

1 Часть I

1) Понятие ВС. Задачи, решаемые ВС. Классифицирующее ВС по типам решаемых задач. Общая модель ВС, состав и назначение отдельных частей.

Вычислительная система (ВС) - это совокупность аппаратных и программных средств, направленное на решение класса задач.

- Аппаратная часть обеспечивает возможность и производительность решения задач
- Программная часть обеспечивает универсальность и расширение класса задач на другие классы.

Задачи решаемые ВС:

1. Руководствующие задания

- погода.
- биология
- гидроэнергетика
- синтез рем
- распознавание образов...

2. Универсальные задания

- a. Научно-техническое расчёты (использование, много операций)
- b. Статистическая обработка (много данных, мало операций)

3. Специализированные задания

- управление объектами.
- сопряжение и согласование узлов ВС
- обработка однотипных данных.

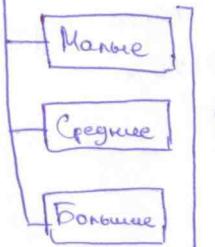
Классифицирующее ВС по типам решаемых задач:

1. Кабор поддерживаемых систем ввода/вывода
2. Кабор поддерживаемых программ.
3. Поддерживающее данные
4. Обслуживание.

Классы ВС

Универсальные ЭВМ

- много сменных ввода/вывода
- неограниченный набор опер.
- нет ограничений на форматы данных
- неограниченное объёме памяти



Программно-ориент. ЭВМ

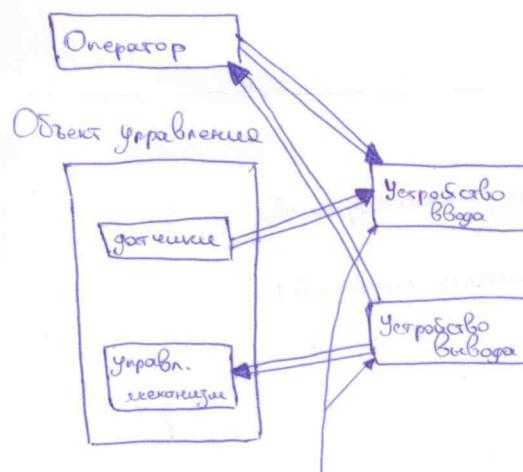
- Ввод/вывод носителей заданы
- набор команд ограничен
- поддержка структур данных
- неограниченное объёме памяти



Специализированные ЭВМ

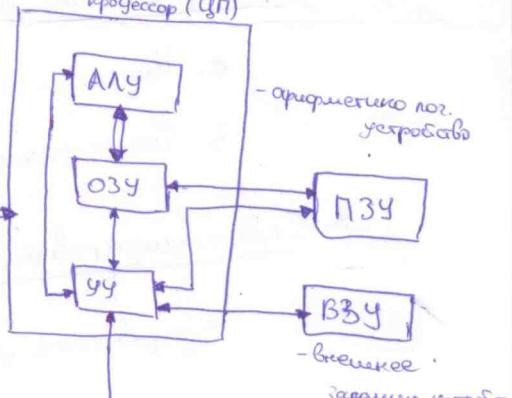
- Ввод/вывод жестко заданы
- маленький, но эффективный набор команд
- однотипные данные
- объём памяти ограничен
- архитектура процессора под задачу
- Супер ЭВМ
- Специализированные ЭВМ

Обобщенная схема ВС



ширина

Централизованный процессор (ЦП)



- арифметико-лог. устройство

- вспомогательное запоминающее устройство

↔ данные

→ управл. сигналы

- 2) Задачи обработки данных. Классифицируем ВС по элементной базе. Вариантно построим систему обработки информации.
4. централизованное засорение обработки инфр.:
- Сбор данных
 - опрос источника
 - преобразование аналогового сигнала в цифровой
 - Первичная обработка
 - масштабирование
 - вынесение правил
 - вычисление зависимостей
 - сравнение значений с пределами допустимыми.
 - Вторичная обработка
 - задачи первичной обработки, но более точные алгоритмы
 - Представление данных измерений или управлений
 - подготовка данных для отправки
 - согласование данных, протоколов, скорости приёма и т.д.

Классификация ВС по элементной базе.

1. Аналоговое - система непрерывного действия

Данное представление непрерывности изображается ЭЛ. Тока и напряжения

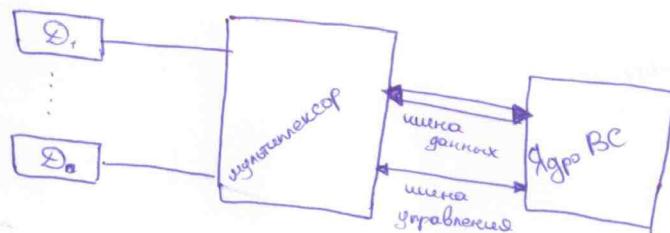
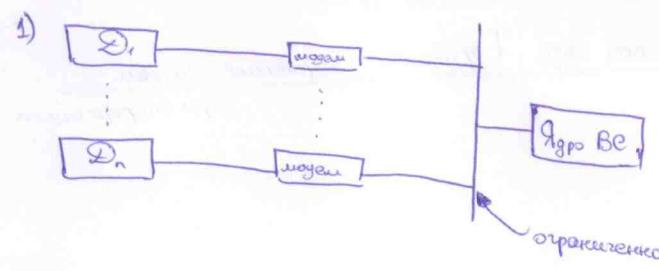
Процессы в них описаны диф. уравнениями, для решения которых эти системы требуются

Решают конкретную задачу.

2. Цифровое - ВС дискретного действия

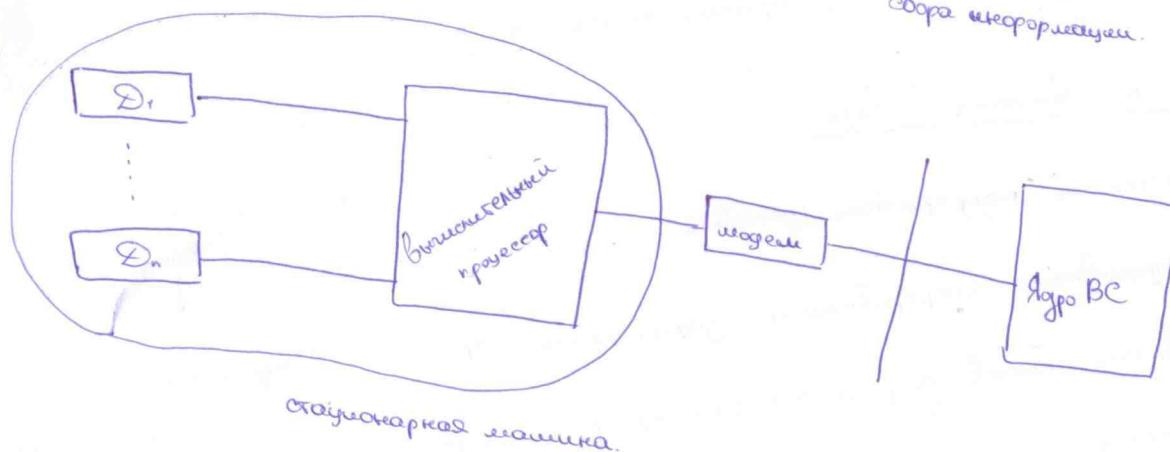
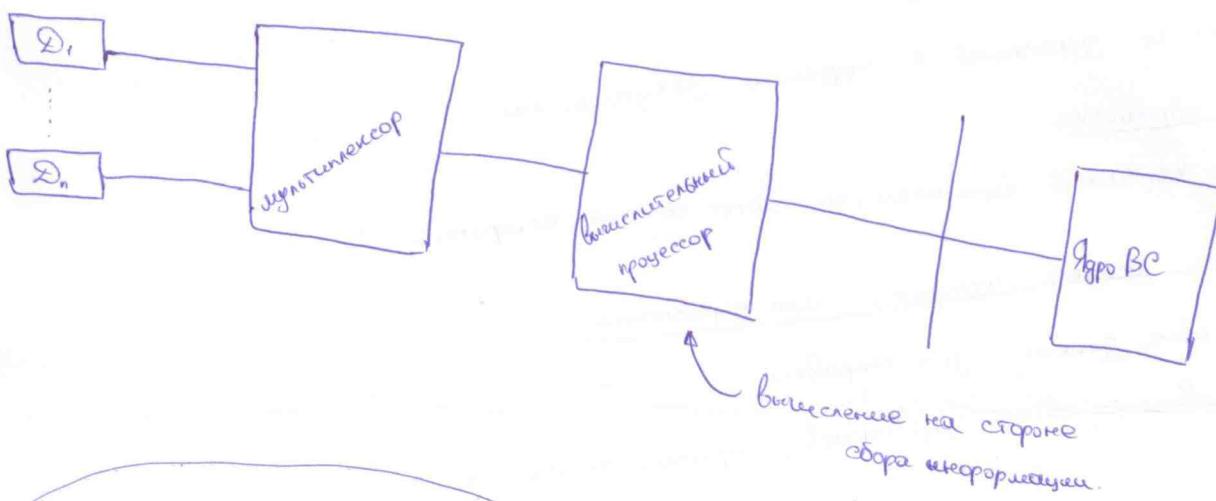
3. Универсальное - ВС смешанного типа.

Варианты построения систем сбора информации:



многотелескоп - узел (устройство) с несколькими входами и один выходом

многопортовый - один вход и несколько выходов



3) Производительность ВС. Способы оценки производительности. Организации, производящие оценку производительности ВС. Связь производительности и стоимости ВС.

Нет единого способа сравнения двух производительных систем между собой. Есть набор методик, единица измерения производительности — время.
В многоядерных системах время складывается из времени исполнения программы и времени выполнения расхолдов системе.

Соответственное время процессора — время ядро-представителя
Тактовая частота — синхронизующий ядро компонент
Пиковая производительность = пиковая производ. одного ядра * кол-во ядер в системе

исполнение набора команд ЦПУ.

Предполагается, что все подсистемы работают в максимальном режиме

Реальная производительность зависит от временного бюджета с особенностями архитектуры ВС
для оценки пиковой производительности используется: MIPS, MFLOPS

Операция — ядро-команду

команда — сложив, инвертировать.

MIPS (million instructions per second)

$$\text{MIPS} = Q/t$$

где

$$t \propto Q, \text{ то } \text{MIPS} \propto$$

"+":

- интуитивно понятно

- "-":
- зависит от набора команд процессора
- зависит от набора команд В программы
- с ростом производ. может падать из-за:
 - а) оптимизирующего компилятора.
 - б) при использовании сопроцессора.

Значит MIPS нужно сравнивать

на специализированных программах.

1. Специализированные ядро-команды, в которых % содержат ядро-команду разных типов соединяется.

классами ядер, на которые ориентирована система.

2. Ядро — барьерные ядро-команды, на которые ориентирована ВС.

