



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

ИУ7, 4-й семестр, 2020 г.

Организация курса

- видео-, аудиозапись и фотосъёмка запрещены
- 2 модуля + экзамен
- 8 лекций, N лабораторных работ
- 38 часов самостоятельной подготовки к лабораторным работам

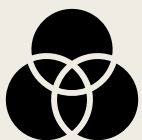
Литература

- Зубков С. В. "Assembler. Для DOS, Windows и Unix"

Цели и программа курса

- Изучение низкоуровневого устройства ЭВМ
- Понимание исполнения программ на аппаратном уровне. Работа процессора
- Умение составлять и читать программы, включая:
 - *составление программы на низком уровне "с нуля"*
 - *взаимодействие программного кода с внешними устройствами*
 - *доп. возможности и расширения современных процессоров*
 - *отладку и реверс-инжиниринг исполняемых файлов*

История создания ЭВМ. Появление вычислителей общего назначения. Архитектура фон Неймана



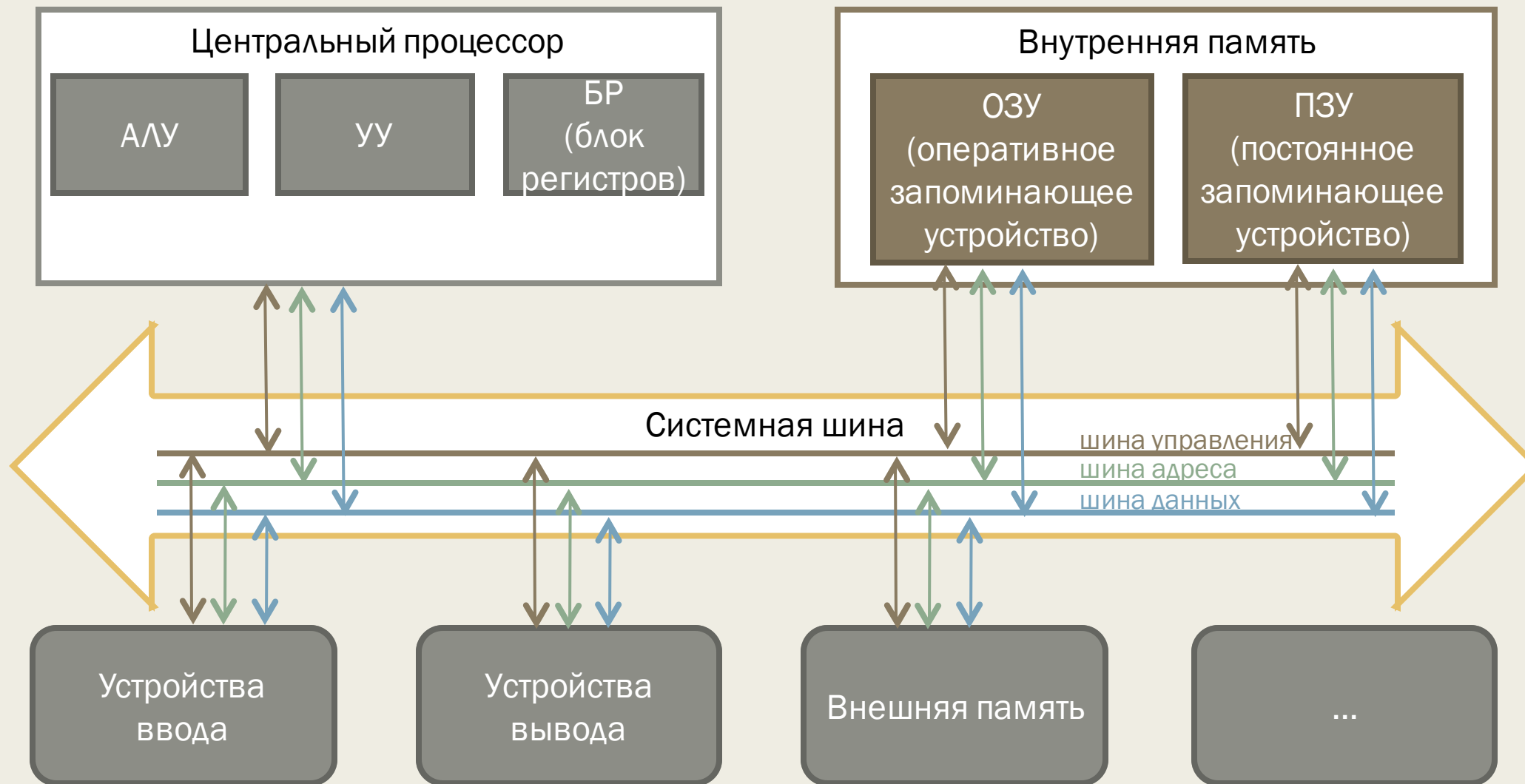
От решения частных вычислительных задач - к
универсальным системам

Принципы фон Неймана:

1. Использование двоичной системы счисления в вычислительных машинах.
2. Программное управление ЭВМ.
3. Память компьютера используется не только для хранения данных, но и программ.
4. Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.
5. Возможность условного перехода в процессе выполнения программы.



Структурная схема ЭВМ



Память. Единица адресации.

Представление символов

Байт - минимальная адресуемая единица памяти

- 8 бит
- диапазон значений - 0..255
- $8 = 2^3 = 10_{16}^2$

Машинное слово — машинно-зависимая величина, измеряемая в битах, равная разрядности регистров/шины данных

Параграф - 16 байт

ASCII (аски́) - *American standard code for information interchange*, США, 1963.

ASCII Code Chart																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- 7-битная кодировка
- первые 32 символа - служебные
- старшие 128 символов 8-битной кодировки - национальные языки

Системы счисления

Двоичная (binary)

- 0, 1, 10, 11, 100, 101...
- $2^8 = 256$
- $2^{10} = 1024$
- $2^{16} = 65536$

Шестнадцатеричная (hexadecimal)

- 0, 1, ..., 8, 9, A, B, C, D, E, F, 10, 11, 12, ..., 19, 1A, 1B...
- $2^4 = 16$
- $2^8 = 256$
- $2^{16} = 65536$

$$1011011011111000_2 = B6F8_{16}$$

Представление отрицательных чисел. Дополнительный код

-00101101 => инверсия и прибавление единицы:

■ 11010010

■ 11010011

-1 => 11111111

-1+1 = 0

11111111 + 1 = (1)00000000

Виды современных архитектур ЭВМ

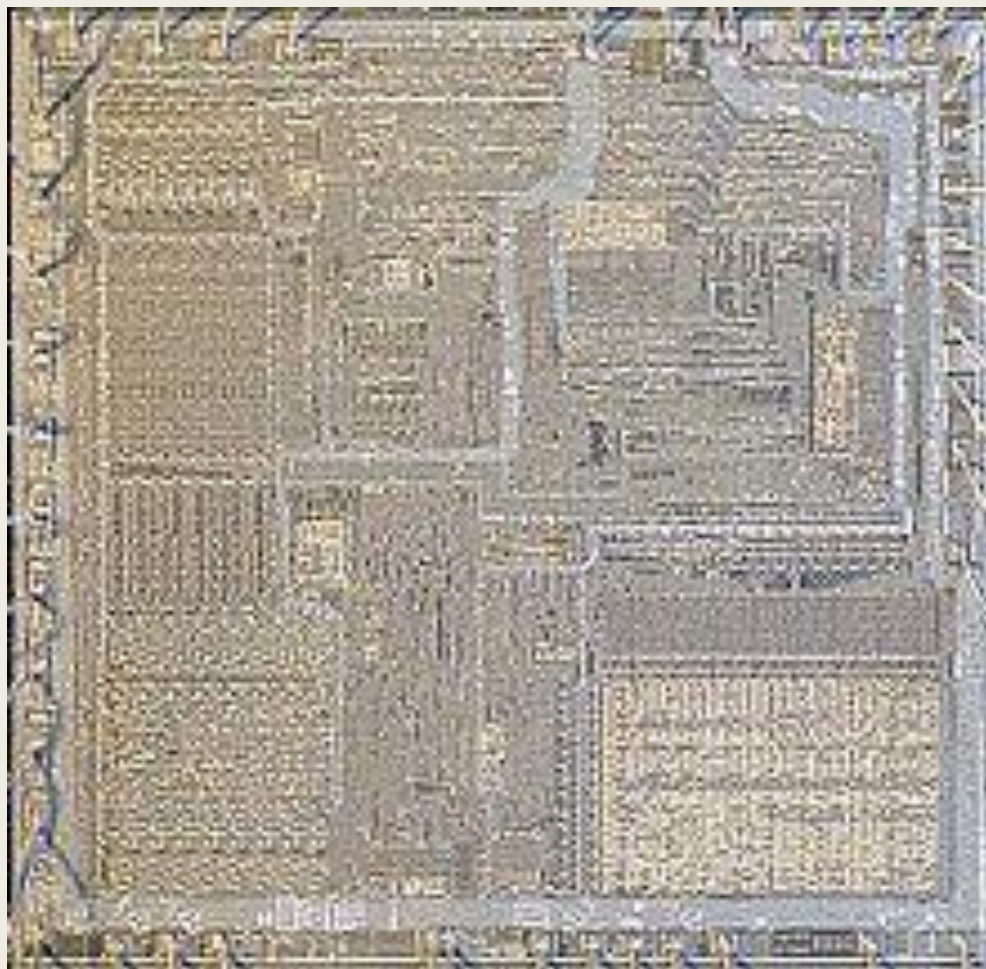
- x86
- x86-64
- IA64
- ARM
- MIPS (в т. ч. Байкал)
- Эльбрус

Семейство процессоров x86

- Микропроцессор 8086: 16-разрядный, 1978 г., 5-10 МГц, 3000 нм
- Предшественники: 4004 – 4-битный, 1971 г.; 8008 – 8-битный, 1972 г.; 8080 – 1974 г.
- Требуется микросхем поддержки
- 80186 – 1982 г., некоторое развитие, интегрированы микросхемы поддержки
- 80286 – 1982 г., 16-разрядный, добавлен защищённый режим
- 80386, 80486, Pentium, Celeron, AMD ... - 32-разрядные, повышение быстродействия и расширение аппаратного функционала (системы команд)
- x86-64 (x64) - семейства с 64-разрядной архитектурой
- Советский аналог - К1810ВМ86, 1985 г.



