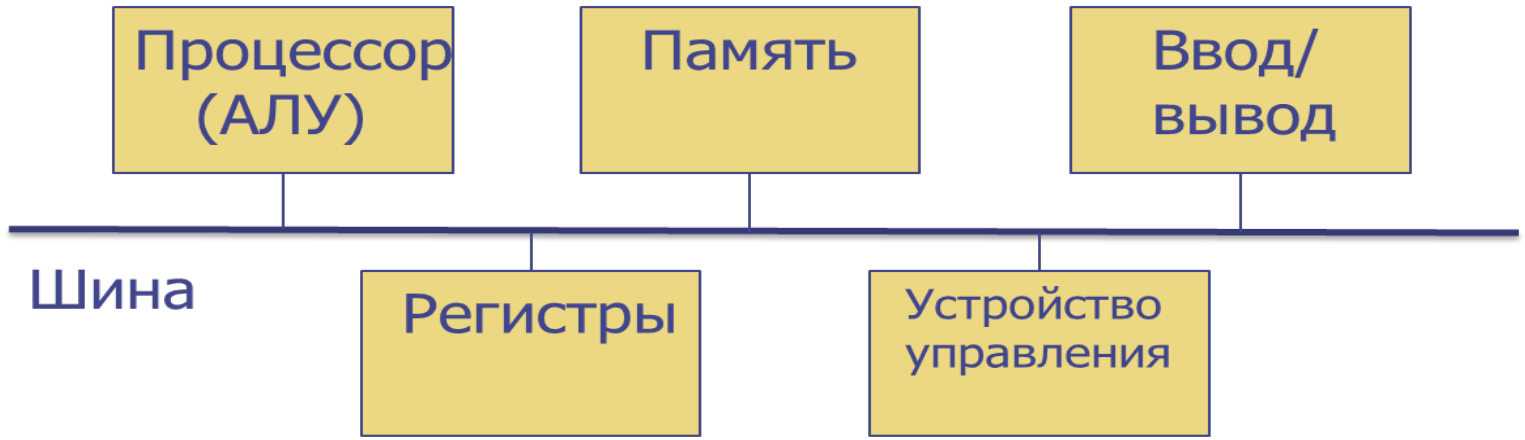
Архитектура ЭВМ

Зав. кафедрой системного программирования СПбГУ
доктор физ-мат наук, профессор
А.Н. Терехов

1. Введение. Что такое архитектура ЭВМ



Впервые эта схема появилась в 1834-1837гг, но и сегодня все ЭВМ имеют такую же архитектуру

2.1. Организация ЭВМ на уровне ассемблера

Допустим, надо выполнить формулу: C = A + B;
На ассемблере IBM mainframe это выглядит так:

```
L R0, AddrA
A R0, AddrB
ST R0, AddrC
```

Исполнительный адрес складывается из базового регистра, возможного индексного регистра и смещения

2.2. Команды и данные

В **гарвардской архитектуре** команды и данные хранятся в разных устройствах памяти

В **архитектуре фон Неймана** команды и данные хранятся в одной и той же памяти

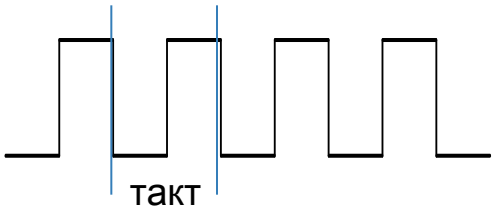
Чтобы остальные устройства не простаивали, пока очередная команда читается из памяти, устройство управления организует предварительное чтение из памяти и дешифрацию команд (обычно 6-8 команд) на фоне работы АЛУ и других устройств – **водопровод (pipeline)**

3.1. Цифровая логика

Идеальный сигнал:

Реальный сигнал:

Clock (Тактовая частота):



3.2. Цифровая логика

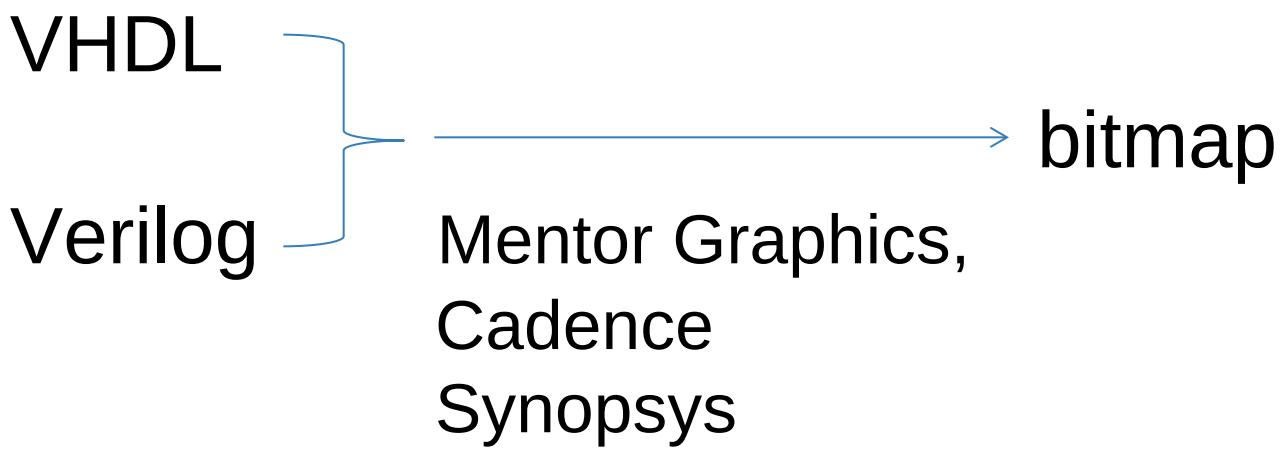
- Вентили **И**, **ИЛИ**, **исключающее ИЛИ**
- **Триггер** – схема с 2 устойчивыми состояниями, т.е. память на 1 разряд
- **Регистр** – несколько триггеров, рассматриваемых как 1 устройство (8,16,32 бита)
- **Счетчик** – регистр, над которым можно выполнять операцию +1

3.3. Кристаллы

- **Жесткий кристалл** – множество (миллионы) вентилей, соединенных в сложную схему и упакованных в единый «камень» (ЧИП)

- **Гибкий кристалл** (FPGA – Field-Programmable Gate Array или ПЛИС - Программируемая Логическая Интегральная Схема) – кристалл, имеющий оперативную память (bitmap), каждый бит которой управляет наличием соединительной линии (1 – есть, 0 – нет) между вентилями схемы. Сегодня соединения делаются не между простыми вентилями, а между более сложными элементами LUT (Lugical Unit)

3.4. Программирование гибких кристаллов



Мы создали свой язык HaSCoL (Hardware and Software Codesign Language), программы на котором в 5-6 раз короче, чем такие же программы на VHDL

4.1. Представление данных

- Бит – 1 разряд (триггер)
- Байт – 8 бит (с 1964 г.) 1 литера
- Слово – 16 и 32 бит (целое число)
- Двойное слово – 64 и 128 бит (вещественное число)

4.2. Вещественные числа

Стандарт IEEE 754 (1985 г.).

Профессор Университета Беркли Вильям Каган

Одинарная точность

знак	порядок		мантисса	
31	30	23	22	0

Порядок 8 бит со сдвигом 127, мантисса не 23, а 24 бита, старшая 1 подразумевается

Двойная точность

знак	порядок		мантисса	
63	62	52	51	0

Порядок 11 бит со сдвигом 1023

4.3. Особенные вещественные числа

- **NaN** (not a number)
  Порядок- все единицы
  Мантисса – не нулевая
- **Бесконечность**
  Порядок – все единицы
  Мантисса – 0
- **Нуль**
  Порядок – 0
  Мантисса – 0
- **Денормализованные числа**
  Порядок – 1
  Мантисса – не равна 0, подразумеваемый старший разряд не 1, а 0. Денормализованные числа дают возможность работать с очень маленькими значениями, но большое замедление

5.1. Представление нечисловых данных, коды литер

ASCII (American Standard Code for Information Interchange) 1963 г., КОИ 8

EBCDIC (Extended Binary Coded Decimal Interchange Code) IBM, ДКОИ 8

Unicode 1991г. 2 байта, UTF 16

UTF 8 (Unicode Transformation Format) 1992г.

В первом байте часть битов указывает количество байтов в коде литеры:

0xxxxxxx – 1 байт, совпадает с ASCII,

110xxxxx – 2 байта,

1110xxxx – 3 байта, т.д. (всего до 6 байтов)

В остальных байтах кода 10xxxxxx

5.2. Представление графических данных

Растровое изображение – совокупность точек, используемых для отображения на экране монитора.

Черно-белое изображение – 1 байт,

0000000 – не подсвечивается

11111111 – черный

Между ними оттенок серого

Цветное RGB – 3 байта, яркость красного, зеленого и синего

В HTML:

FF0000 интенсивный красный

00FF00 интенсивный зеленый

0000FF интенсивный синий

000000 черный

FFFFFF белый

FF – наибольшая яркость

Векторное изображение - набор примитивов из элементарных отрезков кривых с параметрами (координаты узловых точек, радиусы кривизны и пр.), которые описываются математическими формулами.

5.3. Представление записей

Запись (иногда говорят «структура») – последовательность полей произвольных типов

struct {float a; int b; char c;} s;

Если float занимает 8 байтов, int 4 байта, а char 1 байт, то s займет в памяти 13 байтов, но

struct {char c; int b; float a;} s1;

может занять и 20 и 24 байтов, так как современные устройства памяти требуют определенного выравнивания (например, если хочешь прочитать или записать 8-ми байтовое значение, то адрес должен быть кратен 8)

5.4. Доступ к полю записей

struct {float a; int b; char c;} s;

s.a, s.b, s.c – выборка поля записи.