3. Бенчмарки — различные показатели производительности классов ВС.

4. Программы специализированной нагрузки — параллельные генераторы чисел.

Разработкой и стандартизацией

тестов занимается SPEC:

- INT92
- CINT92
- CFP92.

Генерацией бенчмарков занимается

компания AIM Technology

MFLOPS (million float operations per second) - оперирует вещественными операндами.

Операции выполняются всегда одновременно, а кадры каскадов могут откладываться.

MFLOPS вне каскадного горна ≈ 0 .

Для подсчёта MFLOPS используются:

линейные и некоэффициентные узлы и пакеты LINPACK (SLATE)

Корреляционные операции:

$+,-,*,<,>$ — 1

$/,\sqrt{ }$ — 4

$\sin(x), e^x$ — 8.

4) Требования к ВС по надежности и совместимости ПО. Принцип открытых систем.

Организация и структура ВС должна позволять добавлять новые узлы.

"+" :

- добавление нового узла

новые узлы. Беспредельное (максимально)

"-" :

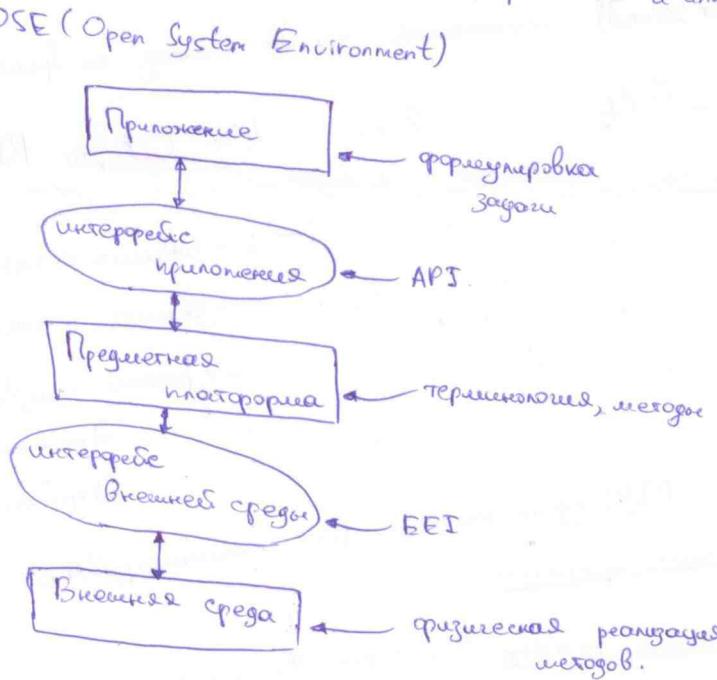
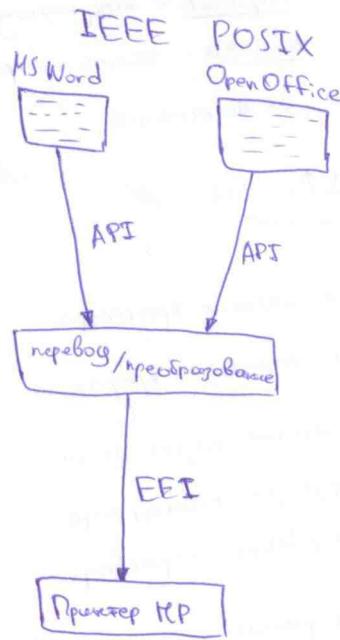
- упрощение топологии замедляет передачу

данных

- Упрощение ПО

[Обратимое изменение функций]

Ключевые открытые системы - являются стандартом или программистским и аппаратным составляющими.



ISP - наборы интегрированных стандартов, задающие стандарты для разных решений.

Profile - набор стандартов.

ISP → переход от общих проблем к группам.

5) Надёжность и отказоустойчивость ВС. Дерево концепций возможностей. Оценка надёжности ВС.

Надёжность — это вероятность того, что ВС справится с поставленной задачей.

Составляющие:

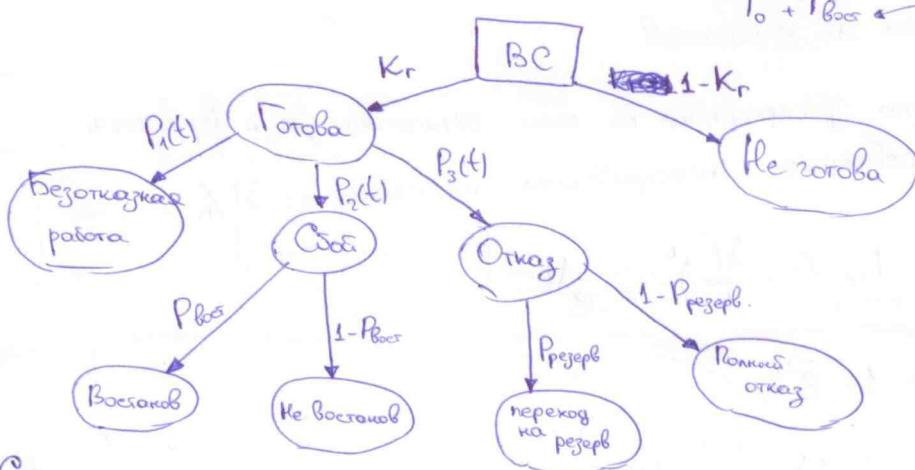
1. Вероятность того, что ВС находится в состоянии готовности к моменту начала запуска цикла управления.
2. Вероятность того, что ВС будет исправной в течение всего цикла управления.
3. Отказоустойчивость — способность ВС с допустимым временным, точностикованным и аппаратурным потерями на присущее отказоустойчивое уровне завершить весь цикл управления.

Измерение надёжности. Дерево концепций возможностей.

Коэффициент готовности ВС

$$K_r = \frac{T_0}{T_0 + T_{\text{вос}}}$$

T_0 ← время наработка на отказ
 $T_0 + T_{\text{вос}}$ ← время ремонта, когда ВС становится



Сбой — самоустраняющийся отказ

Сбой:

- асинхронные — можно восстановить колесницу на которой сбой произошёл
- асинхронные — цикл кешируется небрежно.

При наличии

аппаратного контроля

⇒ неизделие сбои и восстановление как аппаратное управление.

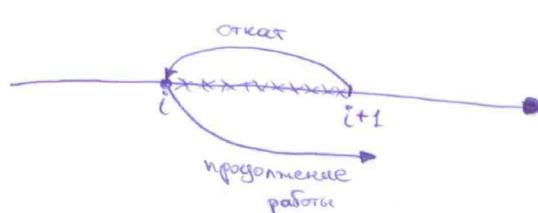
Методы защиты от сбоев:

Защита от сбоев (программная) — "двойной просчёт"

Программа делится на сегменты рекомпилированного обёма.

Перед входом в сегмент устанавливается контрольные точки —

сохранённые заранее предыдущими и текущими колесницами.



Зашита от отказов - резервирование

Для одного узла несколько копий

Рассмотрим систему с одницею узлом: $K_r = 0,9$.

$$\text{Двойной резерв: } K_r^{(2)} = 1 - (1 - 0,9)^2 = 0,99$$

Если кол-во резервов \uparrow , то надежность \uparrow и стоимость \uparrow .
Тройной резерв: $K_r^{(3)} = 1 - (1 - 0,9)^3 = 0,999$.

Вычисление вероятности исправной работы.

λ_c - частота сбоев.

$$\lambda_o - \text{частота отказов} \quad \lambda = \lambda_c + \lambda_o$$

За один цикл управления движущейся t сек кол-во неисправностей: λt

Рассмотрим цикл управления на n сегментов

Считаем, что сбоя равномерно распределены по этим сегментам, тогда вероятность неисправности на сегменте: $\lambda t/n$.

Значит $P_1(t) = \lim_{n \rightarrow \infty} \left(1 - \frac{\lambda t}{n}\right)^n = e^{-\lambda t}$.

$$P_2(t) + P_3(t) = 1 - e^{-\lambda t}$$

Значит

$$P_2(t) = \frac{\lambda_c}{\lambda} \left(1 - e^{-\lambda t}\right)$$

$$P_3(t) = \frac{\lambda_o}{\lambda} \left(1 - e^{-\lambda t}\right)$$

$$P(t) = K_r P_1(t) + K_r P_2(t) P_{\text{бок.}} + K_r P_3(t) P_{\text{резерв.}}$$

Надежность работы
системы в течение t секунд.

6) Архитектура ВС. Уровни архитектуры ВС. Иерархический метод описания ВС. Таблица уровней описаний.

Архитектура — логическая организующая ВС и распределение функциональности по обработке данных
Уровни ВС:

1. Пользовательский: обработка входных данных с точки зрения пользовательской системы.

- звуковой анализ (только интерфейс)

- серийный анализ (интерфейс и частичная структура)

- белый анализ (полное описание внутренней структуры)

2. Функциональный: совокупность алгоритмов и программной составляющей и взаимодействия

- программная модель ядер

- аппаратная модель

3. Логический: ВС как совокупность

интерфейсов

функционирующих между собой.

4. Аппаратный: состав

и описание

аппаратных частей

Использование иерархического способа описания структуре ВС!

Он включает в себя

следующие элементы:

1. система — совокупность эл-тов

для достижения конкретных целей

2. структура системы — фиксированная

совокупность эл-тов системы и связи между ними.

3. Элементы — небольшая часть

системы, структура которой не рассматривается, а имеет значение только функциональность

На верхнем уровне ~~из~~ иерархии ВС рассматривается как независимое целое обладающее избыточной

функциональностью и описание интерфейса

взаимодействия с внешней средой.

Уровень описания	Объект	Структурный базис	Язык описания
Пользовательский	Всесистемный базис и ОС	Команды и операторы	алгоритмические языки
Руководительский	ВС	составные части	макромодули и микромодули
Операторско-автоматический	АЛУ, УЧ	микропрограммные автоматы	язык описания микропрограмм
Логические элементы	комбинационные устройства	логические, запоминающие элементы	теория КА и булева алгебра
Электрические схемы	логические и запоминающие устройства	радио-электрические коммутационные	схема техника, теория эл. схем

7) Логические основы построения ВС. Базовое логическое устройство "Мелкая логика". Пример построения логического элемента, реализующего операцию XOR.

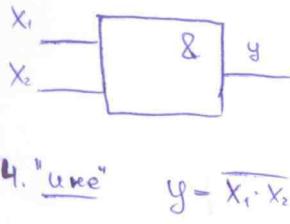
Научной основой построения цифровых устройств является булева алгебра.