Устройство 8086



		MAX MODE		(MIN MODE)
GND	1	40	U _{CC}	
AD14	2	39	AD15	
AD13	3	38	A16/S3	
AD12	4	37	A17/S4	
AD11	5	36	A18/S5	
AD10	6	35	A19/S6	
AD9	7	34	$\overline{\text{BHE}}/\text{S7}$	
AD8	8	33	$\text{MN}/\overline{\text{MX}}$	
AD7	9	32	$\overline{\text{RD}}$	
AD6	10	31	$\overline{\text{RQ}}/\overline{\text{GT0}}$	(HOLD)
AD5	11	30	$\overline{\text{RQ}}/\overline{\text{GT1}}$	(HLDA)
AD4	12	29	$\overline{\text{LOCK}}$	($\overline{\text{WR}}$)
AD3	13	28	$\overline{\text{S2}}$	($\text{M}/\overline{\text{IO}}$)
AD2	14	27	$\overline{\text{S1}}$	($\text{DT}/\overline{\text{R}}$)
AD1	15	26	$\overline{\text{S0}}$	($\overline{\text{DEN}}$)
AD0	16	25	QS0	(ALE)
NMI	17	24	QS1	($\overline{\text{INTA}}$)
INTR	18	23	$\overline{\text{TEST}}$	
CLK	19	22	READY	
GND	20	21	RESET	

Архитектура 8086 с точки зрения программиста



Язык ассемблера

Язык **ассемблера** - машинно-зависимый язык программирования низкого уровня, команды которого прямо соответствуют машинным командам.

Исполняемые файлы. Компиляция. Линковка

- **Исполняемый файл** — файл, содержащий программу в виде, в котором она может быть исполнена компьютером.
- Получение исполняемых файлов: компиляция + линковка.
- Компилятор - программа для преобразования исходного текста другой программы на определённом языке в объектный модуль.
- Компоновщик (линковщик, линкер) - программа для связывания нескольких объектных файлов в исполняемый.

Исполняемые файлы. Запуск программы. Отладчик

- .EXE, .COM. Запуск новой программы операционной системой:
 1. Определение формата файла.
 2. Чтение и разбор заголовка.
 3. Считывание разделов исполняемого модуля (файла) в ОЗУ по необходимым адресам.
 4. Подготовка к запуску, если требуется.
 5. Передача управления на точку входа.
- Отладчик — программа для автоматизации процесса отладки. Может выполнять трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения кода, устанавливать и удалять контрольные точки или условия остановки.

"Простейший" формат исполняемого файла

.COM (command) - простейший формат исполняемых файлов DOS и ранних версий Windows.

- С < 64 Кб

Запуск COM-программы:

- Система выделяет свободный сегмент памяти и заносит его адрес во все сегментные регистры (CS, DS, ES и SS).
- В первые 256 байт этого сегмента записывается PSP.
- Непосредственно за ним загружается содержимое COM-файла без изменений.
- *Указатель стека (регистр SP) устанавливается на конец сегмента.*
- *В стек записывается 0000h (адрес возврата для команды ret).*
- Управление передаётся по адресу CS:0100h, где находится первый байт исполняемого файла.

Классификация команд 8086

- Команды пересылки данных
- Арифметические и логические команды
- Команды переходов
- Команды работы с подпрограммами
- Команды управления процессором

Команда пересылки данных MOV

MOV <приёмник>, <источник>

Источники: непосредственный операнд, РОН, сегментный регистр, переменная (ячейка памяти).

Приёмник: РОН, сегментный регистр, переменная (ячейка памяти).

- MOV AX, 5
- MOV BX, DX
- MOV [1234h], CH
- MOV DS, AX

- MOV [0123h], [2345h]
- MOV DS, 1000h

Целочисленная арифметика, базовые команды

- ADD <приёмник>, <источник> - выполняет арифметическое сложение приёмника и источника. Сумма помещается в приёмник, источник не изменяется.
- SUB <приёмник>, <источник> - вычитание. Аналогично ADD.
- MUL <источник> - умножение (без знака). Умножаются источник и AL/AX, в зависимости от размера источника. Результат помещается в AX либо DX:AX.
- DIV <источник> - деление (без знака). Деление AL/AX на источник. Результат помещается в AL/AX, остаток - в AH/DX.

Побитовая арифметика

- AND <приёмник, источник> - побитовое "И". AND al, 00001111b
- OR <приёмник, источник> - побитовое "ИЛИ". OR al, 00001111b
- XOR <приёмник, источник> - побитовое исключающее "ИЛИ". XOR AX, AX
- NOT <приёмник> - инверсия
- SHL <приёмник>, <счётчик> - сдвиг влево
- SHR <приёмник>, <счётчик> - сдвиг вправо

Команда безусловной передачи управления JMP

JMP <операнд>

- Передаёт управление в другую точку программы, не сохраняя какой-либо информации для возврата.
- Операнд - непосредственный адрес, регистр или переменная.

Пример

```
...  
    XOR    AX, AX  
    MOV    BX, 5  
label1:  
    INC    AX  
    ADD    BX, AX  
    JMP    label1  
...
```



AX	0000	SI	0000	CS	19F5	IP	0100
BX	0000	DI	0000	DS	19F5		
CX	0024	BP	0000	ES	19F5	HS	19F5
DX	0000	SP	FFFE	SS	19F5	FS	19F5

CMD >

0100	33C0	XOR	AX, AX
0102	BB0500	MOV	BX, 0005
0105	40	INC	AX
0106	03D8	ADD	BX, AX
0108	EBFB	JMP	0105
010A	BA1401	MOV	DX, 0114
010D	CD21	INT	21
010F	B44C	MOV	AH, 4C

Взаимодействие программы с внешним миром (ОС, пользователь)

Прерывания:

- аппаратные
- программные

int - генерация программного прерывания.

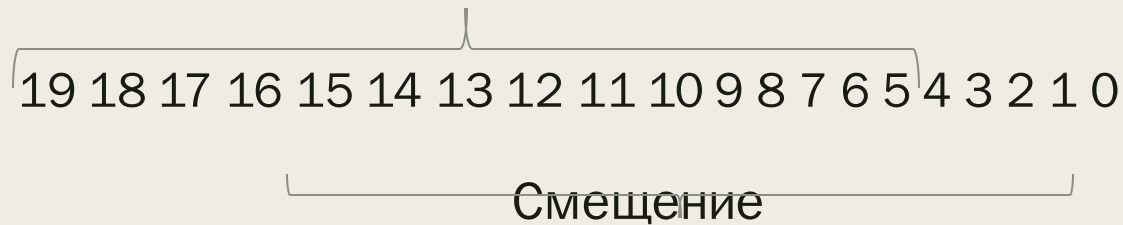
21h - прерывание DOS, предоставляет примерно 70 функций.

Номер функции передаётся через ah, параметры каждой функции и возвращаемый результат описаны в документации.

Память в реальном режиме работы процессора (для 8086)

1 Мб памяти = 2^{20}

Номер параграфа начала сегмента (сегментная часть адреса, сегмент)



CS:IP, DS:BX, SS:SP...

[SEG]:[OFFSET] => физический адрес:
 $SEG * 16 + OFFSET$

5678h:7890h =>

+	56780
	<u>7890</u>
	5E010

Логическая структура памяти.

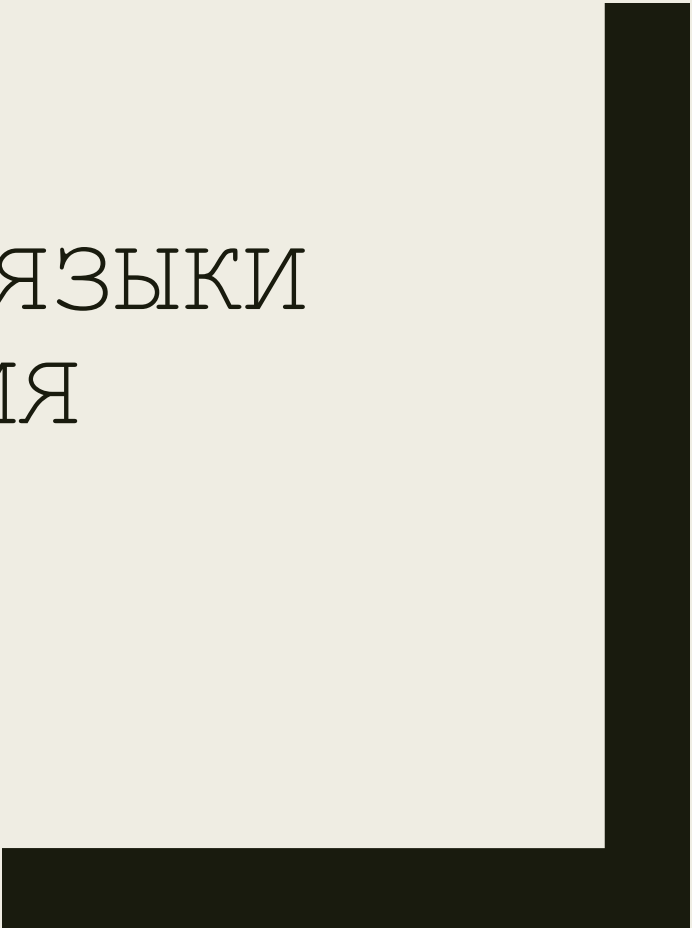
Сегменты

- Сегмент кода (CS)
- Сегменты данных (**DS**, ES, FS, GS)
- Сегмент стека (SS)



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 2
ИУ7, 4-й семестр, 2020 г.



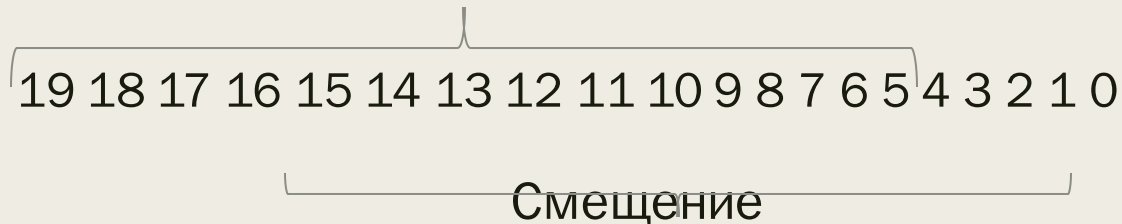
Важное из 1-й лекции и 1-й л/р

- Машинный код - набор кодов операций конкретной вычислительной машины, которые интерпретируются непосредственно процессором.
- Язык ассемблера - низкоуровневый язык программирования, одна команда которого соответствует одной машинной команде.
- Компиляция - процесс перевода программы с языка программирования в машинный код.
- COM-файл - простейший формат исполняемого файла DOS (и немного Windows), который считывается с диска в ОЗУ без изменений и запускается с 1-го байта. Плюсы: простой. Минусы: размер < 64 Кб; слишком простой.

Память в реальном режиме работы процессора (режим 8086)

1 Мб памяти = 2^{20}

Номер параграфа начала сегмента (сегментная часть адреса, сегмент)



CS:IP, DS:BX, SS:SP...

