

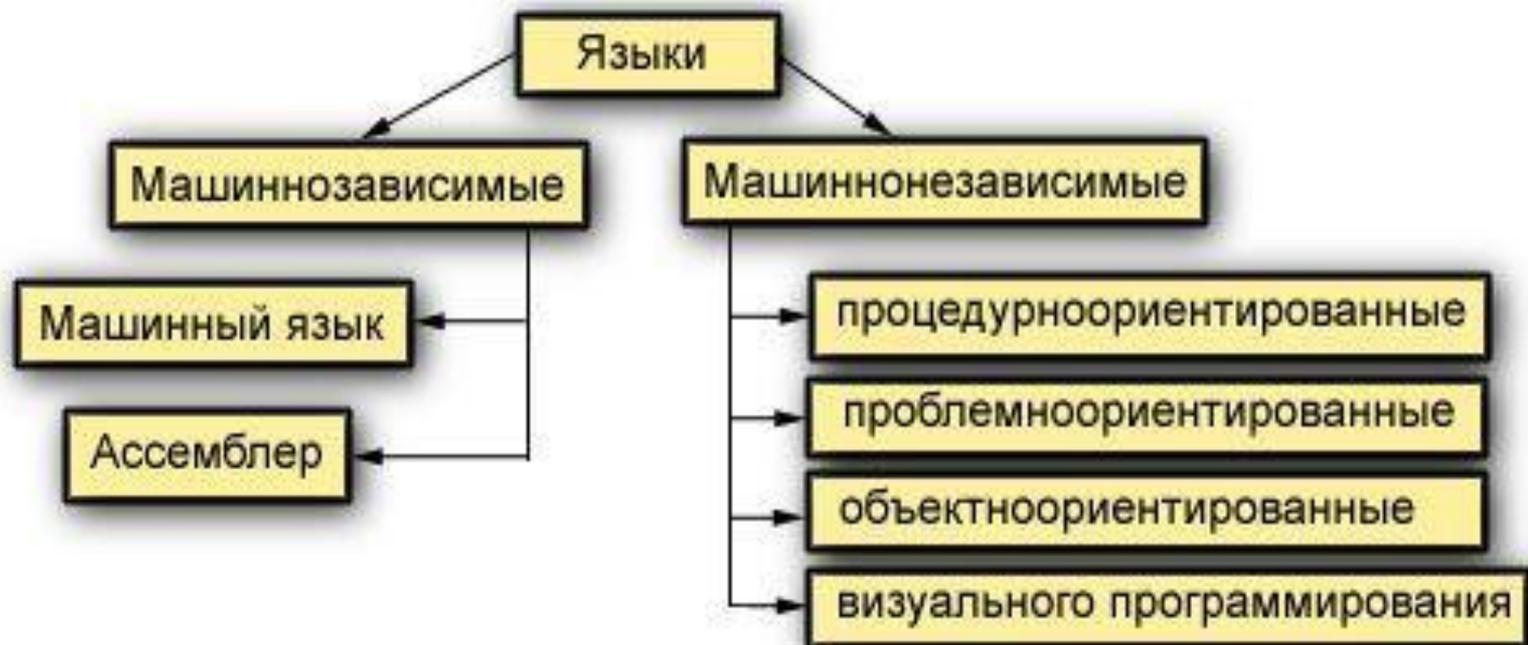
Лекция №1

- Введение.
 - Классификация языков программирования.
 - Машинный язык и язык ассемблера.
 - Структурная схема компьютера.
- 

Литература

- Тищенко В.И. Учебное пособие по курсу «Системное программирование»;
- Тищенко В.И. Лабораторный практикум по курсу «Системное программирование»;
- Тищенко В.И. Лабораторный практикум «Разработка оверлейных и резидентных программ»;
- Юров В. Ассемблер – учебник;
- Юров В. Ассемблер – практикум.

Классификация языков программирования



Машинный язык программирования

Машинный язык программирования – это язык, непосредственно воспринимаемый компьютером. Каждая его команда интерпретируется аппаратурой ЭВМ.

Машинная команда имеет структуру вида:

код операции, операнд1, операнд2

Операнды - это данные или адреса, над которыми будет выполняться действие, определенное кодом операции.

Структура данных в оперативной памяти называется форматом данных.

Ассемблер как язык программирования

Языки ассемблерного типа используют мнемоническое обозначение адресов и кодов операций.

Ассемблер включает в себя:

- машинные команды,
- символические адреса,
- макросы,
- комментарии.

Этапы развития языков программирования

Выделим 5 основных поколений:

1. (конец 50-х г.) – Fortran, Algol;
2. (середина 60-х г.) - Cobol, Lisp;
3. (70-е годы) - PL/1, Pascal;
4. (80-е годы) - Object Pascal, C⁺⁺, Ada;
5. (90-е годы) - Visual Basic, Delphi, Builder.

Функциональные возможности языков и технологии программирования

- с развитием аппаратных средств появились:
функции ввода-вывода,
поддержка файловой системы,
взаимодействие с операционной системой;
- с усложнением задач:
поддержка подпрограмм,
механизм передачи параметров (основа для методологии **структурного программирования**);

Функциональные возможности языков и технологии программирования (продолжение)

- возможность создания больших программ на основе подпрограмм изменили архитектуру языков и подход к компоновке программ (**механизм раздельной трансляции программ и понятие модульности**);
- абстракция данных, типизация и модульность – основа **технологии объектно-ориентированного программирования**;
- появление среды Windows породило **технология визуального программирования**.

Архитектура ЭВМ

Архитектура ЭВМ – это абстрактное представление ЭВМ, которое отражает ее структурную, схемотехническую и логическую организации.

Архитектура ЭВМ

Общие принципы построения ЭВМ

Архитектура ЭВМ

Структурная схема ЭВМ

Средства и способы доступа к элементам структурной схемы ЭВМ

Способы представления и форматы данных ЭВМ

Набор и доступность регистров

Организация и способы адресации памяти

Форматы машинных команд

Набор машинных команд ЭВМ

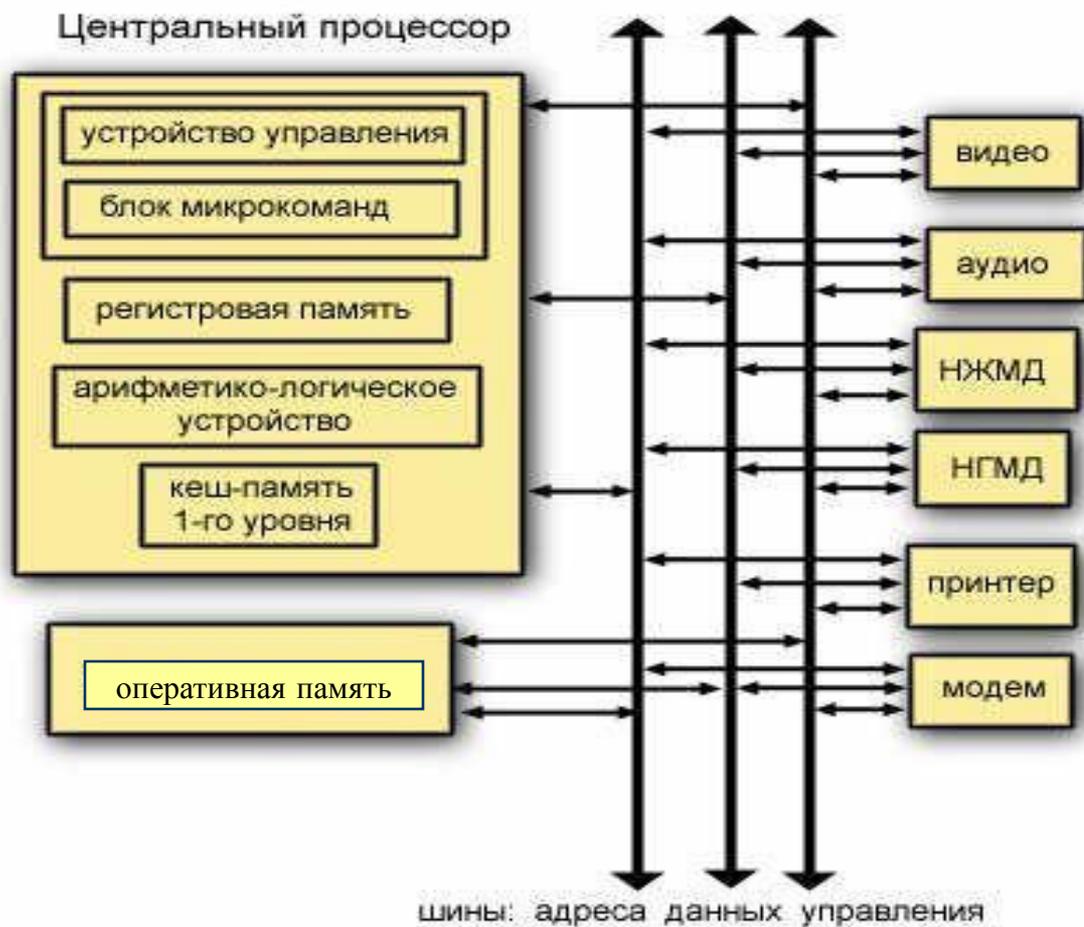
Организация ввода/вывода ЭВМ

Обработка нештатных ситуаций (прерываний)

Принцип Фон-Неймана

Схема выполнения программы в компьютере

Структурная схема компьютера



Лекция №2

- 1.Классификация регистров.
 - 2.Назначение регистров.
 - 3.Адресация памяти.
 - 4.Физическая адресация памяти.
- 

Регистры

Электронное устройство, предназначенное для временного хранения информации, называется регистром.

Расположены на кристалле МП.

Характеризуются размером:

8-разрядные, 16-разрядные, 32-разрядные, 64-разрядные.

Регистры IBM/PC (intel 8086/8088)

Регистры

Регистры данных и адресов

Управляющие регистры

Регистры общего назначения (данных)

Сегментные регистры

Регистры указателей

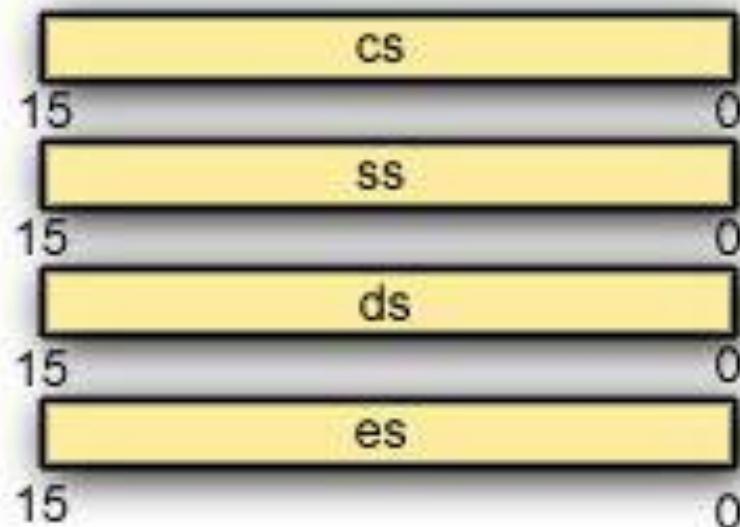
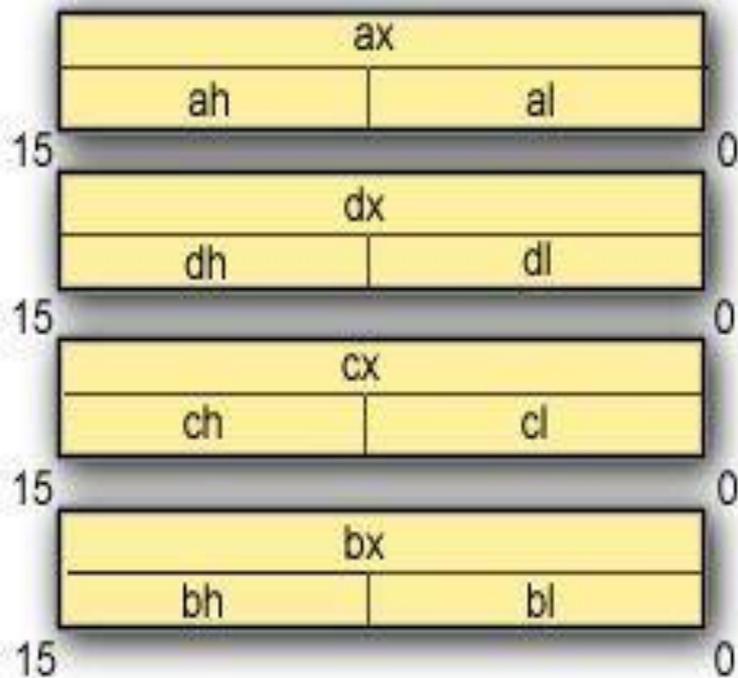
Регистр флагов

Указатель команд **ip**

Регистры данных

Сегментные регистры

Регистры общего назначения:



Регистры индексов и указателей

- sp – указатель стека;
- bp – указатель базы;
- si – индекс источника;
- di -индекс результата.

