

Машинно-зависимые языки программирования, лекция 1

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2025 г.



Организация курса

- 2 модуля + экзамен
- 8 лекций, 12 лабораторных работ
- 76 часов самостоятельной подготовки (по учебному плану)

Литература

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



Цели и программа курса

- изучение низкоуровневого устройства ЭВМ;
- понимание исполнения программ на аппаратном уровне, высокоуровневого устройства и работы процессора;
- умение составлять и читать программы на языках низкого уровня, включая:
 - написание программы на низкоуровневом языке “с нуля”;
 - взаимодействие программного кода с устройствами;
 - использование расширений процессоров;
 - отладку и реверс-инжиниринг исполняемых файлов.

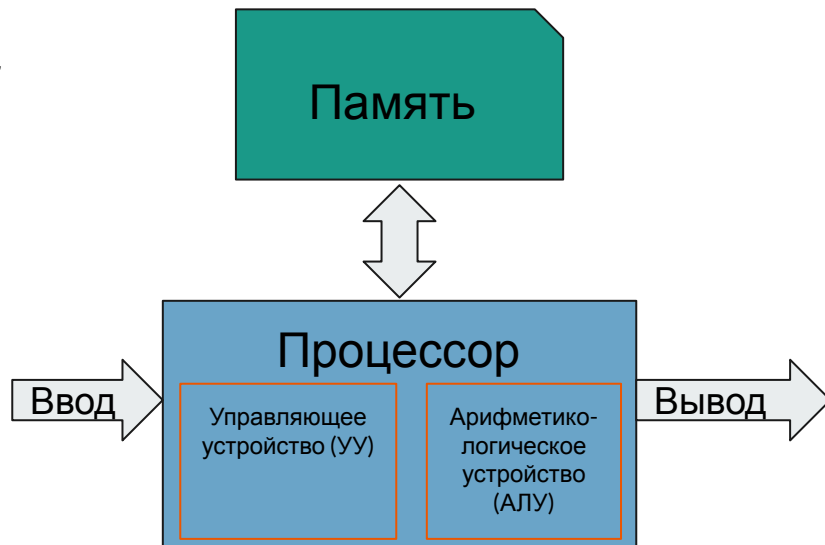
История создания ЭВМ. Появление вычислителей общего назначения.

Архитектура фон Неймана

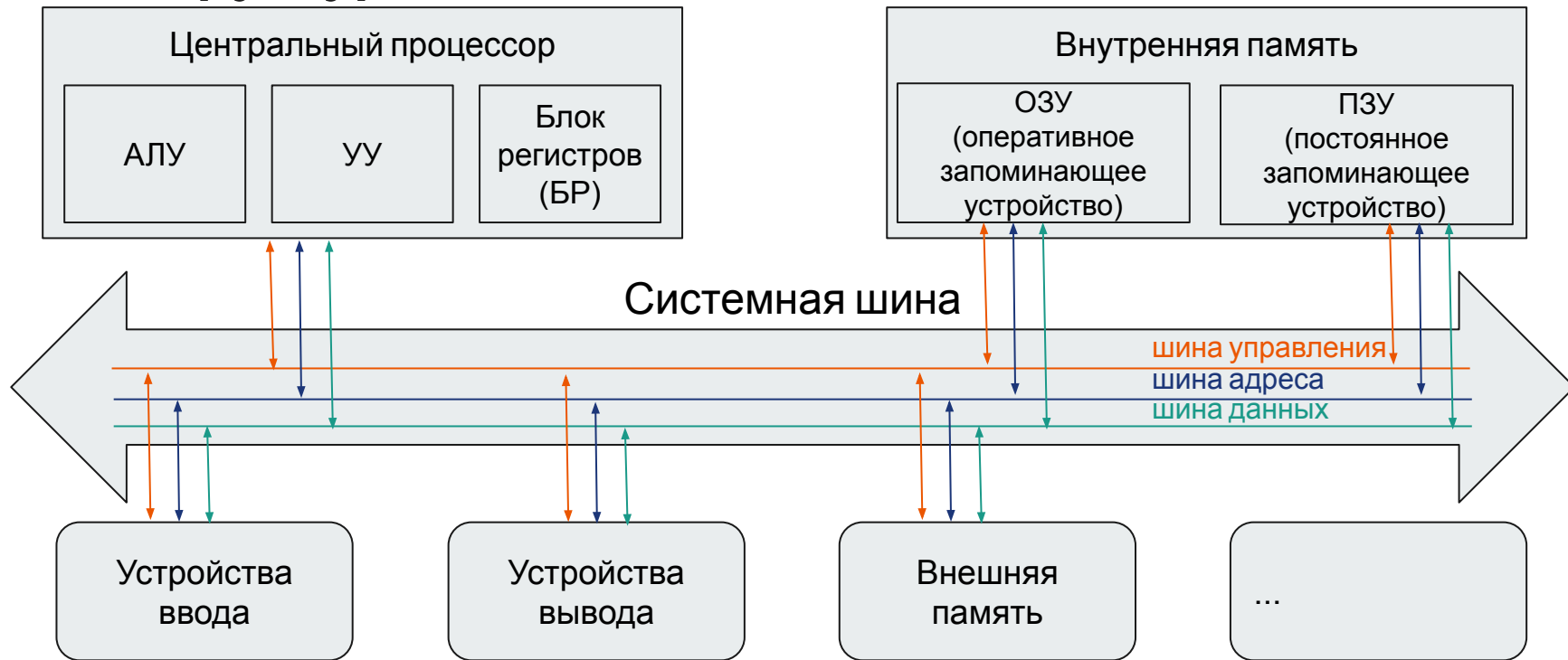
От решения частных вычислительных задач - к универсальным системам

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.

Минимальная адресуемая единица памяти - байт:

- 8 бит
- $2^8=256$ значений (0..255)
- $8 = 2^3$
- $256 = 2^8=10_{16}^2=100_{16}$

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

Параграф - 16 байт

ASCII (ackú) - 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 (в расширенном варианте - 8-битная)
- первые 32 символа - непечатные (служебные)
- старшие 128 символов 8-битной кодировки - национальные языки, псевдографика и т. п.



Системы счисления

Двоичная (binary)

- 0, 1, 10, 11, 100, 101...
- $2^8 = 256$
- $2^{10} = 1024$
- $2^{16} = 65536$
- Суффикс - b. Пример: 1101b

Шестнадцатеричная (hexadecimal)

- 0, 1, ..., 8, 9, A, B, C, D, E, F, 10, 11, 12, ..., 19, 1A, 1B, ...
- $2^4 = 10_{16}$
- $2^8 = 100_{16}$
- $2^{16} = 10000_{16}$
- Суффикс - h (10h - 16). Некоторые компиляторы требуют префикса 0x (0x10)

$$1011011011111000_2 = B6F8_{16}$$



Представление отрицательных чисел

Знак - в старшем разряде (0 - "+", 1 - "-").