[SEG]:[OFFSET] => физический адрес:
 $SEG * 16 + OFFSET$

5678h:7890h =>

+	56780
	<u>7890</u>
	5E010

Логическая структура памяти. Сегменты

- Сегмент кода (CS)
- Сегменты данных (**DS**, ES, FS, GS)
- Сегмент стека (SS)

Команда NOP (no operation)

- Ничего не делает
- Занимает место и время
- Размер - 1 байт, код 90h
- Назначение - задержка выполнения либо заполнение памяти, например, для выравнивания

Структура программы на ассемблере

(Зубков С. В., Assembler для DOS, Windows, ..., глава 3)

- Модули (файлы исходного кода)
- Сегменты (описание блоков памяти)
- Составляющие программного кода:
 - команды процессора
 - инструкции описания структуры, выделения памяти, макроопределения
- Формат строки программы:
 - метка команда/директива операнды ; комментарий

Метки

В коде

- Пример:

```
mov cx, 5
```

```
label1:
```

```
add ax, bx
```

```
loop label1
```

- Метка обычно используется в командах передачи управления

В определении данных

- `label`

- метка `label` тип
- Допустимые типы: BYTE, WORD, DWORD, FWORD, QWORD, TBYTE, NEAR, FAR.

- `EQU, =`

- макрос
- вычисляет выражение в правой части и присваивает его метке

Директивы выделения памяти

- Псевдокоманда - директива ассемблера, которая приводит к включению данных или кода в программу, но не соответствует никакой команде процессора.
- Псевдокоманды определения данных указывают, что в соответствующем месте располагается переменная, резервируют под неё место заданного типа, заполняют значением и ставят в соответствие метку.
- Виды: DB (1), DW (2), DD (4), DF (6), DQ (8), DT (10).
- Примеры:
 - a DB 1
 - float_number DD 3.5e7
 - text_string DB 'Hello, world!'
- DUP - заполнение повторяющимися данными.
- ? - неинициализированное значение.
- uninitialized DW 512 DUP(?)

Структура программы

- Любая программа состоит из сегментов
- Виды сегментов:
 - сегмент кода
 - сегмент данных
 - сегмент стека
- Описание сегмента в исходном коде:
имя SEGMENT READONLY выравнивание тип разряд 'класс'
...
имя ENDS

Параметры директивы SEGMENT

Выравнивание

- BYTE
- WORD
- DWORD
- PARA
- PAGE

Тип

- PUBLIC
- STACK
- COMMON
- AT
- PRIVATE

Класс - любая метка, взятая в одинарные кавычки. Сегменты одного класса расположатся в памяти друг за другом.

Директива ASSUME

- ASSUME регистр:имя сегмента
- Не является командой
- Нужна для контроля компилятором правильности обращения к переменным

```
Data1 SEGMENT WORD 'DATA'  
Var1 DW 0  
Data1 ENDS
```

```
Data2 SEGMENT WORD 'DATA'  
Var2 DW 0  
Data2 ENDS
```

```
Code SEGMENT WORD 'CODE'  
    ASSUME CS:Code  
ProgramStart:  
    mov ax,Data1  
    mov ds,ax  
    ASSUME DS:Data1  
    mov ax,Data2  
    mov es,ax  
    ASSUME ES:Data2  
    mov ax,[Var2]  
  
    .  
    .  
    .  
Code ENDS  
END ProgramStart
```

Модели памяти

.model модель, язык, модификатор

- TINY - один сегмент на всё
- SMALL - код в одном сегменте, данные и стек - в другом
- COMPACT - допустимо несколько сегментов данных
- MEDIUM - код в нескольких сегментах, данные - в одном
- LARGE, HUGE
- Язык - C, PASCAL, BASIC, SYSCALL, STDCALL. Для связывания с ЯВУ и вызова подпрограмм.
- Модификатор - **NEARSTACK**/FARSTACK
- Определение модели позволяет использовать сокращённые формы директив определения сегментов.

Конец программы. Точка входа

.

.

.

END start

- start - имя метки, объявленной в сегменте кода и указывающее на команду, с которой начнётся исполнение программы.
- Если в программе несколько модулей, только один может содержать начальный адрес.

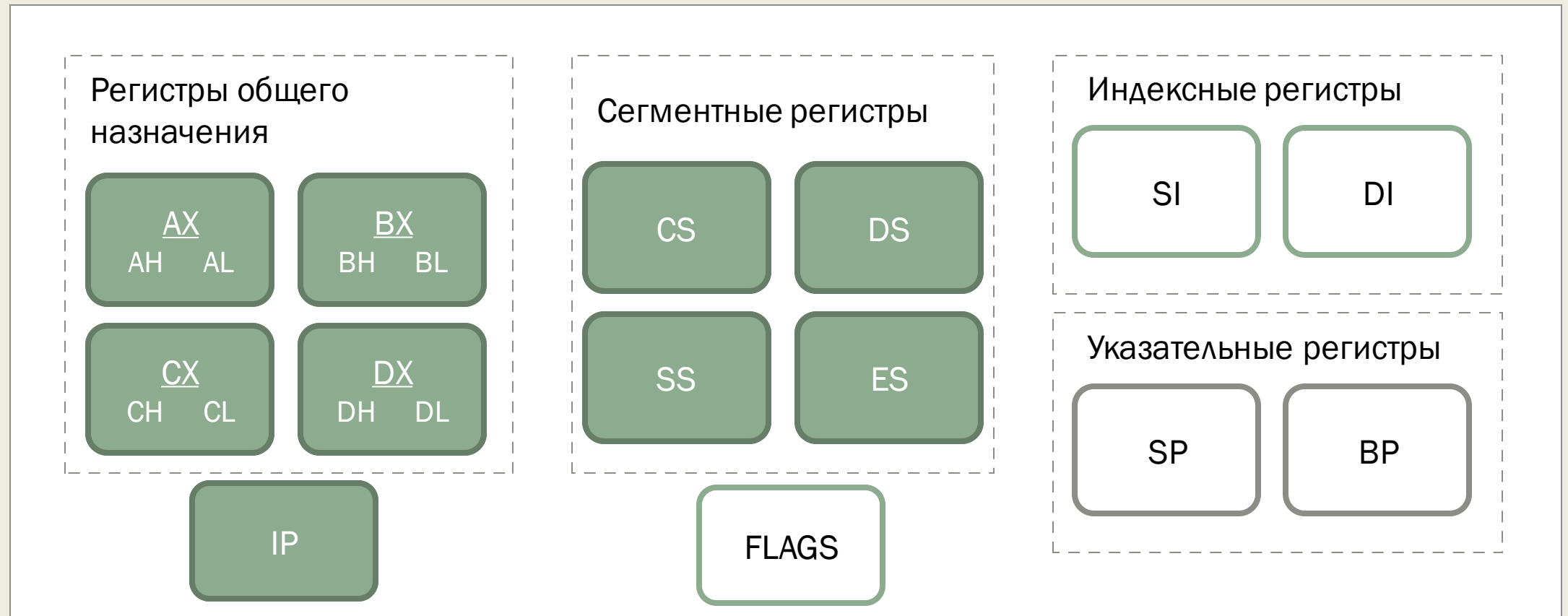
Прочие директивы

- Задание набора допустимых команд: **.8086**, .186, .286, ..., .586, .686, ...
- Управление программным счётчиком
 - ORG значение
 - EVEN
 - ALIGN значение
- Глобальные объявления
 - public, comm, extrn, global
- Условное ассемблирование
 - IF выражение
 - ...
 - ELSE
 - ...
 - ENDIF

Виды переходов для команды JMP

- short (короткий) -128 .. +127 байт
- near (ближний) в том же сегменте (без изменения CS)
- far (дальний) в другой сегмент (со сменой CS)
- Для короткого и ближнего переходов непосредственный операнд (число) прибавляется к IP
- Операнды - регистры и переменные заменяют старое значение в IP (CS:IP)

Регистры. Регистр флагов



Индексные регистры SI и DI

- SI – source index, индекс источника
- DI – destination index, индекс приёмника
- Могут использоваться в большинстве команд, как регистры общего назначения
- Применяются в специфических командах поточной обработки данных

Способы адресации (Зубков, Assembler, ..., глава 2)

- Регистровая адресация (`mov ax, bx`)
- Непосредственная адресация (`mov ax, 2`)
- Прямая адресация (`mov ax, ds:0032`)
- Косвенная адресация (`mov ax, [bx]`). В 8086 допустимы BX, BP, SI, DI
- Адресация по базе со сдвигом (`mov ax, [bx]+2`; `mov ax, 2[bx]`).
- Адресация по базе с индексированием (допустимы BX+SI, BX+DI, BP+SI, BP+DI):
 - `mov ax, [bx+si+2]` - `mov ax, [bx][si]+2`
 - `mov ax, [bx+2][si]` - `mov ax, [bx][si+2]`
 - `mov ax, 2[bx][si]`

Регистр FLAGS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CF	-	PF	-	AF	-	ZF	SF	TF	IF	DF	OF	IOPL		NT	-

- CF (carry flag) - флаг переноса
- PF (parity flag) - флаг чётности
- AF (auxiliary carry flag) - вспомогательный флаг переноса
- ZF (zero flag) - флаг нуля
- SF (sign flag) - флаг знака
- TF (trap flag) - флаг трассировки
- IF (interrupt enable flag) - флаг разрешения прерываний
- DF (direction flag) - флаг направления
- OF (overflow flag) - флаг переполнения
- IOPL (I/O privilege level) - уровень приоритета ввода-вывода
- NT (nested task) - флаг вложенности задач

Команда сравнения CMP

- CMP <приёмник>, <источник>
- Источник - число, регистр или переменная
- Приёмник - регистр или переменная; не может быть переменной одновременно с источником
- Вычитает источник из приёмника, результат никуда не сохраняется, выставляются флаги
- CF, PF, AF, ZF, SF, OF

Команды условных переходов J.. (Зубков, Assembler, ..., глава 2)

- Переход типа short или near
- Обычно используются в паре с CMP
- "Выше" и "ниже" - при сравнении беззнаковых чисел
- "Больше" и "меньше" - при сравнении чисел со знаком

Виды условных переходов (часть 1)

Команда	Описание	Состояние флагов для выполнения перехода
JO	Есть переполнение	OF = 1
JNO	Нет переполнения	OF = 0
JS	Есть знак	SF = 1
JNS	Нет знака	SF = 0
JE JZ	Если равно/если ноль	ZF = 1
JNE JNZ	Не равно/не ноль	ZF = 0
JP/JPE	Есть чётность/чётное	PF = 1
JNP/JPO	Нет чётности/нечётное	PF = 0
JCXZ	CX = 0	

Виды условных переходов (часть 2)

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JB JNAE JC	Если ниже Если не выше и не равно Если перенос	CF = 1	нет
JNB JAE JNC	Если не ниже Если выше или равно Если нет переноса	CF = 0	нет
JBE JNA	Есть ниже или равно Если не выше	CF = 1 или ZF = 1	нет
JA JNBE	Если выше Если не ниже и не равно	CF = 0 и ZF = 0	нет

Виды условных переходов (часть 3)

Команда	Описание	Состояние флагов для выполнения перехода	Знаковый
JL JNGE	Если меньше Если не больше и не равно	SF <> OF	да
JGE JNL	Если больше или равно Если не меньше	SF = OF	да
JLE JNG	Если меньше или равно Если не больше	ZF = 1 или SF <> OF	да
JG JNLE	Если больше Если не меньше и не равно	ZF = 0 и SF = OF	да

Команда TEST

- TEST <приёмник>, <источник>
- Аналог AND, но результат не сохраняется
- Выставляются флаги SF, ZF, PF.

