Основы языка ассемблера

План лекции

1. Устройство компьютера
2. Устройство процессора
3. Режимы работы процессора
4. Регистры процессора
5. Языки ассемблера
6. Формат команд (инструкций)
7. Команды ассемблера
8. Процесс создания программы
9. Адресное пространство процесса
10. Системные структуры и механизмы

Основные понятия

- Ассемблер – транслятор исходного кода программы, на языке ассемблера в машинный код
- Дизассемблер
- Машинный код
- Машинное слово

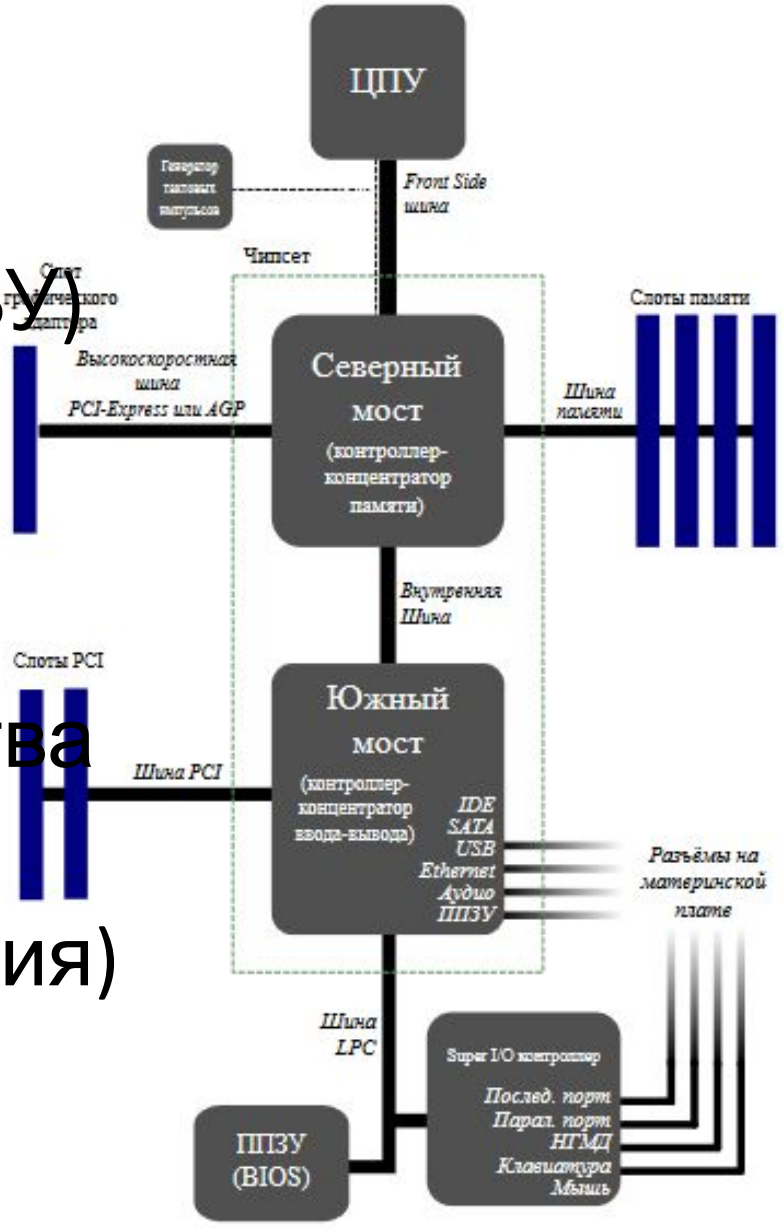
Фон-неймановская архитектура

- **Принцип однородности памяти**
- Принцип адресности
- Принцип программного управления
- Принцип двоичного кодирования