Выборка поля – статический объект, нельзя, например, организовать цикл по полям записи, зато выборка ничего не стоит, так как смещение каждого поля записи можно определить во время трансляции.

Тип поля может быть любой, например, записью или массивом.

5.5. Представление массивов

Массив – это последовательность элементов одинакового типа, например,

[L : U] int A;

Во многих языках программирования границы массива L и U могут вычисляться во время счета, в Паскале и Си – только константы, причем в Си L – всегда равен 0.

Адрес элемента массива A[i] вычисляется по формуле C_0 + *i*d*, где

C_0 - индекс элемента с индексом 0

d – шаг, то есть размер элемента

5.6. Представление многомерных массивов

Бывают многомерные массивы, например,

[L1:U1, L2:U2] int M;

Тогда адрес элемента M[i, j] (иногда говорят «вырезка элемента») вычисляется по формуле

C_0 + *i*d1* + *j*d2*, где

d1– шаг, по первому измерению (в данном случае – длина строки)

d2 – шаг по второму измерению.

Эта формула очевидным образом обобщается на массивы произвольной размерности.

5.7. Векторы Айлифа

В некоторых языках (например, в Си) многомерные массивы представляются специальным образом, например, массив

Int M [U1] [U2];

Представляется как одномерный массив из

U1 элементов, каждый из которых является адресом одномерного массива из U2 элементов, тогда вырезка элемента записывается M[i] [j].

Представление массивов в виде векторов Айлифа позволяет избежать дорогостоящей операции умножения, кроме того, можно сформировать, например, двумерный массив из строк разной длины

6.1. Системы и типы команд ЭВМ

Современные ЭВМ имеют достаточно много регистров, доступ к которым во много раз эффективнее, чем доступ к памяти. Команды ЭВМ делятся по формату (регистр – регистр, регистр – память, память - память) и по типу:

- Работа с памятью (чтение в регистр, запись из регистра в память, семафорное чтение);
- Арифметические команды (+; -; *; /);
- Логические команды (И; ИЛИ; НЕ);
- Команды передачи управления;
- Команда ввода/вывода SIO.

6.2. Команды передачи управления

- Безусловная передача управления

Branch Label

Label – метка в программном коде, куда нужно передать управление

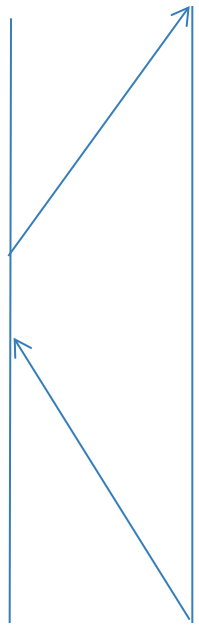
- Условная передача управления

Branch Condition Label

Condition – условие, при выполнении которого необходимо совершить передачу управления, в противном случае управление передается на следующую команду

6.3. Механизм вызова подпрограмм

BAL R, Label



Это команда Branch And Link
Адрес следующей команды записывается в регистр R, а управление передается на метку Label, которая обозначает начало подпрограммы. В конце подпрограммы ставится команда BR R (передача управления по содержимому регистра R). В больших компаниях следуют соглашению о связях, в частности, для возврата из подпрограммы используют всегда один и тот же регистр (например, в IBM – это 14-й регистр). Поскольку внутри подпрограммы могут быть вызовы других подпрограмм, в начале каждой подпрограммы все регистры запоминаются в памяти, а перед возвратом восстанавливаются.

6.4. Прерывания

В некоторых случаях нужно прервать последовательное исполнение текущего процесса и запустить какую-то служебную программу, например,

- Аварийная ситуация (деление на 0, переполнение, выход индекса за границу массива и т.д.)
- Завершение операции ввода/вывода
- Завершение кванта времени, выделенного текущему процессу.

6.5. Обработка прерываний

- Каждое прерывание имеет свой номер
- В конце памяти размещается последовательность команд безусловной передачи управления.
- При возникновении прерывания с номером N, происходит аппаратная передача управления по адресу K-N, где K – адрес конца памяти, а там стоит команда передачи управления на соответствующий обработчик.

Такая косвенная передача управления при возникновении прерывания нужна для возможности подмены обработчиков.

7.1. Функциональная организация устройств

Первые ЭВМ имели прямую аппаратную реализацию для всех своих команд, если какое-то действие, например, сложение использовалось в нескольких командах, для каждой такой команды формировался свой сумматор, т.е. реализация была слишком избыточной с точки зрения объемов аппаратуры. И сегодня суперкомпьютеры реализуются именно так.

В 1951 г. Морис Винсент Уилкс (Кембридж, Англия) предложил принципиально другое решение – каждая команда ЭВМ реализуется в виде последовательности мелких аппаратных действий (микрокоманд). Это позволяет существенно уменьшить избыточность аппаратуры, но уменьшает скорость работы ЭВМ из-за времени выборки микрокоманды из микропамяти и ее интерпретации.

7.2. Микропрограммирование

- На протяжении каждого такта всей ЭВМ управляет регистр микрокоманд – двоичная шкала размером 60-100 разрядов, в которой каждое поле управляет каким-то элементом ЭВМ. Эта шкала и называется микрокомандой
- Группа микрокоманд, реализующая команду ЭВМ, называется микропрограммой
- Микропрограммы хранятся в специальной быстрой микропамяти
- Устройство управления выбирает из водопровода очередную команду, по ее коду определяет адрес начала соответствующей микропрограммы и загружает первую микрокоманду из нее в регистр микрокоманд
- Одно из полей микрокоманды управляет выборкой следующей микрокоманды, которая осуществляется по концу текущего такта

7.3. Реализация микропрограмм

- Первые микропрограммы первых ЭВМ так и готовились в виде длинных последовательностей нулей и единиц
- Позже появились микроассемблеры (самый известный фирмы AMD), но резкого повышения производительности труда микропрограммистов не произошло, для составление микропрограмм одной ЭВМ требовалось десятки человеко-лет
- В нашем коллективе Николай Фоминых придумал и реализовал способ задания микрокоманд на алгоритмическом языке высокого уровня (на Алголе 68), использовав возможности этого языка по созданию специализированных языков на его основе с помощью описания новых типов и операций над данными этих типов.

7.4. Пример микропрограммы

Команда «Ч Смещение» – загрузить на стек слово из памяти по адресу регистр L+ Смещение (1 байт)
(ВД1=> АЛУ (D+L) => БА, КМК);
(АЛУ (вверх), КМК);
(БД => АЛУ (D -> S), КМПУ (Г));
(БД => АЛУ (D -> S), КМП);

ВД1 – взять очередной байт из водопровода
АЛУ – арифметико-логическое устройство
D – входной регистр АЛУ
БА – буфер адреса памяти, запись туда адреса означает начало операции чтения из памяти
КМК – конец микрокоманды, переход на следующую микрокоманду
Вверх – поднять указатель стека S в АЛУ
БД – буфер данных памяти
КМПУ (Г) – условный конец микропрограммы по триггеру готовности памяти
КМП – безусловный конец микропграммы

7.5. Необходимые пояснения

- В методе Николая Фоминых микропрограммист по-прежнему пишет каждую микрокоманду отдельно, но система не даст ему написать операцию с неправильными типами данных
- => - операция пересылки данных по шине компьютера
- -> - операция пересылки внутри АЛУ
- + - это не сложение целых чисел, а обозначение соответствующей операции внутри АЛУ
- Таких операций Николай Фоминых описал более сотни, основной объем этих описаний занимают проверки соответствия запрошенных действий аппаратуре кристаллов
- Микропрограмма – это обычная программа на языке Алгол 68, запуск которой приводит к порождению битовых шкал микрокоманд

8.1. Иерархия памяти

- Регистры, расположены прямо в процессоре
- Кэш память для хранения часто используемых фрагментов памяти
- Основная оперативная память
- Дисковая память
- Память на магнитных лентах (стриммеры)
- В старых ЭВМ активно использовались перфоленты и перфокарты

8.2. Кэш память

- **Кэш-память** содержит копии наиболее часто используемых участков основной памяти и строится на быстрой статической памяти. Кэшируются не отдельные слова, а линейки из последовательности байтов основной памяти (скажем, в архитектуре x86 длина линейки составляет 64 байта).
- Кэш-память является своеобразной hash-таблицей, где hash образуется из адреса основной памяти отбрасыванием нескольких младших битов.
- Линейка, на которую адрес попал, переносится в начало списка, а линейки, на которые долго не было попадания, выбрасываются из списка.

8.3. Принципы работы основной памяти

- **Статическая память** на триггерах, быстрая, но дорогая
- **Динамическая память** на конденсаторах, медленнее, чем статическая, но и значительно дешевле. Поскольку идеальных диэлектриков не бывает, заряд с конденсатора постепенно стекает, поэтому нужно регулярно делать чтение и перезапись (refresh), что еще больше ее замедляет.

8.4. Характеристики основной памяти

- **Latency** – задержка. Если 20 лет назад память была медленнее процессора в 3 – 4 раза, то сегодня – в 15-20 раз
- **Cycle time** – длительность такта (см. слайд 3.1.).
- **Bandwidth** - пропускная способность, т.е. ширина шины доступа процессор-память. *Пример:* первые IBM PC XT базировались на микропроцессоре i8086, процессор и шина 16-ти разрядные. Когда перешли на i8088 с 16-ти разрядным процессором, но с 8-ми разрядной шиной, в несколько раз уменьшили цену персонального компьютера.