Любое цифровое устройство собрано на базе лог. эл-тов

Логический элемент реализует общую булеву функцию $y = f(x_1, \dots, x_n)$

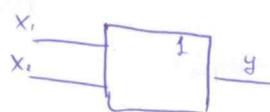
Базовое логическое эл-тво:

$$1. \text{ "и"} \quad y = x_1 \cdot x_2$$



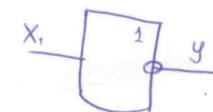
X ₁	X ₂	y
0	0	0
0	1	0
1	0	0
1	1	1

$$2. \text{ "или"} \quad y = x_1 + x_2$$



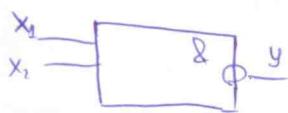
X ₁	X ₂	y
0	0	0
0	1	1
1	0	1
1	1	1

$$3. \text{ "не"} \quad y = \bar{x}_1$$



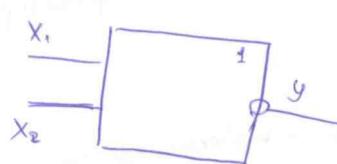
X ₁	y
0	1
1	0

$$4. \text{ "исключающее или"} \quad y = \bar{x}_1 \cdot x_2$$



$$y = x_1 / x_2 - \text{исключающее или!}$$

$$5. \text{ "исключающее или не"} \quad y = \bar{x}_1 + \bar{x}_2$$



$$y = x_1 \cdot \bar{x}_2 - \text{исключающее или не!}$$

Задача: построить АЭ реализующий операцию XOR

$$\begin{matrix} x_2 & x_1 & x_0 & y \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

$$CDK\Phi: y = \bar{x}_2 \bar{x}_1 x_0 + \bar{x}_2 x_1 \bar{x}_0 + x_2 \bar{x}_1 \bar{x}_0 + x_2 x_1 x_0$$

$$CK\Phi: y = (x_2 + x_1 + x_0) (\bar{x}_2 + \bar{x}_1 + \bar{x}_0) (\bar{x}_2 + x_1 + \bar{x}_0) (\bar{x}_2 + \bar{x}_1 + x_0)$$

Не всегда оптимально!

Минимизируют кол-во инверторов в базисах "и не", "или не".

6

8) Минимизируемые функции, реализующие АЭ. Критерий минимизации. Алгоритм минимизации с помощью карты Карто

Признак минимизируемых ДНФ(КНР): попарное склеивание с последовательными помехами

$$X_1 \bar{X}_2 X_3 \vee X_1 X_2 \bar{X}_3 = X_1 X_3 (\bar{X}_2 \vee X_2) = X_1 X_3$$

$$(X_1 + X_2 + \bar{X}_3)(\bar{X}_1 + \bar{X}_2 + \bar{X}_3) = X_1 + \bar{X}_3 + (X_2 \bar{X}_2) = X_1 + \bar{X}_3$$

Карта Карто для минимизируемых ДНФ(КНР)

Карта Карто - способы представления образов переписанной таблицы истинности, эллы которой упорядочены по коду Грэй

(каждые 2 эллы откладываются ровно на один разряд)

Алгоритм минимизации:

1. Выделить на карте смежные области, содержащие интересующие значения для ДНФ - клетки с 1
КНР - клетки с 0

2^n клеток. Поиск

2. Вспенеть термическое состоящее из максимального числа в максимальной области.

Пример: $X_2 \ X_1 \ X_0 \ Y$
 $\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{array}$

$X_2 \ X_1$	00	01	11	10
X_1	0	1 1	0 0	0 0
X_0	1 0	0 0	1 1	1 1

ДНР: $y = \bar{X}_2 \bar{X}_1 + X_2 X_1$

КНР: $y = (X_2 + \bar{X}_1) * (\bar{X}_2 + X_1)$

$$y = f(X_1, X_2, X_3)$$

$X_2 \ X_3$	00	01	11	10
X_1	0	1	1	1
X_0	1	0	0	0

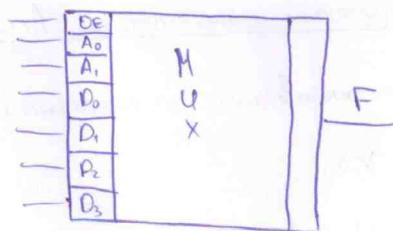
9) Типовое комбинационное АЭ. Мультиплексор, дешиффратор, шифратор/десифратор, цифровой компаратор.

Комбинационные АЭ - это такие АЭ, у которых выходные значения зависят от входовых воздействий подаваемых только в данный момент времени.

Последовательностные АЭ - это такие АЭ, у которых выходные значения зависят от входовых воздействий в данный момент времени и их предыдущих значений.

Мультиплексор (MUX) - комбинационный ЛЭ, предназначенный для передачи данных из нескольких каналов (источников) в один канал.

DE	A ₀	A ₁	F
1	X	X	0
0	0	0	D ₀
0	0	1	D ₁
0	1	0	D ₂
0	1	1	D ₃

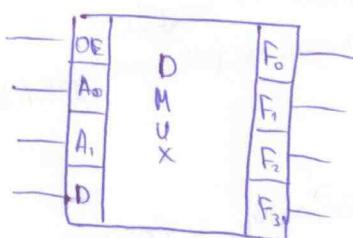


СДНФ для MUX:

$$F = \overline{D} E D_0 \bar{A}_0 \bar{A}_1 \vee \overline{D} E D_1 \bar{A}_0 \bar{A}_1 \vee \\ \vee \overline{D} E D_2 A_0 \bar{A}_1 \vee \overline{D} E D_3 A_0 A_1$$

Демультиплексор (DMUX) - комбинационный ЛЭ, предназначенный для передачи данных от одного источника в несколько каналов.

DE	A ₀	A ₁	F ₀	F ₁	F ₂	F ₃
1	X	X	0	0	0	0
0	0	0	D	0	0	0
0	0	1	0	D	0	0
0	1	0	0	0	D	0
0	1	1	0	0	0	D



$$F_0 = \overline{D} E D \bar{A}_0 \bar{A}_1$$

$$F_1 = \overline{D} E D \bar{A}_0 A_1$$

$$F_2 = \overline{D} E D A_0 \bar{A}_1$$

$$F_3 = \overline{D} E D A_0 A_1$$

Преобразователь кодов (РК)

- это комбинационное ЛЭ, предназначенное для преобразования чисел

Шифратор - расчетный схема РК, переводит коды входа в адресное слово закодированное 2-м битами.

DE	F ₀	F ₁	F ₂	F ₃	A ₀	A ₁
1	0	0	0	0	X	X
0	1	0	0	0	0	0
0	0	1	0	0	0	1
0	0	0	1	0	1	0
0	0	0	0	1	1	1

Демодулятор - переводит адресное слово в коды входа.

Таблица истинности модулятора:

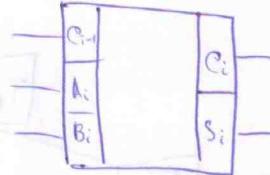
Царрорный компаратор - комбинационный ЛЭ, предназнач. для сравнения чисел представляемых в двоичной записи

A	B	=	>	<
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

10) Сумматоры. Понятий одноразрядный сумматор. Четверть-сумматор, полусложитель. N -разрядный сумматор с последовательным, параллельным и условным переносом.

Одноразрядный сумматор - наименее сложный ЛЭ, выполняющий опер. арифметического сложения над битовыми представлениями зеис.

A	B	C _{i-1}	S _i	C _i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

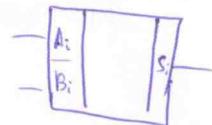


Четверть-сумматор (не заботится о переносе) -

XOR или сложение по mod 2

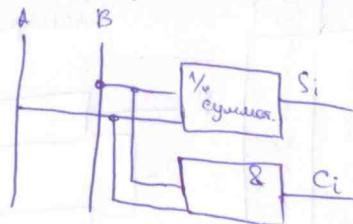
$$y = \bar{a}b + a\bar{b}$$

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

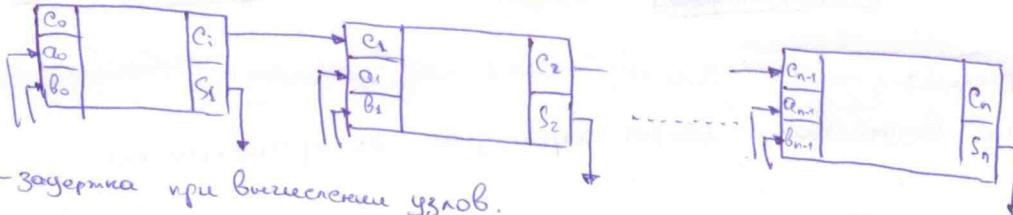


Полусумматор - не учитывает предыдущий перенос, а сам вычисляет

A	B	S _i	C _i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Последовательные сумматоры для n -разрядного числа



Проблема - задержка при вычислении узлов.

Параллельный сумматор - сумма и перенос вычисляется независимо друг от друга.

- Рукавное генерации - возбр. 1, если перенос не зависит от предыдущего
- Рукавное прореживание - возбр 1, если перенос возникает при наличии пред. переноса

$$g_i = \alpha_i \wedge \beta_i$$

$$h_i = \alpha_i \vee \beta_i$$

$$C_0 = g_0 \vee C_{Bx} h_0$$

$$C_1 = g_1 \vee C_0 h_1 = g_1 \vee g_0 h_1 \vee C_{Bx} h_1 h_0, \dots$$

Сумматор с условным переносом - оптимизируется!

Дано: n -разр сумматор

Делим на 2 группы по $\frac{n}{2}$ разрядов

Группа старших разрядов дублируется

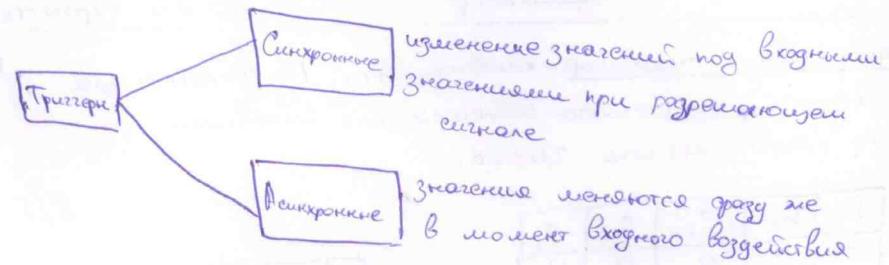
Сумма вычисляется параллельно без учёта переноса

По знаку переноса в инд. группе выбирается старшая группа.

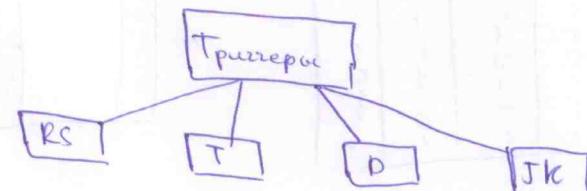
1) Последовательностные АЭ. Триггеры, RS-триггер, T-триггер, D-триггер, JK-триггер.

Триггер - последовательностное АЭ, способное сохранять устойчивое значение выходного сигнала и одновременно менять их под воздействием входов.