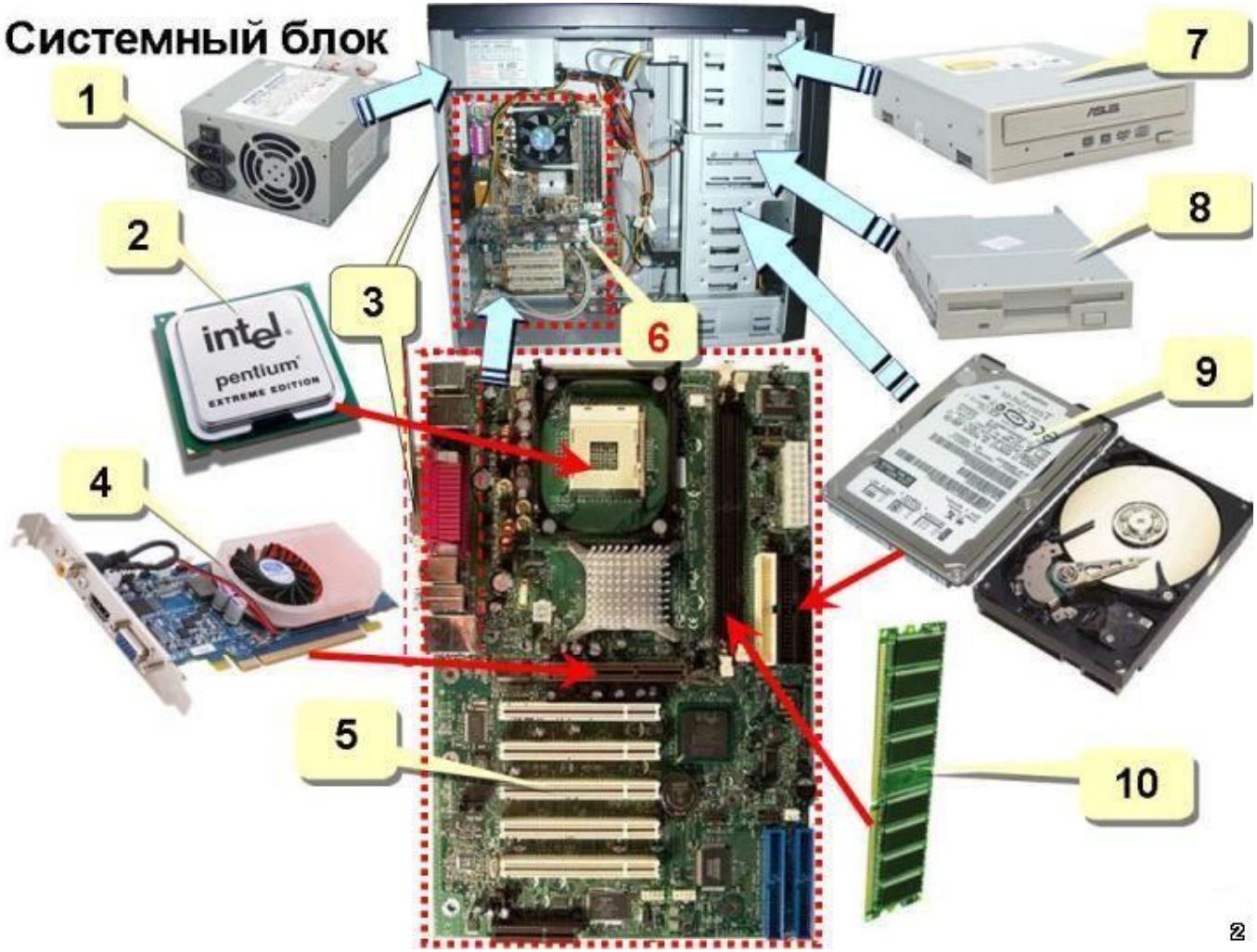
Архитектура компьютера

- ЦПУ
- Оперативная память (ОЗУ)
- Северный мост
- Южный мост
- BIOS (ППЗУ)
- Периферийные устройства
- Системная шина

(адреса, данных, управления)