Возможные способы:

- прямой код
- обратный код (инверсия)
- **дополнительный код (инверсия и прибавление единицы)**

Примеры доп. кода на 8-разрядной сетке

-1:

1. 00000001
2. 11111110
3. 11111111

Смысл: $-1 + 1 = 0$ (хоть и с переполнением):

$$11111111 + 1 = (1)\underline{00000000}$$

-101101:

1. 00101101
2. 11010010
3. 11010011



Виды современных архитектур ЭВМ

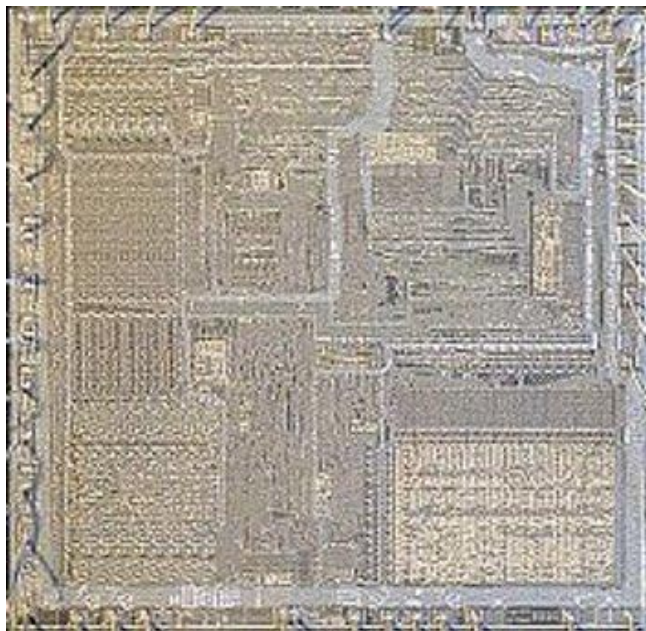
- x86-64: 8086 (16-разр.) ➤ x86 (32-разр.) ➤ x86-64 (64-разр.)
- ARM
- IA64
- MIPS (включая Байкал)
- VLIW (например, Эльбрус)

Семейство процессоров x86 и x86-64

- 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., добавлено несколько команд, интегрированы микросхемы поддержки
- 80286 - 1982 г., 16-разрядный, добавлен защищённый режим
- 80386, 80486, Pentium, Celeron, AMD, ... - 32-разрядные, повышение быстродействия и расширение системы команд
- x86-64 (x64) - семейство с 64-разрядной архитектурой
- Отечественный 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	BHE/S7	
AD8	8	33	MN/MX	
AD7	9	32	RD	
AD6	10	31	RQ/GT0	(HOLD)
AD5	11	30	RQ/GT1	(HLDA)
AD4	12	29	LOCK	(WR)
AD3	13	28	S2	(M/IO)
AD2	14	27	S1	(DT/R)
AD1	15	26	S0	(DEN)
AD0	16	25	QS0	(ALE)
NMI	17	24	QS1	(INTA)
INTR	18	23	TEST	
CLK	19	22	READY	
GND	20	21	RESET	

Архитектура 8086 с точки зрения программиста (структура блока регистров)





Язык ассемблера

Машинная команда - инструкция (в двоичном коде) из аппаратно определённого набора, которую способен выполнять процессор.

Машинный код - система команд конкретной вычислительной машины, которая интерпретируется непосредственно процессором.

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



Операционная система. DOS

Операционная система — программное обеспечение, управляющее компьютерами (включая микроконтроллеры) и позволяющее запускать на них прикладные программы.

DOS (disk operating system, дисковая операционная система) - семейство операционных систем для мейнфреймов, начиная с IBM System/360, и ПК. Наиболее распространённая разновидность для ПК — MS-DOS.


Основные свойства:

- отсутствие ограничений для прикладных программ на вмешательство в работу ОС и доступ к периферийному оборудованию;
- однозадачность;
- текстовый режим работы.



Исполняемые файлы. Компиляция. Линковка

- Исполняемый файл - файл, содержащий программу в виде, в котором она может быть исполнена компьютером (то есть в машинном коде).
- Получение исполняемых файлов обычно включает в себя 2 шага: компиляцию и линковку.
- 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

- В DOS и Windows - расширения .EXE и .COM
- Последовательность запуска 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 (загрузка библиотек).
 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 или удалять контрольные точки или условия остановки.



“Простейший” формат исполняемого файла

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

- не имеет заголовка;
- состоит из одной секции, не превышающей 64 Кб;
- загружается в ОЗУ без изменений;
- начинает выполняться с 1-го байта (точка входа всегда в начале).

Последовательность запуска COM-программы:

1. Система выделяет свободный *сегмент* памяти нужного размера и заносит его адрес во все сегментные регистры (CS, DS, ES, FS, GS, SS).
2. В первые 256 (100h) байт этого сегмента записывается служебная структура DOS, описывающая программу - PSP.
3. Непосредственно за ним загружается содержимое COM-файла без изменений.
4. Указатель стека (регистр SP) устанавливается на конец сегмента.
5. В стек записывается 0000h (начало PSP - адрес возврата для возможности завершения командой ret).
6. Управление передаётся по адресу CS:0100h.



Классификация команд процессора 8086

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



Команда пересылки данных MOV

MOV <приёмник>, <источник>

Источник: непосредственный операнд (константа, включённая в машинный код), 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



Команда безусловной передачи управления JMP

JMP <операнд>

- Передаёт управление в другую точку программы (на другой адрес памяти), не сохраняя какой-либо информации для возврата.
- Операнд - непосредственный адрес (вставленный в машинный код), адрес в регистре или адрес в переменной.



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*



Примеры других команд

- ADD <приёмник>, <источник>, SUB <приёмник>, <источник>
- MUL <источник>, DIV <источник>
- AND <приёмник>, <источник>
- OR <приёмник>, <источник>
- XOR <приёмник>, <источник>
- NOT <приёмник>

Пример

...

XOR AX, AX

MOV BX, 5

label1:

INC AX

ADD BX, AX

JMP label 1



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 внешней средой (ОС, пользователь, ...)

Прерывания - аппаратный механизм для приостановки выполнения текущей программы и передачи управления специальной программе - обработчику прерывания.

Основные виды:

- аппаратные
- программные

int <номер> - вызов (генерация прерывания)

21h - прерывание DOS, предоставляет прикладным программам около 70 различных функций (ввод, вывод, работа с файлами, завершение программы и т.д.)

Номер функции прерыванию 21h передаётся через регистр АН. Параметры для каждой функции передаются собственным способом, он описан в документации. Там же описан способ возврата результата из функции в программу.



Память в реальном режиме работы процессора

Реальный режим работы - режим совместимости современных процессоров с 8086.