по способу управления:



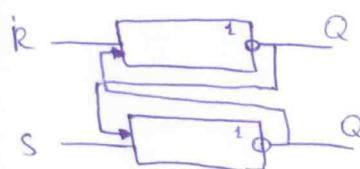
по внутренним логическим связям:



RS: set/reset

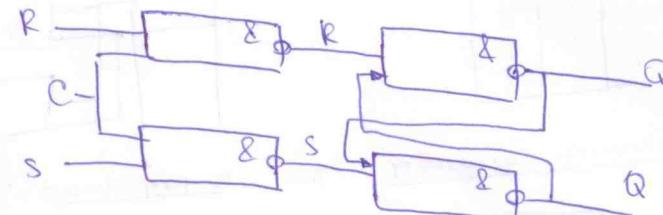
$$y = S \vee \bar{R} Q_{n-1}$$

Асинхронное:



S	R	Q _{n-1}	Q _n
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	X	Keep

Синхронное: Разрешающий сигнал C

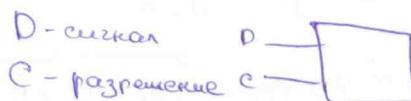


T: переключатель (асинхронный)

под воздействием входного воздействия меняет сохраненное значение на противоположное.

D-задерживающий (синхронный)

На входе:

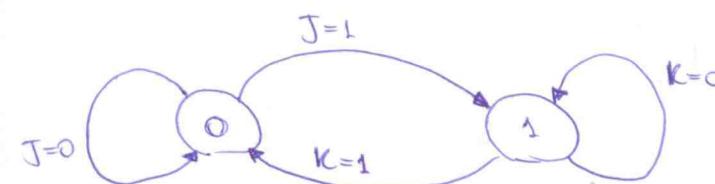


Если C=1, то D- текущий состоян.

Если C=0, то D- предыдущее значение (запоминающее)

JK: jump/kill

Допускает $j=k=1$ и работает в этом режиме как T.

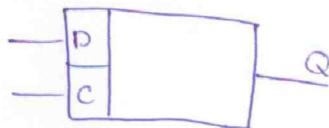


J≈R, K≈S

J	K	Q _{n-1}	Q _n
0	X	0	0
1	X	0	1
X	1	1	0
X	0	1	1

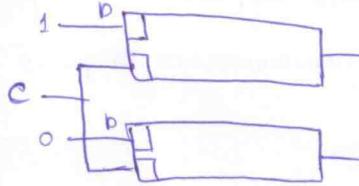
12) Применение организованных регистров. Параллельный и последовательный регистры.
Схемы сдвигов. Статическая память.

Регистр — это последовательностный АЭ предназначененный для хранения двоичной информации.

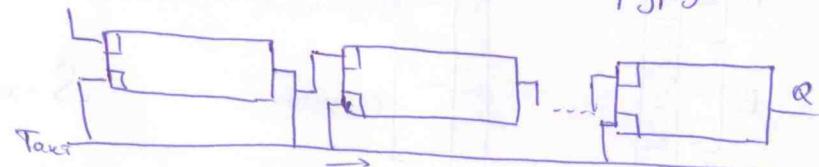


D-Delay триггер.

Параллельный регистр: загружается значение сразу

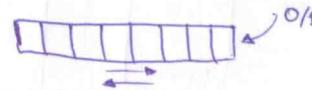


Последовательные регистры: загружается по разрядам

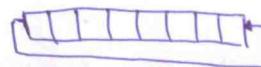


Схемы сдвигов:

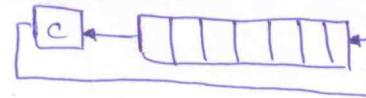
1. параллельный



2. циклический



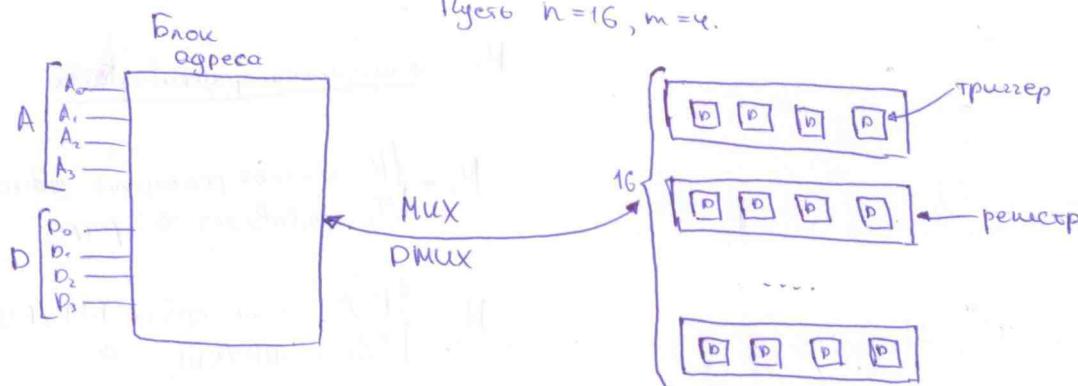
3. циклический сдвиг через флаг переноса



Статическая память — массив регистров общего назначения (РОН)

Мы собираем n регистров по m разрядам

Рассмотрим $n=16$, $m=4$.



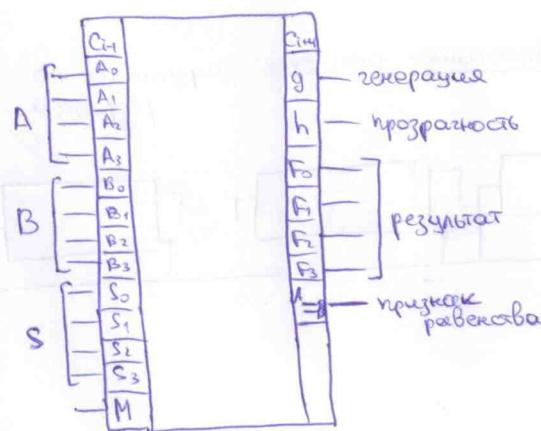
13) Типовая схема элементарного вычислительного устройства. Схема АЛУ.



Возможные операции:

1. Загрузить значение из регистра в другой регистр (2 результата)
2. Изменить на 1 значение какого-то РОН.
3. Субтракция значений РОН.

Схема 4х разрядного АЛУ



M - селектор типа операции.

$M=1 \Rightarrow$ логическая операция

$M=0 \Rightarrow$ арифметико-логическая операция.

S - кодирует код операции

можно составлять таблицу

S	$M=0$	$M=1$
0000	7A	$A+B$
0001	;	;
1111	;	;

14) Структурная схема микропроцессора (МП). Мультиплексоры用于 переноса,用于 сдвига, анализа условий и реализующие или командов.

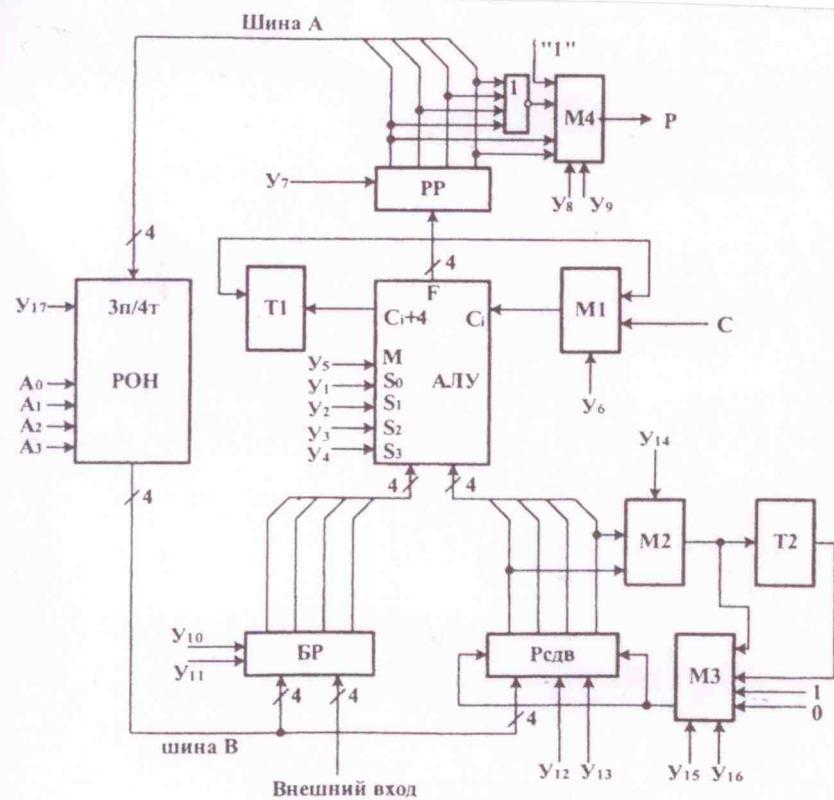


Рис. 3.3. Структурная схема МП

M_1 - мультиплексор用于 переноса

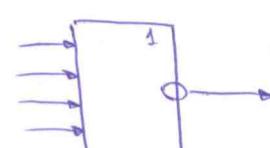
$M_1 = \begin{cases} T_1 \Rightarrow сложение с переносом \\ C == 0 \Rightarrow add \end{cases}$

M_2 - мультиплексор用于 сдвига

$M_2 = \begin{cases} M_5 - обычное реализующее сдвиг \\ T_2 - добавляет еще разряд \end{cases}$

$M_3 = \begin{cases} M_2/T_2 - управ. сдвиг ROR/ROL \\ 1/0 - SHR/SHL. \end{cases}$

M_4 - мультиплексор проверки логических условий.



проверка на 0.

15) Порядок к организации управления выполнением операций. Схема "жёсткой" и программируемой логики. Применение и недостатки обоих подходов.

УУ - генерирует последовательность команд

Основные функции:

1) Управление выполнением операции

2) Считывание команд программы в правильной последовательности, а так же демодуляция и обработка этих команд.

Входные управляемые сигналы обрабатываются в один управл. блок

На вход подаётся код запрашиваемой опер.

На выходе последов. управляемых сигналов.

1. "Жёсткая" логика



- "+":

 - скорость работы
 - возможность оптимизации

- "-":

 - сложность

2. "Программируемая" логика



- "+":

 - легко перепрограммировать

- "-":

 - медленнее.

16) Схема выработки последовательности управляемых сигналов операционных устройств.

