл буферизации заключается в том, что каждом уровне выделяется некоторая область (для «автономного» обмена с дисками без участия процессора). Информационные единицы обмена по мере удаления от процессора увеличиваются.

Организация ОП.

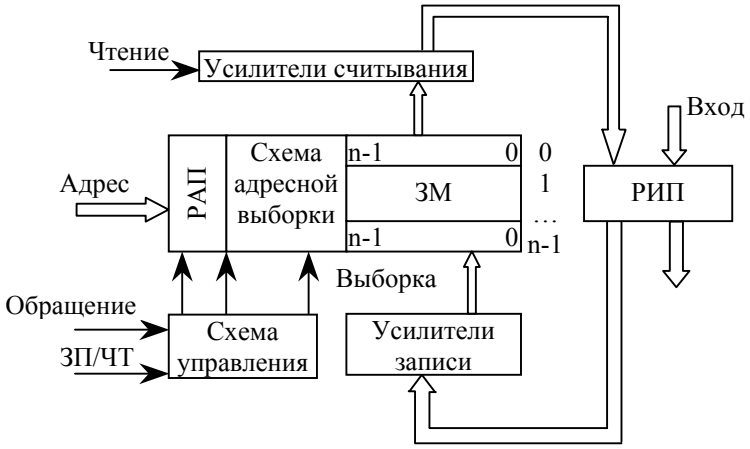
Оперативная память любого вычислителя всегда строится из отдельных модулей памяти, которые конструктивно и функционально закончены. Каждый модуль имеет свое собственное электронное обрамление (устройства, которые обеспечивают запись, выборку, чтение). Емкость каждого модуля ОП фиксирована (в виде степени двойки).

Структура модуля памяти определяется способом организации ОП (способ адресации). Существует 3 разновидности организации памяти:

1) адресная память
2) память со стековой организацией
3) ассоциативная организация ОП

С точки зрения функционального построения, любое ЗУ этого типа представляет собой некоторый массив элементов памяти. Структурные элементы памяти образуют ячейки памяти. Ширина ячеек – ширина выборки из памяти.

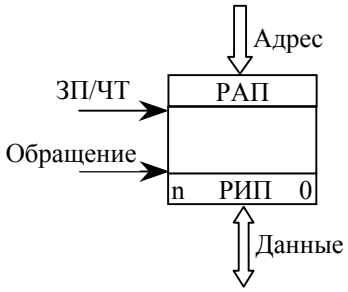
1) В адресной памяти, размещение и поиск информации в массиве запоминания, базируется на основе номера (адреса). Массив запоминания элементов содержит N n-разрядных слов, которые пронумерованы (0…N-1). Электронное обрамление включает в себя регистры для хранения адреса памяти, регистр информации (само слово), схемы адресной выборки (адресации), разрядные усилители для чтения и записи.



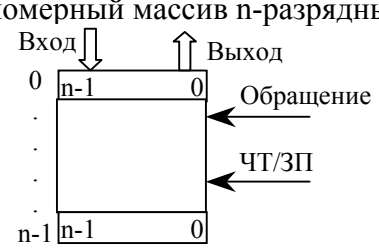
Массив запоминания (ЗМ) содержит N n-разрядных слов. Регистр адреса памяти (РАП) + схема адресной выборки + усилитель считывания + усилитель записи + регистр информации памяти (РИП) + схема управления – электронное обрамление.

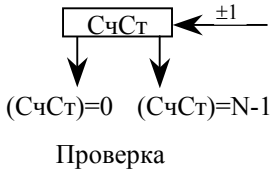
Цикл работы памяти инициируется сигналом обращения к памяти и операцией (ЗП/ЧТ). При инициировании обращения производится дешифрация адреса схемы адресной выборки. Если задана операция чтения, то активизируется усилитель считывания и информация через усилитель считывания предается в регистр информации памяти. Если память требует динамической регенерации, то после регистра информации памяти все поступает в регистр адреса памяти.

Если происходит чтение, то активизируется усилитель записи, который обеспечивает запись из регистра информации памяти в нужное место памяти.



2) Стековая память является безадресной. Ячейки стековой памяти представляют собой одномерный массив n-разрядных ячеек, в котором соседи связаны друг с другом.