Адрес памяти задается двумя значениями – сегмент и смещение, например :

ds:dx, es:bx, ss:sp, ss:bp.

Текущая исполняемая команда
определяется **cs:ip**.

Управляющие регистры

- **ip** – указатель команд,
- **регистр флагов.**

Указатель команд - содержит смещение следующей команды в кодовом сегменте.

Регистры 32-разрядного МП Pentium

- 16 – системных;
 - 16 – пользовательских:
-  8 РОН, индексных и указателей по 32 разряда
eax/ax/ah/al, **ebx/bx/bh/bl**,
edx/dx/dh/dl, **ecx/cx/ch/cl**,
 -  6 сегментных по 16 разрядов:
cs, ds, ss, es, fs, gs;
 -  2 регистра состояния и управления по 32 разряда: **eip/ip** и **eflags/ flags**.

Адресация памяти

Адресация памяти

Физическая

В ассемблере

Способы адресации

Регистровая

Непосредственная

Прямая

Косвенная регистровая

Адресация по базе

Прямая с индексированием

Адресация по базе с индексированием

Некоторые константы

$2^{16} = 64 \text{ кб}$, $2^{32} = 4 \text{ Гб}$, $2^{20} = 1 \text{ Мб}$, $2^{24} = 16 \text{ Мб}$.

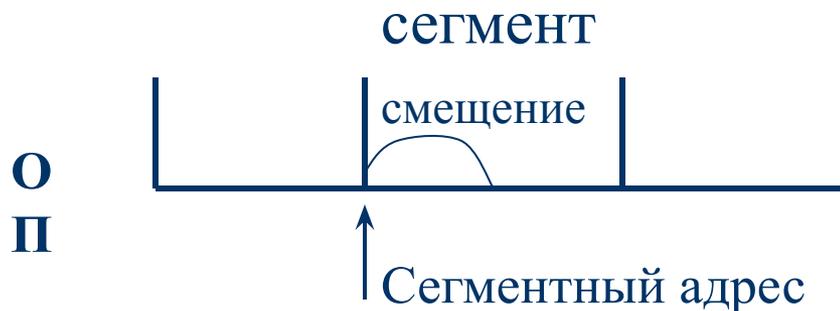
Для МП intel 8088 размер машинного слова – 16 бит или 2 байта, шина имела 20 линий, поэтому адрес 20- разрядный.

Физическая адресация памяти

Адрес, выдаваемый на шину адреса, называется **физическим**.

Физический адрес = (сегментный адрес)*16 + смещение
или

Физический адрес = (сегментный адрес)*10h + смещение.



Пример вычисления физ. адреса

Пусть содержимое сегментного регистра равно $2011h$, смещение равно $15h$, тогда $ФА = 20110h + 15h = 20125h$

Расположение машинного слова в памяти

Младший байт записывается в ячейку с меньшим адресом, старший – в ячейку с адресом на 1 больше.

Пример:

пусть число 1234h размещено с адреса 1927:0000, т.е. занимает ф.а. 19270h и 19271h.

Тогда цифры 34h – по адресу 19270h, а 12h – по адресу 19271h.

Назначение регистров - РОН

AX – аккумулятор.

BX – как вычислительный регистр, но может быть адресным.

CX – счетчик в некоторых командах.

DX – расширитель аккумулятора.

Назначение регистров адресации

**si, di, bp, bx – основное назначение –
хранить 16 – разрядное значение при
формировании адреса.**

Назначение регистров - управляющие регистры

- **ip** – указатель команд,
- **регистр флагов.**

Указатель команд - содержит смещение следующей команды.

Методы изменения порядка выполнения команд:

1. последовательный порядок команд,
2. переход внутри сегмента (**near** – переход),
3. переход в другой сегмент (**far** – переход).

Указатель стека `sp`

Определяет смещение текущей вершины стека.
Адрес стека определяется как **`ss:sp`** или **`ss:bp`**.

Пример загрузки сегментных регистров `cs` в `ds`:

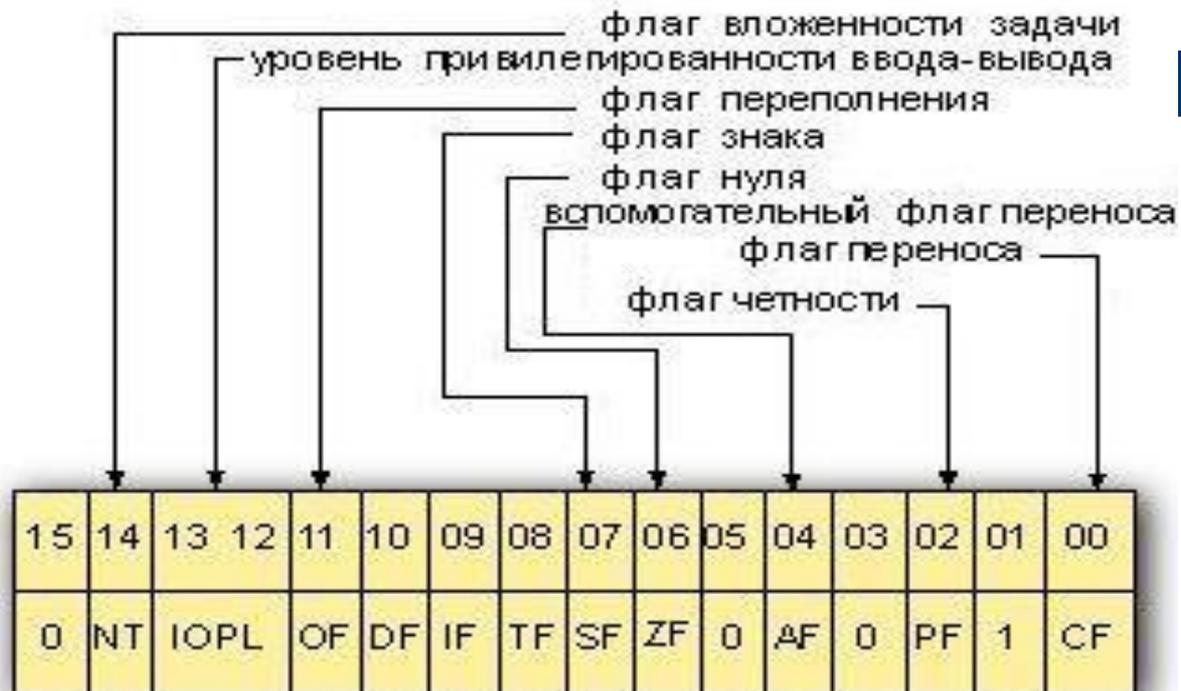
```
а) mov ax, cs  
   mov ds, ax  
в) push cs  
   pop ds
```

Лекция №3

- 1.Регистр флагов.
 - 2.Механизм формирования физического адреса.
 - 3.Форматы данных.
 - 4.Директивы определения данных.
- 

Содержимое регистра flags

ФЛАГИ СОСТОЯНИЯ:



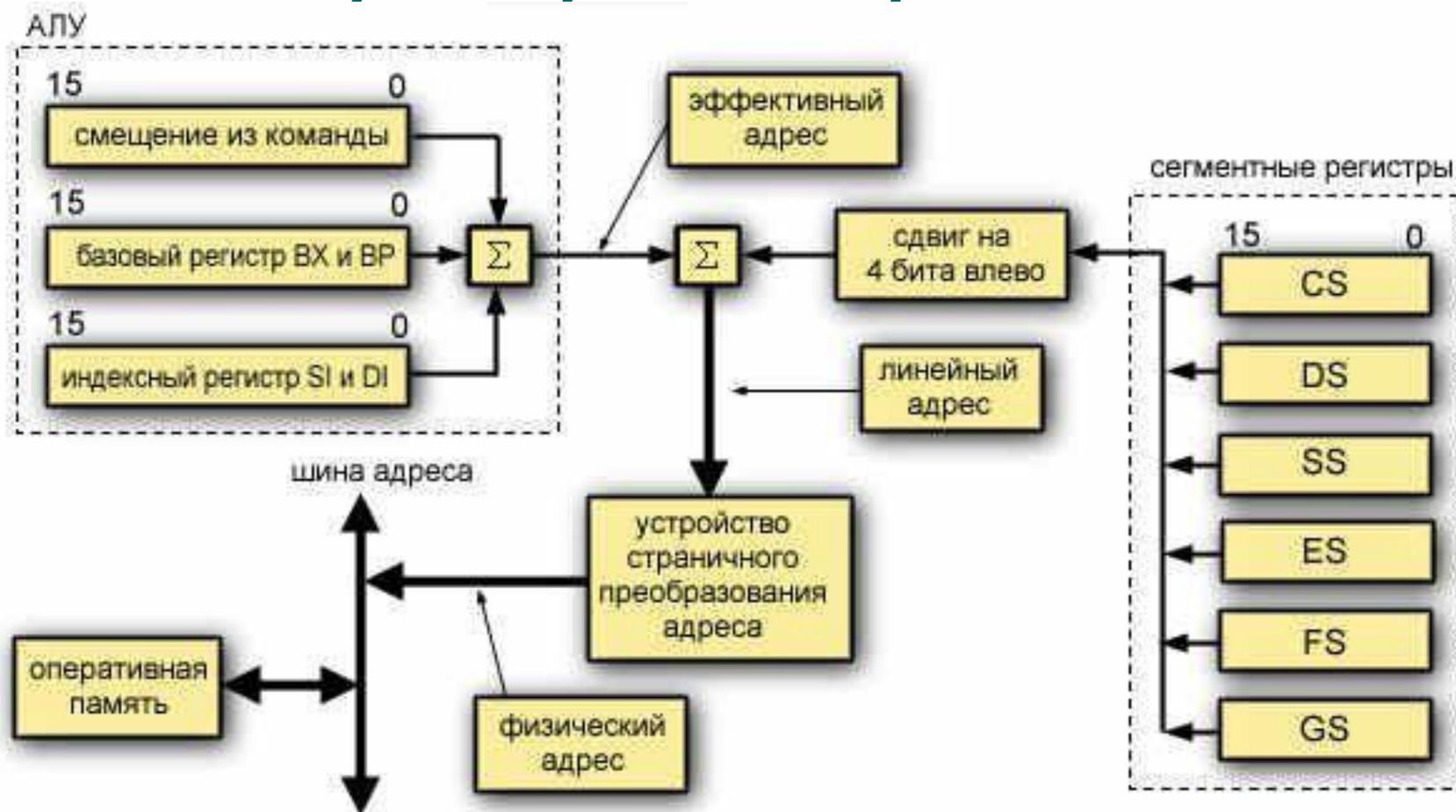
ФЛАГ УПРАВЛЕНИЯ:
флаг направления

СИСТЕМНЫЕ ФЛАГИ:
флаг трассировки
флаг прерывания

Обозначения регистров под отладчиком

имя флага	1	0
переполнение (да/нет) overflow	OV	<u>NV</u>
направление (↑/↓) directory	DN	<u>UP</u>
прерывание (разрешено/запрещено)	<u>EI</u>	DI
знак (-/+) sign	NG	<u>PL</u>
нуль (да/нет) zero	ZR	<u>NZ</u>
дополнительный перенос(да/нет)	AC	<u>NA</u>
четность(чет/нечет) parity	PE	<u>PO</u>
перенос (есть/нет) carry	CY	<u>NC</u>