Для выполнения любой операции МП должен содержать след. этапы:

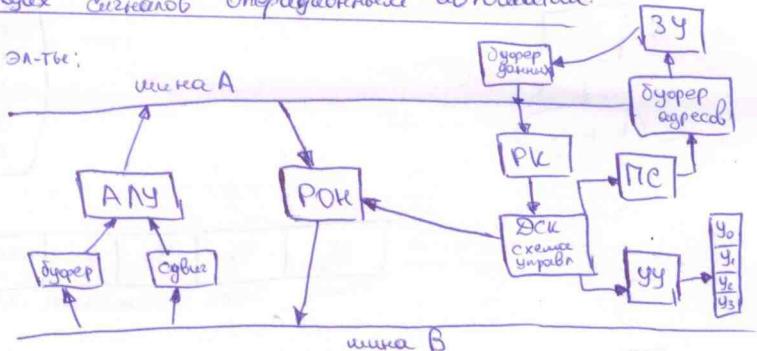
1. Программный счётчик (ПС)

2. Регистр команд (РК)

3. Схема выдачи адресов операндов или между адресами

4. Схема считывания данных с шинами данных

Управляемые устройства с программной логикой содержит все необходимые описания микрокоманд для реализации алгоритма этой операции.



17) Микропрограммы. Роль микрокоманд. Способы кодирования полей микропрограмм. Этапы проектиров. управл. алгоритмов

Микрокоманда состоит из:

1. набора управляемых сигналов: y_1, \dots, y_k
2. адреса следующей команды

Реализация условных переходов к следующим коммандам осуществляется через Вектор признаков(условий)

Подходы формирования управляемого слова:

$$y = \{y_1, \dots, y_k\}$$

1) горизонтальный

разряд слова $n=14$

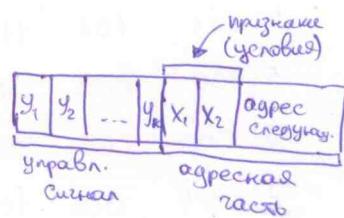
каждому разряду \mapsto 1 управляемый сигнал.

"-": не экономичное использование памяти!

2) вертикальный

кодируется номер управляемого сигнала

"-": за один раз только одна операция!



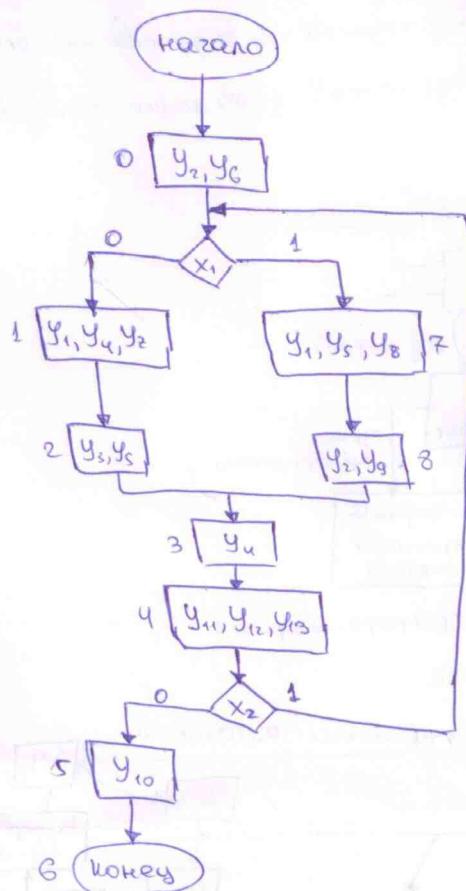
3) смешанный

баланс между кол-вом и длиной разрядов.

У разделяется на группы, которые не могут одновременно вырабатываться

Выделяется $\log_2 n$ разрядов, где n - кол-во эл-тов в группе.

Проектирование управляющего автомата



$$Y = \{Y_1, \dots, Y_{13}\} \quad X = \{X_1, X_2\}$$

горизонтально: 13 разр.

вертикально: 4 разр

Разделение на 3 группы:

$$Y_I = \{Y_1, Y_2, Y_3, Y_{10}, Y_{11}\} \Rightarrow 3 \text{ разр}$$

$$Y_{II} = \{Y_4, Y_5, Y_6, Y_9, Y_{12}\} \Rightarrow 3 \text{ разр}$$

$$Y_{III} = \{Y_7, Y_8, Y_{13}\} \Rightarrow 2 \text{ разр}$$

ног	Y_I	Y_{II}	Y_{III}
000	ε	ε	ε
001	Y_1	Y_4	Y_7
010	Y_2	Y_5	Y_8
011	Y_3	Y_6	Y_{13}
100	Y_{10}	Y_9	—
101	Y_{11}	Y_{12}	—
110	—	—	—
111	—	—	—

ног	X
00	tong 0
01	X_1
10	X_2
11	tong 1

	Y_I	Y_{II}	Y_{III}	X_i	след. агрес
3: 000	3 бита	3 бита	2 бита	2 бита	4 бита
4: 001	00 : 01	00 : 01	00 : 01	0111 (?)	
5: 010	001	01 : 00	0010 (2)		
6: 011	010	00 : 00	0011 (3)		
7: 100	000	00 : 00	xxxx (6)		
8: 101	101	11 : 00	xxxx (9)		
9: 110	100	10 : 00	xxxx		
10: 111	000	00 : 11	xxxx (3)		
11: 000	000	00 : 01	xxxx (7)		
12: 001	010	10 : 00	xxxx (8)		
13: 010	100	00 : 11	xxxx		
14: 011	000	00 : 01	xxxx (7)		
15: 100	000	00 : 10	xxxx (1)		

- 0: 010 011 00 : 01 0111 (?)
 1: 001 001 01 : 00 0010 (2)
 2: 011 010 00 : 00 0011 (3)
 3: 000 001 00 : 00 xxxx (4)
 4: 101 101 11 : 00 xxxx (9)
 5: 100 000 00 : 00 xxxx (6)
 6: 000 000 00 : 00 xxxx
 7: 001 010 10 : 00 xxxx (8)
 8: 010 100 00 : 11 xxxx (3)
 9: 000 000 00 : 01 xxxx (7)
 10: 000 000 00 : 10 xxxx (1)

Этапы появления языков:

1. Машинноориентированное программирование.

- появление первых ЭВМ (программирование в машинных кодах)
- язык ассемблер (автокод)
- необходимо знание архитектуры и ОС

2. Процедурное программирование

- алгоритмизующий процесс решения задачи
- выбор оптимального алгоритма
- начало с конца 50-ых годов. 1957 - Fortran, 1960 - Algol 60.
- решение инженерных задач математиками
- системные программы - ASM.
- алгоритмические языки машинно-независимые

3. Структурное программирование

- Н. Вирт алгоритмы + структуры данных = программы
- языки общего назначения, 1971 - Pascal, 1972 - C.
- системные программы более доступны
- компиляторы C, Fortran \Rightarrow листинг кода на ASM.

4. Модульное программирование

- "спрятать" алгоритмы и данные внутри программной единицы - модули
- 1975 - 1979 Modula
- системные программы Turbo Pascal, Borland C.

5. ООП

- обобщение данных и алгоритмов и их обработка по смысловому признаку.
- 1960 - Simula
- Object C, Pascal, Java, Small Talk

6. Объектное технологическое разработке ПО. (CASE - технологии)

- templates (шаблоны)
- разработка и сборка ПО на основе готовых компонентов и прототипов структур данных
- COM, STL, ATL.
- использование IDE.

2) Общие представления организаций данных. Рассмотрим систему счисления. Правила перевода целых и вещественных чисел из десятичной системы счисления в любую другую систему.

Пусть дано число, состоящее из n целых цифр (q) и k дробных (D).

Продвижение цифр от кула, начиная с десятичной точки влево и вправо, тогда

номерная система счисления с основанием q - это та сама система, в которой каждое число состоит из цифр $D_i, Y_i \in \{0, \dots, q-1\}$, $i \in \{-k, n\}$ - позиций, и

каждое i -значное

Число = $D_n Y_{-k}$

Число:		Y_{-k}	\dots	Y_1	Y_0	\cdot	D_{-1}	\dots	D_{-k}
$i:$	n	\dots	1	0	\cdot	-1	\dots	-1	$-k$

$$\text{Число} = \sum_{i=-k}^n q^i \cdot \text{цифра}_i$$

десятичное

Десятичная система счисления (Decimal)

$$q=10, y \in \{0, \dots, 9\}$$

Представление целого числа:

$$5321 = 5 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10 + 1 \cdot 10^0$$

Представление дробного числа:

$$0,125 = 0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}$$

11 3) Двоичная, восьмеричная и шестнадцатеричная системы счисления. Правила перевода целых и вещественных чисел в десятичную систему. Способ определения четности на языке ASM.

Двоичные системы счисления (Binary)

$q=2$, $y \in \{0,1\}$ Бит - Binary digit.

Число: $117_d \rightarrow x_b$

$$\begin{array}{r} 117 \\ \hline 116 \quad | 2 \\ \underline{-1} \quad \quad 58 \\ \quad \quad 0 \quad | 2 \\ \quad \quad 28 \quad | 2 \\ \quad \quad \underline{-14} \quad \quad 2 \\ \quad \quad \quad 0 \quad | 2 \\ \quad \quad \quad 14 \quad | 2 \\ \quad \quad \quad \underline{-14} \quad \quad 2 \\ \quad \quad \quad \quad 0 \quad | 2 \\ \quad \quad \quad \quad 7 \quad | 2 \\ \quad \quad \quad \quad \underline{-6} \quad \quad 2 \\ \quad \quad \quad \quad \quad 1 \quad | 2 \\ \quad \quad \quad \quad \quad \underline{-2} \quad \quad 1 \\ \quad \quad \quad \quad \quad \quad 1 \end{array}$$

$$117_d = 1110101_b$$

дробное: $0,5_d \rightarrow x_b$

$$\begin{array}{r} \times 0,5 \\ \hline 1,0 \end{array} \Rightarrow 0,5_d = 0,1_b$$

$0,05_d \rightarrow x_b$

$$\begin{array}{r} \times 0,05 \\ \hline 0,1 \\ 0,2 \quad 1,20 \\ 0,4 \quad 1,40 \\ 0,8 \quad 1,80 \\ 1,6 \quad 1,60 \end{array}$$

$$x_b = 0,000011(0011)_b$$

$1110101.01_b \rightarrow x_d$

$$x_d = 2^6 + 2^5 + 2^4 + 2^2 + 2^0 = 117,25$$

Способ определения четности:

I. div в ASM (делим на 2)

```
mov ax, 10  
mov bx, 2  
xor dx, dx  
div bx  
cmp dx, 0  
jz Even.  
Even: ax : dx  
Odd: al : ah
```

II test (логическое уравнение)

```
test ax, 1  
jz Even  
jne Odd
```

III SHR (сдвиг вправо)

```
mov ax, 10  
shr ax, 1  
jc Odd  
jnc Even.
```

SHL (сдвиг влево)

shl ax, 1 - умножает на
следующую единицу.

Шестнадцатеричные системы счисления (Hexadecimal)

$q=16$ $y \in \{0, \dots, 9, A, \dots, F\}$ dec \rightarrow bin \rightarrow hex

одна 16-тичная цифра соответствует четырем двоичным цифрам.