Доступен 1 Мб памяти (2^{20} байт), то есть разрядность шины адреса - 20 разрядов.

Физический адрес формируется из двух частей: **сегмента** и **смещения**. Сегментом называется условно выделенная область адресного пространства определённого размера, а смещением — адрес ячейки памяти относительно начала сегмента. Базой сегмента называется адрес, который указывает на начало сегмента в адресном пространстве.



Структура памяти программы. Виды сегментов. Назначение отдельных сегментных регистров

- Сегмент кода — область памяти, содержащая код программы. За её адресацию отвечает регистр CS. Командой MOV изменить невозможно, меняется автоматически по мере выполнения команд.
- Сегмент данных — область памяти, содержащая переменные и константы программы. Основным регистром для адресации к сегменту данных — DS, при необходимости дополнительных сегментов данных задействуются ES, FS, GS.
- Сегмент стека. Для адресации стека используется регистр SS.



Машинно-зависимые языки программирования, лекция 2

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2025 г.

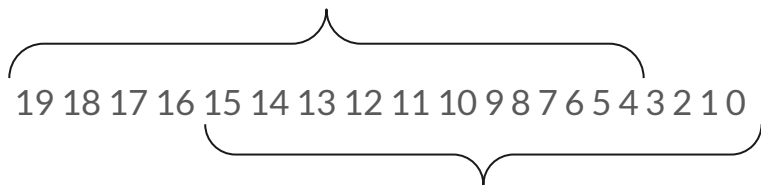


Логическая структура памяти. Сегменты

- Сегмент кода (регистр CS)
- Сегменты данных (основной регистр - DS, для дополнительных сегментов - ES, FS, GS)
- Сегмент стека (регистр SS)

Память в реальном режиме работы процессора - пример

Номер параграфа начала сегмента



[SEG]:[OFFSET] => физический адрес:

1. SEG необходимо побитово сдвинуть на 4 разряда влево (или умножить на 16, что тождественно)
2. К результату прибавить OFFSET

5678h:1234h =>

$$\begin{array}{r} 56780 \\ + 1234 \\ \hline 579B4 \end{array}$$

Вычисление физического адреса выполняется процессором аппаратно, без участия программиста.

Распространённые пары регистров: CS:IP, DS:BX, SS:SP

Память 8086 (20-разрядная адресация)

00000	
00001	
00002	
00003	
00004	
00005	
00006	
00007	
00008	
00009	
0000A	
0000B	
0000C	
0000D	
0000E	
0000F	

Параграф 0

00010	
00011	
00012	
00013	
00014	
00015	
00016	
00017	
00018	
00019	
0001A	
0001B	
0001C	
0001D	
0001E	
0001F	

Параграф 1

...

FFFF0	
FFFF1	
FFFF2	
FFFF3	
FFFF4	
FFFF5	
FFFF6	
FFFF7	
FFFF8	
FFFF9	
FFFFA	
FFFFB	
FFFC	
FFFD	
FFFE	
FFFF	

Параграф FFFF

Сегментная модель памяти 8086

Максимальный размер
сегмента с начальной
сегментной частью адреса
8AFE

00000	
00001	
...	
00010	
00011	
...	
8AFE0	
8AFE1	
8AFE2	
...	
9AFDF	
9AFE0	
...	
FFFFF	

$8AFE:0000 = 8AFE0$

$8AFE:001F = 8AFFF$

$8AFE:1000 = 8BFE0$

$8AFE:FFFF = 8AFE0 + 10000 - 1 = 9AFDF$



Целочисленная арифметика (основные команды)

- ADD <приёмник>, <источник> - выполняет арифметическое сложение приёмника и источника. Сумма помещается в приёмник, источник не изменяется.
- SUB <приёмник>, <источник> - арифметическое вычитание источника из приёмника.
- MUL <источник> - беззнаковое умножение. Умножаются источник и AL/AX/EAX/RAX, в зависимости от размера источника. Результат помещается в AX либо DX:AX/EDX:EAX/RDX:RAX.
- IMUL <источник>; IMUL <приёмник>, <источник>; IMUL <приёмник>, <источник1>, <источник2> - знаковое умножение
- DIV <источник> - целочисленное беззнаковое деление. Делится AL/AX/EAX/RAX на источник. Результат помещается в AL/AX/EAX/RAX, остаток - в AH/DX/EDX/RDX.
- IDIV <источник> - знаковое деление
- INC <приёмник> - инкремент на 1
- DEC <приёмник> - декремент на 1



Побитовая арифметика (основные команды)

- 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



Структура программы на ассемблере

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

- Модули (файлы исходного кода)
 - Сегменты (описание блоков памяти)
 - команды процессора;
 - инструкции описания структур данных, выделения памяти для переменных и констант;
 - макроопределения.

Полный формат строки:

метка команда / директива операнды ; комментарий



Метки

В коде

- Пример:

```
mov cx, 5

label1:

    add ax, bx

    loop label1
```

- Метки обычно используются в командах передачи управления

В данных

- `label`
 - *метка label тип*
 - Возможные типы: BYTE, WORD, DWORD, FWORD, QWORD, TBYTE, NEAR, FAR.
- `EQU, =`
 - *label EQU выражение*
 - макрос
 - вычисляет выражение в правой части и приравнивает его метке

Директивы выделения памяти

- Директива - инструкция ассемблеру, влияющая на процесс компиляции и не являющаяся командой процессора. Обычно не оставляет следов в формируемом машинном коде.
- Псевдокоманда - директива ассемблера, которая приводит к включению данных или кода в программу, но не соответствующая 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(?)`



Описание сегментов программы

- Любая программа состоит из сегментов
- Виды сегментов:
 - сегмент кода
 - сегмент данных
 - сегмент стека
- Описание сегмента - директива SEGMENT:

```
имя SEGMENT [READONLY] [выравнивание] [тип] [разрядность] ['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 будут расположены в памяти друг за другом.



Модели памяти

`.model` модель, язык, модификатор

- Модели:
 - 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.



Завершение описания модуля. Точка входа

.

.

.

END [точка_входа]

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



Сегментный префикс. Директива ASSUME

- Для обращения к переменной процессору необходимо знать обе составляющие адреса: и сегментную, и смещение.

Пример полной записи - DS:Var1

- Директива `ASSUME регистр:имя сегмента` устанавливает значение сегментного регистра по умолчанию