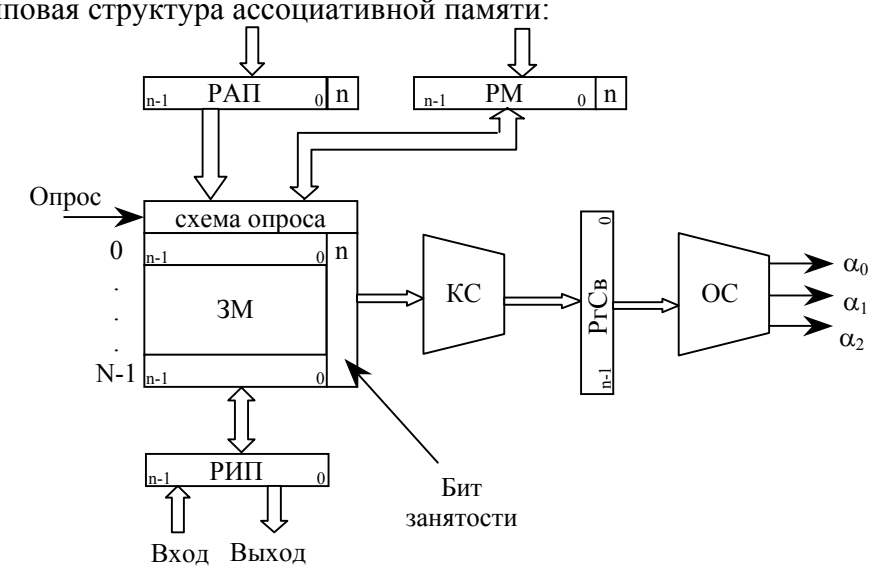
Для операций с памятью доступна только 0 ячейка. Операция с памятью инициируется сигналом обращения.

Каждая операция записи, инициируемая сигналом обращения к памяти, приводит к тому, записанные данные помещаются в 0 ячейку памяти. При этом все ранние записи в памяти слова автоматически сдвигаются на 1 адрес ниже. Операция чтения, инициируемая сигналом обращения, приводит к тому, что на выходе памяти формируется значение слова, находящиеся в 0 ячейке памяти. При этом все имеющиеся слова сдвигаются на одно слово вверх. Счетчик стека нужен только для контроля заполнения и очищения стека. Техническая реализация стековой памяти оказывается сложнее адресной памяти. Стековая память используется достаточно широко (короткий стек из микропрограммирования). Чаще всего

применяется не стековая память, а адресное поле, которое функционирует по принципу стека.

3) ассоциативная память. Исторически последняя. Является представителем многофункциональных запоминающих устройств (возможна обработка данных без процессора в памяти). Отличительная особенность: поиск любой информации в ЗМ производится не по адресу, а по ассоциативным признакам (признакам опроса). Поиск производится одновременно по всем ячейкам ЗМ.

Типовая структура ассоциативной памяти:



РАП - регистр ассоциативного опроса, РМ - регистр маски, РгСв - регистр связи.

У каждой ячейки памяти существует бит занятости (1 - есть или 0 - нет слова). Опрос производится на основе двух признаков: признака ассоциативного опроса (в РАП) и маски, которая находится в РМ. Поиск производится по незамаскированным разрядам. Поиск начинается с сигнала опроса (производится опрос всего ЗМ). Результаты опроса фиксируются в РгСв через комбинационную схему (КС).

Связи устанавливаются: $РгСв[j]:=\overline{\&_{i=0}^{n-1}\{РгАП[i]\oplus ЗМ[j,i]\cup РгМ[i]\}}$

Если некоторые биты замаскированы, они не участвуют в операции (1). По результатам опроса с помощью формирования считывания (ФС), формируется 3 сигнала:

$$\alpha_0=\overline{\&_{j=0}^{N-1} РгСв[j]}$$

(нет слова, отвечает признак опроса)

$\alpha_1=\bigcup_{j=0}^{N-1}\{РгСв[j]\ \&_{\substack{l=0\\ l\neq j}}^{N-1} РгСв[l]\}$ только одно слово $\alpha_2=\overline{\alpha_0}\overline{\alpha_1}$,

более 1 слова множественный ответ.

С помощью них выполняются операции чтения и записи.

Чтение:

если α_0 - результат пустое множество;

если α_1 - слово считывается в РИП; если

если α_2 - процедура чтения может строиться по разному:

1) считывается первый из элементов ЗМ, удовлетворяющий признаку запроса
2) считывание всех элементов, применяются методы множественного считывания (последовательное чтение), результат множество слов.

Запись: опрос по полю занятости. Ищется свободная ячейка, если результат α_1 или α_2 - производится запись в первую свободную ячейку. Сложности возникают, когда результатом является α_0.

С точки зрения структуры, любая основная память компьютера может быть построена, как одноблочная, либо как многоблочная (сейчас одноблочная память практически не используется). Многоблочную память строят из однотипных блоков.