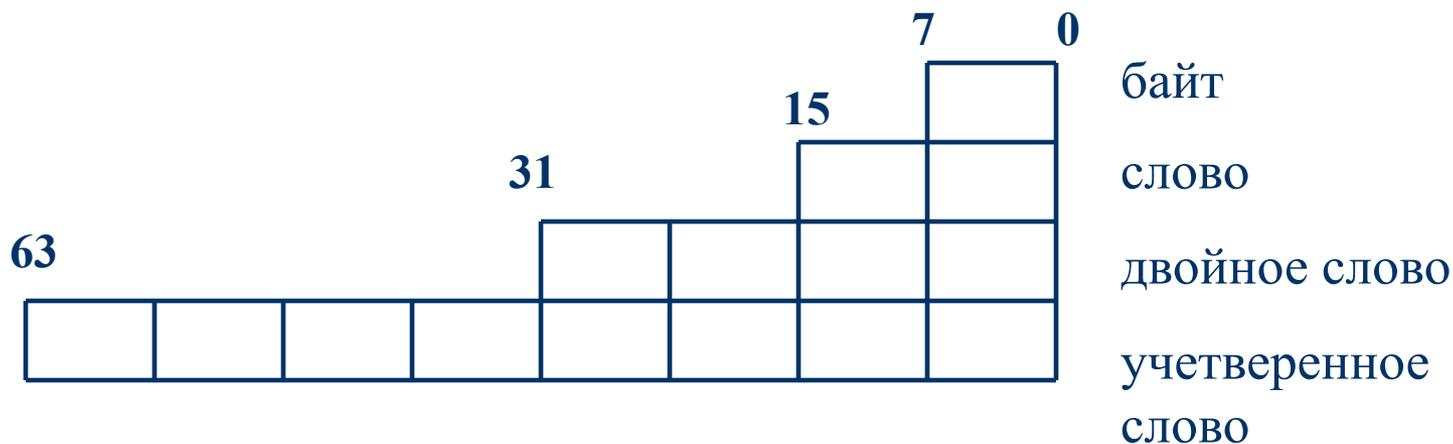
Механизм формирования физического адреса в реальном режиме



Типы данных

Классификация данных по разрядности

(поддерживается МП на аппаратном уровне)



Типы арифметических данных: логическая структура (см. стр.12-14 пособия)

Форматы арифметических данных



Форматы данных сопроцессора

8 регистров данных длиной 80 бит.

Оперирует 7 типами данных:

- 3 типа целых (слово 16 бит, короткое целое 32 бита, длинное целое 64 бита);
- 3 типа вещественных (короткое 32 бита, длинное 64 бита, временное 80 бит):
знак, характеристика, мантисса
1 бит 8, 11 или 15 бит 23, 52 или 64 бита;
- Упакованные двоично-десятичные числа.

Операторы ассемблера

Общий вид оператора ассемблера:

метка КОП операнд_1,операнд_2

Оператором может быть:

- машинная команда;*
- директива транслятора;*
- макрокоманда;*
- комментарий.*

Директивы транслятора для определения данных в ассемблере

формат директивы:

имя **d** $\left[\begin{array}{c} \mathbf{b} \\ \mathbf{w} \\ \mathbf{d} \end{array} \right]$ — выражение

a dw 10 ; в десятичной системе счисления

b dw 10h ; использование шестнадцатеричной
системы счисления

Выражение

в директиве определения данных

может быть:

- константой:

```
ABC1 dw 1234h
```

- СПИСКОМ:

```
ABC2 db 1,2,3
```

- строкой:

```
ABC3 db 'stroka'
```

- с операцией дублирования:

```
ABC4 db 4 dup (0)
```

Лекция №4

-Режимы адресации в ассемблере.



Режимы адресации

Режимы адресации

Регистровая

Непосредственная

Прямая

Косвенная регистровая

Адресация по базе

Прямая с индексированием

По базе с индексированием

Способы задания операндов в операторах ассемблера

1. Регистровая адресация

`mov ax,bx`

`mov al,dl`

2. Непосредственная адресация

`mov ax,1234h`

`mov cl,'a'`

`mov ah,5`

3. Прямая адресация

```
.data
```

```
ABC dw 1234h
```

```
.code
```

```
mov dx,ABC
```

Пусть адрес ABC = ds:0000,

тогда команда под отладчиком выглядит так:

mov dx, word ptr [0000] или **mov dx, [0000]**

Для вычисления адреса операнда по умолчанию используется сегментный регистр **ds**.

4. Косвенная регистровая адресация

Смещение, которое вычисляется аппаратно для доступа к операнду в памяти, называется исполнительным (эффективным) адресом (ЕА)

.data

ABC dw 1234h

.code

.....

mov bx, offset ABC

mov ax,[bx]

Для вычисления адреса операнда по умолчанию используется сегментный регистр - **ds**

5. Адресация по базе

Эффективный адрес EA вычисляется:

$$EA = \begin{cases} [bx] + \text{индексное смещение} \\ [bp] + \text{индексное смещение} \end{cases}$$

Размер индексное смещение (сдвига) - 0, 1 или 2 байта.

Пример: обращение к 2-му элементу массива слов

`.data`

```
ARRAY dw 1,2,3,4,5
```

`.code`

```
mov bx,offset ARRAY
```

```
mov ax,[bx]+2
```

6. Прямая с индексированием

Эффективный адрес EA вычисляется:

$$EA = \begin{cases} [si] + \text{индексное смещение} \\ [di] + \text{индексное смещение} \end{cases} + \text{смещение прямого адреса}$$

Пример: загрузить 5-й элемент массива байтов в регистр al.

```
.data
```

```
table db 0Ah,0Bh,0Ch,0Dh,0Eh,0Fh
```

```
.code
```

```
mov di,0002 или mov di,2
```

```
mov al, table[di+2]
```

7. По базе с индексированием

Эффективный адрес EA вычисляется:

$$EA = \begin{cases} [bx] & [si] \\ [bp] & [di] \end{cases} + \text{индексное смещение}$$

Пример: задан массив записей, каждая запись состоит из 6 слов.
Переслать 3-е слово 2-ой записи в регистр ax.

```
.data
```

```
table dw 1,2,3,4,5,6,7,8,9,10,11,12,.....
```

```
.code
```

```
mov si,12 или mov si,2*6
```

```
mov bx,offset table
```

```
mov ax, [bx+si+4 ]
```

Форма записи операнда

Для адресации по базе с индексированием
возможны следующие комбинации регистров:

$[bx][si] + \text{сдвиг}$ совместно с регистром ds

$[bx][di] + \text{сдвиг}$ совместно с регистром ds

$[bp][si] + \text{сдвиг}$ совместно с регистром ss

$[bp][di] + \text{сдвиг}$ совместно с регистром ss

Используются разные формы записи операнда:

$[bx][di] + 4$, $[bx+di] + 4$, $[bx+di+4]$

Сводная таблица используемых регистров адресации

r/m	адрес операнда	reg	регистры	
000	[bx+si+инд. смещение]	000	ax	al
001	[bx+di+инд. смещение]	001	cx	cl
010	[bp+si+инд. смещение]	010	dx	dl
011	[bp+di+инд. смещение]	011	bx	bl
100	[si+инд. смещение]	100	sp	ah
101	[di+инд. смещение]	101	bp	ch
110	[bp+инд. смещение]	110	si	dh
111	[bx+инд. смещение]	111	di	bh

Размеры индексного смещения

- сдвиг отсутствует, длина поля равна 0;
- занимает 1 байт, значение в диапазоне -128 до +127;
- занимает 2 байта, значение в диапазоне -32768 до +32767.

Лекция №5

- Принципы и свойства архитектуры ЭВМ.
- Иерархия памяти.

Принципы архитектуры

- 1. Принцип хранимой программы**
(Джона фон Неймана) – код программы и данные хранятся в оперативной памяти (ОП).
- 2. Принцип микропрограммирования** – для каждой команды есть набор действий- сигналов, генерируемых для ее выполнения.
- 3. Принцип адресности** – пространство ОП линейно, т.е. совокупность последовательно пронумерованных ячеек памяти. Номер ячейки – ее адрес (0,1,2,...).
- 4. Принцип программного управления** – последовательное выполнение команд программы. Для изменения порядка выполнения используются специальные команды.

Принципы архитектуры

5. **Принцип однородности памяти** – для процессора нет принципиальной разницы между данными и командами. Над командами можно выполнять такие же действия как и над данными.
6. **Принцип двоичного кодирования** – необходимо четко разделять пространство данных и команд, т.к. процессор трактует двоичную информацию в зависимости от назначения адресного пространства .
7. **Принцип безразличия к целевому назначению данных** – процессору не важна логическая нагрузка обрабатываемых данных.

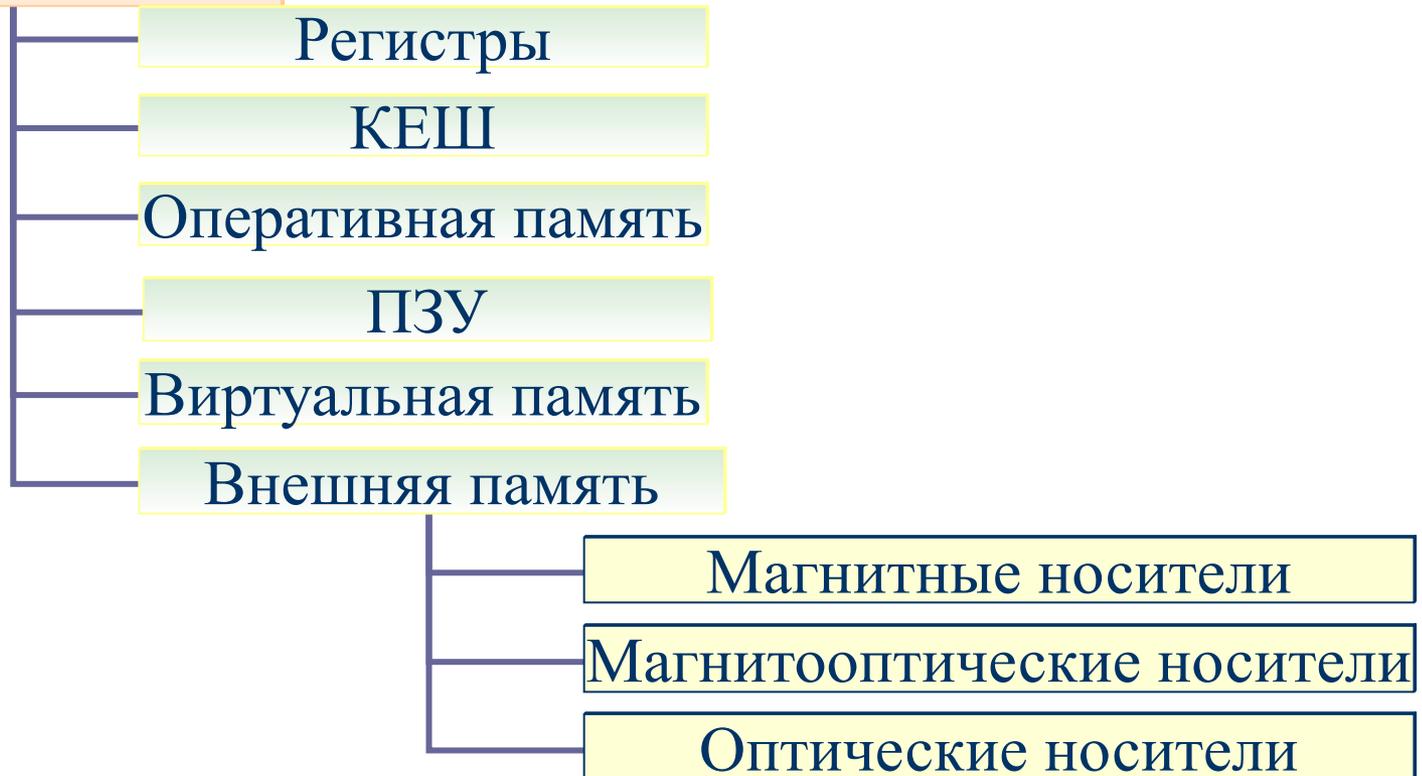
Индивидуальные особенности процессоров

Способы кэширования кода и данных:

- в i486 – один блок встроенного кеша размером 8 кб для кодов и данных;
- в Pentium - два блока по 8 кб, один для кода, другой для данных. Возможен одновременный доступ к коду и данным.

Иерархия памяти

Иерархия памяти



Кеш - память

Располагается между основной памятью и процессором.

Основное назначение - улучшение эффективной скорости взаимодействия с памятью и увеличение быстродействия процессора.