```
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
```



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

...

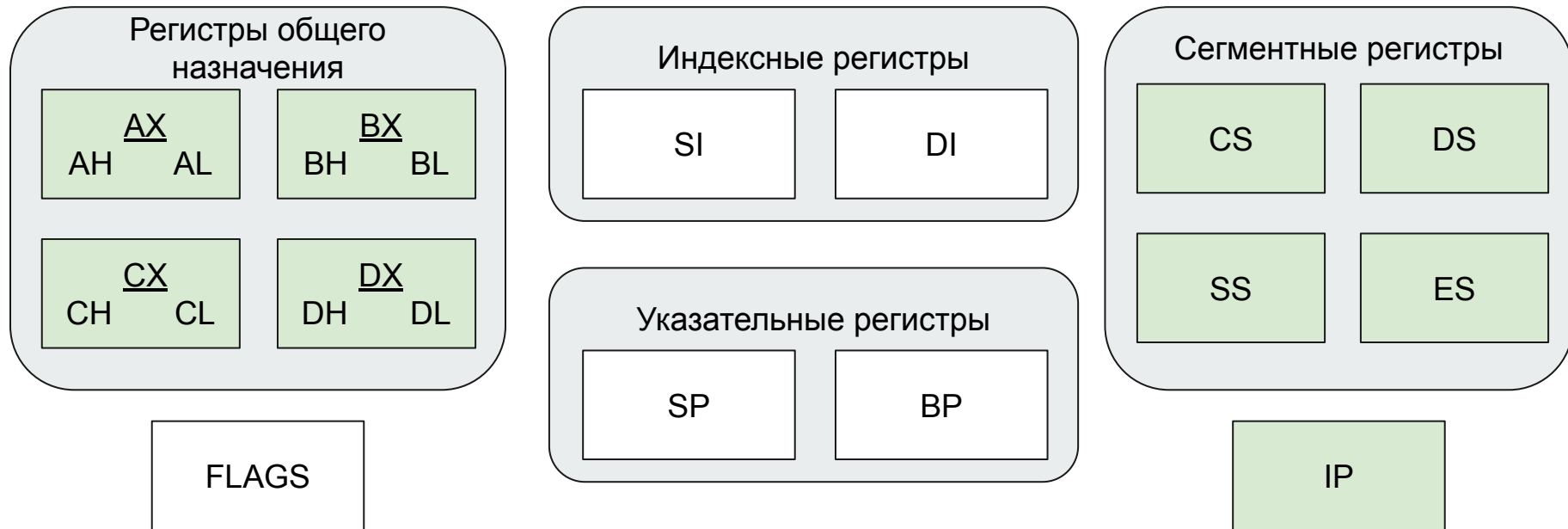
ENDIF



Виды переходов для команды JMP

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

Архитектура 8086 с точки зрения программиста (структура блока регистров)





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 С. В. Assembler для DOS, Windows, Unix, глава 2)

- непосредственная адресация (`mov ax, 2`)
- регистровая адресация (`mov ax, bx`)
- прямая адресация (`mov ax, ds:[0032]`)
- регистровая косвенная адресация (`mov ax, [bx]`)
- 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]`
- адресация с масштабированием `mov ax, [si*4]`
- адресация с масштабированием и смещением `mov ax, [bx][si*4]+10h`



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	-

Флаги состояния:

- 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
- OF (overflow flag) - флаг переполнения

Управляющий флаг:

- DF (direction flag) - флаг направления

Системные флаги:

- IF (interrupt enable flag) - флаг разрешения прерываний
- TF (trap flag) - флаг трассировки
- IOPL (I/O privilege flag) - 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 Jcc

(Зубков С. В. Assembler для DOS, Windows, Unix, глава 2)

cc - condition code

- Переход типа short или near
- Обычно используются в паре с CMP
- Термины “выше” и “ниже” - при сравнении беззнаковых чисел
- Термины “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	Если 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



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

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2025 г.



Команда TEST

TEST <приёмник>, <источник>

- Аналог AND, но результат не сохраняется
- Выставляются флаги SF, ZF, PF



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



NEG - изменение знака

NEG <приёмник>

Переводит число в дополнительный код.



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
```



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/MOVSW <приёмник>, <источник> - копирование
- CMPS/CMPSБ/CMPSW <приёмник>, <источник> - сравнение
- SCAS/SCASБ/SCASW <приёмник> - сканирование (сравнение с AL/AX)
- LODS/LODSБ/LODSW <источник> - чтение (в AL/AX)
- STOS/STOSБ/STOSW <приёмник> - запись (из AL/AX)

Пояснение принципа работы на примере команды movsb:

1. Байт копируется из памяти по адресу DS:SI в память по адресу ES:DI.
2. SI и DI инкрементируются (декрементируются, если установлен DF).



Строковые операции: префиксы повторения

Префиксы: REP/REPE/REPZ/REPNE/REP NZ

Пример копирования 10 байт из одного массива в другой:

```
mov cx, 5  
rep movsw
```



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)

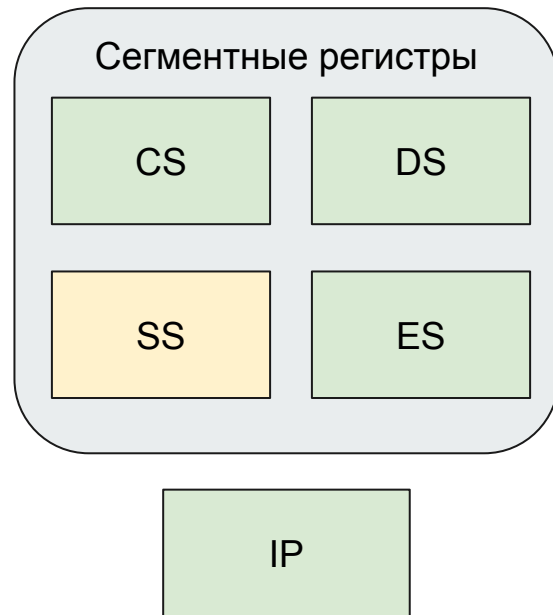
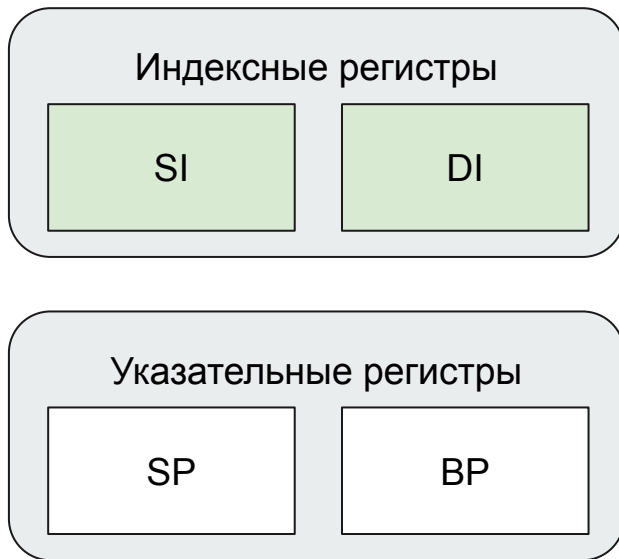
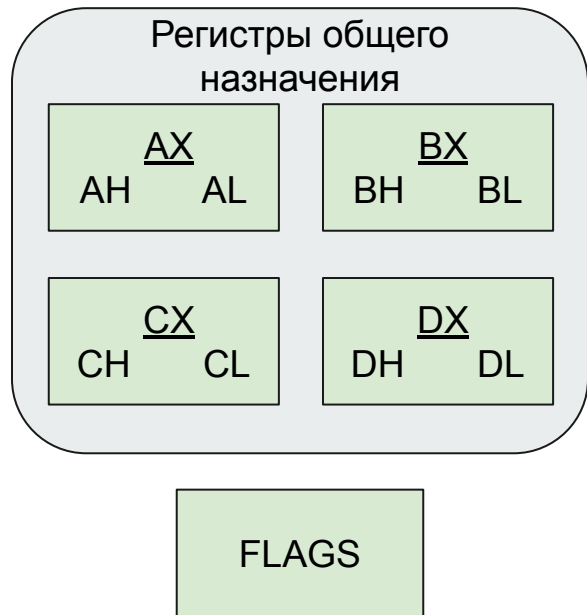