Прерывания

- Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передаётся программе-обработчику возникшего прерывания.
- Виды прерываний:
 - аппаратные (асинхронные) - события от внешних устройств;
 - внутренние (синхронные) - события в самом процессоре, например, деление на ноль;
 - программные - вызванные командой `int`.

Прерывание DOS 21h

- Аналог системного вызова в современных ОС
- Используется наподобие вызова подпрограммы
- Номер функции передаётся через AH

Прерывание DOS - вывод на экран в текстовом режиме

Функция	Назначение	Вход	Выход
02	Вывод символа в stdout	DL = ASCII-код символа	-
09	Вывод строки в stdout	DS:DX - адрес строки, заканчивающейся символом \$	-

Прерывание DOS - ввод с клавиатуры

Функция	Назначение	Вход	Выход
01	Считать символ из stdin с эхом	-	AL – ASCII-код символа
06	Считать символ без эха, без ожидания, без проверки на Ctrl+Break	DL=FF	AL – ASCII-код символа
07	Считать символ без эха, с ожиданием и без проверки на Ctrl+Break	-	AL – ASCII-код символа
08	Считать символ без эха	-	AL – ASCII-код символа
10 (0Ah)	Считать строку с stdin в буфер	DS:DX - адрес буфера	Введённая строка помещается в буфер
0Bh	Проверка состояния клавиатуры	-	AL=0, если клавиша не была нажата, и FF, если была
0Ch	Очистить буфер и считать символ	AL=01, 06, 07, 08, 0Ah	



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 3

"Команды процессора x86. Стек. Подпрограммы"

ИУ7, 4-й семестр, 2020 г.



CMOVcc - условная пересылка данных

CMOVcc <приёмник>, <источник>

Условия аналогичны Jcc

XCHG - обмен операндов между собой

XCHG <операнд1>, <операнд2>

Выполняется над двумя регистрами либо регистром и переменной

XLAT/XLATB - трансляция в соответствии с таблицей

XLAT [адрес]

XLATB

Помещает в AL байт из таблицы по адресу DS:BX со смещением относительно начала таблицы, равным AL.

Адрес, указанный в исходном коде, не обрабатывается компилятором и служит в качестве комментария.

Если в адресе явно указан сегментный регистр, он будет использоваться вместо DS.

LEA - вычисление эффективного адреса

LEA <приёмник>, <источник>

Вычисляет эффективный адрес источника и помещает его в приёмник.

Позволяет вычислить адрес, описанный сложным методом адресации.

Иногда используется для быстрых арифметических вычислений:

```
lea bx, [bx+bx*4]
```

```
lea bx, [ax+12]
```

Эти вычисления занимают меньше памяти, чем соответствующие MOV и ADD, и не изменяют флаги.

Двоичная арифметика.

ADD, ADC, SUB, SBB

ADD, SUB не делают различий между знаковыми и беззнаковыми числами.

ADC <приёмник>, <источник> - сложение с переносом. Складывает приёмник, источник и флаг CF.

SBB <приёмник>, <источник> - вычитание с займом. Вычитает из приёмника источник и дополнительно - флаг CF.

```
add ax, cx  
adc dx, bx
```

```
sub ax, cx  
sbb dx, bx
```

Арифметические флаги - CF, OF, SF, ZF, AF, PF

IMUL, MUL IDIV, DIV

Умножение чисел со знаком:

IMUL <источник>

IMUL <приёмник>, <источник>

IMUL <приёмник>, <источник1>, <источник2>

Целочисленное деление со знаком:

IDIV <источник>

Результат округляется в сторону нуля, знак остатка совпадает со знаком делимого.

INC, DEC

INC <приёмник>

DEC <приёмник>

Увеличивает/уменьшает приёмник на 1.

В отличие от ADD, не изменяет CF.

OF, SF, ZF, AF, PF устанавливаются в соответствии с результатом.

NEG - изменение знака

NEG <приёмник>

Десятичная арифметика

DAA, DAS, AAA, AAS, AAM, AAD

- Неупакованное двоично-десятичное число - байт от 00h до 09h.
- Упакованное двоично-десятичное число - байт от 00h до 99h (цифры A..F не задействуются).
- При выполнении арифметических операций необходима коррекция:
 - $19h + 1 = 1Ah \Rightarrow 20h$

```
inc al
```

```
daa
```


Логические команды AND, OR, XOR, NOT, TEST

Логический, арифметический,
циклический сдвиг.

SAR, SAL, SHR, SHL, ROR, ROL, RCR, RCL

- SAL тождественна SHL
- SHR зануляет старший бит, SAR - сохраняет (знак)
- ROR, ROL - циклический сдвиг вправо/влево
- RCR, RCL - циклический сдвиг через CF

Операции над битами и байтами

BT, BTR, BTS, BTC, BSF, BSR, SETcc

- BT <база>, <смещение> - считать в CF значение бита из битовой строки
- BTS <база>, <смещение> - установить бит в 1
- BTR <база>, <смещение> - сбросить бит в 0
- BTC <база>, <смещение> - инвертировать бит
- BSF <приёмник>, <источник> - прямой поиск бита (от младшего разряда)
- BSR <приёмник>, <источник> - обратный поиск бита (от старшего разряда)
- SETcc <приёмник> - выставляет приёмник (1 байт) в 1 или 0 в зависимости от условия, аналогично Jcc

Организация циклов

- LOOP <метка> - уменьшает CX и выполняет "короткий" переход на метку, если CX не равен нулю.
- LOOPE/LOOPZ <метка> - цикл "пока равно"/"пока ноль"
- LOOPNE/LOOPNZ <метка> - цикл "пока не равно"/"пока не ноль"

Декрементируют CX и выполняют переход, если CX не ноль и если выполняется условие (ZF).

Строковые операции: копирование, сравнение, сканирование, чтение, запись

Строка-источник - DS:SI, строка-приёмник - ES:DI.

За один раз обрабатывается один байт (слово).

- MOVS/MOVSБ/MOVSВ <приёмник>, <источник> - копирование
- CMPS/CMPSБ/CMPSВ <приёмник>, <источник> - сравнение
- SCAS/SCASБ/SCASВ <приёмник> - сканирование (сравнение с AL/AX)
- LODS/LODSБ/LODSВ <источник> - чтение (в AL/AX)
- STOS/STOSБ/STOSВ <приёмник> - запись (из AL/AX)

- Префиксы: REP/REPE/REPZ/REPNE/REPZ

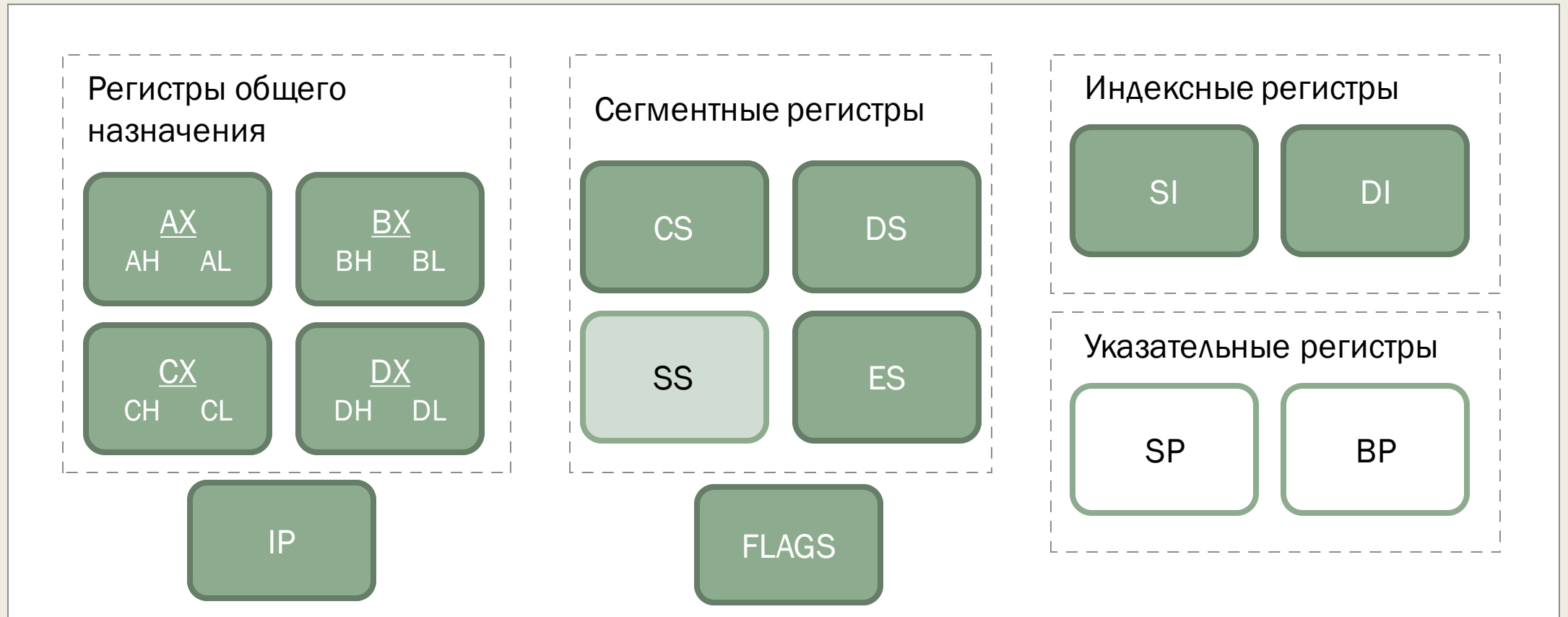
Управление флагами

- STC/CLC/CMC - установить/сбросить/инвертировать CF
- STD/CLD - установить/сбросить DF
- LAHF - загрузка флагов состояния в AH
- SAHF - установка флагов состояния из AH
- CLI/STI - запрет/разрешение прерываний (IF)

Загрузка сегментных регистров

- LDS <приёмник>, <источник> - загрузить адрес, используя DS
- LES <приёмник>, <источник> - загрузить адрес, используя ES
- LFS <приёмник>, <источник> - загрузить адрес, используя FS
- LGS <приёмник>, <источник> - загрузить адрес, используя GS
- LSS <приёмник>, <источник> - загрузить адрес, используя SS

Регистры. Стек



Стек

- LIFO/FILO (last in, first out) - последним пришёл, первым ушёл
- Сегмент стека - область памяти программы, используемая её подпрограммами, а также (вынужденно) обработчиками прерываний
- SP - указатель на вершину стека
- В x86 стек "растёт вниз", в сторону уменьшения адресов. При запуске программы SP указывает на конец сегмента

Команды непосредственной работы со стеком

- PUSH <источник> - поместить данные в стек. Уменьшает SP на размер источника и записывает значение по адресу SS:SP.
- POP <приёмник> - считать данные из стека. Считывает значение с адреса SS:SP и увеличивает SP.
- PUSHA - поместить в стек регистры AX, CX, DX, BX, SP, BP, SI, DI.
- POPA - загрузить регистры из стека (SP игнорируется)

CALL - вызов процедуры, RET - возврат из процедуры

CALL <операнд>

- Сохраняет адрес следующей команды в стеке (уменьшает SP и записывает по его адресу IP либо CS:IP, в зависимости от размера аргумента)
- Передаёт управление на значение аргумента.