Подразделяется на кеш I уровня (на кристалле) и кеш II уровня (на кристалле или вне его).

Концепция кеш – памяти

Предвосхищение наиболее вероятного использования процессором данных из ОП путем их копирования в кеш-память. Данные передаются блоками, состоящими из нескольких слов.

Среднее время доступа к кеш – памяти:
в 3 раза быстрее, чем к ОЗУ,
в 10 раз быстрее, чем к ПЗУ.

Оперативная память (ОП)

Физическая память, к которой МП имеет доступ по шине адреса, называется ОП.

Выполнена ОП на дешевых и медленно действующих полупроводниковых устройствах.

Реально реализована как последовательность ячеек – байтов. Каждому **байту** соответствует уникальный **адрес**, называемый **физическим**.

Аппаратный механизм управления ОП

Он обеспечивает:

- компактность хранения адреса в команде;
- гибкость механизма адресации;
- защиту адресного пространства задачи;
- поддержку виртуальной памяти.

Модели использования ОП

МП аппаратно поддерживает модели:

- сегментированную;
- страничную.

Режимы работы МП:

- режим реальных адресов;
- защищенный;
- виртуального процессора.

Реальный режим работы МП intel 8086/88. Основные понятия

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

Сегмент – независимый, поддерживаемый на аппаратном уровне блок памяти.

Программно возможен непосредственный доступ только к 4(6) сегментам. Каждый сегмент имеет специальное назначение.

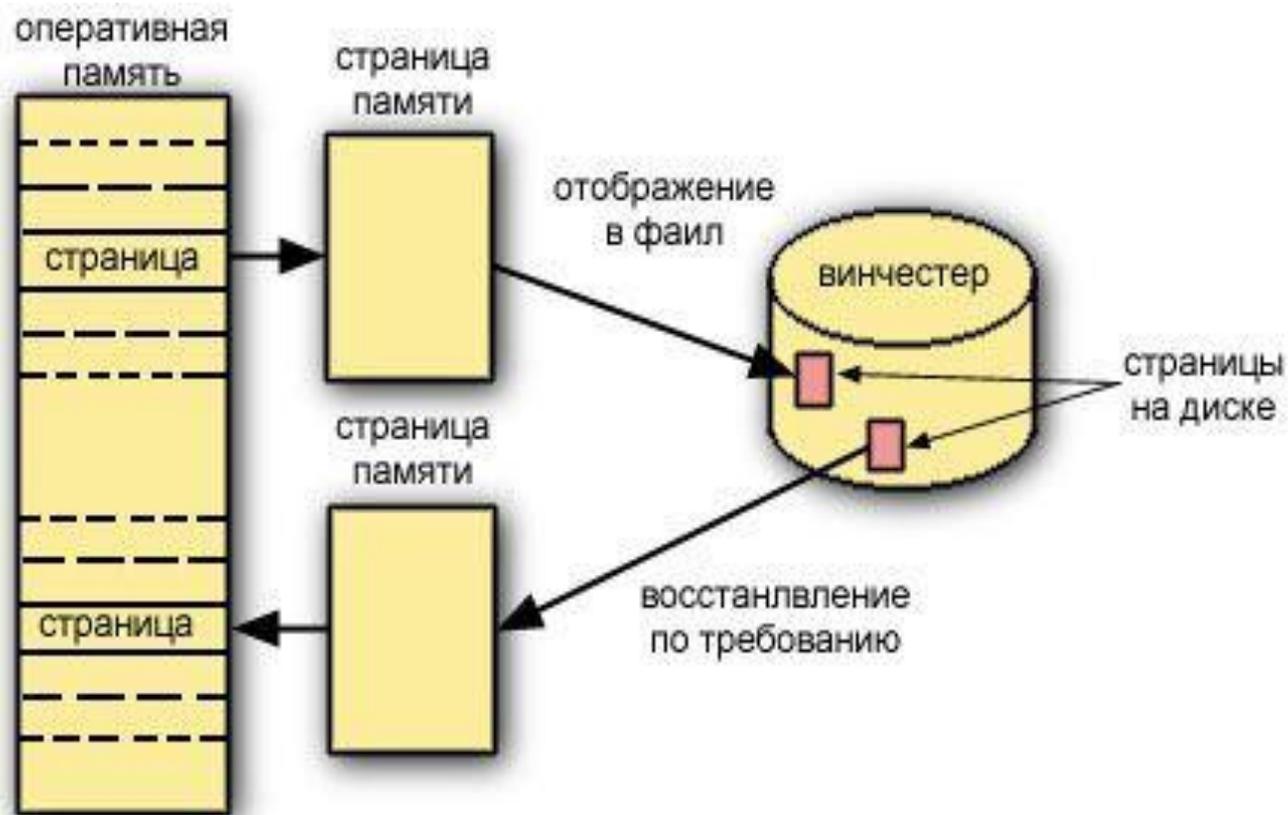
Размещение сегментов программы

Первоначально начальные физические адреса сегментов программы неизвестны.

ОС распределяет память и помещает адреса:
в реальном режиме - в сегментные регистры;
в защищенном – в специальную дескрипторную таблицу.

Под **физическим** (линейным) адресом понимается **адрес, выдаваемый на шину адреса.**

Виртуальная память



Лекция №6

- Виды операндов, поддерживаемые транслятором.
 - Структура команды мп intel 80386.
 - Байт mod r/m.
 - Формат команды mov (1-2).
- 

Общий вид оператора ассемблера

метка КОП операнд_1,операнд_2

Оператором ассемблера может быть:

- *машинная команда;*
- *директива транслятора;*
- *макрокоманда;*
- *комментарий.*

Виды операндов, поддерживаемые транслятором TASM

- Постоянные или непосредственные (число, строка):
`ABC equ 3` или `ABC1=ABC1+120/5`;
- Адресные (сегмент:смещение): `ds:0001`;
- Перемещаемые – адреса, непривязанные к конкретному физическому адресу памяти;
- Текущее значение счетчика адреса (`$`): `mov ax,$+5`;
- Регистровые: `ax`, `cx`, `dx` и т.д.;
- Базовый и индексный операнды: `bx`, `di`, `si`;
- Арифметические операторы в выражении: `+`, `-`, `*`, `/`;

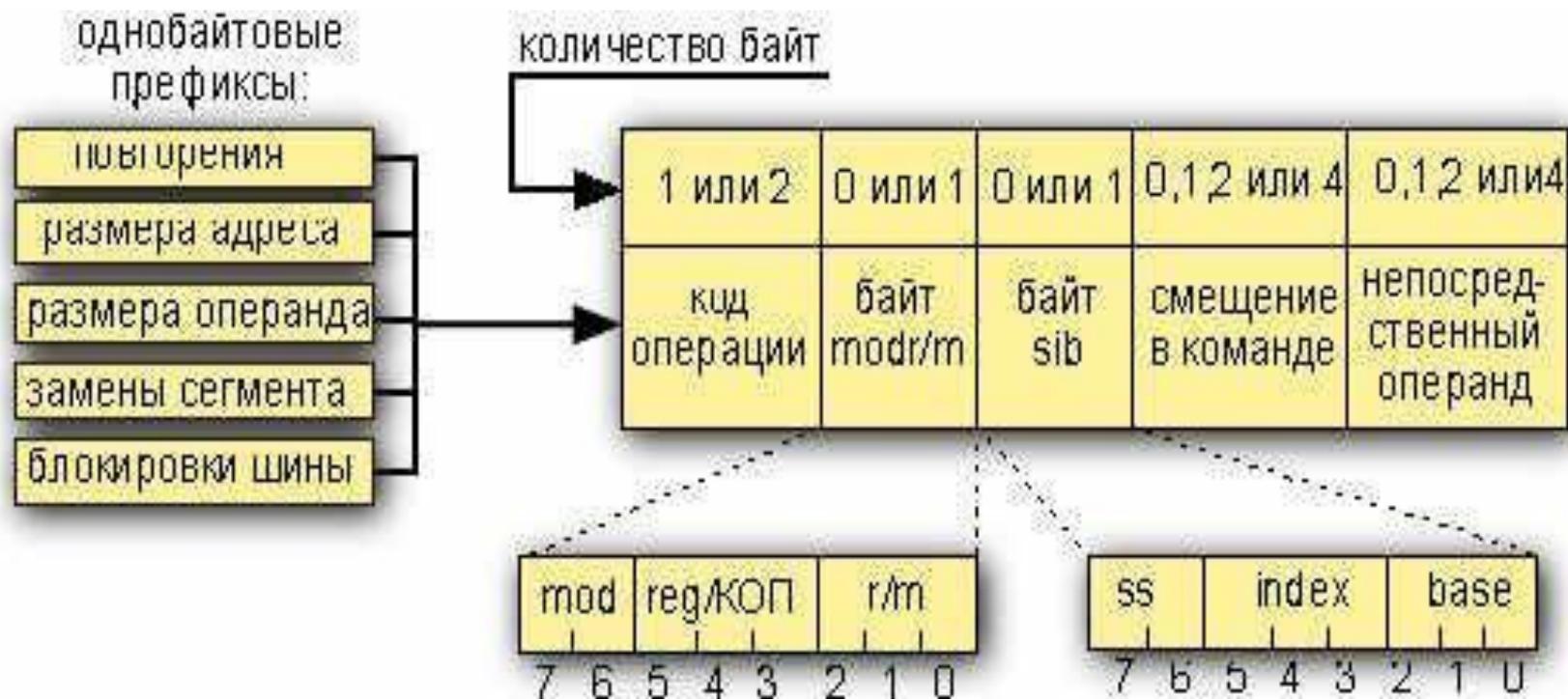
Виды операндов (продолжение)

- Операторы сравнения: `eq`, `ne`, `lt`, `le`, `gt`, `ge` и т.д.
- Логические операторы: `and`, `or`, `xor`.
- Индексный оператор `[]`: `mov ax, ABC [si]`
- Оператор определения типа (приписывает операнду указанный тип):
 - формат:** тип ptr выражение
 - пример: `abc dw 12h`
 `mov al, byte ptr abc`
- Получение сегментной части адреса или смещения:
 - `mov ax, seg ABC`
 - `mov dx, offset ABC`

Машинная команда



Формат машинной команды МП intel 80386



Префиксы

Каждый префикс может занимать 1 байт.
Есть 5 типов префиксов:

- повторения (в цепочечных командах);
- размера адреса;
- размера операнда;
- замены сегмента;
- блокировки шины.

Структура байта mod r/m

| mod | reg/коп | r/m |

Поле mod – 2 бита, кодировка:

00 – индексное смещение отсутствует;

01 – смещение занимает 1 байт;

10 – смещение занимает 2 байта;

11 – поле r/m определяет регистр.

Поля reg и r/m имеют длину по 3 бита.

Определяют регистр и способ адресации.

Сводная таблица используемых регистров адресации

r/m	адрес операнда	reg	регистры	
000	[bx+si+инд. смещение]	000	ax	al
001	[bx+di+инд. смещение]	001	cx	cl
010	[bp+si+инд. смещение]	010	dx	dl
011	[bp+di+инд. смещение]	011	bx	bl
100	[si+инд. смещение]	100	sp	ah
101	[di+инд. смещение]	101	bp	ch
110	[bp+инд. смещение]	110	si	dh
111	[bx+инд. смещение]	111	di	bh

Особые случаи при кодировании байта $\text{mod } r/m$

- При регистровой адресации поле $\text{mod} = 11$, а в поле r/m - код второго регистра.
- Для прямой адресации поле $\text{mod} = 00$, $r/m=110$ и за байтом $\text{mod } r/m$ стоят 2 байта, указывающих смещение прямого адреса.