Загрузка сегментных регистров

- 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

Приёмник - регистр, источник - переменная

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)
- PUSHF - поместить в стек содержимое регистра флагов
- POPF - загрузить регистр флагов из стека



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

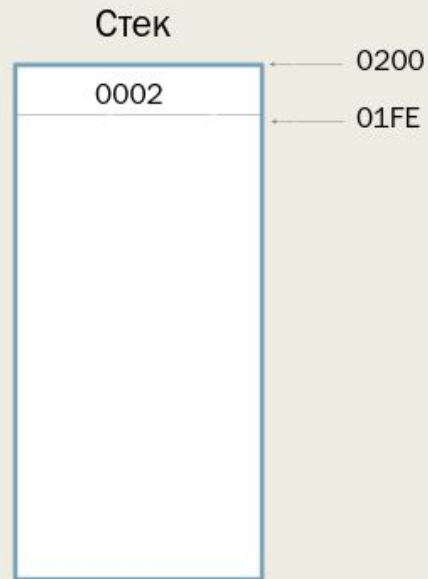


BP – base pointer

- 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

	0. SP = 0200
0000: PUSH <u>ABCDh</u> ; передача параметра	1. SP = 01FE
0002: CALL P1	2. SP = 01FC
0004: POP DX	4. SP = 0200
0006: MOV BX, AX	
...	
P1:	
0123: MOV BP, SP ; <u>ss:[bp]</u> - адрес возврата	
; <u>ss:[bp+2]</u> - параметр	
...	
0223: MOV AX, 5	
0225: RET	3. SP = 01FE

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

```
0002: CALL P1
```

```
0004: POP DX
```

```
0006: MOV BX, AX
```

444

P1:

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

```
;ss:[bp+2] - параметр
```

• • •

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

```
0225: RET
```

Стек

ABCD

0004

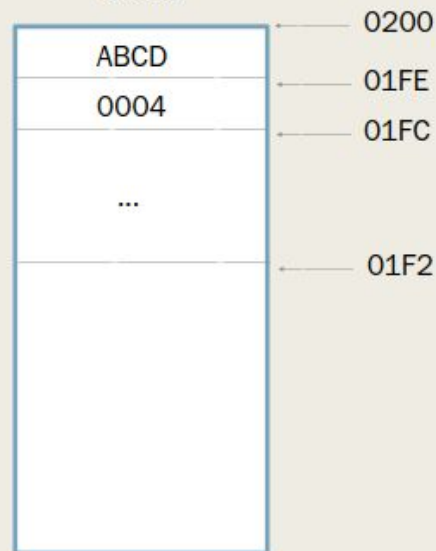
Пример вызова подпрограммы №3

```
0. SP = 0200

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

3. SP = 01F2
4. SP = 01FC
5. SP = 0200
```

Стек





Использование стека подпрограммами

Стековый кадр (фрейм) — механизм передачи аргументов и выделения временной памяти с использованием аппаратного стека. Содержит информацию о состоянии подпрограммы.

Включает в себя:

- параметры
- адрес возврата (обязательно)
- локальные переменные



Соглашения о вызовах (calling conventions)


Описания технических особенностей вызова подпрограмм, определяющие:

- способ передачи параметров подпрограммам;
- способ передачи управления подпрограммам;
- способ передачи результатов выполнения из подпрограмм в точку вызова;
- способ возврата управления из подпрограмм в точку вызова.



Распространённые соглашения

- cdecl
- pascal
- stdcall (Win32 API)
- fastcall
- thiscall - вызов нестатических методов C++ (this через ECX)
- Microsoft x64



Машинно-зависимые языки программирования, лекция 4

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2025 г.



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`.



Маскирование прерываний

Внешние прерывания, в зависимости от возможности запрета, делятся на:

- **маскируемые** — прерывания, которые можно запрещать установкой соответствующего флага;
- **немаскируемые** (англ. Non-maskable interrupt, NMI) — обрабатываются всегда, независимо от запретов на другие прерывания



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 полного адреса возврата (адреса следующей команды) - 6 байт
- 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



Установка обработчика прерывания в DOS

- int 21h
 - AH=35h, AL= номер прерывания - возвращает в ES:BX адрес обработчика (в BX 0000:[AL*4], а в ES - 0000:[AL*4+2].)
 - AH=25h, AL=номер прерывания, DS:DX - адрес 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 (команда проверки переполнения)
- 5 - при невыполнении условия в команде BOUND (команда контроля индексов массива)
- 6 - недопустимая (несуществующая) инструкция
- 7 - отсутствует FPU
- 8 - таймер
- 9 - клавиатура
- 10h - прерывание BIOS



Прерывание BIOS 10h

АН = 00h	установка видеорежима, код в AL
АН = 02h	установить позицию курсора
АН = 08h	считать символ и атрибуты из позиции курсора
АН = 09h	записать символ и атрибуты в позицию курсора
АН = 0Ch	задать пиксель
АН = 0Dh	прочитать цвет пикселя



Резидентные программы

- Резидентная программа - та, которая остаётся в памяти после возврата управления DOS
- Завершение через функцию 31h прерывания 21h / прерывание 27h
- DOS не является многозадачной операционной системой
- Резиденты - частичная реализация многозадачности
- Резидентная программа должна быть составлена так, чтобы минимизировать используемую память



Способы вызова старого обработчика прерывания

1 Вызов в начале командой CALL

2.1 Безусловная передача управления в конце командой JMP на переменную

2.2 Безусловная передача управления в конце командой JMP с сохранением адреса в машинном коде команды



Завершение с сохранением в памяти

- **int 27h**
 - DX = адрес первого байта за резидентным участком программы (смещение от PSP)
- **int 21h, ah=31h**
 - AL - код завершения
 - DX - объём памяти, оставляемой резидентной, в параграфах