RET/RETN/RETF <число>

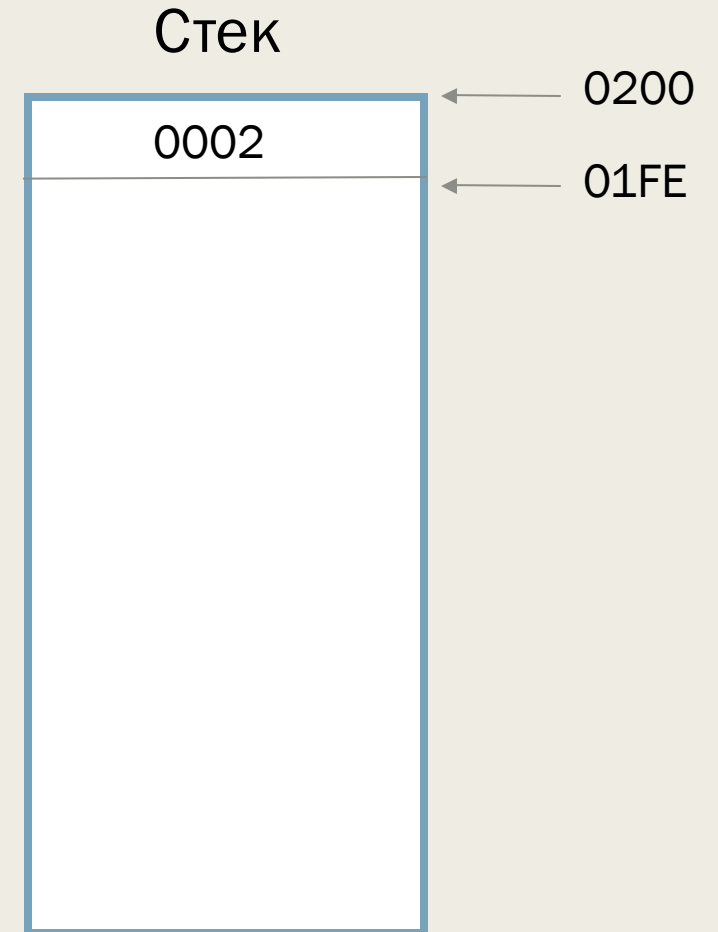
- Загружает из стека адрес возврата, увеличивает SP
- Если указан операнд, его значение будет дополнительно прибавлено к SP для очистки стека от параметров

BP – base pointer

- Используется в подпрограмме для сохранения "начального" значения SP
- Адресация параметров
- Адресация локальных переменных

Пример вызова подпрограммы №1

```
                                0. SP = 0200
0000: CALL P1                  1. SP = 01FE
0002: MOV BX, AX
...
P1:
0123: MOV AX, 5
0125: RET                      2. SP = 0200
```



Пример вызова подпрограммы №2

0000: PUSH ABCDh ;передача параметра

0002: CALL P1

0004: POP DX

0006: MOV BX, AX

...

P1:

0123: MOV BP, SP ;ss:[bp] - адрес возврата

;ss:[bp+2] - параметр

...

0223: MOV AX, 5

0225: RET

0. SP = 0200

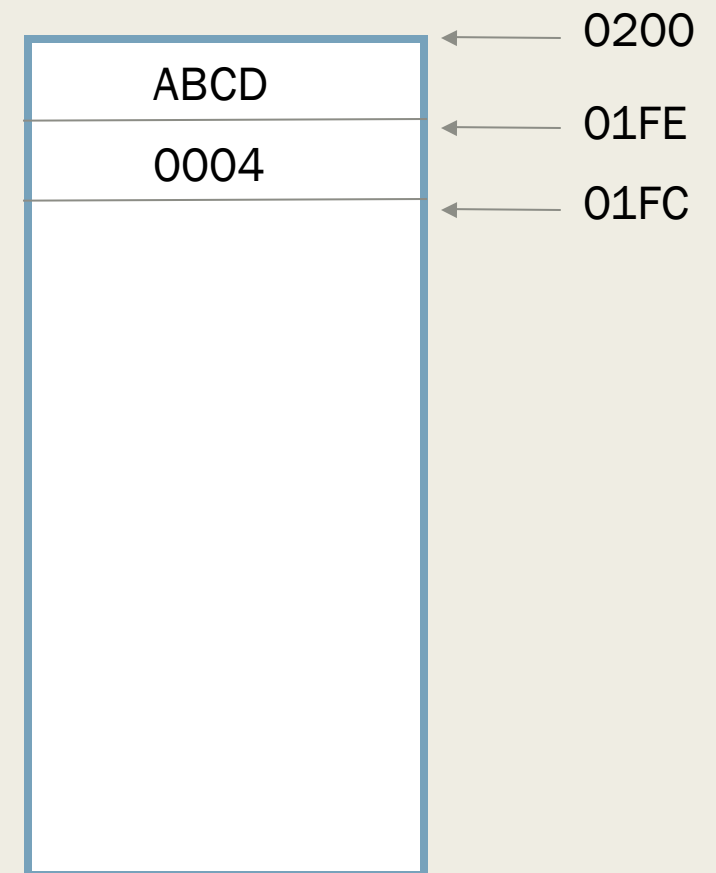
1. SP = 01FE

2. SP = 01FC

4. SP = 0200

3. SP = 01FE

Стек



Пример вызова подпрограммы №3

```
0000: PUSH ABCDh ;передача параметра
```

```
0002: CALL P1
```

```
0004: MOV BX, AX
```

```
...
```

```
P1:
```

```
0123: MOV BP, SP ;ss:[bp] - адрес возврата
```

```
;ss:[bp+2] - параметр
```

```
0125: SUB SP, 10 ; ss:[bp-1 .. bp-10] - локальные переменные
```

```
...
```

```
0221: ADD SP, 10
```

```
0223: MOV AX, 5
```

```
0225: RET 2
```

0. SP = 0200

1. SP = 01FE

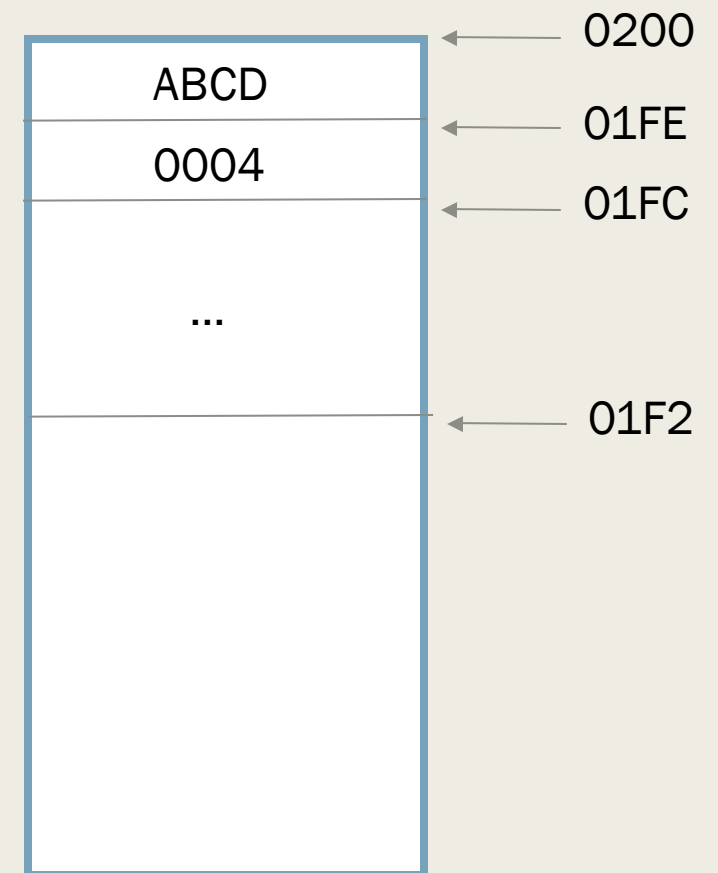
2. SP = 01FC

3. SP = 01F2

4. SP = 01FC

5. SP = 0200

Стек



Вопросы к рубежному контролю

1. Регистры общего назначения.
2. Сегментные регистры. Адресация в реальном режиме. Понятие сегментной части адреса и смещения.
3. Регистры работы со стеком.
4. Структура программы. Сегменты.
5. Прерывание 21h. Примеры ввода-вывода.
6. Стек. Назначение, примеры использования.
7. Регистр флагов.
8. Команды условной и безусловной передачи управления.
9. Организация многомодульных программ.
10. Подпрограммы. Объявление, вызов.
11. Арифметические команды.
12. Команды побитовых операций.
13. Команды работы со строками.



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 4

"Обработка прерываний. Работа с портами ввода-вывода"

ИУ7, 4-й семестр, 2020 г.

Прерывания (повтор)

- Прерывание - особая ситуация, когда выполнение текущей программы приостанавливается и управление передаётся программе-обработчику возникшего прерывания.
- Виды прерываний:
 - аппаратные (асинхронные) - события от внешних устройств;
 - внутренние (синхронные) - события в самом процессоре, например, деление на ноль;
 - программные - вызванные командой `int`.

Процессоры x86, x86-64

8086 (1978 г.) -> 80186 (1982 г.)

-> 80286 (1982 г.) добавлен защищённый режим

-> **80386** (1985 г.) архитектура стала 32-разрядной

-> 80486 (1989 г.) -> Pentium -> ... -> (современные процессоры)

"Реальный" режим (режим совместимости с 8086)

- обращение к оперативной памяти происходит по реальным (действительным) адресам, трансляция адресов не используется;
- набор доступных операций не ограничен;
- защита памяти не используется.

"Защищённый" режим

- обращение к памяти происходит по виртуальным адресам с использованием механизмов защиты памяти;
- набор доступных операций определяется уровнем привилегий (кольца защиты): системный и пользовательский уровни

Режим V86

Таблица векторов прерываний в реальном режиме работы процессора

- Вектор прерывания — номер, который идентифицирует соответствующий обработчик прерываний. Векторы прерываний объединяются в таблицу векторов прерываний, содержащую адреса обработчиков прерываний.
- Располагается в самом начале памяти, начиная с адреса 0.
- Доступно 256 прерываний.
- Каждый вектор занимает 4 байта - полный адрес.
- Размер всей таблицы - 1 Кб.