$117_d \rightarrow x_h$

$$x_b = 1110101 \Rightarrow x_h = \frac{01110101}{75} = 75h.$$

Восьмеричные системы счисления (Octal)

$q=8$ $y \in \{0, \dots, 7\}$ 3 группы 8 bin.

$$117,25_d = \underline{\underline{001}}.\underline{\underline{110101}}.\underline{\underline{010}} = 165,2 \text{ oct.}$$

4) Расположение числа в регистре. Алгоритм перевода num. кодов символов в число.
Алгоритм перевода числа в num. кодов символов.

Байт - двоичный разряд

Байт - 4 байта идущих двоичных разряда.

Байт - 8 байтов идущих двоичных цифр: FFh = 225d.

Слово - это 2 байта FFFF = 65535.

Двоичное слово - 4 байта FFFFFFFFh = 4,294,967,295.

Вывод числа в ASM

Для вывода: mov AX, 1
int 21h \Rightarrow в al код символа \Rightarrow al - '0' = цифра.

Input:

```
push bx  
push dx  
xor bx,bx  
mov cx,10
```

inpt:
mov ah,1
int 21h

cmp al,13
jz exit

xchg ax,bx

mul cx

xchg ax,bx

xor ah,ah

sub al,30h

add bx,ax

jmp inpt

exit

mov ax,bx

pop dx

pop bx

ret.

Output:

AX = 10
StrBuf db 8 dup(' ')
db '\$'

stosb ^{AX} _{AL} β es:[di].

printdec:

pusha
mov di, offset StrBuf 5
std.

std \leftarrow
cl d \rightarrow

cycle:

xor dx,dx
mov cx,10
div cx
add dx,'0'
xchg ax,dx
stosb
xchg ax,dx
test ax,ax
jnz cycle

mov dx, offset StrBuf
mov ah,9
int 21h
popa
ret.

5) Стандартные адресации.

Адресаение — способ вычисления адресов операндов в памяти.

1. Регистровое адресование

AX - Accumulator

BX - Base Register

CX - Count Register

DX - Data Register

DI - Destination Index

SI - Source Index

SP - Stack Pointer

BP - Base Pointer

общего
назначения

CS - Code Segment

DS - Data Segment

ES - Extra Segment

386
FS
GS

SS - Stack Segment

сегментные

mov ax, BX.

2. Именованное адресование

mov ax, 2

3. Прямое адресование

Если адрес переменной известен, то его можно подавать как число.

ES:001 - переменная

mov ax, ES:001.

Макрос define (byte, word)

A db 5

B dw 7

mov AX,B // По умолчанию сегмент берется по DS

Для других сегментов:

mov ax, ES:B.

секущее ке более

16 бит в реальном

32 бит в защищённом

режиме

4. Косвенное адресование

Заданный смещение подается в регистр

mov ax, [bx+di]

на 386 можно было использовать только: bx+di, bx+si, bp+di, bp+si.

Компьютер с 386 можно использовать регистры общего назначения.

По-умолчанию берется DS, но если используются BP, EBP, ESP, то

используется сегмент стека SS.

5. Адресація по базе со смещением

`Mov ax, [bx+2]` ~ memory bytes кисточкой или мечкой (dh)

`Mov ax, [bx]+2 ~ mov ax, 2[bx].`

использование:

1) обращение к элементам структур.

BX - адрес начала структур

offset - смещение от начала.

2) обращение к параметрам и/и передаваемым через стек.

SP - тек. энт в стеке(указатель)

`push c
push b
push a` `mov bp, sp
mov ax, [bp+2]`

3) обращение к элементам одномерных байтовых массивов

`Mov ax, A[bx].`

6. Адресація с масштабированием.

`Mov ax, [EBX * $\frac{1}{2}$]` используются только 32х бит. регистры обзываются масштабированием.

7. Адресація по базе с индексированием

`Mov AX, [BX+SI+2]`

`Mov AX, [BX + SI]+2.`

`Mov AX, 2[BX+SI]`

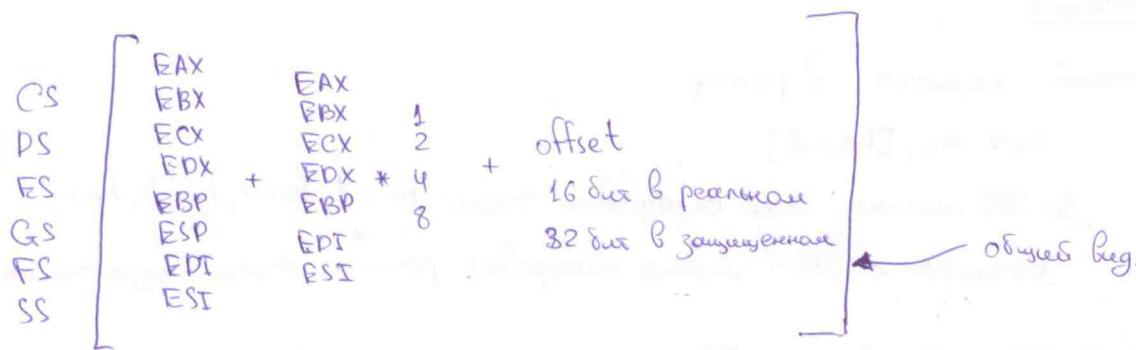
`Mov AX, [BX][SI]+2`

`Mov AX, [BX][SI+2]`

использование: индексирование двумерных байтовых массивов.

$X_{m \times n} : X[i,j] = i \times n + j \quad X[BX][SI], BX = i \times n, SI = j.$

8. Адресація по базе с индексированием и масштабированием.



BP - называемо SS

назад DS.

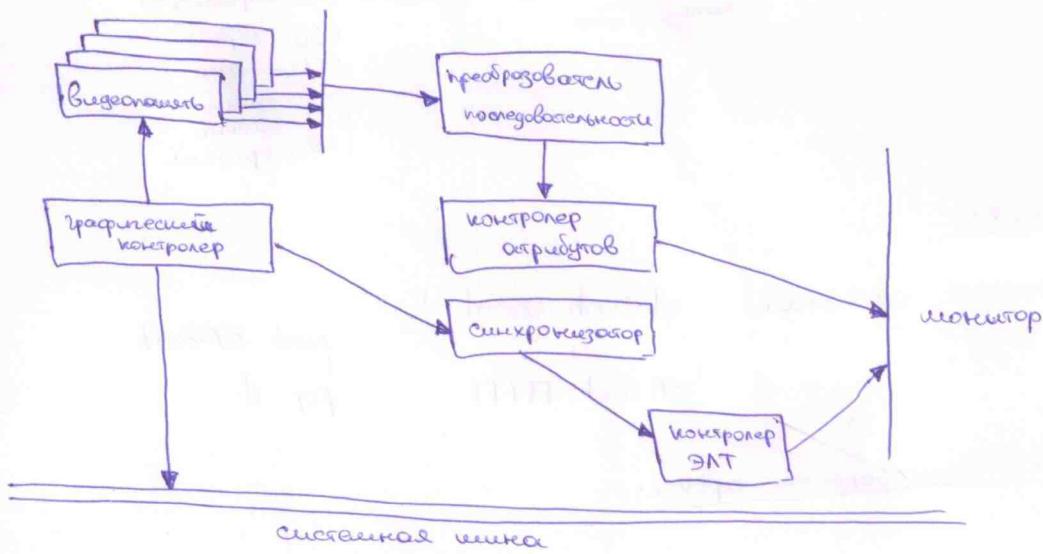
б) Архитектура видеоджадтера VGA. Текстовое решение. Задание символа для отображения.

Архитектура видеоджадтера VGA. Принцип работы с видеопамятью на АЗМ.

10h - работа с видеоджадтером.

VGA (Video Graphics Adapter) - непосредственные работы с изображением на экране.

Архитектура VGA:



Графический контроллер (ГК) - взаимодействие между ЦП и видеопамятью. Может привлекать битовые операции.

Видеопамять (ВП) - выводит содержимое сразу на экран.

256 DRAM на чипе по 64 Кбайт.

Предобразователь последовательности (ПП) - переводит содержимое ВП в посл. форм. и передаёт в ГА.

Контролер атрибутов (КА) с помощью кэш-памяти формирует содержимое ячеек.

Синхронизатор отвечает за временные синхронизацию переключения цветовых слоёв.

Контролер ЭЛТ формирует последовательность сигналов.

Текстовые решения:

Символы расположаются на экране в ячейках 9x16 пикселей.

Протоколы: 8x16 - VGA (1987)

8x14 - EGA (1985)

8x8 - CGA (1981)

Режимы:

1. 40x25 строк, 16 цветов, 360x400 - расширенное.

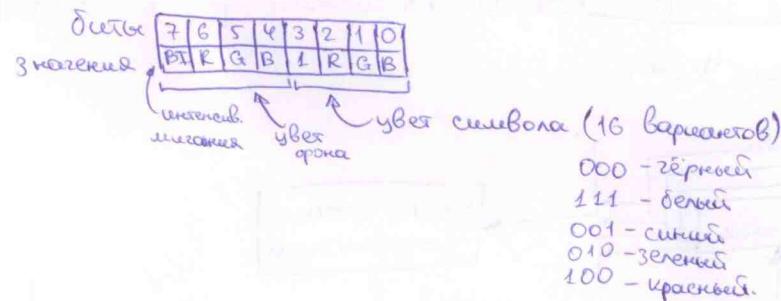
2. 80x25, 16 , 120x400.

3. 80x25, монохромный, 120x400.

Задание символа:

В 2х байтах

AL = "код символа"
AX: AH = "атрибут" (8 бит)



Процессы работы с буфером символов:

Абсолютное смещение: 0B800h:0000h
;
0B800h: FFFFh

push 0B800h
pop ds.

Явное размещение символов в произвольной строке:

dseg

myS db 'Hello', '\$'

start:

push 0B800h

pop es

mov di, 0

lea bx, myS

mov ah, 9

loop:

mov al, [bx]

cmp al, '\$'

jz exit.

mov es:[di], al

inc bx

add di, 2

inc ah

and ah, 7

or ah, 8

jmp loop.

exit:

14 | 7) Принцип модульного программирования. Согласование имен и атрибутов модулей.

Программа должна быть разделена на несколько самостоятельных частей.

Особенности:

- Управляемость кода
- Взаимодействие модулей.

Модули могут быть на различных ЯП и транслироваться отдельно.
Как согласовать части кода?

1. Согласование имен и атрибутов сегментов.

Сегмент кода: TEXT.

Сегмент данных: DATA.

Стек: STACK

Конец сегмента	segment	Readonly где ввода данные ввода	смещение PAGE-256 V PARA-16 BYTE WORD DWORD	тип AT PRIVATE COMMON V PUBLIC	разрядность USE16 USE32	'класс'] слово, которое определяет сегмент
{	CODE DATA STACK					

<конец сегмента> ends

_DATA word public USE16 'DATA'.