9.1. Виртуальная память

- **Virtualis** (лат.) означает «возможный», объект, который не существует, но может возникнуть.
- **Страница** – фрагмент оперативной памяти фиксированного размера (скажем, 2 Кб)
- **Математический адрес** – пара (N, D), где N – номер страницы, D – смещение нужного адреса от начала страницы
- **Таблица страниц** – таблица, входом которой является номер страницы, а содержимым – адрес страницы в оперативной памяти, если она находится в этой памяти, или информация о ее месте на диске, если этой страницы нет в памяти
- **Главный принцип виртуальной память** – использование прямых физических адресов запрещено, только относительные математические.

9.2. Реализация виртуальной памяти

- Перед каждым использованием математического адреса для обращения к памяти его нужно преобразовать в физический адрес
- По номеру страницы обращаемся к содержимому таблицы страниц
- Если страница присутствует в памяти, то просто добавляем к ее адресу начала смещение, если же нет – то вызывается процедура операционной системы по подкачке нужной страницы. Если свободной памяти для этой страницы нет, то приходится откачивать несколько страниц на диск

9.3. Способы ускорения виртуальной памяти

- Если нужной страницы не оказалось в оперативной памяти, говорить о быстродействии не приходится
- Оказывается, при работе любой программы быстро формируется рабочее множество страниц, к которым осуществляется 90-95% обращений
- Но даже, если страница в памяти находится, преобразование математический адрес → физический адрес сильно замедляет работу
- TLB (Translation lookaside buffer) – специализированная кэш-память, где находятся адреса часто используемых страниц.

9.4. Обработка ошибок в памяти

- **Ошибки** можно компенсировать только избыточной информацией
- Самый простой способ - к каждому байту информации добавить бит четности, если сумма по модулю 2 всех битов, включая дополнительный, равна 0, все в порядке, этот способ помогает только фиксировать ошибку, но никак не помогает в ее исправлении
- **Код Хэмминга**, например, на 16 разрядов можно добавить 5 контрольных разрядов (контрольные суммы в 1, 2, 4, 8, 16 позициях), тогда можно не только обнаружить одиночную ошибку, но и исправить ее
- Существуют и другие, более сложные коды.

10.1. Основы ввода/вывода

- **Канал** – специализированный процессор, отвечающий за обмен данными с определенным устройством
- **Блок данных** – порция данных фиксированного размера
- **Запись** – единица обработки данных, обычно блок состоит из многих записей
- Как уже говорилось, центральный процессор запускает операцию обмена, выполняя команду SIO с параметром – номером канала, после чего центральный процессор и канал работают параллельно

10.2. Буферизация

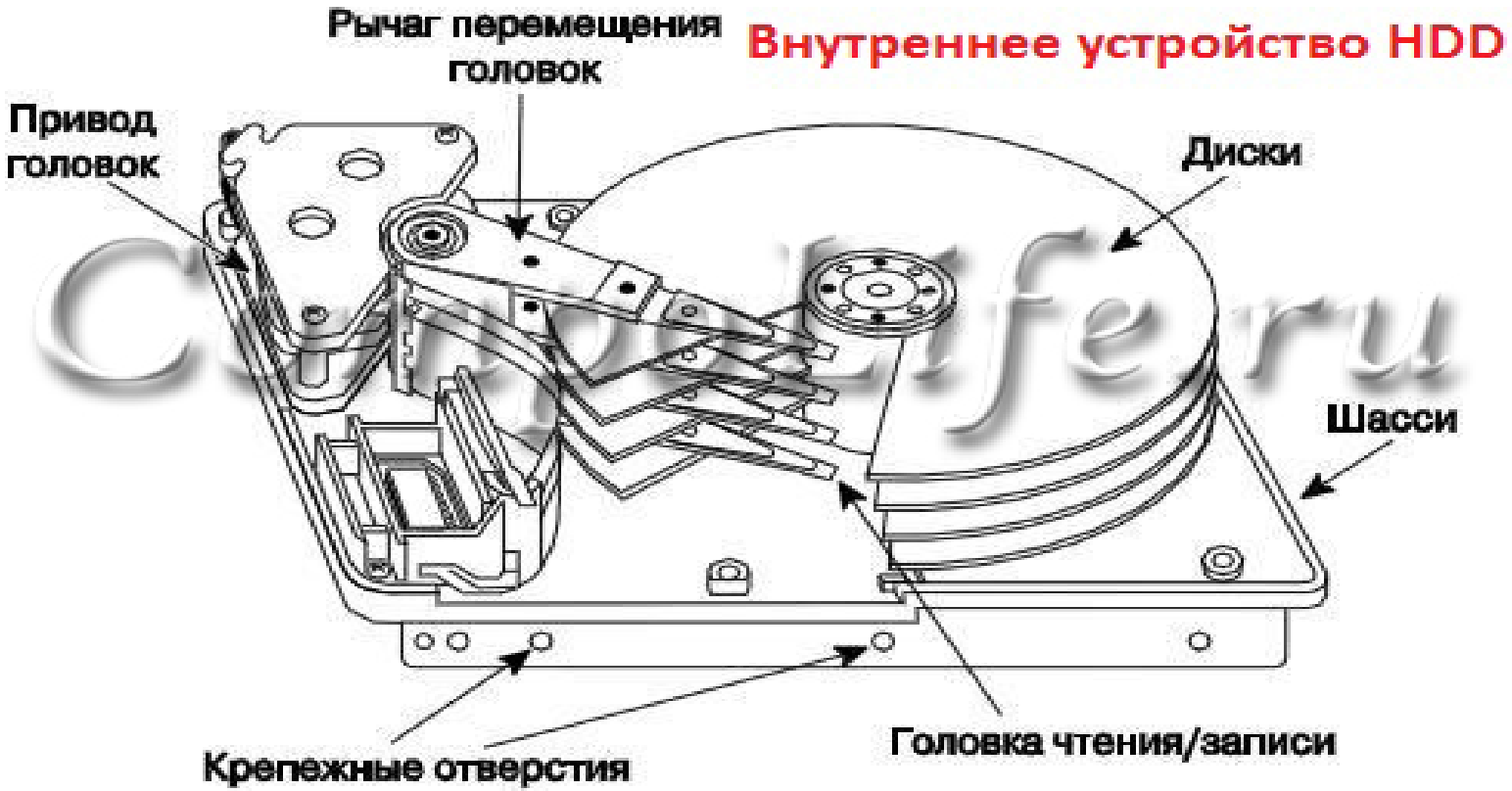
- Пусть есть большой файл, который нужно последовательно обработать
- Чтобы не терять время на чтение блока, можно отвести два или больше буферов, тогда задержка будет только на чтение первого блока файла, затем прочитанный буфер обрабатывается, а следующие параллельно с этим читаются
- Нужно управлять размером и количеством буферов, чтобы время обработки одного буфера было примерно равно времени его чтения, тогда центральный процессор будет постоянно загружен.

10.3. Завершение операции ввода/вывода

- **Программируемый ввод/вывод** – центральный процессор сам регулярно опрашивает определенные регистры устройства в ожидании завершения операции. Иногда это действие называют сканированием, такая система ввода/вывода применяется во встроенных системах, когда сканируемых устройств много
- **Ввод/вывод по прерыванию** – канал по завершении операции вызывает прерывание центрального процессора, во время обработки которого запускается следующая операция обмена, этот способ более эффективен.

11.1. Внешние накопители - 1

Жесткий диск – HDD (Hard Disk Drive), иногда говорят «винчестер»



http://compolife.ru/ustrojstvo-kompjutera/chto-takoe-zhestkij-disk.html

11.2. Внешние накопители - 2

Стримеры – ленточные накопители, используемые для хранения больших объемов данных, например, для создания резервных копий дисков

Ленточное хранилище данных в CERN (фото: hardware.slashdot.org).

11.3. Внешние накопители - 3

- **CD ROM** – (*Compact Disc Read-Only Memory*) разновидность компакт-дисков (CD) с записанными на них данными, доступными только для чтения
- **CD-RW** - (*Compact Disc-ReWritable)* разновидность компакт-диска (CD) для многократной записи информации

https://ru.wikipedia.org/wiki

11.4. Шины

- **SCSI** (Small Computer System Interface) SCSI-стандарты определяют команды, протоколы и электрические и оптические интерфейсы для объединения на одной шине различных по своему назначению устройств.
- **USB** (*Universal Serial Bus*) – один из наиболее распространенных последовательных интерфейсов с универсальным разъемом
- **PCI-Express** – шина третьего поколения, обеспечивающая существенно более высокую скорость передачи данных

11.5. Прямой доступ к памяти

DMA – direct memory access. В первых ЭВМ любое обращение к памяти шло через центральный процессор. С ростом объемов дисковой памяти и скорости их работы стало понятно, что хорошо бы разгрузить процессор от этой работы. Именно с этой целью появились устройства DMA, организующие обмен данными между дисками и оперативной памятью, минуя центральный процессор, что позволило существенно ускорить этот процесс.