Срабатывание прерывания

- Сохранение в текущий стек регистра флагов и адреса возврата (адреса следующей команды)
- Передача управления по адресу обработчика из таблицы векторов
- Настройка стека?
- Повторная входимость (реентерабельность), необходимость запрета прерываний

IRET - возврат из прерывания

- Используется для выхода из обработчика прерывания
- Восстанавливает FLAGS, CS:IP
- При необходимости выставить значение флага обработчик меняет его значение непосредственно в стеке

Перехват прерывания

- Сохранение адреса старого обработчика
- Изменение вектора на "свой" адрес
- Вызов старого обработчика до/после отработки своего кода
- При деактивации - восстановление адреса старого обработчика

Некоторые прерывания

- 0 - деление на 0
- 1 - прерывание отладчика, вызывается после каждой команды при флаге TF
- 3 - "отладочное", int 3 занимает 1 байт
- 4 - переполнение при команде INTO
- 5 - при невыполнении условия в команде BOUND
- 6 - недопустимая (несуществующая) инструкция
- 7 - отсутствует FPU
- 8 - таймер
- 9 - клавиатура
- 10h - прерывание BIOS

Резидентные программы

- Резидентная программа - та, которая остаётся в памяти после возврата управления DOS
- Завершение через функции 31h/27h
- DOS не является многозадачной операционной системой
- Резиденты - частичная реализация многозадачности
- Резидентная программа должна быть составлена так, чтобы минимизировать используемую память

Порты ввода-вывода

- Порты ввода-вывода - механизм взаимодействия программы, выполняемой процессором, с устройствами компьютера.
- IN - команда чтения данных из порта ввода
- OUT - команда записи в порт вывода
- Пример:
 - IN al, 61h
 - OR al, 3
 - OUT 61h, al



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 5

"32-, 64-разрядные процессоры семейства x86"

ИУ7, 4-й семестр, 2020 г.

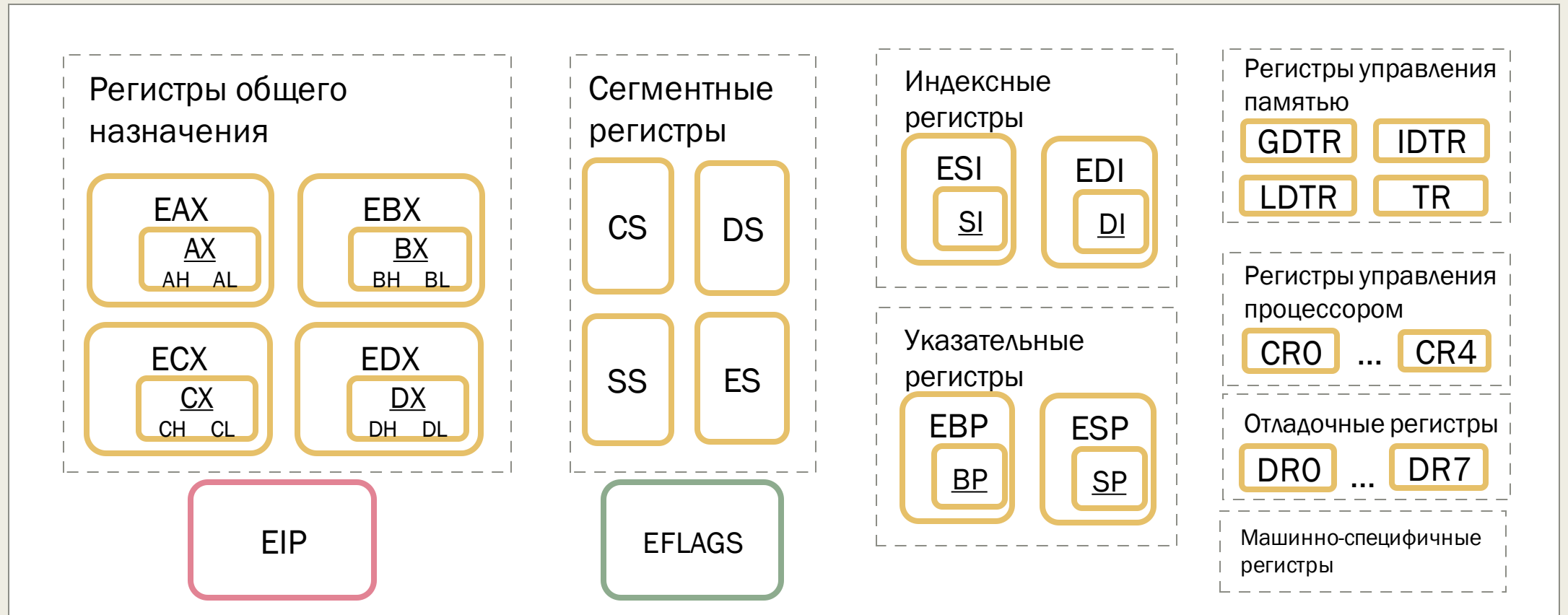
32-разрядные процессоры (386+)

Производство x86: 1985 - ~2010

32-разрядные:

- Регистры, кроме сегментных
- Шина данных
- Шина адреса ($2^{32} = 4\text{Гб ОЗУ}$)

Регистры x86



Система команд

- Аналогична системе команд 16-разрядных процессоров
- Доступны как прежние команды обработки 8- и 16-разрядных аргументов, так и 32-разрядных регистров и переменных
- Пример:

```
mov eax, 12345678h
xor ebx, ebx
mov bx, 1
add eax, ebx      ; eax=12345679h
```

Модели памяти

- Плоская - код и данные используют одно и то же пространство
- Сегментная - сложение сегмента и смещения
- Страничная - *виртуальные* адреса отображаются на физические постранично
 - виртуальная память — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (файл, или раздел подкачки)
 - основной режим для большинства современных ОС
 - в x86 минимальный размер страницы - 4096 байт
 - основывается на *таблице страниц* - структуре данных, используемой системой виртуальной памяти в операционной системе компьютера для хранения сопоставления между виртуальным адресом и физическим адресом. Виртуальные адреса используются выполняющимся процессом, в то время как физические адреса используются аппаратным обеспечением. Таблица страниц является ключевым компонентом преобразования виртуальных адресов, который необходим для доступа к данным в памяти.

Управление памятью в x86

- В сегментных регистрах - селекторы
 - 13-разрядный номер дескриптора
 - какую таблицу использовать - глобальную или локальную
 - уровень привилегий запроса 0-3
- По селектору определяется запись в одной из таблиц дескрипторов сегментов
- При включённом страничном режиме - по таблице страниц определяется физический адрес страницы либо выявляется, что она выгружена из памяти, срабатывает исключение и операционная система подгружает затребованную страницу из "подкачки" (swap)

Поддержка многозадачности

TSS (Task State Segment — сегмент состояния задачи) — специальная структура в архитектуре x86, содержащая информацию о задаче (процессе). Используется ОС для диспетчеризации задач, в т. ч. переключения на стек ядра при обработке прерываний и исключений

Исключения

- **Исключения** (Exceptions) подразделяются на отказы, ловушки и аварийные завершения.
- **Отказ** (fault) — это исключение, которое обнаруживается и обслуживается **до** выполнения инструкции, вызывающей ошибку. После обслуживания этого исключения управление возвращается снова на ту же инструкцию (включая все префиксы), которая вызвала отказ. Отказы, используемые в системе виртуальной памяти, позволяют, например, подкачать с диска в оперативную память затребованную страницу или сегмент.
- **Ловушка** (trap) — это исключение, которое обнаруживается и обслуживается **после** выполнения инструкции, его вызывающей. После обслуживания этого исключения управление возвращается на инструкцию, следующую за вызвавшей ловушку. К классу ловушек относятся и программные прерывания.
- **Аварийное завершение** (abort) — это исключение, которое не позволяет точно установить инструкцию, его вызвавшую. Оно используется для сообщения о серьезной ошибке, такой как аппаратная ошибка или повреждение системных таблиц.

Регистры управления памятью

- GDTR: 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов (GDT) и 16-битный размер (лимит, уменьшенный на 1)
- IDTR: 6-байтный регистр, содержит 32-битный линейный адрес начала таблицы глобальных дескрипторов обработчиков прерываний (IDT) и 16-битный размер (лимит, уменьшенный на 1)
- LDTR: 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий текущую таблицу локальных дескрипторов
- TR: 10-байтный регистр, содержит 16-битный селектор для GDT и весь 8-байтный дескриптор из GDT, описывающий TSS текущей задачи

Регистр EFLAGS

- FLAGS и ещё 5 специфических флагов

Регистры управления процессором

- CR0 - флаги управления системой
 - PG - включение режима страничной адресации
 - управление отдельными параметрами кеша
 - WP - запрет записи в страницы "только для чтения"
 - NE - ошибки FPU вызывают исключение, а не IRQ13
 - TS - устанавливается процессором после переключения задачи
 - PE - включение защищённого режима
- CR1 - зарезервирован
- CR2 - регистр адреса ошибки страницы - содержит линейный адрес страницы, при обращении к которой произошло исключение #PF
- CR3 - регистр основной таблицы страниц
 - 20 старших бит физического адреса начала каталога таблиц либо 27 старших бит физического адреса начала таблицы указателей на каталоги страниц, в зависимости от бита PAE в CR4
 - Управление кешированием и сквозной записью страниц
- CR4 - регистр управления *новыми возможностями процессоров* (с Pentium)

Отладочные регистры

- DR0..DR3 - 32-битные линейные адреса четырёх возможных точек останова по доступу к памяти
- DR4, DR5 - зарезервированы
- DR6 (DSR) - регистр состояния отладки. Содержит причину останова
- DR7 (DCR) - регистр управления отладкой. Управляет четырьмя точками останова

Машинно-специфичные регистры

- Управление кешем
- Дополнительное управление страничной адресацией
- Регистры расширений процессора: MMX и т.д.

Системные и привилегированные команды

- Выполнение ограничено, в основном, нулевым кольцом защиты
- LDGT, SDGT
- LLDT, SLDT
- LTR, STR
- LIDT, SIDT
- MOV CR0..CR4 или DR0..DR7, <источник>
- ...