Порты ввода-вывода

- Порты ввода-вывода - отдельное адресное пространство для 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
```



Список основных портов ввода-вывода

000-00F Контроллер DMA
010-01F Контроллер DMA (PS/2)
020-02F Главный контроллер прерываний
030-03F Подчиненный контроллер прерываний
040-05F Programmable Interval Timer (PIT)
060-06F Контроллер клавиатуры
070-071 Часы реального времени
080-083 DMA Page Register
090-097 Programmable Option Select (PS/2)
0A0-0AF PIC #2
0C0-0CF DMAC #2
0F0-0FF Математический сопроцессор, PCJr Disk Controller
100-10F Programmable Option Select (PS/2)
170-17F Hard Drive 1 (AT)
1F0-1FF Hard Drive 0 (AT)
200-20F Game Adapter
210-217 Expansion Card Ports
278-27F Parallel Port 3

2A2-2A3 Clock
2B0-2DF EGA/Video
2E2-2E3 Data Acquisition Adapter (AT)
2E8-2EF Последовательный порт COM4
2F8-2FF Последовательный порт COM2
300-31F Prototype Adapter, Periscope Hardware Debugger
360-36F Network
370-377 Контроллер дисководов FDD
378-37F Параллельный порт LPT2
380-38F SDLC Adapter
390-39F Cluster Adapter
3B0-3BF Monochrome Adapter
3BC-3BF Параллельный порт LPT1
3C0-3CF EGA/VGA
3D0-3DF Color Graphics Adapter
3E0-3EF Последовательный порт COM3
3F0-3F7 Контроллер дисководов FDD
3F8-3FF Последовательный порт COM1



Структуры

название_структуры struct

поля структуры

название_структуры ends