11.6. Арбитраж

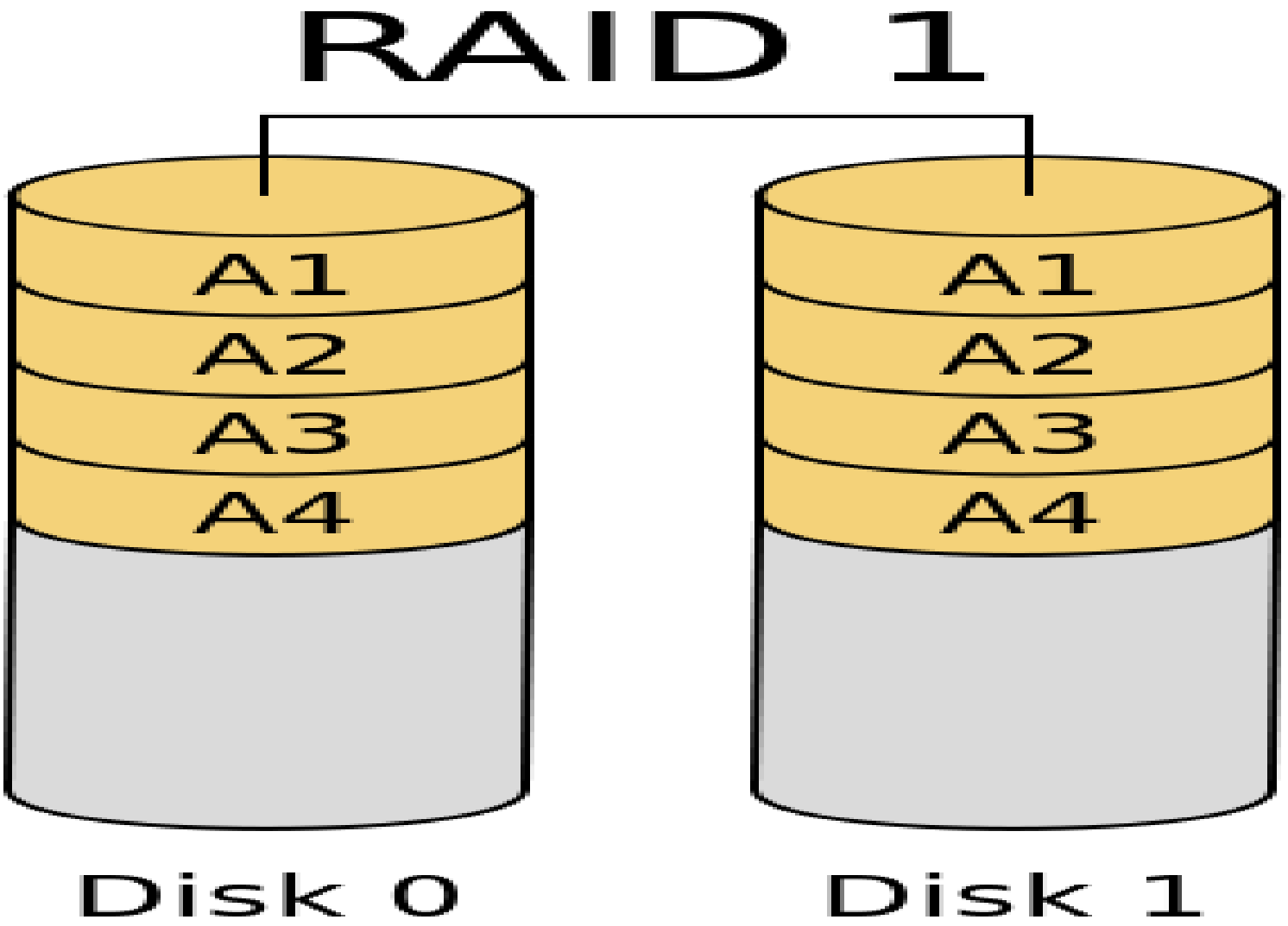
Иногда возникают коллизии при обращении к каким-либо устройствам, чаще всего, к памяти, когда к одному устройству в один и тот же такт обращаются несколько устройств. Для разрешения этих коллизий придуманы специальные электронные схемы, получившие название «Арбитраж». Например, если к памяти обращается процессор за данными и водопровод за командой, приоритет отдается процессору

12.1. RAID-архитектуры

RAID - Redundant Array of Independent Disks («избыточный (резервный) массив независимых дисков»), т.е. несколько дисков рассматриваются как один повышенной надежности.

- RAID 1 — зеркальный дисковый массив;
- RAID 2 — зарезервирован для массивов, которые применяют код Хемминга;
- RAID 3 и 4 — дисковые массивы с чередованием и выделенным диском чётности, RAID 4 применяется редко;
- RAID 5 — дисковый массив с чередованием и отсутствием выделенного диска чётности.

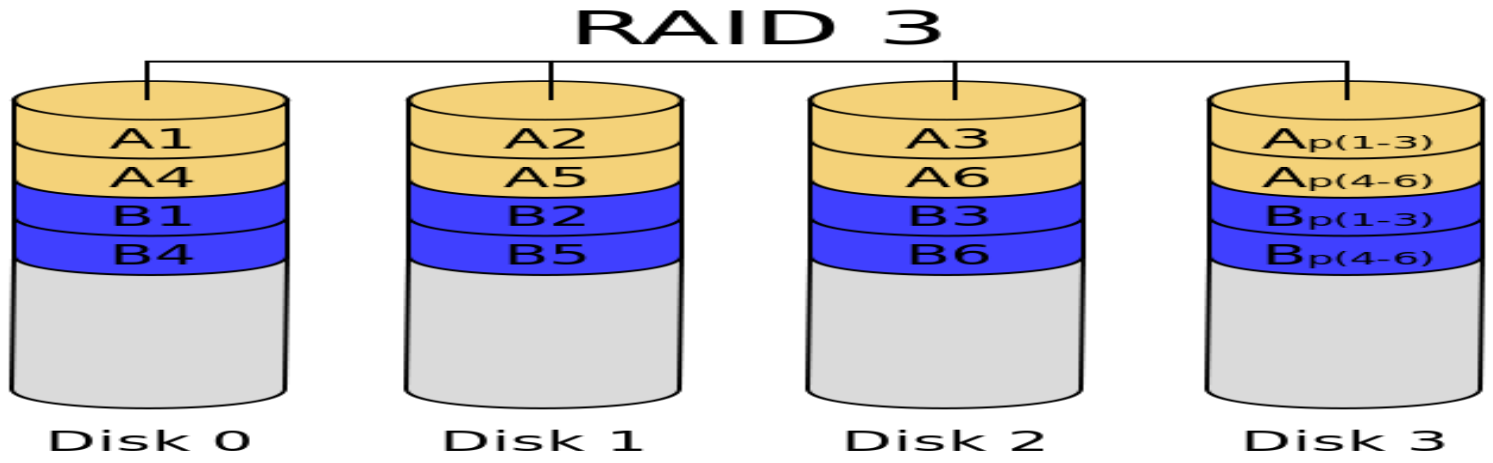
12.2. RAID 1



https://ru.wikipedia.org/wiki/RAID

12.3. RAID 3

Данные хранятся на n-1 диске, причем разбиваются на куски размером меньше сектора диска. На одном диске хранятся блоки четности этих кусков, что дает возможность восстановить данные в случаях сбоев. Ранее применялся RAID-2, в котором на один диск данных использовалось n-1 дисков четности, что давало возможность исправления ошибочных фрагментов, но это решение оказалось слишком затратным.



https://ru.wikipedia.org/wiki/RAID

12.4. RAID 5

Основным недостатком уровней RAID от 2-го до 4-го является невозможность производить параллельные операции записи, так как для хранения информации о чётности используется отдельный контрольный диск. RAID 5 не имеет этого недостатка. Блоки данных и контрольные суммы циклически записываются на все диски массива, нет асимметричности конфигурации дисков. Контрольные суммы формируются с помощью операции XOR, что дает возможность восстановить информацию в случае сбоя одного из дисков.

RAID 5 обеспечивает высокую скорость чтения — выигрыш достигается за счёт независимых потоков данных с нескольких дисков массива, которые могут обрабатываться параллельно.

13.1. RISC – Архитектура

- Reduced Instruction Set Computer
- Проект «RISC» в Университете Беркли был начат в 1980 году под руководством Дэвида Паттерсона и Карло Секвина.
- В 1981 году, Джон Хеннесси начал аналогичный проект в Стэнфордском университете.
- Принципы RISC
  - *Все команды имеют одинаковую длину*
  - *Все (или почти все) команды исполняются за один такт*
  - *Много регистров*
  - *Работа с памятью только в двух командах – Load и Store*
  - *Программирование только на АЯВУ*

RISC

13.2. Водопровод (prefetching)

- Еще раз, с удовольствием, повторю, что схему опережающего чтения команд из памяти придумал и опубликовал в 1955 г. в США советский ученый С.А. Лебедев. Он назвал эту схему «водопроводом», так что термин «pipeline» является прямым переводом с русского, позже в англоязычной литературе стали использовать другой термин «prefetching»
- Поскольку по статистике каждая 6-8 исполняемая команда является передачей управления, делать аппаратный буфер водопровода на большее число позиций не целесообразно
- Если 20 лет назад память была в 4-5 раз медленнее процессора, то сегодня уже в 15-20 раз, поэтому актуальность водопровода только возрастает.

13.3. Предсказывание переходов

- После любой команды передачи управления водопровод нужно заново разгонять, при этом процессор будет простаивать, поэтому в условных предложениях и циклах нужно запускать два водопровода (после then и else, начало цикла и команды после цикла)
- Современные ЭВМ умеют собирать динамическую статистику переходов и на более вероятные ветви удлинять глубину водопровода, а на менее вероятные – укорачивать.