Страничная адресация - преобразование линейного адреса в физический

- Линейный адрес:
 - биты 31-22 - номер таблицы страниц в каталоге
 - биты 21-12 - номер страницы в выбранной таблице
 - биты 11-0 - смещение от **физического** адреса начала страницы в памяти
- Каждое обращение к памяти требует двух дополнительных обращений!
- Необходим специальный кеш страниц - TLB
- Каталог таблиц/таблица страниц:
 - биты 31-12 - биты 31-12 физического адреса таблицы страниц либо самой страницы
 - утрибуты управления страницей

Механизм защиты

- Механизм защиты - ограничение доступа к сегментам или страницам в зависимости от уровня привилегий
- К типам сегментов реального режима (код, стек, данные) добавляется TSS - сегмент состояния задачи. В нём сохраняется вся информация о задаче на время приостановки выполнения. Размер - 68h байт.
- Структура:
 - селектор предыдущей задачи
 - Регистры стека 0, 1, 2 уровней привилегий
 - EIP, EFLAGS, EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, CS, DS, ES, FS, HS, SS, LDTR
 - флаги задачи
 - битовая карта ввода-вывода (контроль доступа программы к устройствам)

64-разрядные процессоры (x86-64)

AMD - с 2001, Intel - с 2003

■ Режимы работы:

- Legacy mode - совместимость с 32-разрядными процессорами
- Long mode – 64-разрядный режим с частичной поддержкой 32-разрядных программ. Рудименты V86 и сегментной модели памяти упразднены

■ Регистры:

- *целочисленные 64-битных регистры общего назначения - RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP;*
- *новые целочисленные 64-битных регистры общего назначения R8 — R15*
- *64-битный указатель RIP и 64-битный регистр флагов RFLAGS.*



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 6

"Математический сопроцессор. Расширения процессоров x86"

ИУ7, 4-й семестр, 2020 г.

Сопроцессор (FPU – Floating Point Unit)

- Изначально - отдельное опциональное устройство на материнской плате, с 80486DX встроен в процессор
- Операции над 7-ю типами данных
 - целое слово (16 бит)
 - короткое целое (32 бита)
 - длинное слово (64 бита)
 - упакованное десятичное (80 бит)
 - короткое вещественное (32 бита)
 - длинное вещественное (64 бита)
 - расширенное вещественное (80 бит)

Форма представления числа с плавающей запятой в FPU

- Нормализованная форма представления числа ($1, \dots * 2^{\text{exp}}$)
- Экспонента увеличена на константу для хранения в положительном виде
- Пример представления 0,625 в коротком вещественном типе:
 - $1/4 + 1/8 = 0,101b$
 - $1,01b * 2^{-1}$
 - Бит 31 - знак мантииссы, 30-23 - экспонента, увеличенная на 127, 22-0 - мантиисса без первой цифры
 - **001111110**010000000000000000000000
- Все вычисления FPU - в расширенном 80-битном формате

Особые числа FPU

- Положительная бесконечность: знаковый - 0, мантисса - нули, экспонента - единицы
- Отрицательная бесконечность: знаковый - 1, мантисса - нули, экспонента - единицы
- NaN (Not a Number):
 - *qNaN (quiet)* - при приведении типов/отдельных сравнениях
 - *sNaN (signal)* - переполнение в большую/меньшую сторону, прочие ошибочные ситуации
- Денормализованные числа (экспонента = 0): находятся ближе к нулю, чем наименьшее представимое нормальное число

Регистры FPU

- R0..R7, адресуются не по именам, а рассматриваются в качестве стека ST. ST соответствует регистру - текущей вершине стека, ST(1)..ST(7) - прочие регистры
- SR - регистр состояний, содержит слово состояния FPU. Сигнализирует о различных ошибках, переполнениях
- CR - регистр управления. Контроль округления, точности
- TW – 8 пар битов, описывающих состояния регистров: число, ноль, не-число, пусто
- FIP, FDP - адрес последней выполненной команды и её операнда для обработки исключений

Исключения FPU

- Неточный результат - произошло округление по правилам, заданным в CR. Бит в SR хранит направление округления
- Антипереполнение - переход в денормализованное число
- Переполнение - переход в "бесконечность" соответствующего знака
- Деление на ноль - переход в "бесконечность" соответствующего знака
- Денормализованный операнд
- Недействительная операция

Команды пересылки данных FPU

- FLD - загрузить вещественное число из источника (переменная или ST(n)) в стек. Номер вершины в SR увеличивается
- FST/FSTP - скопировать/считать число с вершины стека в приёмник
- FILD - преобразовать целое число из источника в вещественное и загрузить в стек
- FIST/FISTP - преобразовать вершину в целое и скопировать/считать в приёмник
- FBLD, FBSTP - загрузить/считать десятичное BCD-число
- FXCH - обменять местами два регистра (вершину и источник) стека

Базовая арифметика FPU

- FADD, FADDP, FIADD - сложение, сложение с выталкиванием из стека, сложение целых. Один из операндов - вершина стека
- FSUB, FSUBP, FISUB - вычитание
- FSUBR, FSUBRP, FISUBR - обратное вычитание (приёмника из источника)
- FMUL, FMULP, FIMUL - умножение
- FDIV, FDIVP, FIDIV - деление
- FDIVR, FDIVRP, FIDIVR - обратное деление (источника на приёмник)
- FPREM - найти частичный остаток от деления (делится ST(0) на ST(1)). Остаток ищется цепочкой вычитаний, до 64 раз

Базовая арифметика FPU (продолжение)

- FABS - взять модуль числа
- FCHS - изменить знак
- FRNDINT - округлить до целого
- FSCALE - масштабировать по степеням двойки ($ST(0)$ умножается на $2^{ST(1)}$)
- FXTRACT - извлечь мантиссу и экспоненту. $ST(0)$ разделяется на мантиссу и экспоненту, мантисса дописывается на вершину стека
- FSQRT - вычисляет квадратный корень $ST(0)$

Команды сравнения FPU

- FCOM, FCOMP, FCOMPP - сравнить и вытолкнуть из стека
- FUCOM, FUCOMP, FUCOMPP - сравнить без учёта порядков и вытолкнуть
- FICOM, FICOMP, FICOMP - сравнить целые
- FCOMI, FCOMIP, FUCOMI, FUCOMIP (P6)
- FTST - сравнивает с нулём
- FXAM - выставляет флаги в соответствии с типом числа

Трансцендентные операции FPU

- FSIN
- FCOS
- FSINCOS
- FPTAN
- FPATAN
- F2XM1 – $2^x - 1$
- FYL2X, FYL2XP1 – $y * \log_2 x$, $y * \log_2(x+1)$

Константы FPU

- FLD1 – 1,0
- FLDZ – +0,0
- FLDPi – число Π
- FLDL2E – $\log_2 e$
- FLDL2T – $\log_2 10$
- FLDLN2 – $\ln(2)$
- FLDLG2 – $\lg(2)$

Команды управления FPU

- FINCSTP, FDECSTP - увеличить/уменьшить указатель вершины стека
- FFREE - освободить регистр
- FINIT, FNINIT - инициализировать сопроцессор / инициализировать без ожидания (очистка данных, инициализация CR и SR по умолчанию)
- FCLEX, FNCLEX - обнулить флаги исключений / обнулить без ожидания
- FSTCW, FNSTCW - сохранить CR в переменную / сохранить без ожидания
- FLDCW - загрузить CR
- FSTENV, FNSTENV – сохранить вспомогательные регистры (14/28 байт) / сохранить без ожидания
- FLDENV - загрузить вспомогательные регистры
- FSAVE, FNSAVE, FXSAVE - сохранить состояние (94/108 байт) и инициализировать, аналогично FINIT
- FRSTOR, FXRSTOR - восстановить состояние FPU
- FSTSW, FNSTSW - сохранение CR
- WAIT, FWAIT - обработка исключений
- FNOP - отсутствие операции

Команда CPUID (с 80486)

Идентификация процессора

- Если EAX = 0, то в EAX - максимальное допустимое значение (1 или 2), а EBX:ECX:EDX – 12-байтный идентификатор производителя (ASCII-строка).
- Если EAX = 1, то в EAX - версия, в EDX - информация о расширениях
 - EAX - модификация, модель, семейство
 - EDX: наличие FPU, поддержка V86, поддержка точек останова, CR4, PAE, APIC, быстрые системные вызовы, PGE, машинно-специфичный регистр, CMOVss, **MMX**, **FXSR (MMX2)**, **SSE**
- Если EAX = 2, то в EAX, EBX, ECX, EDX возвращается информация о кэшах и TLB

MMX (1997, Pentium MMX)

Увеличение эффективности обработки больших потоков данных (изображения, звук, видео...) - выполнение простых операций над массивами однотипных чисел.

- 8 64-битных регистров MM0..MM7 - **мантиссы регистров FPU**. При записи в MMn экспонента и знаковый бит заполняются единицами
- Пользоваться одновременно и FPU, и MMX не получится, требуется FSAVE+FRSTOR
- Типы данных MMX:
 - учетверённое слово (64 бита);
 - упакованные двойные слова (2);
 - упакованные слова (4);
 - упакованные байты (8).
- Команды MMX перемещают упакованные данные в память или обычные регистры целиком, но арифметические и логические операции выполняют поэлементно.
- *Насыщение* - замена переполнения/антипереполнения превращением в максимальное/минимальное значение

Команды пересылки данных MMX

- MOVD, MOVQ - пересылка двойных/четверённых слов
- PACKSSWB, PACKSSDW - упаковка со знаковым насыщением слов в байты/двойных слов в слова. *Приёмник -> младшая половина приёмника, источник -> старшая половина приёмника*
- PACKUSWB - упаковка слов в байты с беззнаковым насыщением
- PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ - распаковка и объединение старших элементов источника и приёмника через 1

Арифметические операции MMX

- PADDB, PADDW, PADDD - поэлементное сложение, перенос игнорируется
- PADDSB, PADDSW - сложение с насыщением
- PADDSB, PADDSW - сложение с насыщением
- PADDUSB, PADDUSW - беззнаковое сложение с насыщением
- PSUBB, PSUBW, PDUBD - вычитание, заём игнорируется
- PSUBSB, PSUBSW - вычитание с насыщением
- PSUBUSB, PSUBUSW - беззнаковое вычитание с насыщением
- PMILHW, PMULLW - старшее/младшее умножение (сохраняет старшую или младшую части результата в приёмник)
- PMADDWD - умножение и сложение. Перемножает 4 слова, затем попарно складывает произведения двух старших и двух младших

Команды сравнения MMX

- PCMPREQB, PCMPREQW, PCMPREQD - проверка на равенство. Если пара равна - соответствующий элемент приёмника заполняется единицами, иначе - нулями
- PCMPGTB, PCMPGTW, PCMPGTD - сравнение. Если элемент приёмника больше, то заполняется единицами, иначе - нулями

Логические операции MMX

- PAND - логическое И
- PANDN - логическое НЕ-И (штрих Шеффера) (источник*НЕ(приёмник))
- POR - логическое ИЛИ
- PXOR - исключающее ИЛИ

Сдвиговые операции MMX

- PSLLW, PSLLD, PSLLQ - логический влево
- PSRLW, PSRLD, PSRLQ - логический вправо
- PSRAW, PSRAD - арифметический вправо

Расширение SSE (Pentium III, 1999)

Решение проблемы параллельной работы с FPU

- 8 128-разрядных регистров
- Свой регистр флагов
- Основной тип - вещественные одинарной точности (32 бита)
- Целочисленные команды работают с регистрами MMX
- Команды:
 - Пересылки
 - Арифметические
 - Сравнения
 - Преобразования типов
 - Логические
 - Целочисленные
 - Упаковки
 - Управления состоянием
 - Управления кэшированием
- Развитие: SSE2, SSE3...