Область действия команды mov

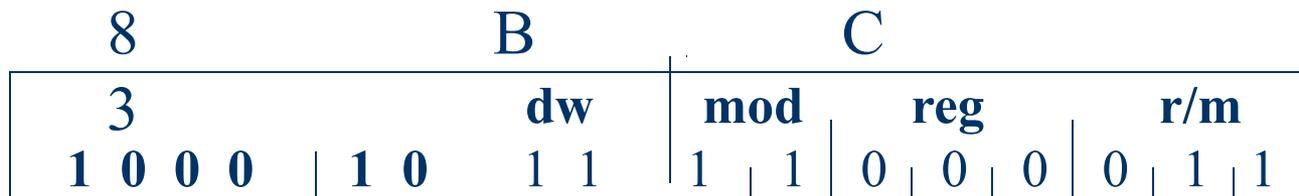


```
mov ax, @data  
mov ds, ax
```

Форматы машинных команд

1. Пересылка регистра/памяти в/из регистра:

`mov ax,bx`



2 байта



Параметры **d** и **w**

- **d** определяет направление перемещения:

$$d = \begin{cases} 0 & \text{– из регистра,} \\ 1 & \text{– в регистр.} \end{cases}$$

- **w** определяет размер операнда:

$$w = \begin{cases} 0 & \text{– 1 байт,} \\ 1 & \text{– 2 байта.} \end{cases}$$

Лекция №7

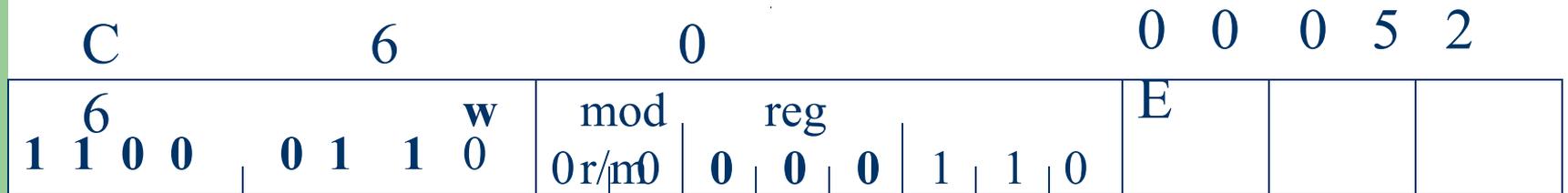
- Форматы команды MOV
 - Префикс замены сегмента.
 - Система машинных команд МП intel 8086 (п.1-2).
- 

Форматы машинных команд

2. Непосредственный операнд в регистр/память

`mov ABC,46`

Пусть смещение ABC равно 0005



Длина команды 5 байт

Смещение
ABC

(46)₁₀

Форматы машинных команд

3. Непосредственный операнд в регистр
`mov si,1000` (1000=03E8h)



Длина 3 байта

|
s
i

Форматы машинных команд

4. Память в регистр

`mov ax,exword` ; если адрес `exword = ds:0002`, то

A 1 02 00

1 0 1 0	0 0 0 1	адрес младший	адрес старший
---------	---------	------------------	------------------

Длина 3 байта

Форматы машинных команд

5. Аккумулятор в память

`mov exbyte,al ; если адрес exbyte = ds:0004, то`

A 2 04 00

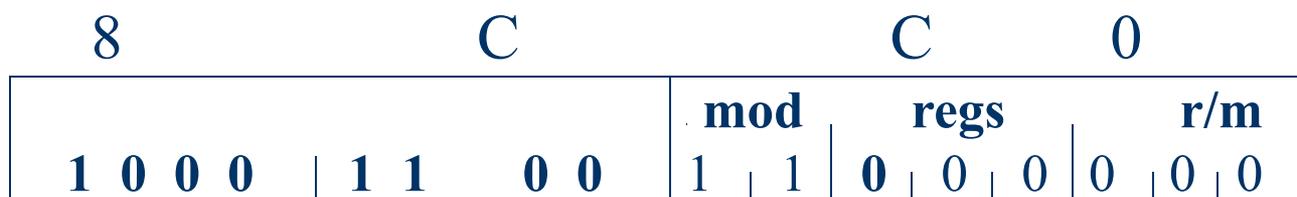
1 0 1 0	0 0 1	w	адрес младший	адрес старший
		0		

3 байта

Форматы машинных команд

7. Сегментный регистр в регистр/память:

`mov ax,es`



2 байта

regs	описание
00	es
01	cs
10	ss
11	ds

Префикс замены сегмента

Префикс занимает 1 байт и имеет вид:

001 regs 110

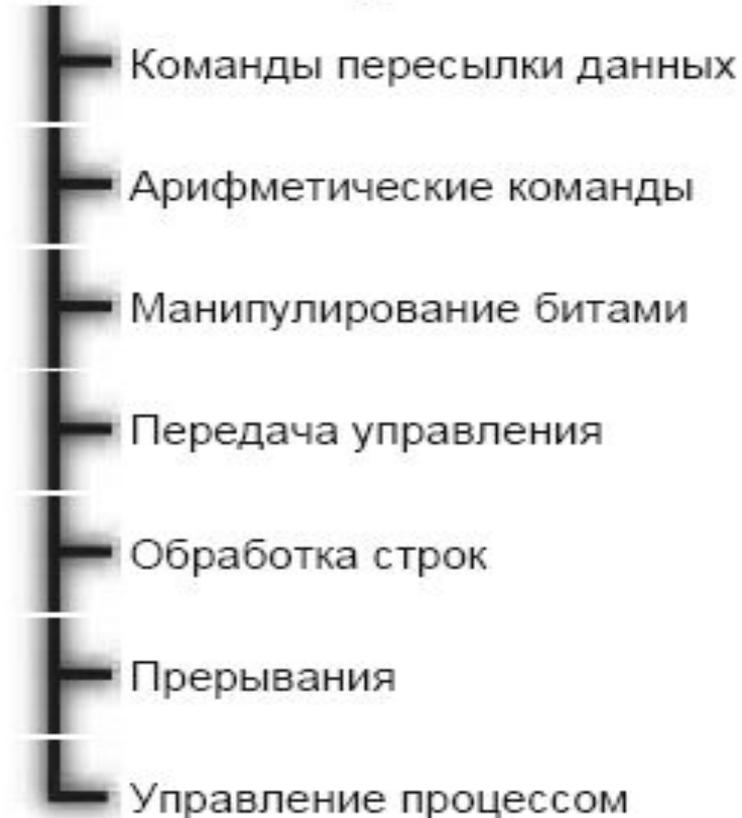
Кодировка сегментных регистров:

regs	описание
00	es
01	cs
10	ss
11	ds

Пример: переменная `var` определена в сегменте `cs`, тогда
в исходном модуле - **inc cs:var**,
под отладчиком - **cs: inc var**

Система команд микропроцессора

Машинные команды



1. Команды общего назначения

Выполняют обмен информации между регистрами, ячейками памяти и портами ввода-вывода:

а) команды пересылки данных (mov, lea, xchg);

пример:

```
mov bx,offset abc
```

```
lea bx,abc
```

```
xchg ax,bx
```

б) стековые операции (push, pop, pushf, popf).

Пример: `push ds`

```
pop es
```

СТЕК

Стек – это область оперативной памяти, специально выделенная для временного хранения данных программы.

Работает по принципу LIFO в сторону уменьшения адресов.

Используются 3 регистра:

ss - сегментный регистр,

sp – указатель стека,

bp – указатель базы кадра стека.

Организация стека в оперативной памяти

----- 0000:0000

.....

----- Сегмент стека SS:0000

Вершина стека SS:SP

Дно стека SS:FFFF

.....

Работа стековых команд

push – запись значения в вершину стека:

1. **sp=(sp)-2;**
2. Запись операнда по адресу **ss:sp**.

pop – извлечение из вершины стека:

1. Извлечение операнда по адресу **ss:sp**
2. **sp=(sp)+2;**
3. Запись операнда.

pusha – групповая запись в стек регистров
ax, cx, dx, bx, sp, bp, si, di

popa – групповое извлечение из стека в регистры

Работа стековых команд с регистром флагов

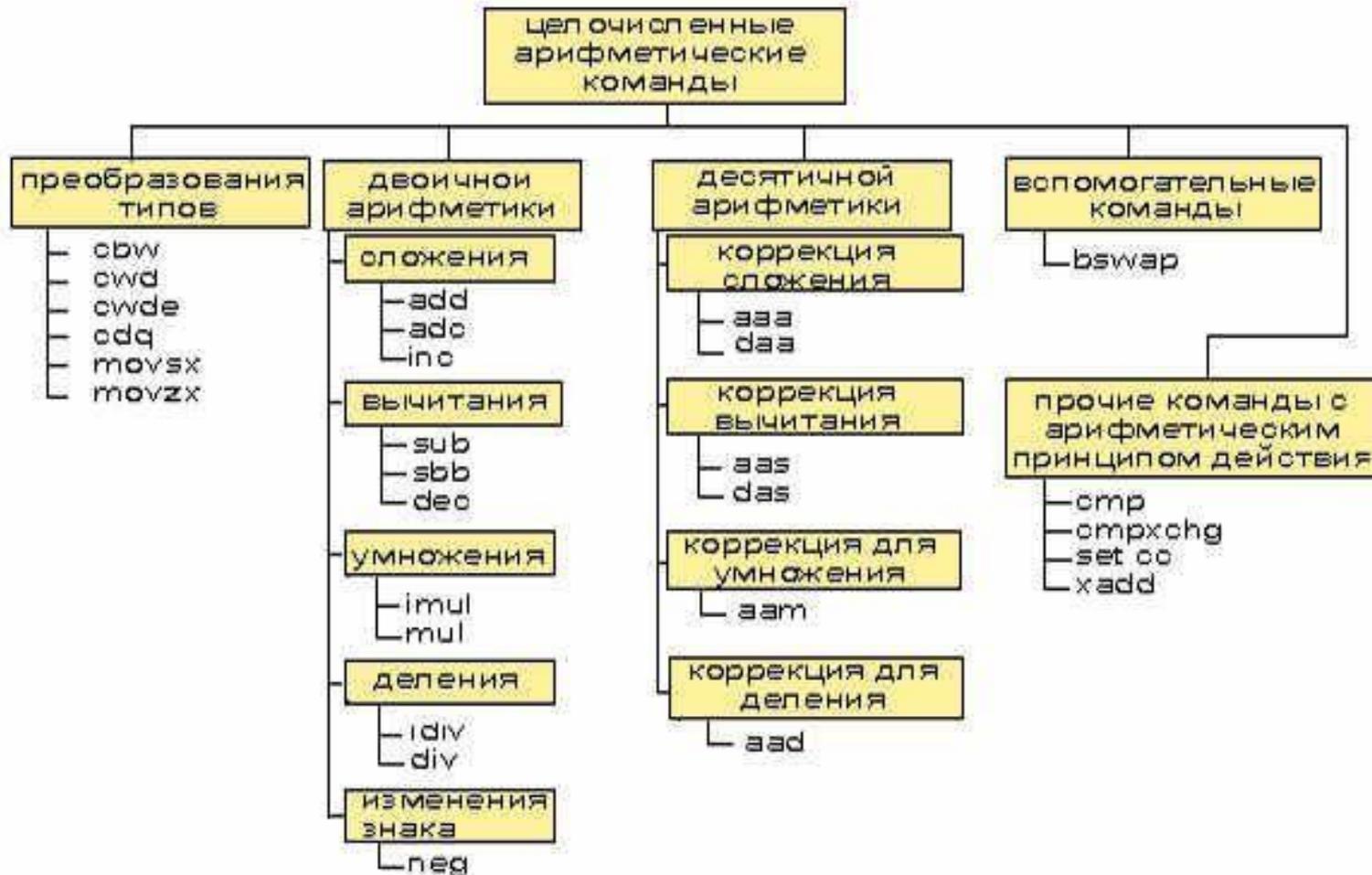
pushf – загрузка содержимого регистра флагов в вершину стека:

1. **sp=(sp)-2;**
2. Запись содержимого регистра флагов по адресу **ss:sp**.

popf – извлечение информации из вершины стека и загрузка в регистр флагов:

1. извлечение операнда по адресу **ss:sp**;
2. **sp=(sp)+2;**
3. загрузка в регистр флагов.