13.4. Чередование (interleaving)

- Одно из наиболее частых и времяемких действий – это работа с массивами, при этом идет обращение к последовательным адресам памяти
- Можно сделать несколько (4,8, …) банков памяти, каждый со своим устройством управления, причем адреса идут последовательно по банкам, а не внутри одного банка, т.е. адрес 0 - в первом банке, адрес 1 – во втором и т.д., если банков 4, то адрес – 4 снова в первом банке
- В векторных операциях можно за один такт прочитать или записать столько слов памяти, сколько банков памяти имеется в наличии.

14.1. Введение в аппаратный параллелизм уровня машинных команд

ILP Instruction-Level Parallelism

- Мы уже говорили о **водопроводе** – своеобразном конвейере, читающем команды из памяти впрок
- Позже появились **конвейеры исполнения команд**, начиная с Intel 80486 (1989 г.)
- **Суперскалярное выполнение** – если ЭВМ имеет несколько сумматоров, умножителей и других устройств, можно выполнять несколько команд параллельно, первым суперскалярным процессором стал Intel Pentium (1993 г.)
- **Спекулятивное выполнение**, например,
If a >0 then x := y+z; здесь можно начать исполнение присваивания параллельно с проверкой условия, но собственно запись в x произойдет только, когда станет известной истинность условия
- **Предсказание переходов** – ЭВМ может собрать статистику условных передач управления и увеличивать буфер водопровода более вероятных веток

14.2. Многопроцессорные и альтернативные архитектуры-1:

- **SIMD** – single instruction, multiple data, несколько ЭВМ каждый такт выполняют одну команду, но со своими данными, одна выделенная ЭВМ раздает команды на исполнение.

Пример: ILLIAC IV (1975 г.) 64 компьютера. Был советский аналог ПС 2000

- к архитектуре SIMD можно отнести векторные ЭВМ, например, CDC STAR-100, Cray-1, Fujitsu AP1000 и AP3000, была оригинальная советская векторная ЭВМ ПС 3000

14.3. Многопроцессорные и альтернативные архитектуры-2:

- **MIMD** - multiple instruction, multiple data

Пример: транспьютер фирмы Inmos (1985 г.)
микропроцессор с 4-мя каналами, по которым он может соединяться с такими же микропроцессорами, для транспьютеров был придуман специальный язык параллельного программирования ОККАМ

Пример: Intel Xeon Phi (2012 г.)

14.4. Многопроцессорные и альтернативные архитектуры-3:

- **VLIW** - very long instruction word (1980 г.), одна длинная машинная команда содержит в себе много инструкций. Это существенно компиляторный подход в отличие от аппаратного параллелизма – только оптимизирующий компилятор может заполнить все или почти все позиции команды. Первые компьютеры с такой архитектурой – это Cydrome, MultiFlow, DSP C6000, Эльбрус 2000, Эльбрус S

14.5. Многопроцессорные и альтернативные архитектуры-3:

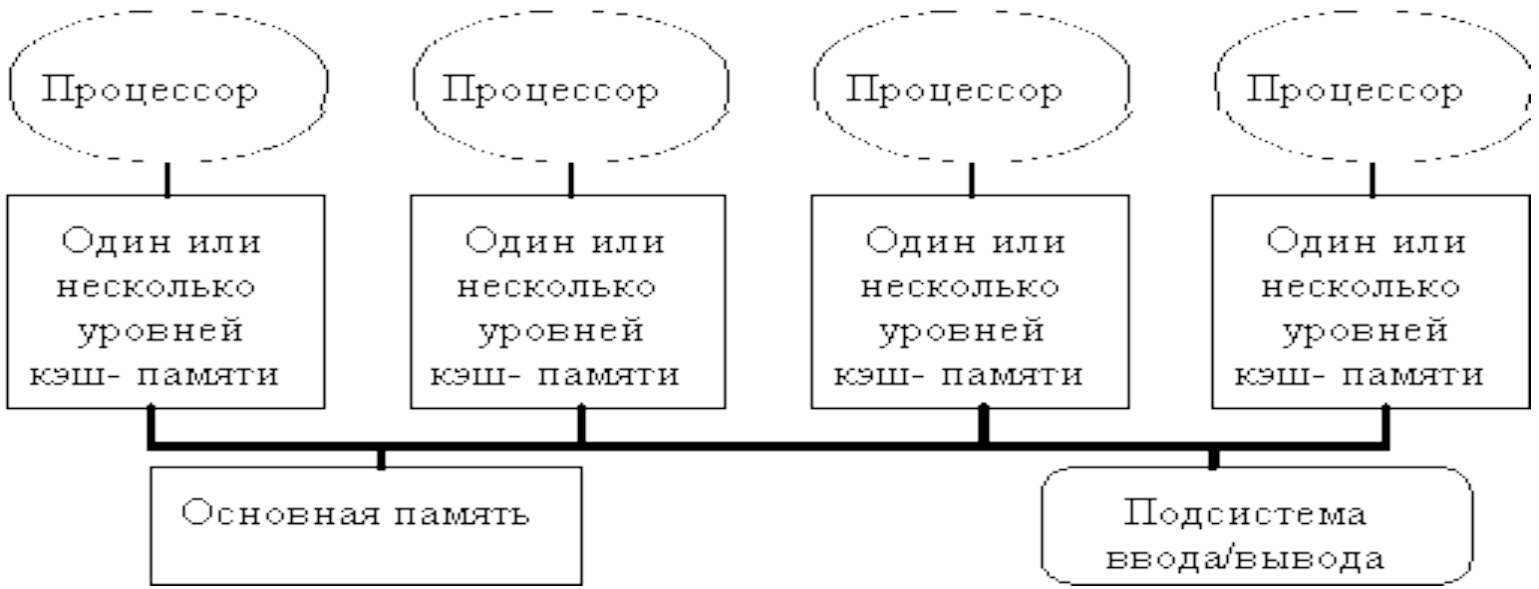
- При всех достоинствах VLIW по сравнению с аппаратной реализацией, возникли и проблемы, например, синхронизация кэша для разных действий внутри одной длинной команды. Для решения этих проблем в 1997 г. фирмы HP и Intel предложили развитие архитектуры VLIW
- **EPIC** - explicitly parallel instruction computing. Первый пример Itanium фирмы Intel
- Предикация – команды из разных веток условного предложения запускаются параллельно, но со специальными полями условий
- Спекулятивная загрузка в регистры из памяти, команды Load запускаются существенно раньше, но перед реальным использованием регистра

15.1. Систолические архитектуры

Это несколько аппаратных конвейеров, состоящих из небольших вычислительных элементов (систол), «заточенных» под конкретную задачу. Все систолы работают от единого тактового генератора. Систолические архитектуры применяются для задач массовой обработки данных (обработка сигналов, машинная графика, биоинформатика, базы данных).

Современные систолические решения основаны на парадигмах SIMD/MIMD, являются репрограмируемыми и реконфигурируемыми.

15.2. Архитектуры с общей памятью



http://bourabai.ru/dbt/servers/glava_10.htm

Реального ускорения в N раз (N - количество процессоров) добиться невозможно из-за конфликтов обращений к общей памяти. Ситуацию несколько сглаживает наличие у каждого процессора большой кэш-памяти.

15.3. Обеспечение соответствия кэша и памяти

Кэш (Cache) – относительно небольшая быстрая память для часто используемых данных и команд.

Кэш состоит из набора записей (обычно несколько десятков слов), являющихся копией соответствующих слов основной памяти.

Кэш – это hash-таблица, входом в которую является адрес без нескольких младших битов. Если какой-то элемент данных нашелся в кэше, он помещается в начало списка элементов с одинаковым hash. Редко используемые данные выпадают из кэша

16.1. Современные архитектуры. Мобильные устройства

Современный мобильный телефон – это мощная ЭВМ с огромной памятью и высокой производительностью.
Главные преимущества мобильного устройства (по сравнению с персональной ЭВМ) – всегда включен, всегда в кармане, всегда в сети!
Сегодня многие пользователи предпочитают искать информацию в интернете на мобильном устройстве, а не с помощью персональной ЭВМ.
Тем не менее у мобильных устройств есть и принципиальные ограничения связанные с форм-фактором: маленькие размеры клавиатуры, маленький экран, часто протоколы связи с базовой станцией (GPRS, EDGE, 3G) имеют существенно меньшую пропускную способность по сравнению с локальными сетями ЭВМ.

16.2. Современные архитектуры. Встроенные системы - 1

Встроенная система – это программно-аппаратная система, в которой одна или несколько ЭВМ управляют каким-то специализированным оборудованием (цифровая телефонная станция, ракета, сердечный стимулятор и т.д.).

16.3. Современные архитектуры. Встроенные системы - 2

Многие авторы признают программирование СРВ самой трудной задачей нашей специальности по следующим причинам:

1. Необходимость укладываться в жёсткие временные ограничения, вне рамок которых важная информация просто пропадает;
2. Обычно СРВ управляются не одной ЭВМ, а многими, соединёнными в сложноустроенные сети, поэтому в СРВ входит вся многообразная тематика сетей (протоколы, устойчивость к сбоям и отказам, реконфигурация, пропускная способность и т.д.);

16.4. Современные архитектуры. Встроенные системы - 3

3. Любая цифровая телефонная станция включает в себя большие базы данных (абоненты, блоки аппаратуры, учёт трафика и т.д.), поэтому и проблематика БД, сама по себе весьма не простая, в полном объёме включается в СРВ;
4. Часто (особенно для больших станций) станцией управляет один или несколько операторов, поэтому все проблемы человеко-машинных интерфейсов также приходится решать в процессе проектирования цифровых телефонных станций;
5. Самая большая трудность при отладке СРВ заключается в неповторяемости ситуаций. Цифровая телефонная станция может успешно работать 2 года, а потом вдруг «вылететь», причём понять причину отказа очень трудно.