```
point struct
```

```
    x word ?
```

```
    y word ?
```

```
point ends
```



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 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 5

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2025 г.



32-разрядные процессоры (386+)

Производство x86: 1985 - ~2010

32-разрядные:

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

Режимы работы

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)

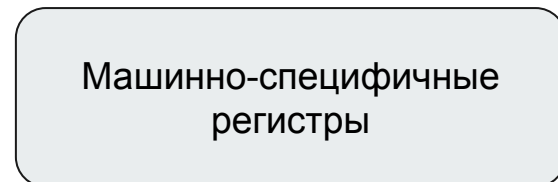
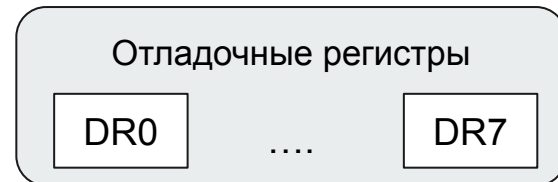
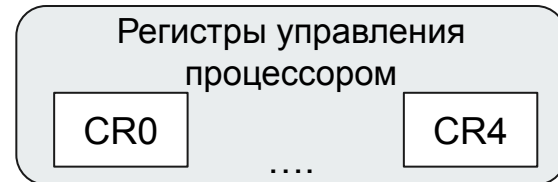
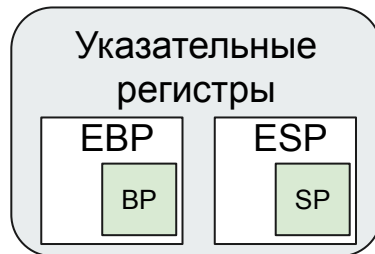
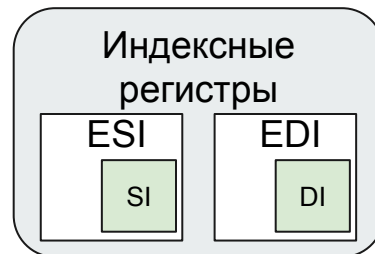
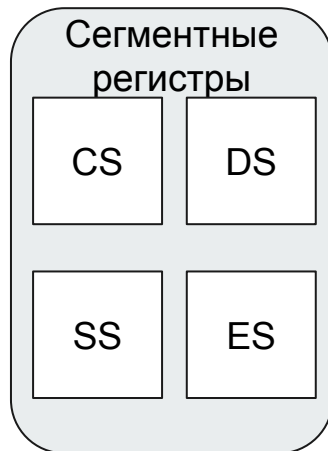
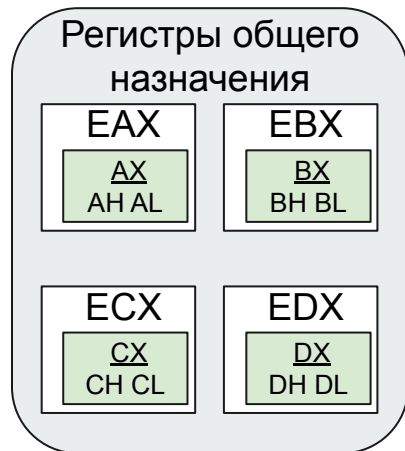
- обращение к оперативной памяти происходит по реальным (действительным) адресам, трансляция адресов не используется;
- набор доступных операций не ограничен;
- защита памяти не используется.

"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, ...

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:

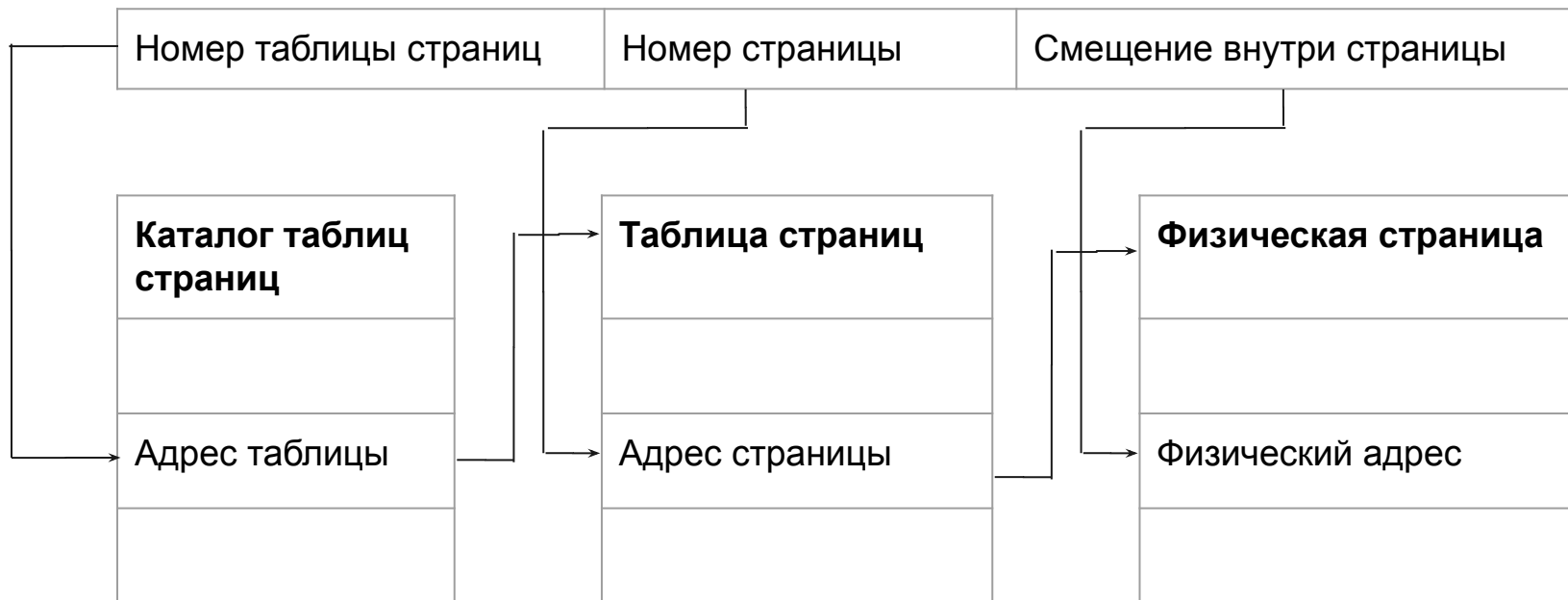
```
mov eax, 12345678h
xor ebx, ebx
mov bx, 1
add eax, ebx      ; eax=12345679h
```




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.

Страничная организация 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, использующиеся 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.



Регистр EFLAGS

FLAGS + 5 специфических флагов



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



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
- LGDT, SGDT
- 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
 - атрибуты 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-битные регистры общего назначения R8 — R15 (с подрегистрами rNd - 32 бита, rNw - 16 бит, rNb - 8 бит)
 - 64-битный указатель RIP и 64-битный регистр флагов RFLAGS.



Виды трансляторов ассемблера

- MASM
- TASM
- NASM
- FASM
- YASM
- as
- ...



AT&T-синтаксис

Синтаксис стандартного ассемблера для UNIX - `as`

Основные отличия от Intel-синтаксиса:

1. Имена регистров предваряются префиксом `%`.
2. Обратный порядок операндов: вначале источник, затем приёмник.
3. Размер операнда задается суффиксом, замыкающим инструкцию.
4. Числовые константы записываются в Си-соглашении.
5. Для получения смещения метки используется префикс `$`.



Создание оконных приложений на ассемблере под x86

Системный вызов — обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции.

Для реализации оконных приложений необходима линковка с соответствующими библиотеками и использование как их функций, так и системных вызовов.



EXE-файлы в Windows

Структура файла формата PE:

1. DOS-секция
2. PE-заголовок
 - NT header: сигнатура, указатели на основной и опциональный заголовки
 - File header: допустимая архитектура, кол-во секций, время создания, указатель на таблицу символов
 - Optional header: точка входа, секция кода, секция данных, базовый адрес, количество каталогов
3. Таблица секций
4. Секции



ELF (Executable and Linking format)

1. Заголовок 52/64 байта, начинающийся с сигнатуры 0x7f ELF:
 - класс файла, метод кодирования, версия заголовка, расширения ABI
 - тип файла
 - архитектура
 - версия формата
 - адрес точки входа
 - смещение таблиц заголовков программы и заголовков секций
 - число заголовков и секций
2. Таблица заголовков
3. Таблица заголовков секций
4. Секции и сегменты



Mach-O

1. Заголовок
 - сигнатура 0xfeedface/0xfeedfacf
 - тип процессора, подтип
 - количество и размер команд загрузки
 - флаги
2. Команды загрузки (указания, как и куда загружать блоки файла)
3. Сегменты, в каждом до 255 секций




Дизассемблирование. Реверс-инжиниринг

Дизассемблер - транслятор, преобразующий машинный код, объектный файл или библиотечные модули в текст программы на языке ассемблера.

Дизассемблирование - процесс получения текста программы на ассемблере из программы в машинных кодах.

Реверс-инжиниринг (обратная разработка) — исследование готовой программы с целью понять принцип работы, поиска недокументированных возможностей или внесения изменений.



Машинно-зависимые языки программирования, лекция 6

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2025 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:

- $1/2 + 1/8 = 0,101\text{b}$
- $1,01\text{b} * 2^{-1}$
- Бит 31 - знак мантииссы, 30-23 - экспонента, увеличенная на 127, 22-0 - мантиисса без первой цифры
- 00111110010000000000000000000000

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 $2^{ST(1)}$)
- FEXTRACT - 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, CMOVcc, **MMX**, **FXSR (MMX2)**, **SSE**
- Если EAX = 2, то в EAX, EBX, ECX, EDX возвращается информация о кэшах и TLB



MMX (Multimedia Extensions - 1997, Pentium MMX)

Single Instruction, Multiple Data: одна инструкция — множество данных

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
- 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 (Streaming SIMD Extensions - Pentium III, 1999)

- Решение проблемы параллельной работы с FPU
- 8 128-разрядных регистров (16 в x64)
- Свой регистр флагов
- Основной тип - вещественные одинарной точности (32 бита)
- Целочисленные команды работают с регистрами MMX
- 70 команд:
 - Пересылки
 - Арифметические
 - Сравнения
 - Преобразования типов
 - Логические
 - Целочисленные
 - Упаковки
 - Управления состоянием
 - Управления кэшированием



SSE2 (2000 г., Pentium 4)

- Развитие и SSE, и MMX, окончательная замена MMX
- Числа двойной точности (2 64-битных в одном регистре)
- 144 новых команды в дополнение к 70 из первой версии SSE



SSE3 (2004 г.), SSE4 (2007-2008 г.)

SSE3:

- 13 новых инструкций
- горизонтальная работа с регистрами (сложение и вычитание значений в одном регистре)

SSE4:

- 54 новых инструкции (47 SSE4.1 + 7 SSE 4.2)
- опции gcc, начиная с версии 4.3: -msse4.1, -msse4.2, -msse4 (оба набора)
- ускорение видеокодеков
- вычисление CRC32
- обработка строк



Расширение AVX (Advanced Vector Extensions)

AVX 2008/2011:

- регистры увеличены со 128 (XMM) до 256 (YMM0-YMM15) бит;
- SSE-инструкции используют младшую половину YMM-регистров, не меняя старшую часть;
- “неразрушающие” (трёхоперандные) инструкции: $c = a + b$ вместо $a = a + b$

AVX2 2013:

- FMA - Fused Multiply-Add

AVX512 2013:

- AVX3 (2013 г.) - 512-битное расширение (регистры ZMM0-ZMM31).




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

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2025 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 – 1985 г.

Современные версии архитектуры - ARMv7 (32-разрядная), ARMv8 (64-разрядная).

ARMv9 - перспективная архитектура с поддержкой векторных инструкций SVE2.

Профили: Classic, Microcontroller, Real-time, Application (последняя буква в архитектуре)

Регистры общего назначения ARMv8.1:

- R0-R29 (Xnn - 64-разрядный алиас, Wnn - 32-разрядный алиас младшей половины)
- SP
- LR (R30) (регистр связи)
- PC (счётчик команд)



Профили ARM

Указывается в последней букве архитектуры (например, -A).

- Application - поддерживает виртуальную память с помощью блоков управления памятью;
- Microcontroller - отсутствуют блоки управления памятью;
- Real-time - поддержка режима реального времени;
- Classic (не реализуется, был основным профилем до архитектуры v6).



Режимы исполнения

AArch64 (2011 год):

- 64-разрядные инструкции;
- поддержка 32-разрядных.

AArch32:

- 32-разрядные инструкции;
- инструкции Thumb32.



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



Уровни исключений ARMv8

- EL0 (user) — пользовательские программы
- EL1 (kernel) — ядро ОС
- EL2 (hypervisor) — гипервизоры виртуальных машин
- EL3 (secure monitor) — наиболее привилегированный уровень



Наборы команд ARM

- A32 (32-разрядные)
- Thumb (16-разрядные, более компактные)
- Thumb2 (16- и 32-разрядные)
- A64 (32-разрядные)



Расширения

- VFP v1-v5
- SIMD, NEON, SVE
- AES, SHA



Current Program Status Register (CPSR)

Bits	Name	Function
[31]	N	Negative condition code flag
[30]	Z	Zero condition code flag
[29]	C	Carry condition code flag
[28]	V	Overflow condition code flag
[27]	Q	Cumulative saturation bit
[26:25]	IT[1:0]	If-Then execution state bits for the Thumb IT (If-Then) instruction
[24]	J	Jazelle bit
[19:16]	GE	Greater than or Equal flags
[15:10]	IT[7:2]	If-Then execution state bits for the Thumb IT (If-Then) instruction
[9]	E	Endianness execution state bit: 0 - Little-endian, 1 - Big-endian
[8]	A	Asynchronous abort mask bit
[7]	I	IRQ mask bit
[6]	F	FIRQ mask bit
[5]	T	Thumb execution state bit
[4:0]	M	Mode field



Быстрые (FIQ) и обычные (IRQ) прерывания

Fast interrupt - режим для получения данных от оборудования, минимизирующий задержки:

- скорость обработки выше;
- допустима работа только одного обработчика одновременно;
- может быть только одно;
- обработчик может устанавливаться непосредственно по адресу вектора;
- не может вызывать другие прерывания.

Standart interrupt - все прочие прерывания.



Базовые команды ARM

Команды пересылки данных: LDR, STR, MOV

Арифметические команды: ADD, SUB, MUL

Команды деления отсутствуют. Замена для деления на константу - умножение на заранее вычисленную степень 2, затем сдвиг.

Побитовые операции: AND, ORR, XOR, LSL, LSR, ASR, ROR, RORX...

Команда сравнения CMP



Команды ветвления B, BL, BLX, Bnn

- B (Branch) - переход
- BL (Branch with link) - переход с сохранением адреса возврата в LR
- BLX - переход с переключением системы команд
- BEQ, BNE, BLT, BLE, BGT, BGE...

Допускаются команды push lr, pop pc.



Вызов программного прерывания

SWI immed_8 (0..255)

Переводит процессор в Supervisor mode, CPSR сохраняется в Supervisor Mode SPSR, управление передаётся обработчику прерывания по вектору.



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



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



Машинно-зависимые языки программирования, лекция 8

Каф. ИУ7 МГТУ им. Н. Э. Баумана, 2024 г.



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. Задействовано ~200 команд.

Является одновременно и стековой, и регистровой.

Java Runtime Environment (JRE) - минимальная реализация виртуальной машины. Включает саму виртуальную машину и библиотеку классов.

javac - компилятор байткода, javar - дизассемблер файлов классов Java



Java virtual machine - система команд

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).

Java virtual machine - структура команд

Возможные суффиксы:

i
l
s
b
c
f
d
a