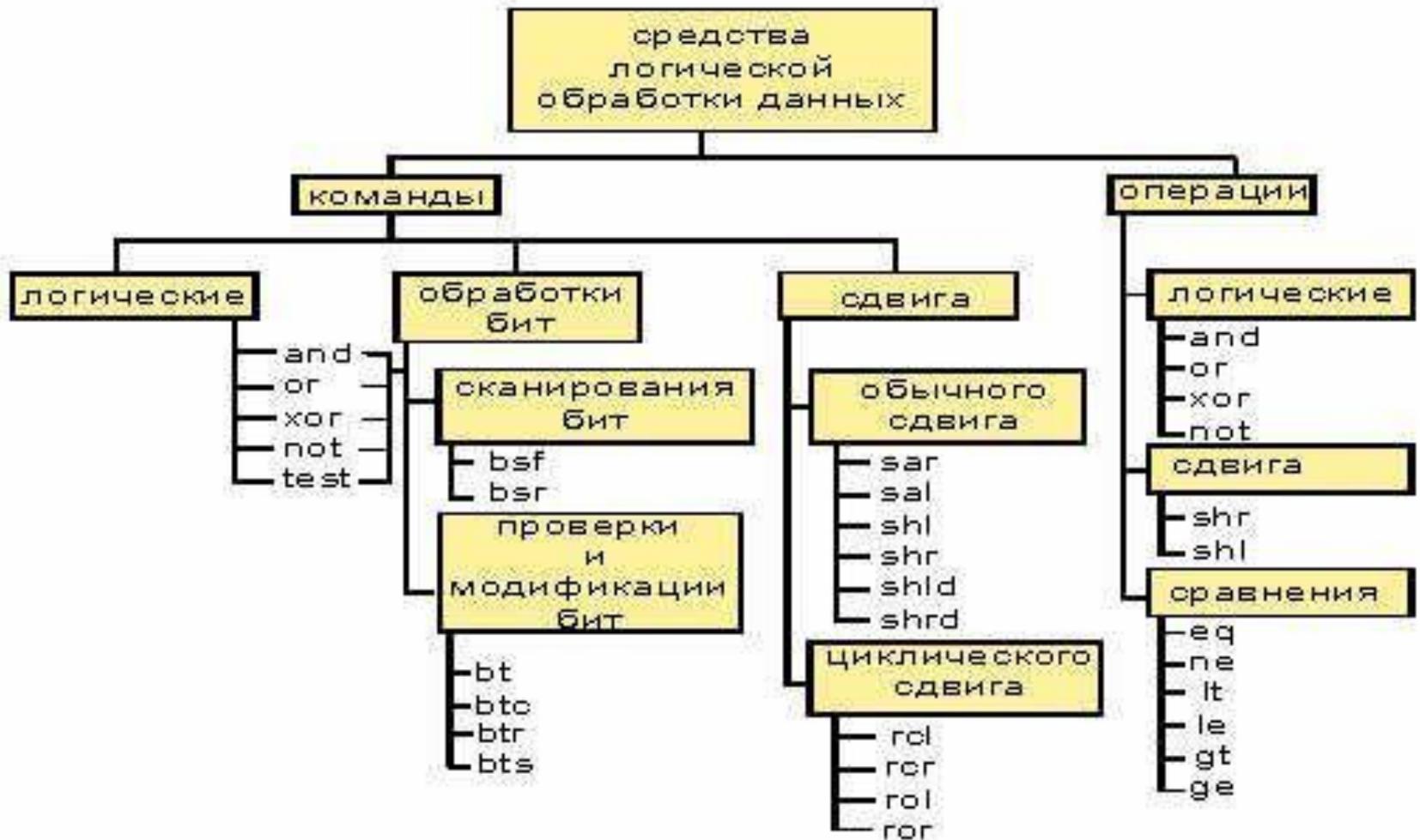
2. Арифметические операции над целыми двоичными числами



Лекция №8

- Система команд процессора (п.3-7).
 - Прерывания.
 - Классификация прерываний.
- 

3. Команды манипулирования битами



4. Команды передачи управления

Типы адресов: short, near, far

Существует 4 способа передачи управления.

Команды перехода

Длина смещения

- безусловного перехода: `jmp` 2, 4 байта
- условного перехода: `j xxx` 1 байт
- перехода с вызовом процедур: `call` 2, 4 байта
- управления циклами: `loop` 1 байт

Пример: `jmp short label`

Замена длины смещения в условном переходе

Если относительный адрес перехода превышает 128 б, то меняют команду (а) на две команды (в):

а)

```
    cmp dl,'a'  
    jz zero
```

continue:

.....

zero:

.....

в)

```
    cmp dl,'a'  
    jnz continue
```

```
    jmp zero
```

Continue:

.....

Zero:

Директива JUMPS (nojumps)

Замену команд для условного перехода можно сделать с помощью директивы транслятора jumps.

Тогда транслятор автоматически заменит условный переход на пару переходов.

Команда цикла LOOP

Использует значение регистра CX, уменьшает его при каждом шаге на 1 и проверяет на 0.

Если не равно 0, то переход по адресу операнда.

```
    mov cx,10
    mov ax,0
abc:  inc ax
      .....
      loop abc
ddd:  .....
```

5. Команды обработки строк

Перемещают, сравнивают, сканируют строки данных.

Работают с последовательностями элементов размером в байт, слово, двойное слово.

Используются с префиксом повторения (**rep**).

Например, команда **movs**:

movsb

movsw

movsd

Алгоритм работы команд обработки строк

1. Установить флаг **df** командами **cld** (по возрастанию) или **std** (по убыванию);
2. Загрузить адреса цепочек в **ds:si** и **es:di**;
3. Загрузить в **cx** количество элементов для обработки;
4. Выполнить команду (например, **movsw**) с префиксом **rep**:
rep movsw

6. Команды прерывания

Для обработки специфической ситуации существует 3 команды:

int, iret, into

7. Команды управления процессом

Назначение: установка и сброс флагов, изменение режима функционирования процессора.

Например,

cld – сбросить флаг направления (флаг=0),

std – установить флаг направления (флаг=1).

Прерывание. Основные понятия

1. **Прерывание** – это сигнал, заставляющий микропроцессор менять обычный порядок исполнения команд.
2. **Прерывание** называется ситуация, приводящая к временному или окончательному прекращению выполнения команд одной программы и переходу к другой программе.

Назначение

Механизм прерываний обеспечивает эффективное взаимодействие устройств ввода-вывода с микропроцессором.

Обработка прерываний – это прерогатива программирования на ассемблере.

Микропроцессор может распознать **256 типов** прерываний.

Для каждого типа разработана своя **программа обработки**, называемая **обработчиком прерываний**.

Вектора прерываний

Адрес программы обработки прерывания конкретного типа называется вектором прерываний. Размер – 4 байта: сегмент: смещение

Все векторы собраны в таблицу векторов прерываний.

Размер таблицы $4 \cdot 256 = 1024$ байта.

Расположена в младших адресах памяти.

Команды прерываний

2 команды вызова прерываний:

int тип_прерывания

into (прерывание по переполнению).

1 команда возврата - **iret**

Лекция № 9

- Схема обработки прерываний.
 - Функции `int 21h` для работы с файлами.
 - Примеры использования команды `int 21h` для работы с файлами.
- 

Классификация прерываний

Прерывания

Прерывания BIOS

Программные прерывания BIOS

Аппаратные прерывания BIOS

Прерывания DOS

Программные прерывания DOS

Аппаратные прерывания DOS

Классификация прерываний

По месту возникновения:

внешние (аппаратные),
внутренние (программные).

По типу системных ресурсов:

BIOS,
DOS.

Прерывания BIOS (тип 0 -1f)

- Векторы прерываний микропроцессора (деление на 0, переполнение);
- Векторы прерываний микроконтроллера (системный таймер, клавиатура, гибкий диск);
- Входные точки процедур системы BIOS (обмен данными с клавиатурой, дисплеем, ...);
- Вызов процедур пользователя;
- Указатели системных таблиц (параметры гибкого и жесткого дисков).

Прерывания DOS (типы с 20h)

- 20h – завершение программ;
- 21h- вызов функций DOS;
- 23h- обработка клавиш Ctrl+ Break;
- 25h- абсолютное чтение с диска;
- 26h- абсолютная запись на диск.

INT 21h – вызов функций DOS

Номер функции задают в регистре **ah**, дополнительную информацию через другие регистры, например, **ds:dx**.

Выходная информация выдается через регистр **al** или другие регистры.

Функции `int 21h` для работы с файлами

`3ch` - Создание файла;

`6ch` - Создание и открытие существующего;

`3dh` - открытие;

`3eh` - закрытие;

`3fh` – чтение файла;

`40h` - запись в файл;

`42h` – позиционирование указателя записи.

Создание файла через функцию `3ch`

Входные данные:

`ah` – `3ch`,

`ds:dx` - адрес ASCIIZ-строки с именем файла,
`cx` - атрибут файла: 0 – обычный.

Выходные данные:

если `cf=0`, то в `ax` - логический номер файла
(дескриптор файла),

если `cf=1`, то в `ax` - код ошибки:

3 – нет указанного пути,

4 – нет свободного дескриптора, 5 – отказано в доступе.

Обработка флага cf

Функция **3ch** на выходе передает код ошибки при **cf=1**.

Для обработки этой ситуации используются команды:

jc – переход при **cf=1**;

jnc - переход при **cf=0**.

Атрибут файла

Задается в регистре побитно:

0 бит = 1 – только для чтения,

0 бит = 0 – обычный;

1 бит = 1 – скрытый файл;

2 бит = 1 – системный файл;

и т.д.

Лекция №10

- Примеры работы с прерываниями.
 - Структура программного сегмента.
 - Префикс программного сегмента (PSP).
 - Буфер обмена DTA.
- 

Пример1: использования команды `int 21h` для создания файла через функцию `3ch`

```
.data
  handle dw 0
  filename db 'myfile.txt',0
  point_fname dd filename
.code
  .....
  xor cx,cx          ; обнуляем cx
  lds dx,point_fname ; формируем указатель на файл
  mov ah,3ch
  int 21h
  jc exit           ; переход по ошибке
  mov handle,ax
  .....
exit:
```

Открытие существующего файла через функцию `3dh`

Входные данные:

`ds:dx` - адрес ASCIIZ-строки с именем файла,

`al = 0` для чтения,

`al = 1` для записи,

`al = 2` для чтения и записи.

Выходные данные:

`ax` - логический номер файла.

Открытие существующего и создание нового файла через функцию bsh

Входные данные:

сх - атрибут файла, 0- для обычного файла,
bh – режим доступа, 2 – чтение-запись;
dh – **1** - для существующего файла открыть доступ, **10h** – создать и открыть;
ds:si - адрес ASCIIZ-строки с именем файла.

Выходные данные:

ah - логический номер файла.

Заккрытие файла

Входные данные:

АН -3Еh,

ВХ - логический номер файла.

Чтение из или запись в файл

Входные данные:

ah=3Fh - для чтения, **40h** - для записи в файл,

bx - логический номер файла,

cx - число считываемых байтов,

ds:dx - адрес буфера ввода-вывода.

Выходные данные:

ax - число фактически считанных (или записанных) байтов.

Пример2: использования команды int 21h для создания файла через bch

```
.data
handle dw 0
filename db 'myfile.txt', 0
point_fname dd filename
.code
.....
xor cx,cx                ; атрибут файла
    mov bx,2              ; режим доступа: чт-запись
    mov dx,1              ; если суц-ет, то открыть
lds si,point_fname      ; формируем указатель на файл
mov ah, 6ch
int 21h
jnc ABC                  ; переход, если существует
```

Пример2- продолжение

mov dx,10h ; создать и открыть

mov ah,6ch

int 21h

jc exit ; переход по ошибке

ABC: mov handle, ax

.....

exit: ; обработка ошибки

Пример 3: вывод строки на экран

```
.data
filename db 'ВЫВОД строки',13,10,$
.....
.code
    mov ax,@data
        mov ds,ax
        mov dx, offset filename
    mov ah, 9h
    int 21h
```

Чтение системной даты

Функция **2ah** засылается в **ah**.

Выходные данные:

cx – год в формате типа 2013,

dh – месяц,

dl – день.

Структура программного сегмента

Область памяти, начинающаяся с минимального адреса загрузки программы пользователя, называется **программным сегментом**.

Этот адрес определяется при вызове программы, т.е. при выполнении операции OS - **EXEC**.

Программный сегмент состоит из:

- **префикса программного сегмента (PSP)**, расположенного по смещению 0000;
- **тела загрузочного модуля** (по смещению 0100h).

Структура PSP

смещение	длина	пояснение
+0	2	int 20h (cd20h)
+2	2	вершина доступной памяти (параграф)
+6	4	размер прогр -ного сегмента в байтах
0Ah	4	} область сохранения векторов } прерываний 22h, 23h, 24h типов
0Eh	4	
12h	4	
+2Ch	2	адрес среды DOS (номер параграфа)
+50h	5	команда вызова диспетчера DOS
+80h	1	длина командной строки
+81h	7Fh	буфер ДТА , который содержит командную строку

Пояснения к таблице

Прерывания :

22h – завершение процесса;

23h – нажатие клавиш Ctrl+Break;

24h – фатальная ошибка.