16.5. Направления развития архитектуры процессоров -1

Долгие годы соблюдался закон Мура «каждые полтора года производительность процессоров удваивается». Достигалось это за счет увеличения количества вентилей в кристалле, уменьшение размеров вентилей и шага между соединительными линиями. Сегодня лучшие фабрики по производству кристаллов работают с шагом 18-22 нм, но при шаге меньше 10 нм уже будут наблюдаться квантовые эффекты, когда вместо значений 0 и 1 придется рассматривать континуум непрерывных значений от 0 до 1.

16.6. Направления развития архитектуры процессоров -2

Кардинальнее направление развития – параллельность (много процессоров, много ядер в одном кристалле, много блоков памяти). Уже сейчас есть 64 ядерные процессоры, в ближайшее время появятся процессоры с тысячами ядер, но здесь инженеры существенно опережают программистов-технологов. Есть технологии MPI, OpenMP и некоторые другие, но все они далеки от совершенства. Создать параллельную программу по-прежнему очень трудно.

17.1. Почему мы решили создать свою ЭВМ. История вопроса

- Мы начали работать с военными в 1980 г. Основным направлением деятельности было повышение производительности труда программистов за счет внедрения подмножества языка Алгол 68, ориентированного на задачи коммутации. Мы назвали его А68К.
- За 5 лет мы разработали более 20-ти кросс-трансляторов для специализированных ЭВМ УК 1010, СУВК СС, СУВК СМ, НЕВА и др.
- Все эти ЭВМ устойчивы к сбоям питания, тряске, сушке и т.п., но программировать для них совершенно невозможно. Мне решили, что спасение утопающих дело рук самих утопающих! Математики могут сделать ЭВМ для себя лучше, чем инженеры

17.2. HLL компьютеры

- Мы уже знали термин HLL – High Level Language Computer (компьютер, ориентированный на алгоритмические языки высокого уровня). Эти компьютеры аппаратно поддерживают основные конструкции языков программирования, поэтому трансляция в их коды существенно упрощается.
- Существовавшие на середину 80-х гг. HLL компьютеры (МИР,Burroughs, Эльбрус) были существенно дороже аналогичных по мощности традиционных компьютеров. Отрицательный рекорд поставила серия кристаллов, ориентированная на реализацию языка АДА, iAPX432 – производительность была в 100 раз хуже, чем могла бы быть при такой же технологии.
- Нужна была свежая идея

17.3. Архитектуры HLL на примере организации УВК Самсон

- Основная идея управляющего вычислительного комплекса Самсон может быть выражена простой схемой:

Программа на АЯВУ -> компилятор -> УВК -> результат

Причем никаких обходных путей нет.

- И компилятор, и УВК разрабатываются в одном коллективе, поэтому всегда можно договориться, что если что-то проверено компилятором, то УВК эти проверки не повторяет.
- Поскольку мы хотели большинство проверок выполнить во время компиляции, мы решили ограничить множество входных языков: Алгол 68, Ада, Модула 2 и другие языки статического типа

17.4. Разделение обязанностей между компилятором и УВК Самсон

- Поскольку программирование на ассемблере не предусматривается, многие ошибки типа неправильные тип операции, адресация, номер регистра и т.п. в аппаратуре не предусмотрены – *компилятор не может ошибиться!*
- С другой стороны наличие в УВК команд типа вызов процедуры, вырезка элемента массива, цикл любой самой сложной природы резко упрощает компилятор, например, если есть 13 действий и 4 типа адресации, то в УВК будет 52 команды без каких-либо исключений. Слишком сильно мы настрадались при разработке компиляторов для других ЭВМ (в IBM/360 есть команды AH, SH, MH, но нет команды DH)

17.5. Стековая архитектура УВК Самсон

- В УВК Самсон все действия выполняются в стеках:

Стек целых (16 позиций по 16 разрядов)
Стек вещественных (8 позиций по 32 разряда)
Стек адресов (16 позиций по 24 разряда)

- Никаких проверок исчезновения или переполнения стека, традиционных для других HLL компьютеров (есть единственное исключение – команда вызов процедуры следит за возможностью переполнения стека), не предусмотрено – за все отвечает компилятор.
- Он же отвечает за корректность использования стеков

17.6. Работа с памятью

- Память УВК Самсон разбита на сегменты переменной длины (работать с внешней фрагментацией мы умеем, а с внутренней, которая возникает при использовании страничного метода - нет).
- Каждый процесс представлен сегментом кода и сегментом данных. В сегменты кода никто не пишет, поэтому параллельные процессы могут использовать один и тот же сегмент кода.
- В сегменте данных процесса размещается стек статик вызванных процедур. На начало сегмента данных указывает регистр G (global), на начало статики текущей процедуры – регистр L (Local)

18.1. Целочисленная арифметика

- УВК Самсон имеет безадресную систему команд, с памятью работают только команды Load и Store (отдельные команды для целых, вещественных и адресов).
- Все команды устроены следующим образом: берут один или два операнда из нужного стека и туда же кладут результат.
- Команды:
- +,-,*, /
- +r, -r, *r, /r – для вещественных чисел
- Важная оптимизация: пусть есть формула a-f(x), если сразу загрузить в стек a, то эта позиция стека будет занята, возможно, долго, пока вычисляется f(x). Для некоммутативных команд – и / ввели команды –обр и /обр. Тогда в формуле a-f(x) сначала вычисляется f(x), только потом загружается a, а затем командой –обр получается нужный результат

18.2. Логические команды

- Во многих ЭВМ логические команды устроены странным образом, например, в IBM/360 есть разряды CC PSW, в которых сохраняются признаки от предыдущей команды. Это удобно для команд условной передачи управления, например,
- BC 1100, Метка (это переход по = или <), но очень не удобно для логических команд.
- Реализация формулы a<b & c>d потребует длинной цепочки команд.
- В УВК Самсон значение истина представляется 1 в стеке целых, а ложь – 0. Любое сравнение вырабатывает 1 или 0.
- Логические команды: AND, OR, XOR снимают со стека целых две позиции, выполняют команду и кладут результат на верхушку стека. Команда НЕ берет одну позицию и кладет результат на стек целых.
- Вещественные сравнения снимают две позиции со стека вещественных, но результат кладут на стек целых
- Таким образом, в УВК Самсон программирование логических команд ничем не отличается от программирования арифметических команд

18.3. Передачи управления

- Исполняемые команды располагаются в специальном сегменте кодов, длина которого не превышает 64 Кбайт.
- П M16 – это команда безусловной передачи управления на адрес, отстоящий на M16 байтов от начала командного сегмента.
- П0 M16 – это команда условной передачи управления, переход произойдет если с верхушки стека целых снимается 0, иначе будет исполняться следующая команда.
- П1 M16 – условный переход по единице в стеке целых
- Переходы по длинному 16-ти разрядному смещению встречаются довольно редко, поэтому мы ввели переходы с коротким смещением.

18.4. Короткие передачи управления

- PC – это счетчик команд, то есть адрес команды, следующей за исполняемой
- B M8 безусловная передача вперед от PC на M8 байтов.
- B0 M8 условный переход по 0 в стеке целых.
- B1 M8 условный переход по 1 в стеке целых
- H M8 безусловная передача назад от PC на M8 байтов. Коротких условных переходов назад нет.
- По нашей и международной статистике в любой программе
  40% занимают команды чтения из памяти
  20% занимают команды передачи управления
  10% занимают команды сравнения
- Таким образом, частым сочетанием являются пары =, B0 M8 (эти пары возникают при трансляции условных предложений и циклов while), мы ввели команду B= M8, тем самым мы выиграли 10% длины кода и два такта.
- Исключительно для целей ортогональности были введены команды B!=, B<,B<=, B>, B>=, хотя они встречаются реже.

18.5. Циклы

- Самыми частыми циклами являются for i to n do и to n do
- to n do реализуется следующим образом: перед входом в цикл n загружается на верхушку стека целых, затем исполняются команды

НСЧ выход
Тело цикла
КСЧ начало тела цикла

Команда НСЧ проверяет, что верхушка стека (счетчик) меньше n, если это не так, то цикл не будет исполняться ни разу.

Команда КСЧ вычитает из верхушки стека 1, если результат больше 0, передает управление на начало цикла, иначе управление передается на следующую команду.

- for i to n do реализуется следующим образом: перед входом в цикл на верхушку стека целых загружаются 1 (начало цикла) и n

НЦ выход
Тело цикла
КЦ начало тела цикла

Аналогично, НЦ проверяет, нужно ли исполнять тело цикла хоть раз, КЦ прибавляет к подверхушке стека 1 и проверяет, что результат меньше или равен верхушке стека.

19.1. Работа с массивами

- Как уже говорилось, исполнение вырезки a[i] эквивалентно исполнению формулы C0+i*d, где C0 – адрес элемента массива с индексом 0, а d – это шаг (размер одного элемента). Умножение – это довольно дорогая операция.
- Если учесть, что наиболее частыми массивами являются массивы из литер, целых и вещественных чисел, то в дополнение к команде ИНД, исполняющей выше приведенную формулу, мы добавили ИНДБ (байты), ИНДЦ (целые числа) и ИНДП (вещественные числа). Соответственно в этих командах умножать надо на 1, 2 и 4, что можно сделать и без всякого умножения. Транслятор может определить тип элемента и сгенерировать соответствующую команду, сэкономив на умножении.