2) Согласование интерфейсов вызовов подпрограмм. Команды CALL/RET. Способы указания адресов вызовов подпрограмм. Могут писать и их чуть с выражением вызовов подпрограммы.

2.1. Согласование адресов вызовов и возвратов.

CALL - вызвать

RET - Вернуться

CALL NEAR - внутрисегментный

CALL FAR - межсегментный вызов.

RET NEAR - ближний

RET FAR - дальний возврат.

При NEAR командах адрес вызова и возврата меняется только в регистре указателя на инструкцию (команду) (IP/EIP) - WORD.

При FAR командах адрес состоит из двух регистров es:IP - DWORD

I В командах CALL/RET

RET: RETN, RETF - рекомендуется использовать в proc

Подпрограмма - это независимый участок кода

CALL <адрес кага>

Еще адрес в том же сегменте → NEAR

Еще адрес в другом сегменте → FAR.

II. A label near
B label far

call a \Leftrightarrow NEAR
call b \Leftrightarrow FAR

call word ptr [bx], call dword ptr [bx]
near far

III Директива proc

pt proc near/far
pt end proc

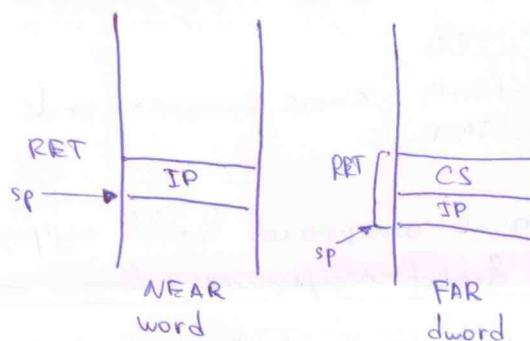
Более предпочтительна!

IV Директива .model

- model <имя> <размер> *Челодиректор на стек*
FAR STACK - стек от сегмента данных
✓ NEAR STACK

название	описание	Баг Баговка
tiny	1 сегмент кода 16 разряд. 2 сегмента данных, каждый память до 16 Кб	NEAR
small	1 сегмент кода и сегментов данных 16 разряд., gpl pos	NEAR
medium	* сегментов кода 1 сегментов данных	FAR
large/huge	и сегментов кода и сегментов данных	FAR
flat	1 сегмент кода 1 сегмент данных 32 разр., Windows $\leq 4 \text{ Гб}$	NEAR

Стек:



g) Способы передачи аргументов в ПП. Соответствие способов передачи аргументов. Особенности передачи параметров через стек

Механизмы:

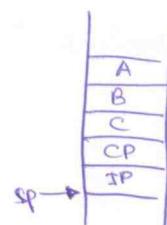
- по значению - число
- по ссылке - адрес
- по Возвр. значению - передаётся адрес, делает локальную копию, записывает результат.
- по результату - адрес для записи результата
- по ширине - друг. Высокомощное адрес параметра
- отложенные Возвращение - функция висит. адрес по необходимости.

Использование стека:

- Регистры
- Общая память
- Стек ✓
- В потоке кода
- В блоке параметров (ds)

Стек:

push A
push B
push C
call pt
add sp, 6 // очистка стека



p1:
push bp
mov bp, sp
mov ax, [bp+4] \leftrightarrow C
[bp+6] \leftrightarrow B
[bp+8] \leftrightarrow A.

ret 6 // очистка стека

A EQU [bp+4]
mov ax, A.

15

10) Стандартные вызовы ПП. Способъ задания стилей вызовов. Параметры дженеричные PROC.
 Вызов программы из модуля. Директивы extern/public.

Стандартные вызовы ПП определяют порядок передачи параметров в скрипте и способ создания со стека.

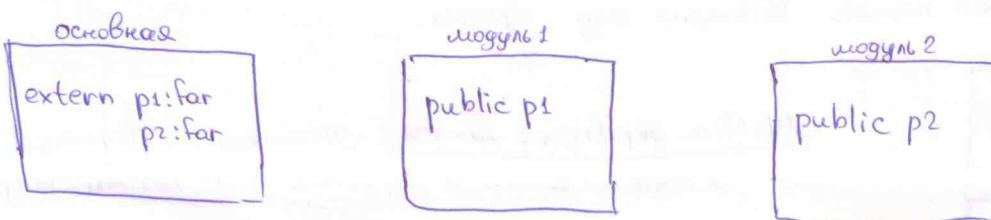
Язык	Порядок передачи	Способ передачи
no language	слева -> направо	Вызываемое ПП
C/C++	справа ->лево	на栈 cell в коге
stdcall	справа ->лево	Вызываемое ПП

model small Pascal

A proc near pascal

A end proc

EXTERN / PUBLIC



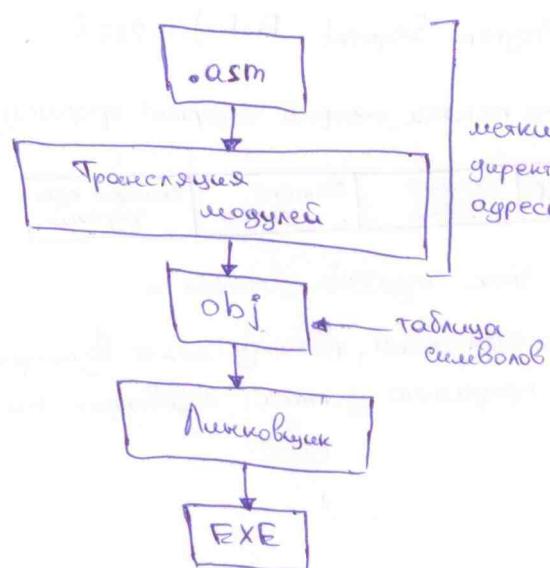
A proc near c p1:word, p2:dword

ARG p4: dword = l(4)

local l1: byte, l2: word

uses AX, BX, CX.

A end proc.



11) Управление памятью. Распределение памяти в реальном режиме. Распределение Conventional memory после загрузки DOS. Роль блока управления памятью MCB.

В реальный режиме в x86-архитектуре ОП распред. след. образом:

0h	Блокирое прерываний (256 dword за-таб) - 1кб	1
40h	данные память - СНОКБ	2
400h	Видеопамять	3
	Верхнее память	4
	использованное память расширение BIOS	5
	использованное память расширение BIOS	6
	использованное память расширение BIOS	
FFFFh	ROM BIOS	6

1. Адрес обработчиков прерываний
2. Рабочая память
3. Физические хранилище образ экрана
4. до 1Мб Upper Memory Block (UMB)
XMS, EMS - интерфейс управляемый
5. High Memory Area
6. ROM - Read Only Memory

Программное запускающее сразу после включения питания:

- POST - Power On Self Test.
- Setup - конфигуратор - настройка основных узлов
- Универсальный загрузчик - инициализирующее и загружает ОС
- BIOS - Base Input Output System.

После загрузки DOS рабочая память выглядит след. образом:

Блокирое прерываний
Данные DOS и BIOS
Ядро DOS
режимная часть command.com
CMB - Control Memory Block
окружение
CMB
режимная прогр.
CMB
режимная прогр.
CMB
свободный блок
не режущая часть command.com

CMB (Блок управления памятью) - занимает 256 б (стаки)
и содержит данные о состоянии блока памяти, который
за него следят.

CMB содержит 3 поля:

1. PSP - адрес 1 парagraфа в начале блока
2. Размер блока в параграфах (1 параграф = 16 б)
3. Пробник котчевого блока

PSP (Program Segment Prefix) - 256 б

Роль блока памяти, который содержит программу:

CMP	PSP	сегмент стека	сегмент данных	сегмент кода программы
-----	-----	---------------	----------------	------------------------

Программный блок отдаётся членам

Рабочая память организована, поэтому может возникнуть нехватка памяти \Rightarrow программа должна освобождать неиспользов. часть блока памяти.

12) Область CMW. Руководство DOS для работы с памятью. Область MMX. Интегрированная EMS и XMS.

48h: Блок

ah = 48h

BX - размер блока в байтах

Если CF=0 \Rightarrow всё хорошо и адрес B AX

Иначе CF=1 \Rightarrow ошибка и сообщение B AX

AX == 7 - CMW разрушено

\Rightarrow AX == 8 - не хватило памяти

можно определить количество занятой BX блоком памяти

49h: освобождение блок

ah = 49h

ES - смещение адреса блока, который нужно освободить

CF = 0 \Rightarrow всё хорошо

CF = 1 \Rightarrow ошибка.

AX == 7 - CMW разрушено

AX == 9 - B ES неверный адрес

4Ah: изменить размер блока

ah = 4Ah

BX - новый размер

ES - смещение адреса блока, размер которого изменяется

CF = 0 \Rightarrow всё хорошо

CF = 1 \Rightarrow ошибка

AX == 7 - CMW разрушено

AX == 8 - не хватило памяти

AX == 9 - неверно указано адрес

50h:

ah = 50h - позволяет текущему процессу претвориться другим процессом

BX - смещение адрес PSP.

Текущий процесс претворится другим процессом!

В DOS 5.0 появилась возможность управлять UMB (Upper Memory Block)

58h: изменение стратегии выделения памяти

Биты 2-0:

в BX записано

- 00 - первый подкодовый
- 01 - второй подкодовый
- 11 - последний подкодовый.

4-3:

- 00 - обычное память
- 01 - UMB
- 11 - UMB + раздел памяти.

High Memory Area - 65520 байт.

Интерфейс EMS (Expanded Memory)

Доступ к памяти осуществляется через "окно" и стратегию по 16 Кбайт.

int 67h:

Порядок работы с EMS:

1. Запросить EMS память и получить манипулятор стратегии.

0-12 - кон-бо стратегии по 16 Кб.

2. Отобразить стратегию как "окно".

Найти в рабочем стеке из запрошенных ~~стратегий~~

3. Получить адрес "окна"

BX - сегментный адрес окна

Интерфейс XMS (Extended Memory) - используя для пересылок данных.

Порядок работы с XMS:

1. Проверить наличие XMS.

2. Получить адрес функции-обработчика
dword ptr FuncXMS.

3. 09h - пересылка данных

Общее с письмом:

форматируя структуру данных:

а. источник.

б. сколько хотим переслать

в. на какой адрес

call FuncXMS.

(3) Режимы работы микропроцессора. Основное характеристики реального и защищённого режимов.

Начиная с процессоров 80286 вводится поддержка многоуровневости и защищённости архитектуры.

В целях совместимости с 8086 этот процессор содержит 2 режима работы:

1) реальный режим (режим реальной адресации) - эмуляция 8086

2) защищённый режим (режим виртуальной адресации) - использование всех возможностей

До момента загрузки ядра ОС процессор работает как 32x разрядный 8086.

Загружаются компоненты BIOS.

Возможна инициализация отдельных компонент.

ОС DOS - однозадачная и не переводит в защищённый режим. Это можно сделать программно.