Область сохранения вектора имеет вид:

2 байта – смещение, 2 байта – сегмент.

Адрес среды - занимает 2 байта , задается сегментным адресом.

Вызов диспетчера функций DOS (5 байт):

call – 1 б, адрес обработчика - 4 б (смещение, сегмент).

Номер функции выбирается из регистра ah.

Лекция №11

- Буфер обмена ДТА.
 - Системное окружение.
 - Структура dos и карта памяти.
- 

Буфер DTA

DTA (Disk Transfer Area) – рабочий буфер обмена с диском.

Содержит **символы командной строки** после имени программы, включая все пробелы, разделители и код **0dh**.

Пример вызова программы в командной строке:
abc.exe bbbbb 5

Содержимое dta: **20424242424220350d**

При нажатии клавиши **Enter** в DTA записывается код **0dh**.

Максимальная длина командной строки 140 байт.

Среда dos или системное окружение (environment)

Передаваемая среда является копией родительского процесса.

Представляет собой последовательность строк ASCIIZ в виде:

параметр=значение0

Общая длина строк – до 32 кб, по умолчанию -512 б.

Пример окружения

Строки среды		в мнемонике ассемблера
-----		-----
Имя_1=значение_1<00>		db 'comspec=c:\command.com',0
Имя_2=значение_2<00>		db 'prompt=\$p\$g",0
.....	
Имя_n=значение_n<00>		db 'path=d:\;c:\apps',0
<00>		db 0
<0100>		dw 1
_имя_программы<00>		db 'c:\abc\abc.exe',0
<00>		db 0

Основные смещения в PSP для программирования

- 2ch – адрес среды;
- 80h – длина рабочего буфера;
- 81h – начальное смещения для содержимого командной строки.

Способы завершения программы

- `int 20h`;
- переход по адресу 0000 в программном сегменте;
- `int 21h` с `ah=4ch`;
- Косвенный переход по адресу 0050 в PSP.

Загрузка регистров ехе- файла

- **DS, ES** указывают на начало PSP;
- **IP, SP** получают значения, указанные при редактировании программы в заголовке загрузочного модуля;
- **CS, SS**, получают значения из заголовка загрузочного модуля, модифицированные на адрес начала программного сегмента.

Загрузка регистров com- файла

- **DS, ES, CS, SS** указывают на начало PSP;
- **IP** равен 0100h;
- **SP** указывает на конец программного сегмента, длина сегмента в ячейке 6 PSP уменьшается, чтобы освободить пространство для стека;
- в вершину стека записывается нулевое слово.

Структура dos

1. Блок начальной загрузки (**boot record**);
2. Интерфейс с BIOS;
3. Встроенные команды dos;
4. Командный процессор (command.com).

Структура dos - 1) блок начальной загрузки

Блок начальной загрузки занимает 1 сектор.

Размещается:

на дискете - на нулевом треке;

на диске – в первом секторе раздела dos.

Структура dos – 2) интерфейс с bios

Содержит команды взаимодействия с bios.
Обеспечивает интерфейс низкого уровня с подпрограммами работы с устройствами через bios.

Структура dos –

3) встроенные команды dos

Команды dos обеспечивают пользовательским программам интерфейс высокого уровня.

Включают операции :

- управления файлами;
- распределения оперативной памяти;
- блочного обмена с дисками;
- управления двигателем нгмд и др.

Командный процессор

Состоит из трех частей:

1. **резидентной;**
 2. **инициализации;**
 3. **нерезидентной.**
- (1) содержит подпрограммы обработки прерываний (22h, 23h, 24h) и подзагрузки нерезидентной части.
- (2) включает обработку файла autoexec.bat и определяет начальный адрес загрузки программы пользователя.

3. Нерезидентная часть командного процессора

Состоит из:

- интерпретатора команд;
- системного загрузчика нерезидентных команд и программ.

Загрузчик вызывается операцией OS **exec** - вызов и загрузка программ.

Карта распределения памяти в dos

- 0000:0000 таблица векторов прерываний;
- 0040:0000 глобальные переменные BIOS;
- 0050:0000 глобальные переменные DOS;
- хххх:0000 BIO.COM;
- хххх:0000 DOS.COM;
- хххх:0000 2 части command.com;
- хххх:0000 резидентные программы;
- хххх:0000 нерезидентные программы и команды;
- хххх:0000 нерезидентная часть command.com

Лекция №12

- Операторы ассемблера;
 - директивы данных;
 - варианты вызова процедур.
 - директивы управления листингом.
- 

Операторы программы на ассемблере

[метка] мнемоника [операнды]
 (код операции)

Оператор ассемблера может быть:

- машинной командой;
- псевдооператором или директивой транслятора;
- макрокомандой;
- комментарием.

Директивы (псевдооператоры) транслятора ассемблера TASM

[метка] мнемоника [операнды]
 (код операции)

Псевдооператоры (директивы ассемблера)
подразделяются на 2 класса:

1. директивы данных;
2. директивы управления листингом.

Директивы данных

1. Определение идентификаторов:

имя `equ` выражение

Пример: `count equ cx`

`N equ 1024`

2. Определение данных:

резервируют память - `db` (1 байт), `dw` (2 байта),

`dd` (4 байта), `df`, `dp` (6 байтов),

`dq` (8 байтов), `dt` (10 байтов)

Директивы данных.

3. Определение сегмента

Директивы **segment** и **ends** – определяют в программе начало и конец сегмента.

Формат директив:

имя **segment** [тип подгонки,] [тип связи,] ‘класс’,
длина адреса

Например, ABC segment para public ‘code’,use16

.....

ABC ends

Определение сегментного регистра

Регистр адресации задается директивой **assume**.

Формат:

assume сегментный_регистр:имя_сегмента [,]

Пример: `mycode segment para public 'code'`
 `assume cs:mycode,ds:mydata`
 `begin: mov ax,seg mydata`
 `mov ds,ax`
 `.....`
 `mycode ends`

Директивы данных.

4. Определения процедур

Формат: имя **proc** [атрибут дальности]

.....

ret

имя **endp**

Атрибут дальности: **near** или **far**

Пример: `.code`
`startABC proc near`

.....

`ret`

`startABC endp`

Механизм вызова – сохранение контекста программы в стеке.

Контекст программы – это информация о состоянии программы в точке вызова процедуры.

Вызов процедур

(call имя_процедуры)

Пусть ABC1 - имя процедуры типа near,

ABC2 – имя процедуры типа far, adr1, adr1 – ссылки:

adr1 dw offset ABC1

adr2 dw offset ABC2,seg ABC2

Варианты вызова процедур:

- Прямая адресация в сегменте (ближний вызов): call near
ptr ABC1
- Прямая адресация между сегментами (дальний вызов):
call far ptr ABC2
- Косвенная адресация в сегменте:
call word ptr adr1
- Косвенная адресация между сегментами:
call dword ptr adr2

Директивы данных.

5. Определение внешних ссылок

public имя – делает указанное имя доступным для других программных модулей, которые впоследствии могут загружаться вместе с данным модулем.

extrn имя: тип - описывает идентификатор, определенный в другом модуле и описанный там в операторе public.

Тип для данных: byte, word, dword

Тип для процедур: near, far

include имяфайла - вставка в текущий файл текста из файла

Пример связи модулей по данным

1 модуль:

```
public ABC
```

```
ABC dw 1234h
```

2 модуль:

```
extrn ABC:word
```

```
.....
```

```
mov ax,ABC
```

Директивы данных.

6. Директива управления трансляцией

end имя_программы

Директивы управления листингом

page [число строк] [число столбцов]

10-255, **57** 60-132, **80**

title текст1

subttl текст2

Лекция №13

- Возможности макросов.
 - Основные понятия.
 - Классификация директив макросов.
- 

Возможности макросредств

Исходный
модуль
test.asm

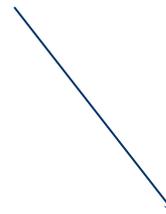
A
B
C
ABC
D

Макроопределение
ABC
в test.asm

E
F

Макрорасширение
в test.asm

A
B
C
E
F
D



Возможности макросредств

Исходный
модуль
test.asm

A
B
Select
0
C
D
Select
1

Макроопределение
Select x
в test.asm

E
if x =
0
F
end
if x =
1
G
end

Макрорасширение

в test.asm

A
B
E
F
C
D
E
G

Основные понятия макросов

Поименованный набор операторов ассемблера называется **макрокомандой**.

Группа команд, определяющая действие макрокоманды, называется **макроопределением**.

Процесс включения в исходный текст программы команд из макроопределения называется **макрорасширением** или **макрогенерацией**.

Этапы использования макросов

1. Определение макрокоманды.
2. Вызов макрокоманды.

Определение макрокоманды

Формат:

имя MACRO список_формальных_параметров

тело макроопределения

ENDM

Список_формальных_параметров

состоит из элементов вида:

имя_формального_аргумента [тип]

где тип:

- **:REQ** – для параметров, задаваемых явно;
- **=<строка>** для стандартного значения параметра.

Примеры прототипа макроса

1. ADD_Word MACRO OP1:REQ,OP2:**REQ**,SUM

2. ADD_Word MACRO OP1,OP2,SUM=dx

Вызов макрокоманды

Вызов макрокоманды осуществляется **по имени** макроса с указанием **списка фактических параметров**.

Процесс замены формального параметра соответствующим фактическим значением при макрорасширении называется **подстановкой аргументов**.

Пример: сложение двух значений размером в слово

Определение:

```
ADD_Word MACRO  OP1,OP2,SUM
             MOV   AX,OP1
             ADD   AX,OP2
             MOV   SUM,AX
             ENDM
```

Макрокоманда:

```
ADD_Word BX,CX,DX
```

Макрорасширение:

```
MOV AX,BX
ADD AX,CX
MOV DX,AX
```

Продолжение примера

Второй вариант макрокоманды:

```
ADD_Word price, tax, cost
```

Макрорасширение:

```
MOV ax, price
```

```
ADD ax, tax
```

```
MOV cost, ax
```

Классификация директив макросов

1) директивы общего назначения:

MACRO, ENDM, LOCAL

2) директивы повторения: **REPT, IRP, IRPC**

3) условные: **IF, IFDEF, IFDIF, IFIDN, IFB, IFNB**

4) выхода: **EXITM**

5) управления листингом: **LALL, SALL, XALL**

1. Директивы общего назначения

```
aaa macro bbb  
    local next  
    push cx  
    mov cx,bbb
```

```
..... ; группа команд
```

```
next: loop next  
    pop cx  
    endm
```

Вызов макроса

Макрокоманда:

aaa 100

Макрорасширение:

push cx

mov cx, **100**

..... ; группа команд

??0001: loop ??0001

pop cx

Лекция №14

- Директивы повторения.
 - Условные директивы.
 - Макрооперации.
- 

2. Директивы повторения

REPT выражение

.....

ENDM

Пример: зарезервировать L байтов и присвоить им знач-я от 1 до L.

```
ALLOCATE MACRO TLABEL, L  
    TLABEL EQU THIS BYTE  
    VALUE = 0  
    REPT L  
        VALUE = VALUE+1  
        DB VALUE  
    ENDM
```

ENDM

Использование макрокоманды allocate

Вызов:

```
.data
```

```
Allocate table,40
```

Макрорасширение в точке вызова:

```
table EQU THIS BYTE
```

```
VALUE = 0
```

```
VALUE = VALUE+1
```

```
DB 1
```

```
VALUE = VALUE+1
```

```
DB 2
```

```
.....  
VALUE = VALUE+1
```

```
DB 40
```

Директивы повторения

IRP параметр ,<список аргументов>

.....

ENDM

Пример создания таблицы из четырех слов:

.data

IRP ABC ,<1,2,3,4>

DW ABC *ABC

ENDM

Расширение макроса для создания таблицы из четырех слов

```
.data
```

```
DW 1 *1
```

```
DW 2 *2
```

```
DW 3 *3
```

```
DW 4 *4
```

Директивы повторения

IRPC параметр, <строка>

Пример: IRPC RG, <ABCD>

PUSH RG&X

ENDM

Макрорасширение : PUSH AX

PUSH BX

PUSH CX

PUSH DX

3. Условные директивы

IF задает начало условно ассемблируемого блока , если <выражение> **истинно** или имеет **ненулевое значение**.