19.2. Хорошие массивы

- В УВК Самсон каждый массив занимает отдельный сегмент, который начинается с паспорта (16 байтов). В языке Си все массивы начинаются с индекса 0, но в других языках это не так, причем во время трансляции нижняя граница не всегда известна. Нужно как-то передать информацию о нижней границе команде вырезки во время счета.
- Мы решили, что если генератор массива видит, что нижняя граница 0 (а если 1, то вставляем фиктивный элемент, тем самым приводя нижнюю границу к 0), такой массив объявляется хорошим, и он представляется не адресом начала сегмента, а адресом, смещенным на 17. Команда вырезки в первом же такте определяет четность индекса, если он нечетный, то прибавлять С0 не нужно (1 вычитается на фоне других действий).
- Таким образом, команды ИНДБ, ИНДЦ и ИНДП, примененные к хорошему массиву, исполняются всего за 3 такта, исполнение же команды ИНД или этих команд для не хороших массивов, требует более 30-ти тактов.

20.1. Виртуальная память УВК Самсон - 1

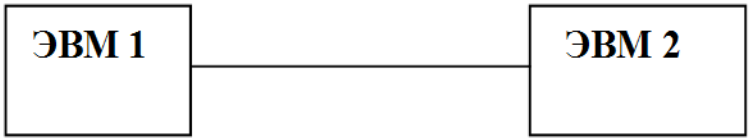
- При рассмотрении 8-го вопроса «Виртуальная память», мы видели, что преобразование математического адреса в физический требуется порядка 10-ти тактов (чтение адреса, чтение строчки таблицы сегментов и сложение).
- Такого рода действие в УВК Самсон выполняется командой ЧА (читать адрес), которая загружает полученный физический адрес на верхушку адресного стека. В нарушение всех правил организации виртуальной памяти мы разрешаем физический адрес использовать много раз без дополнительных преобразований. Поскольку и аппаратуру, и операционную систему и транслятор реализовывал один и тот же коллектив, мы договорились, что если на какой-то сегмент есть хоть одна ссылка из адресного стека, то этот сегмент не подлежит перекачке на диск или даже перемещению в оперативной памяти. Сегментов может быть до 32К, поэтому фиксация 5-10 из них в оперативной памяти не будет заметна.

20.2. Виртуальная память УВК Самсон - 2

- Обратное преобразование физического адреса в математический, осуществляемое командой ПА (писать адрес) заметно сложнее, чем преобразование математического адреса в физический, поскольку трудно найти адрес начала сегмента, на который ссылается адрес.
- Мы нашли простое решение: в дополнение к микросхеме, внутри которой находится стек адресов из 16-ти позиций по 24 разряда, мы поместили еще одну микросхему, где есть 16 позиций по 16 разрядов. Указатель стека адресов указывает на обе микросхемы, команда ЧА, кроме преобразования адресов, помещает во вторую микросхему номер сегмента, соответственно пришлось сделать шину шириной в 40 разрядов. Команда ПА по номеру сегмента обращается в таблицу сегментов, находит там адрес начала сегмента и вычитает его из физического адреса, получая тем самым смещение.

21.1. Архитектуры ЭВМ с повышенной надежностью

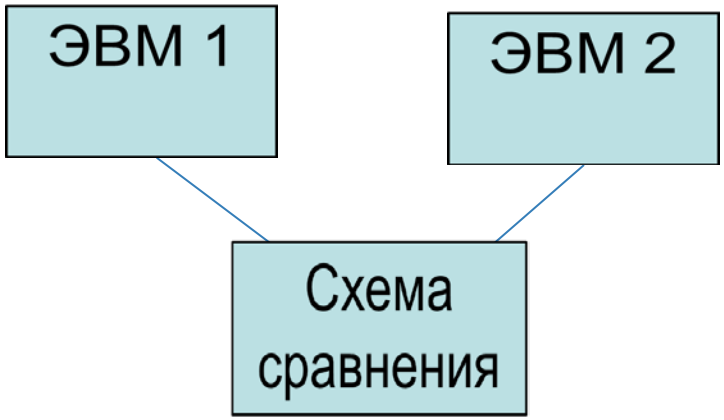
- Надежность – это всегда избыточность.
- В тех случаях, когда нужно обеспечить повышенную надежность, применяют дублированные, троированные и даже учетверенные схемы.
- Дублированные ЭВМ:



Если одна из ЭВМ выходит из строя, вторая перехватывает ее работу. Разделяют горячее и холодное резервирование, при горячем резервировании обе ЭВМ все время работают и получают одну и ту же информацию, поэтому в случае выхода из строя одной из них, другая мгновенно может продолжить полноценную работу. При холодном резервировании резервная ЭВМ простаивает, не потребляет энергию, но при переходе управления на нее требуется определенное время для загрузки текущей информации.

21.2. Дублированные ЭВМ

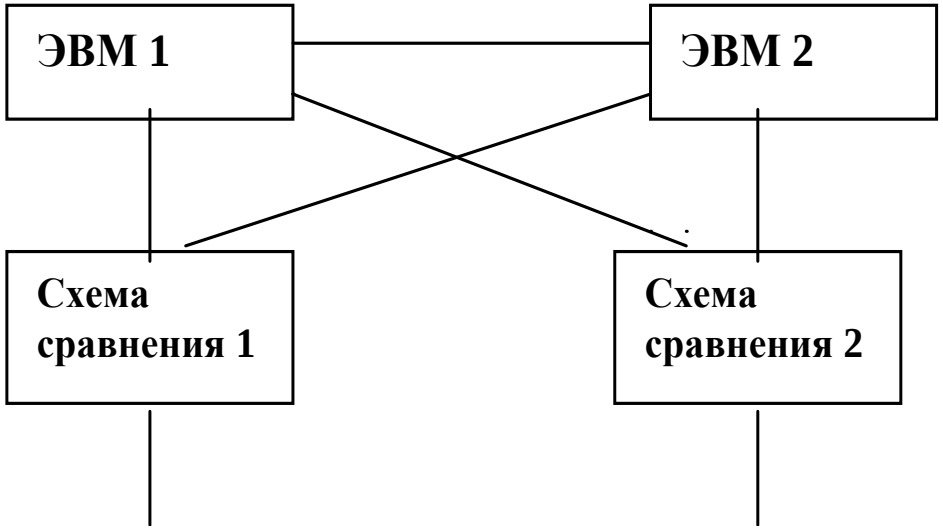
- Как понять какая из ЭВМ дала сбой? Для этого применяют специальные схемы сравнения, которые регулярно сравнивают результаты работы двух ЭВМ:



- Но даже, если схема сравнения выдала сигнал о несовпадении данных, совершенно не ясно какая из 2-х ЭВМ врет, более того, известно, что случайные сбои встречаются в 10000 раз чаще, чем постоянные отказы. Если был случайный сбой, то простое повторение действия приводит систему в идеальное состояние.

21.3. Дублированные ЭВМ - 2

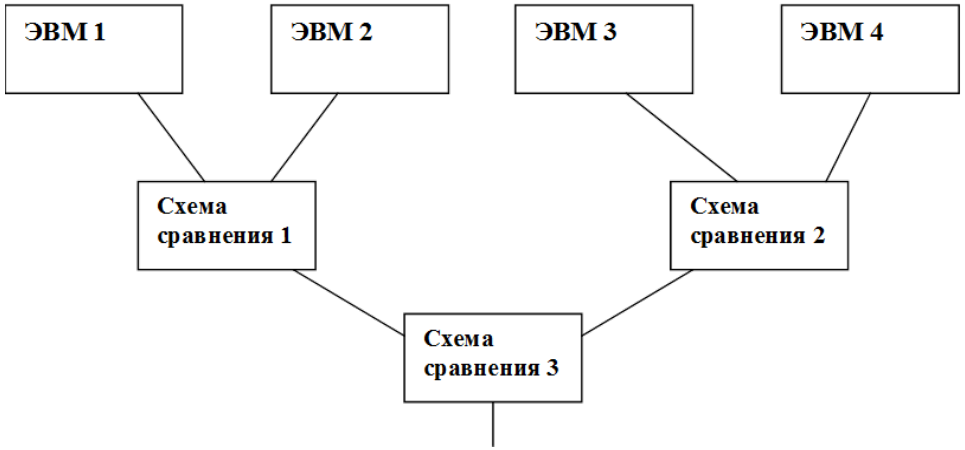
- А если сломалась схема сравнения? Можно предложить следующую схему:



- Но тогда нужна схема сравнения схем сравнения и т.д.
- На наш взгляд, это тупиковое решение.

21.4. Учетверенные ЭВМ

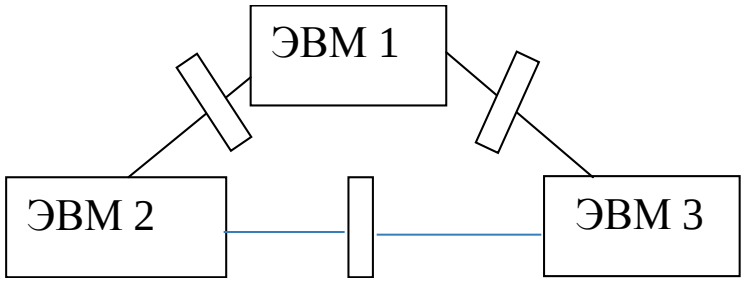
- По такой схеме работала управляющая ЭВМ американского космического корабля Шаттл:



- На наш взгляд, это не слишком удачная схема, во-первых, схема сравнения 3 является узким местом, выход из строя которого приводит к отказу системы в целом, во-вторых, если, например, выйдет из строя ЭВМ1, то схема сравнения 1 будет опираться только на одну ЭВМ2, что понижает уверенность в корректности этого результата.