Расширение AES (Intel Advanced Encryption Standard New Instructions; AES-NI, 2008)

Цель - ускорение шифрования по алгоритму AES

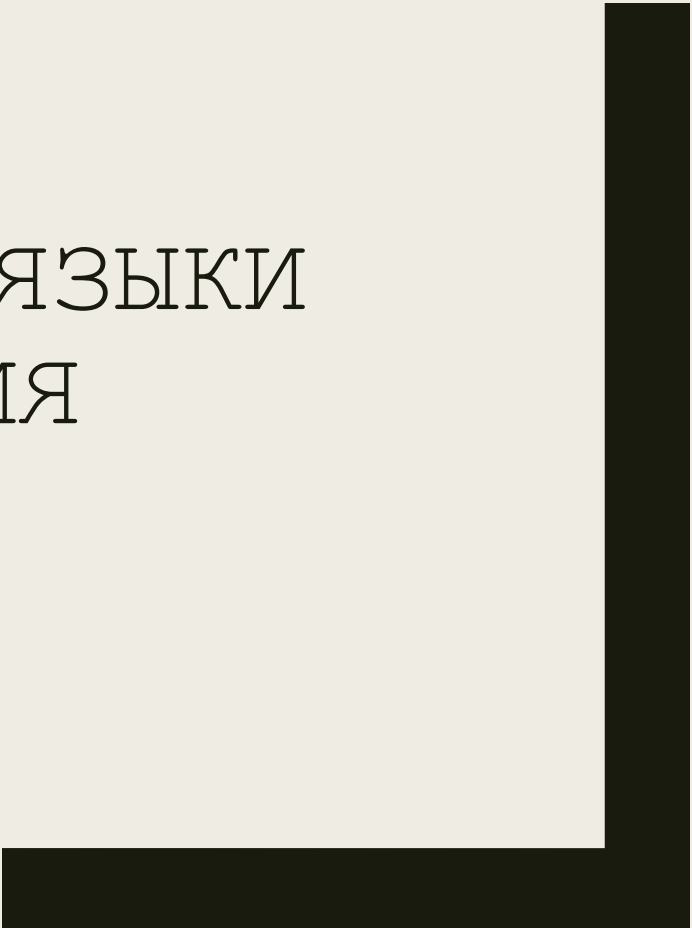
■ Команды:

- раунда шифрования;
- раунда расшифровывания;
- способствования генерации ключа



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 7
"Макроопределения"
ИУ7, 4-й семестр, 2020 г.



Макроопределения

Макроопределение (макрос) - именованный участок программы, который ассемблируется каждый раз, когда его имя встречается в тексте программы.

- Определение:

имя MACRO параметры

.....

ENDM

- Пример:

load_reg MACRO register1, register2

push register1

pop register2

ENDM

Директива присваивания =

Директива присваивания служит для создания целочисленной макропеременной или изменения её значения и имеет формат:

Макроимя = Макровыражение

- Макровыражение (или Макровыражение, или Константное выражение) - выражение, вычисляемое препроцессором, которое может включать целочисленные константы, макроимена, вызовы макрофункций, знаки операций и круглые скобки, результатом вычисления которого является целое число
- Операции: арифметические (+, -, *, /. MOD), логические, сдвигов, отношения

Директивы отождествления EQU, TEXTEQU

Директива для представления текста и чисел:

Макроимя EQU нечисловой текст и не макроимя ЛИБО число

Макроимя EQU <Операнд>

Макроимя TEXTEQU Операнд

■ Пример:

X EQU [EBP+8]

MOV ESI,X

Макрооперации

- `%` – вычисление выражение перед представлением числа в символьной форме
- `<>` – подстановка текста без изменений
- `&` – склейка текста
- `!` – считать следующий символ текстом, а не знаком операции
- `::` – исключение строки из макроса

Блоки повторения

- REPT число ... ENDM - повтор фиксированное число раз

- IRP или FOR:

IRP form,<fact_1[,fact_2,...]> ... ENDM

Подстановка фактических параметров по списку на место формального

- IRPC или FORC:

IRPC form,fact ... ENDM

Подстановка символов строки на место формального параметра

- WHILE:

WHILE cond ... ENDM

Директивы условного ассемблирования

- IF:
IF c1
...
ELSEIF c2
...
ELSE
...
ENDIF
- IFB <par> - истинно, если параметр не определён
- IFNB <par> - истинно, если параметр определён
- IFIDN <s1>,<s2> - истинно, если строки совпадают
- IFDIF <s1>,<s2> - истинно, если строки разные
- IFDEF/IFNDEF <name> - истинно, если имя объявлено/не объявлено

Директивы управления листингом

- Листинг - файл, формируемый компилятором и содержащий текст ассемблерной программы, список определённых меток, перекрёстных ссылок и сегментов.
- `TITLE`, `SUBTTL` - заголовок, подзаголовок на каждой странице
- `PAGE` высота, ширина
- `NAME` - имя программы
- `.LALL` - включение полных макрорасширений, кроме `;;`
- `.XALL` - по умолчанию
- `.SALL` - не выводить тексты макрорасширений
- `.NOLIST` - прекратить вывод листинга

Комментарии

comment @

... многострочный текст ...

@



МАШИННО-ЗАВИСИМЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Лекция 8
"Не-x86 ассемблеры"
ИУ7, 4-й семестр, 2020 г.

RISC-архитектура

Ранние архитектуры процессоров (комплексные, CISC (Complex instruction set computer)):

- большее количество команд
- разные способы адресации для упрощения написания программ на ассемблере
- поддержка конструкций языков высокого уровня

Недостатки: на практике многие возможности CISC используются компиляторами ЯВУ ограниченно, а их поддержка затратна.

RISC (*reduced instruction set computer*):

- сведение набора команд к простым типовым
- большее количество регистров (возможно за счёт общего упрощения архитектуры)
- стандартизация формата команд, упрощение конвейеризации

Семейство процессоров ARM

Свыше 90% рынка процессоров для мобильных устройств

ARMv1 – 1983 г.

Современные - ARMv7, ARMv8.

Регистры общего назначения ARMv7:

- R0-R12
- R13 – SP
- R14 – LR (регистр связи)
- R15 – PC (счётчик команд)

Регистры R8–R12 существуют в двух экземплярах:

- для режима обработки быстрого прерывания
- для остальных режимов

Регистры LR и SP для каждого режима свои (6-7 пар)

Режимы ARM

- User mode — обычный режим выполнения программ. В этом режиме выполняется большинство программ.
- Fast Interrupt (FIQ) — режим быстрого прерывания (меньшее время срабатывания).
- Interrupt (IRQ) — основной режим прерывания.
- System mode — защищённый режим для использования операционной системой.
- Abort mode — режим, в который процессор переходит при возникновении ошибки доступа к памяти (доступ к данным или к инструкции на этапе prefetch конвейера).
- Supervisor mode — привилегированный пользовательский режим.
- Undefined mode — режим, в который процессор входит при попытке выполнить неизвестную ему инструкцию

Наборы команд ARM

- Базовый
- Thumb (16-разрядные, более производительные)
- Thumb2 (32-разрядные)

Команды ветвления B, BL, BLX

- B (Branch) - переход
- BL (Branch with link) - переход с сохранением адреса возврата в LR
- BLX - переход с переключением системы команд

Архитектура VLIW. Эльбрус-8С

VLIW (*very large instruction word*) - продолжение идей RISC для многопроцессорных систем

- В каждой инструкции явно указывается, что должно делать каждое ядро процессора

Эльбрус-8С:

- 8 ядер
- В каждом ядре - 6 арифметико-логических каналов со своими АЛУ и FPU, до 24 операций за такт
- Спецификация опубликована 30.05.2020

• Широкая команда Эльбруса

Широкая команда - набор элементарных операций, которые могут быть запущены на исполнение в одном такте.

Доступны:

- 6 АЛУ (возможности различны)
- Устройство передачи управления
- 3 устройства для операций над предикатами
- 6 квалифицирующих предикатов
- 4 устройства асинхронного для команд чтения данных
- 4 32-разрядных литерала для констант

Определяющие свойства архитектуры "Эльбрус"

- Регистровый файл (рабочие регистры) - 256 регистров (32 для глобальных данных и 224 для стека процедур)
 - механизм регистровых окон: вызывающая подпрограмма выделяет вызываемой область в своём регистровом окне; на начало указывает регистр WD
 - пространство регистров подвижной базы - пространство в текущем окне, на начало указывает регистр BR
- Предикатный файл - 32 регистра со значениями true/false
- Подготовка передачи управления (disp) - подготовка к переходам при ветвлении для исключения задержек
- Асинхронный доступ к массивам

Java. Java virtual machine (JVM)

Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины.

Байт-код Java — набор инструкций, исполняемых виртуальной машиной Java. Каждый код операции байт-кода — один байт.

Группы инструкций:

- загрузка и сохранение (например, ALOAD_0, ISTORE),
- арифметические и логические операции (например, IADD, FCMPL),
- преобразование типов (например, I2B, D2I),
- создание и преобразование объекта (например, NEW, PUTFIELD),
- управление стеком (например, DUP, POP),
- операторы перехода (например, GOTO, IFEQ),
- вызовы методов и возврат (например, INVOKESTATIC, IRETURN).

Платформа .NET. CLR, CIL

.NET (2002) - платформа, основанная на CLR (Common Language Runtime, общезыковая исполняющая среда).

CLR — исполняющая среда для байт-кода CIL (MSIL), в которой компилируются программы, написанные на .NET-совместимых языках программирования.

CIL (Common Intermediate Language) — «высокоуровневый ассемблер» виртуальной машины .NET., основанный на работе со стеком.

```
ldloc.0      // push local variable 0 onto stack
ldloc.1      // push local variable 1 onto stack
add          // pop and add the top two stack items then push the
result onto the stack
stloc.0      // pop and store the top stack item to local variable
0
```

WebAssembly (wasm)

WebAssembly — это бинарный формат инструкций для стековой виртуальной машины, предназначенной для компиляции программ на ЯВУ для WEB.

Исходный код на C	«линейный ассемблерный байт-код»	бинарный код WASM
<pre>int factorial(int n) { if (n == 0) return 1; else return n * factorial(n-1); }</pre>	<pre>get_local 0 i64.eqz if i64 i64.const 1 else get_local 0 get_local 0 i64.const 1 i64.sub call 0 i64.mul end</pre>	<pre>20 00 50 04 7e 42 01 05 20 00 20 00 42 01 7d 10 00 7e 0b</pre>