Формат:

IF <выражение>

.....

ENDIF

В выражении можно использовать операции:

EQ, NE, GT, GE, LT, LE

Примеры вызова процедуры ReadBuf при DoBuf \neq 0

1. Без расширения оператора условия

```
.data
```

```
BufNum dw 5
```

```
DoBuf db 0
```

```
.code
```

```
.....
```

```
if DoBuf
```

```
mov ax, BufNum
```

```
call ReadBuf
```

```
endif
```

```
.....
```

Примеры вызова процедуры ReadBuf при DoBuf \neq 0

2. С расширением условного оператора

BufNum dw 5

DoBuf db 1

.code

.....

if DoBuf

mov ax, BufNum

call ReadBuf

endif

.....

расширяется в:

mov ax, 5

call ReadBuf

Условные директивы IFB , IFNB

Операторы альтернативной обработки пустых операторов:

IFB <параметр>

IFNB <параметр>

Параметр всегда задается в угловых скобках и определяет имя формального аргумента макроса.

Примеры условных директив

Пример макроопределения:

```
PRINT_T MACRO MSG
```

```
    IFB <MSG>
```

```
        MOV SI, DEFMSG
```

```
    ELSE
```

```
        MOV SI, MSG
```

```
    ENDIF
```

```
        CALL SHOW_T
```

```
ENDM
```

.....

```
DEFMSG db 'no'
```

макрокоманда:

```
PRINT_T 'y'
```

макрорасширение:

```
MOV SI, 'y'
```

```
SHOW_T
```

макрокоманда:

```
PRINT_T
```

макрорасширение:

```
MOV SI, 'no'
```

```
SHOW_T
```

Пример извлечения параметров из стека

```
POPREGS MACRO REG1, REG2
    IFNB <REG1>
        POP REG1
    ENDIF
    IFNB <REG2>
        POP REG2
    ENDIF
ENDM
```

Вызов и расширение:

1) POPREGS ax

POP ax

2) POPREGS ,bx

POP bx

3) POPREGS ax, bx

POP ax

POP bx

Условные директивы IF1, IF2, IFDEF

IF1

IF2

Пример: IF1 INCLUDE TEXTMACRO.TXT

Ассемблирование, если символическое имя
определено:

IFDEF символическое имя

Пример: IFDEF **SIZE**

BUF db **SIZE** DUP(?)

endif

Условные директивы IFDIF, IFIDN

Ассемблирование, если параметры различны:

IFDIF <параметр1><параметр2>

Ассемблирование, если параметры
тождественны:

IFIDN <параметр1><параметр2>

Макрооперации

1. & - операция замещения
2. ;; - подавление комментария
3. % - вычисление выражения
4. ! - операция literalного ввода символа
5. <> - операция literalного ввода строки

Макрооперация замещения

1. **Операция замещения**

Формат: & имя параметра

Пример:

```
makemsg MACRO str, n  
    msg&n db '&str'  
endm
```

Вызов макроса:

```
makemsg <Введите значение:>,5
```

Расширение:

```
msg5 db 'Введите значение:'
```

Макрооперации

3. Вычисление выражения

Формат: %выражение

Пример: makemsg <строка>,%3+5

4. Операция литерального ввода символа

Формат: !символ

Пример: makemsg <нельзя вводить число!>100>,3

5. Операция литерального ввода строки

Формат: <строка>

Лекция №15

- Директивы управления листингом.
 - Упрощенные директивы TASM.
 - Модели памяти.
 - Этапы разработки программы.
 - Отладчик Turbo Debugger (td).
- 

Директивы управления листингом

- lall;
- xall;
- sall.

Упрощенные директивы `iasm`

model [модификатор] **модель памяти** [имя кодового сегмента]

Модификатор: `use16`, `use32`, `dos`

`iasm` создает идентификаторы: `@code`, `@data`, `@stack`,

Упрощенные директивы определяют сегменты:

`.code` - кода

`.stack` - стека

`.data` - инициированных данных типа `near`

`.far data` - инициированных данных типа `far`

`.const` – постоянных данных

`.data?` - неинициированных данных типа `near`

Модели памяти

- tiny
- small
- medium
- compact
- large
- huge

Сегменты для модели памяти

модель	Кол-во код. сегментов	Кол-во сег. данных	Ссылки по управл.	Ссылки на данные
tiny	1	-	near	near
small	1	1	near	near
medium	n	1	far	near
compact	1	n	near	far
large	n	n	far	far
Huge (32p)	n	n	far	far
Flat (win32)	1	-	near	near

Порядок выполнения программы на ассемблере

- **tasm** имя_исх._модуля [, имя_объектного_модуля] [, имя_lst] [, имя_crf] [опции]

tasm имя , , ,

tasm имя /l /c

- **tlink** список объектных модулей [, имя_exe_файла] [, имя_map_файла] [, список lib_файлов] [опции]

- **debug** или **td**

Для **td**:

tasm /zi имя_исходного модуля

tlink /v имя_объектного модуля

Этапы разработки программы на ассемблере.

1. Постановка и формулировка задачи

- Назначение и требования к программе;
- представление исходных данных и результатов;
- структура входных и выходных данных;
- ограничения и допущения на исходные и выходные данные.

Этапы разработки программы на ассемблере. 2.

Проектирование

- формулировка ассемблерной модели задачи;
- выбор метода реализации;
- разработка алгоритма реализации;
- разработка структуры программы в соответствии с моделью памяти.

Этапы разработки программы на ассемблере.

3. Кодирование

- уточнение структуры данных и определение ассемблерного представления формата;
- программирование;
- комментирование текста программы и составление предварительного описания программы.

Этапы разработки программы на ассемблере. 4. **Отладка и тестирование**

- составление тестов для проверки правильности работы программы;
- обнаружение, локализация и устранение ошибок в программе, выявленных в тестах;
- корректировка кода программы и описания.

Этапы разработки программы на ассемблере. 5. **Эксплуатация и сопровождение**

- настройка программы на конкретные условия использования;
- обучение пользователей работе с программой;
- сбор сведений о сбоях в работе программы;
- модификация программы.

Отладчик Turbo Debugger (td)

Для работы в Turbo Debugger (td)

необходимо создать загрузочный модуль:

tasm /zi имя

tlink /v имя

Управление работой в отладчике ведется посредством меню двух типов:

- глобальное (вызов по F10);
- локальное (вызов по Alt-F10 или правой кнопкой мыши).

Запуск программы на выполнение

Используется один из четырех режимов:

- безусловное выполнение (F9);
- по шагам:
 - а) F7: Run|Trace into - с пошаговым выполнением процедур и прерываний;
 - б) F8: Run|Trace over - процедуры и прерывания выполняются как одна команда;
- до текущего положения курсора (F4);
- с установкой точек прерывания (breakpoints).

Установка точек прерывания

Сначала устанавливаются точки прерывания курсором и F2, затем – F9.

Прервать выполнение программы – Ctrl+F2

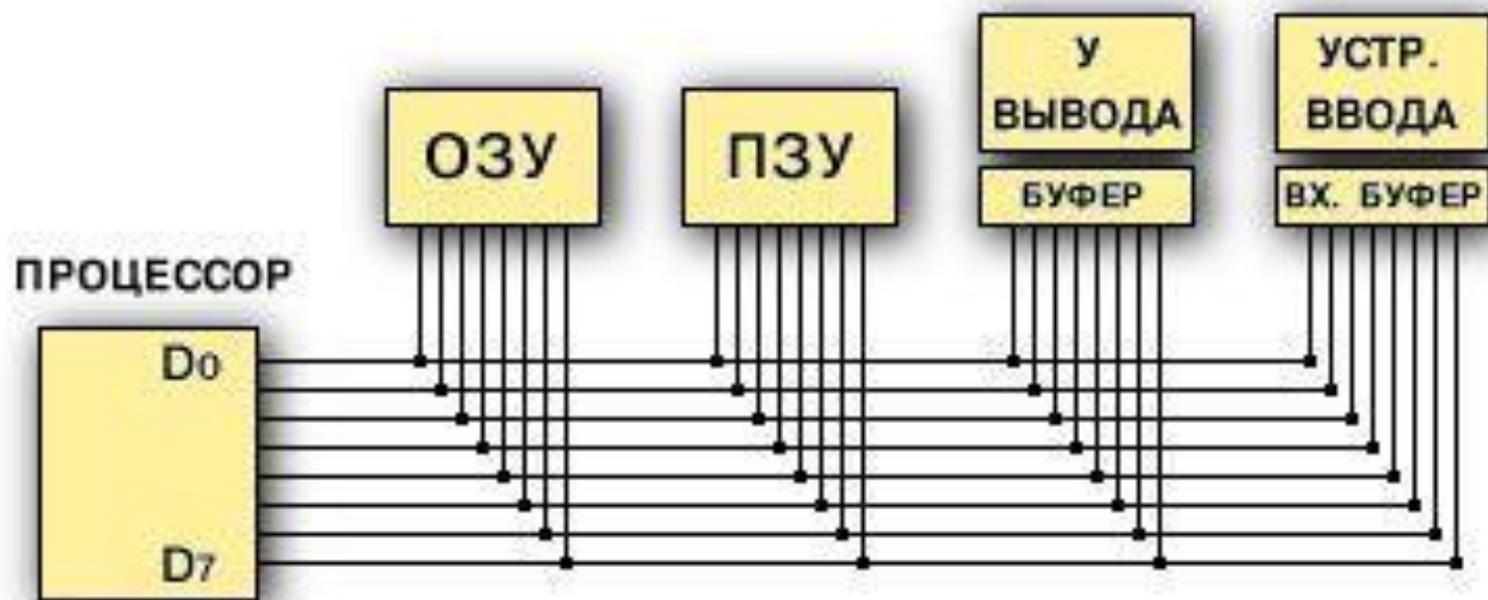
Подсистема ввода/вывода

Организация подсистемы ввода/вывода



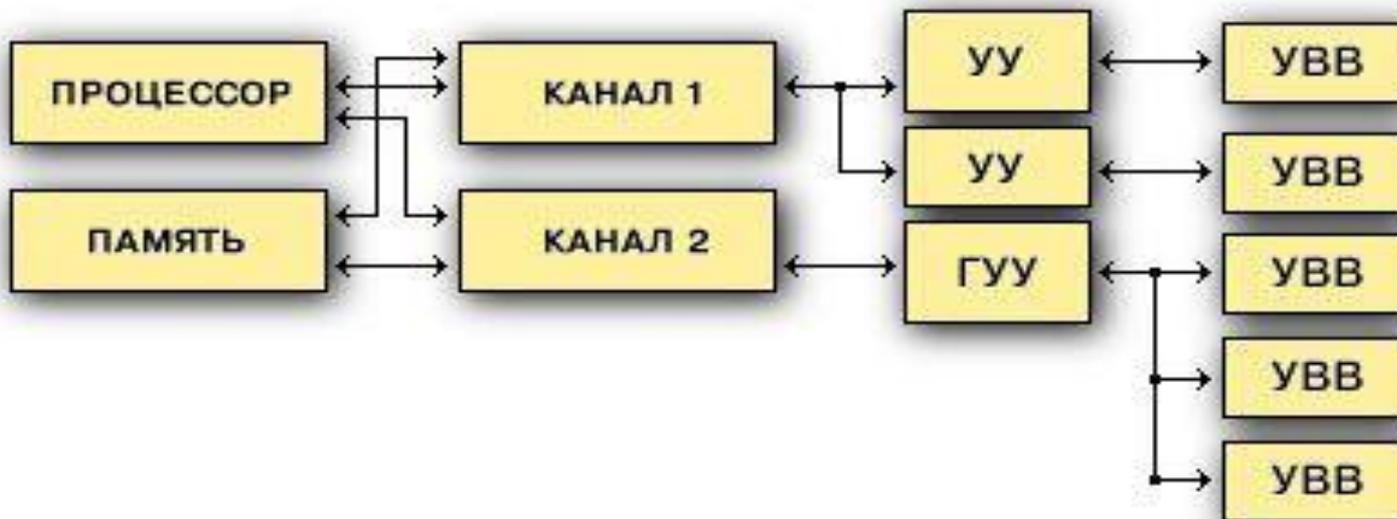
Шинная организация ввода/вывода

Типичное подключение устройств к шине данных.



Канальная организация ввода/вывода

Канальная организация.



Лекция №16

- Подсистема ввода/вывода
- 