21.5. Троированные ЭВМ

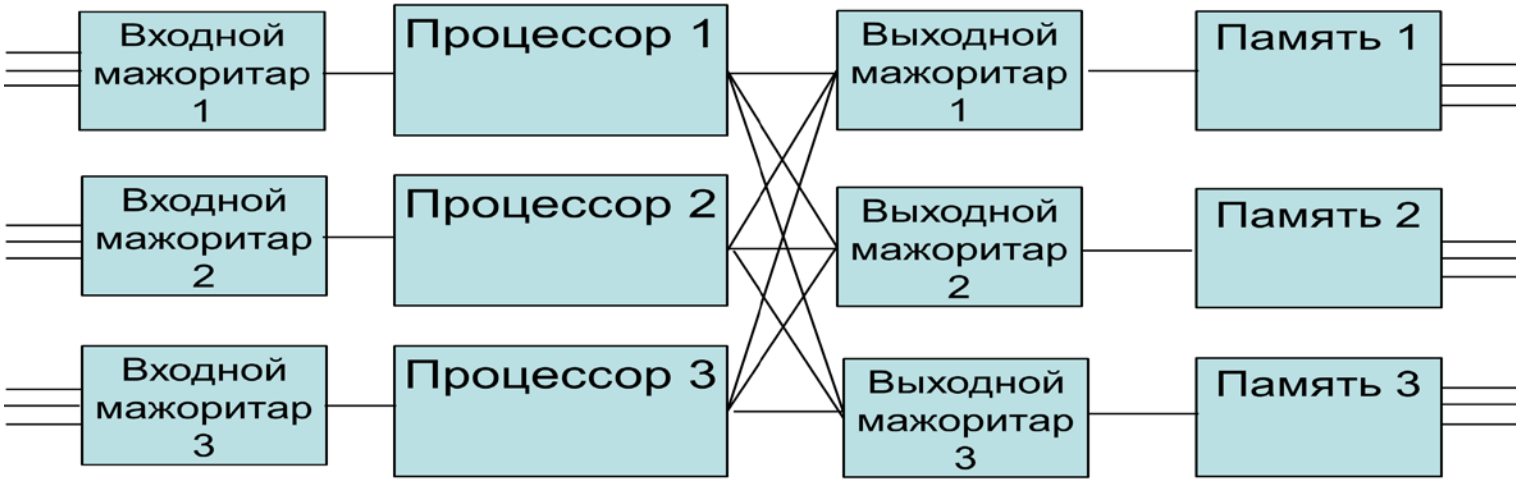
- Вышеприведенная критика дублированных и учетверенных ЭВМ привела нас к идеи создания троированных комплексов, в которых нет ни одного устройства, выход из строя которого приводит к отказу всей системы.
- Первоначально мы остановились на следующей схеме:



- Три обычных ЭВМ, каждые, скажем, 100 мс посылают соседям информацию о своих результатах. Три схемы сравнения могут определить, какая из трех ЭВМ врет, простым голосованием. Но эта схема оказалась не пригодной к использованию, поскольку сломавшаяся ЭВМ, могла испортить свою память и даже выдать какие-то команды во вне до момента срабатывания схемы сравнения.

21.6. Троированный УВК Самсон

- В конце концов, мы остановились на следующей схеме:



- В этой схеме главную роль играют мажоритары – схемы, которые получают три входных сигнала, сравнивают их, если один из этих сигналов отличается от двух других, то на выход подается правильный сигнал (выход двух устройств одновременно практически не возможен). Выходные мажоритары проверяют корректность работы процессоров, а входные мажоритары, которые получают данные от всех трех памятей, проверяют работы этих памятей.

22.1. Архитектура IBM/360 как пример классической архитектуры

- Серия ЭВМ IBM/360 была объявлена в 1964 г. Новшества:
- Это не одна машина, а целая группа ЭВМ, отличающиеся производительностью и ценой, но с абсолютно одинаковой архитектурой и системой команд
- Операционная система является неотъемлемой частью ЭВМ
- каналы ввода/вывода работают независимо от центрального процессора

Собственно, ОС и каналы применялись в машинах IBM и раньше, но впервые они стали применяться в серии ЭВМ.

- В IBM/360 есть 16 32-х разрядных целых регистра (0-15) и 4 64-х разрядных регистров для чисел с плавающей запятой.

22.2. PSW в архитектуре IBM/360

- Есть специальный регистр PSW (Program Status Word), в котором есть адрес следующей, исполняемой команды (24 бита), и 2 разряда кодов условия CC (Condition Code), устанавливаемые по результатам исполнения последней команды:

00 – нуль или равно, 01 – отрицательный, 10 – положительный, 11 – переполнение.

Соответственно в команде BC (Branch on Condition, переход по условию) можно установить 4-х битовую маску:

1000 – равно, 0100 – меньше, 0010 – больше, 0001 - переполнение. Команда может собрать переход по любой комбинации условий, например, маска 1100 соответствует переходу по меньше или равно.

22.3. Команды в архитектуре IBM/360

- Первым байтом команды всегда является код операции
- Регистровые команды занимают 2 байта (код операции и два 4-х битовых номера регистра), например, SR 1,2 означает вычесть содержимое 2-го регистра из 1-го.
- Команды формата память-память занимают 6 байтов, например, MVC 8 *a, b*, означает переслать 8 байтов из адреса *b* в адрес *a*.
- Самые частые команды – команды формата регистр-память, например,

L 1, *a*

означает загрузить слово из 4-х байтов, размещенное по адресу *a*, в 1-й регистр, а команда

A 1, *a*

добавить к 1-му регистру содержимое слово по адресу *a*.

- Еще один пример формата регистр-память: MVI 255, *a.* Эта команда имеет однобайтовый непосредственный операнд (255), который засылается в байт по адресу *a.*

22.4. Адресация в архитектуре IBM/360

- Адрес в команде (то, что на предыдущем слайде мы обозначали буквами *a* и *b*) занимает 2 байта:

Базовый регистр (4 бита) и
Смещение (12 битов)

- Дело в том, что предполагается, что память (максимальный размер 16Мб) разбита на страницы размером 4К, возможно, с большими пересечениями, чтобы добраться до нужного адреса надо загрузить адрес страницы, в которой находится этот адрес в один из целых регистров (кроме нулевого) и задать нужное смещение.

22.5. Формат команды регистр-память

- Код операции (8 бит)
- Номер регистра первого операнда, он же результат (4 бита)
- Номер индексного регистра (4 бита)
- Номер базового регистра (4 бита)
- Смещение (12 бит)

К моменту создания УВК Самсон мы имели многолетний опыт работы на оригинальной IBM/360, а потом на ее копии ЕС ЭВМ. Нас страшно раздражало, что практически в любой команде до половины битов – это нули. Индексный регистр используется редко, 15 базовых регистров никому не нужны, в УВК Самсон их всего 2, а 4095 байтов в смещении – это слишком много для процедур (обычно в процедуре 5-10 локальных переменных) и слишком мало для глобальных данных.

23.1. Архитектура ARM как наиболее массовая в настоящее время

- Архитектура ARM - Advanced RISC Machine
- Разработчик: компания ARM Holdings, лицензирует дизайн процессора производителям оборудования, своего производства не имеет, продает только IP ядра.
- Много однотипных взаимозаменяемых регистров.
- Глубокий конвейер из простых однотактовых операций.
- Режимы процессора ARM

ARM (32 бита)
Thumb-1 (16 бит)
Thumb-2 (смешанный 16/32 бита)
Jazelle (8 бит) – аппаратная реализация Java-байткода.

- Регистры r0 - r15

PC (r15) – Program Counter
LR (r14) – Link Register
SP (r13) – Stack Pointer

- Вещественные регистры (FPU и векторного сопроцессора)

23.2. Флаги условий в архитектуре ARM

- CPSR (Current Program Status Register): флаги условий, обработка прерываний

V oVerflow Переполнение (знаковое)
C Carry Перенос бита (беззнаковый)
Z Zero Нулевой результат (равенство)
N Negative Отрицательное значение (бит 31 установлен)

- Практически все команды ориентированы на спекулятивное исполнение, условие приписывается после команды,

и она будет выполнена, только если условие истинно:
add**ge** r1, r1, r1

- Установкой флагов можно управлять, если команда ALU имеет суффикс ’S’, то флаги будут

установлены в соответствии с результатом команды:
sub**s** r1, r2, #1

23.3. Модификация операндов в архитектуре ARM

- Второй аргумент ALU-команд может быть представлен в виде

ARG2 = R shift_op B, где
R – регистр
B – величина сдвига (0-31)
shift_op – один из LSL, ASL, RSL, ROR или RRX
add r1, r1, r1 lsl #2 // r1 = r1 + r1 *4 = r1 * 5
Величина сдвига может быть представлена не только константой, но и младшим байтом какого-то регистра, т.е. вычисляться

- Второй аргумент ALU-команд может быть также константой, которая кодируется 12 битами (8 бит значение, 4 бита величина сдвига):

CONST_32 = CONST_8 << (2 * N), 0 <= N < 16

23.4. Команды работы с памятью

- Команды загрузки/сохранения LDR / STR:

ldr r1, [r2, #+/-imm12]
ldr r1, [r2, +/-r3, shift imm5]
ldr r1, [pc, #256]
ldr r1, [sp, r2, asl #2]

23.5. Вызов функций

- Вызов функций выполняется при помощи команды

bl – branch with link
Текущее значение регистра pc сохраняется в регистре lr
Возврат из функции выполняется с помощью команды bx lr

some_func: bl some_func
push {lr}
… some_func:
pop {pc}
bx lr