```
outer:  
for (int i = 2; i < 1000; i++) {  
    for (int j = 2; j < i; j++) {  
        if (i % j == 0)  
            continue outer;  
    }  
    System.out.println(i);  
}
```

```
0:  iconst_2  
1:  istore_1  
2:  iload_1  
3:  sipush 1000  
6:  if_icmpge 44  
9:  iconst_2  
10: istore_2  
11: iload_2  
12: iload_1  
13: if_icmpge 31  
16: iload_1  
17: iload_2  
18: irem  
19: ifne 25  
22: goto 38  
25: iinc 2, 1  
28: goto 11  
31: getstatic #84; // Field java/lang/System.out:Ljava/io/PrintStream;  
34: iload_1  
35: invokevirtual #85; // Method java/io/PrintStream.println:(I)V  
38: iinc 1, 1  
41: goto 2  
44: return
```



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
```

ildasm, ilasm - дизассемблер/ассемблер промежуточного языка (intermediate language)

CIL - пример

```
static void Main ( string [] args )
{
    for ( int i = 2; i < 1000; i++ )
    {
        for ( int j = 2; j < i; j++ )
        {
            if ( i % j == 0 )
                goto outer;
        }
        Console.WriteLine( i );
        outer++;
    }
}
```

```
.assembly primes{}
.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    .maxstack 2
    .locals init (int32 v_0,
                  int32 v_1)

        ldc.i4.2
        stloc.0 // v_0=2
        br.s     IL_001f
    IL_0004: ldc.i4.2
        stloc.1 // v_1=2
        br.s     IL_0011
    IL_0008: ldloc.0 // v_0
        ldloc.1 // v_1
        rem     // v_0 % v_1
        brfalse.s IL_001b
        ldloc.1 //v_1
        ldc.i4.1
        add
        stloc.1 // v_1+=1
    IL_0011: ldloc.1 //if v_1<=v_0
        ldloc.0
        blt.s     IL_0008
        ldloc.0
        call     void [mscorlib]System.Console::WriteLine(int32)
    IL_001b: ldloc.0 //v_0
        ldc.i4.1
        add
        stloc.0 //v_0+=1
    IL_001f: ldloc.0
        ldc.i4     0x3e8
        blt.s     IL_0004 //if v_0<=1000
        ret
}
```



WebAssembly (wasm)

WebAssembly — это бинарный формат инструкций для стековой виртуальной машины, предназначенной для компиляции программ на ЯВУ (C, C++, C#, Go, TypeScript/AssemblyScript, Kotlin, Pascal, Rust, D, Erlang) для WEB.



WebAssembly - пример

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>