Характеристики реального режима:

① Адресация 1 Мб памяти.

Размерированные адреса: номер сегмента * 16 + смещение

Всего: $1M \times 64K = 16G$.

8086 - 20 битная адресная шина $\Rightarrow 2^{20} = 1M$

80286 - 24 бита $\Rightarrow 2^{24} = 16M$ (но в целях совместимости выделен 19 на 21 бит)

② Сегментная ядроть (поддержка)

Размер сегмента не превышает 64 Kб.

В сегментных регистрах хранится адрес ~~сегмента~~ сегмента.

③ Нет поддержки многоуровней.

Нет списка контролируемых машинных машинных программ.

④ Нет аппаратного контроля доступа к памяти

Любая программа может обратиться к любому физ. адресу без ограничений.

Характеристики защищённого режима:

Цель: обеспечить поддержку многоуровневости

Задачи:

1. Использовать машинное машинные программы.

2. В случае необходимости обеспечить корректное взаимодействие между программами.

Могутся иерархиче рабочие с памятью

Могутся фрагменты и список программно-аппаратных компонентов управляющих в организацию доступа к памяти

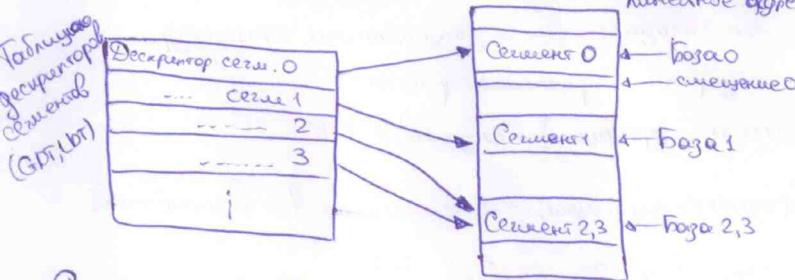
Сегментная ядроть оснащена и кэшем сегментов с надежностью дополнительные характеристики:

1. размер сегмента.

2. набор атрибутов физической физ. того, чтобы

передать заслуга контроля сегментации.

14) Задокументированный режим работы микропроцессора. Принципы сегментной и страницной адресации памяти.



Сегмент соответствует процессу.

Процесс видит не всё адресное пространство, а только область выделенного ему сегмента.
Это изолирует процессов друг от друга.

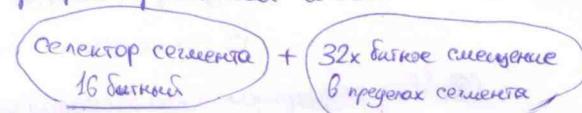
Разделение сегментов заменяется ядром ОС.

Дескриптор сегмента — это 8-байтовая структура, которая содержит:

1. Логический адрес сегмента в памяти
2. Длину сегмента.
3. Уровень привилегий сегмента.
4. Тип сегмента, определяющий назначение (текущие, записи, исполн.)

С помощью этих атрибутов осуществляется контроль доступа к сегменту и произвольная программа не может обратиться к привилегированному сегменту!

На этапе физической памяти адреса осуществляется проверка привилегий и если всё хорошо, то адрес физируется след. образом:



Логическая ячейка адресована 16 Мб линейных адресов, а для виртуальной до 4 Гб в 8x разделенных сегментах.

Селектор сегмента загружается в сегментные регистры.

Формат селектора:

- 0-1: описание запрашиваемых привилегий сегмента или тек. привилегии тек. сегмента когда
- 2: признак того из какой таблицы форм (GDT, LDT)
- 3-15: номер дескриптора в таблице.

Дескрипторы сегментов группируются по 32 разные таблицы в зависимости от назначения сегментов.
Адрес таблицы сохраняется в специальном регистре

Сегмент может быть выделен произвольного размера.

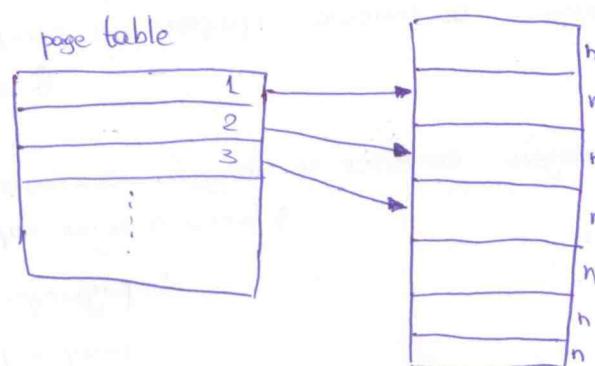
Страницы разделяются одинаковой длины (от 4Кб до 16Мб)

При страницной адресации используется

таблица страниц

Страницы памяти характеризуются

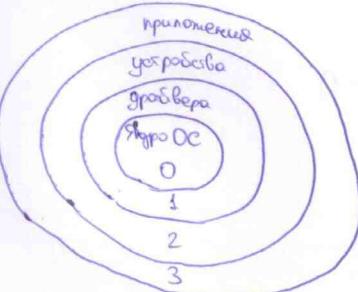
длиной и архитектурой доступа



15) Организация доступа к памяти. Уровни привилегий. Запрос доступа к памяти на уровне сегментов. Запрос доступа к памяти при страницкой адресации.

Запрос доступа может осуществляться в 2х способах:

1. Попытка загрузки селектора в сегментовый регистр
2. Обращение к странице памяти.

Существует 4 уровня привилегий (количество зон):  0, ..., 3
Внешний
Внутренний

В современных ОС используются уровни 0 и 3.

Ограничения налагдаются на:

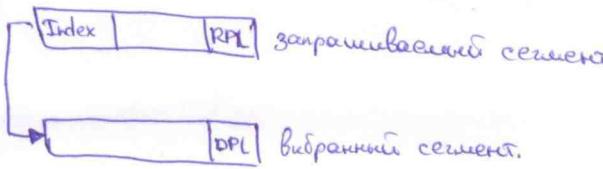
- использование памяти
- порты ввода/вывода
- использование некоторых структур.

При попытке загрузки селектора в сегментовый регистр происходит проверка:

1. берётся текущий сегмент кода

CPL и проверяется уровень привилегии

2.



Чем выше значение, тем выше доступ

16) Взаимодействие программ с ОС

ОС предоставляет различный набор услуг: Ввод/Выход, работа с файлами и т.д.

Обращение к прикладной программы к ОС наз. системные вызовы (СВ)

СВ состоит из следующих действий:

1. Подготовка данных, формирование запроса.
2. Передача управления ОС
3. После возврата из ОС - анализ результатов

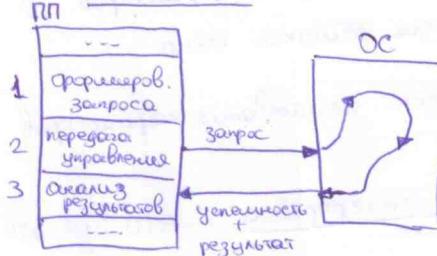
Порядок СВ в различных ОС отличается

Стандартные СВ - это системные интерфейсы (СИ)

Стандартные варианты организации СВ:

1. Программные прерывания
2. API функции.

Схема СВ:



Программные прерывания - варианты интерфейса СВ. Используются в DOS

Основные характеристики:

1. Данные для запроса формируются и передаются через регистры.

Для сложных структур используются указатели, которые передаются через регистры: EAX, EBX, ES.

2. Передача управления к ОС осуществляется через команду процессора int, которая входит в систему команд большинства процессоров

3. Результаты возвращаются в ПП через регистры.

Признак успешности выполнения операции: флаг переноса ($CF = 1$)

Используются регистры: AX, BX, EAX, EBX.

Параметры команды int: номер регистра прерывания.

Если $CF = 0$, то в AX код ошибки!

В процессорах с двухшаговой архитектурой в процессе передачи управления ОС происходит естественный переход, который решается на уровне команды int.

API функций - варианты интерфейса СВ. Сейчас является стандартом.

API - Application Programming Interface.

Делится на две категории OSF и POSIX.

Особенности API функций:

1. Запрос: цель функции и набор аргументов
Передача данных осуществляется через аргументы функции.

2. Передача управления ОС осуществляется командой call, а обратно ret.

3. Результаты выше Board передаются через регистры или память

Если $AX \leq 0 \Rightarrow$ ошибка

Иначе \Rightarrow результат в EBX, ECX, ES.

Сложность заключается в соглашении о вызовах:

- порядок передачи параметров в стек
- порядок очистки стека.
- порядок ожидавших параметров из стека

Задача API интерфейса - это убрать разрыв между СВ и вызовами собственных модулей.

17) Особенности организации Windows-приложений.

1. 16 разрядное DOS-приложение (нет ограничений на реализованную архитектуру)

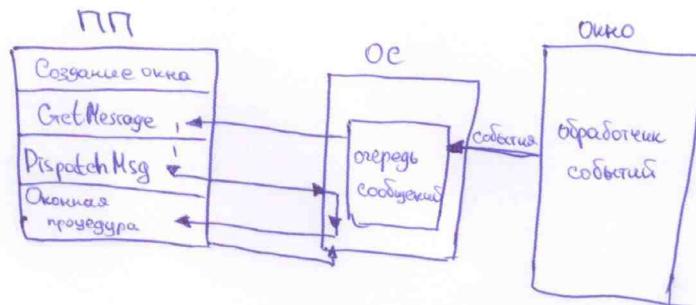
2. Консольное Windows-приложение

используются 32 разрядные регистры и стеки.
OS с помощью API функций.

3. Окноное Windows-приложение

Использует заданную структуру, которую нельзя нарушать.

Окно - это отдельный объект под управлением OS



18) Числа с плавающей запятой в FPU.

FPU (Floating point unit) - сопроцессор до i486 обладающий своим набором регистров и своим набором команд

Название	Кол-во разрядов (бит)	Мантисса	Смещение нормы	Делегате нормы, знако.
Single precision	32	23	127 [8]	2^{-126}
double precision	64	52	1023 [11]	2^{-1022}
double extended precision	80	64	16386 [14]	2^{-16385}

Пример:

$$123,375 = 1111011,011 = 2^6 \cdot 1,111011011.$$

Single смещение + норма

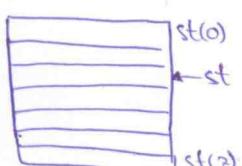
0 10000101 111011011	127+6	23
--------------------------	-------	----

double:

0 1023+6 ...

Регистры FPU

R02 - R01 - 80 битные.



SR - регистр состояния (16 бит)

TW - регистр тегов (8 бит по 2 бита)

00 - число запр. в регистре

01 - запр. 0

10 - запр. не число

11 - нет числа в регистре

CR - регистр управления (16 бит)

19] RC - окружесткие

00 - к ближайшему целому

01 - + ∞

10 - - ∞

11 - к нулю

9] DC - точность

00 - single

01 - не видимка

10 - double

11 - double extended

fld, fld

fst, fstp.