Устройство персонального компьютера

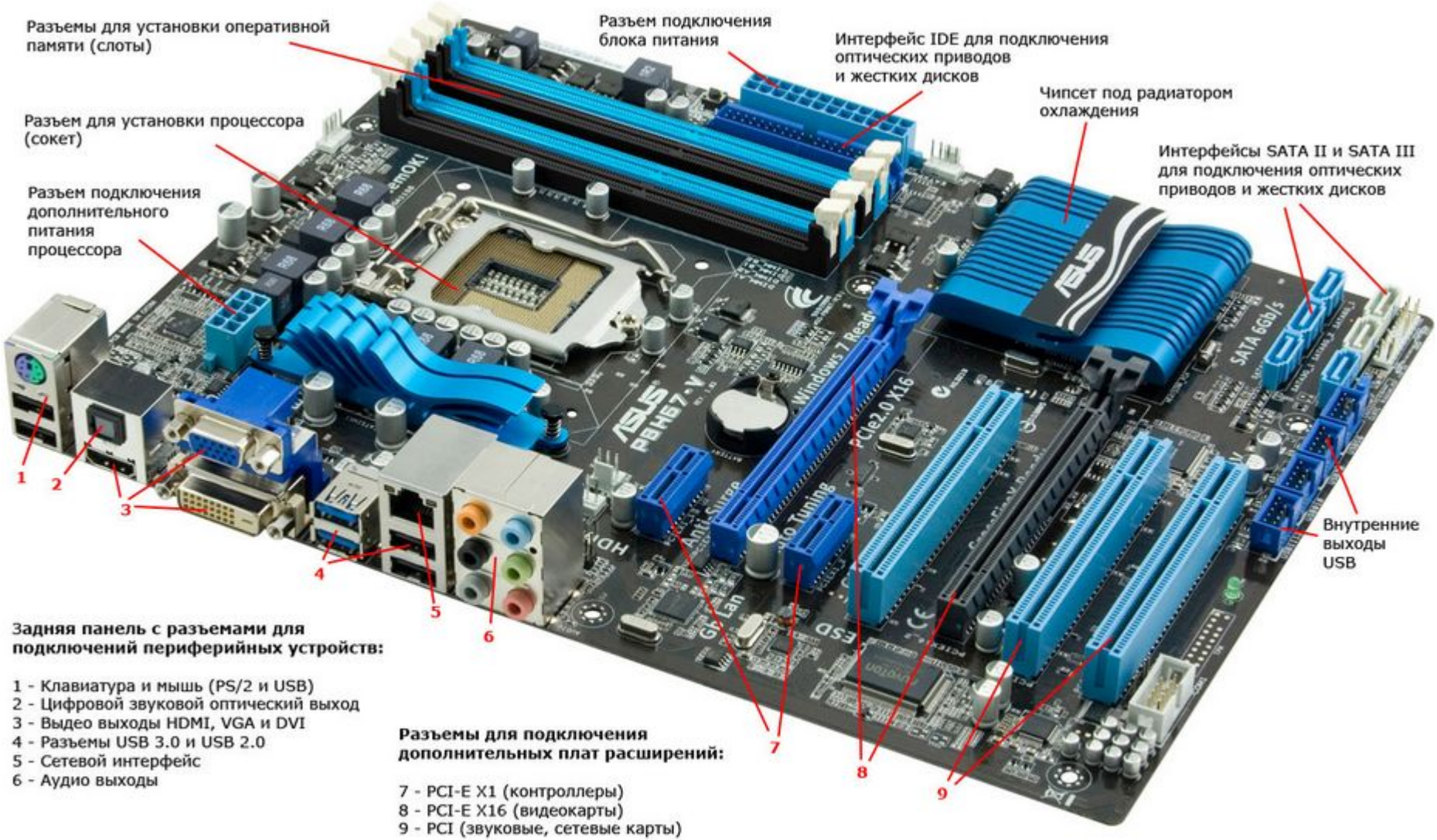


Процессор

- Регистры
- АЛУ

- RSB
- Кэш-память (кода и данных)
- TLB
- L1, L2, L3

Материнская плата



Задняя панель с разъемами для подключений периферийных устройств:

1 - Клавиатура и мышь (PS/2 и USB)
2 - Цифровой звуковой оптический выход
3 - Выдео выходы HDMI, VGA и DVI
4 - Разъемы USB 3.0 и USB 2.0
5 - Сетевой интерфейс
6 - Аудио выходы

Разъемы для подключения дополнительных плат расширений:

7 - PCI-E X1 (контроллеры)
8 - PCI-E X16 (видеокарты)
9 - PCI (звуковые, сетевые карты)

Основные компоненты материнской платы

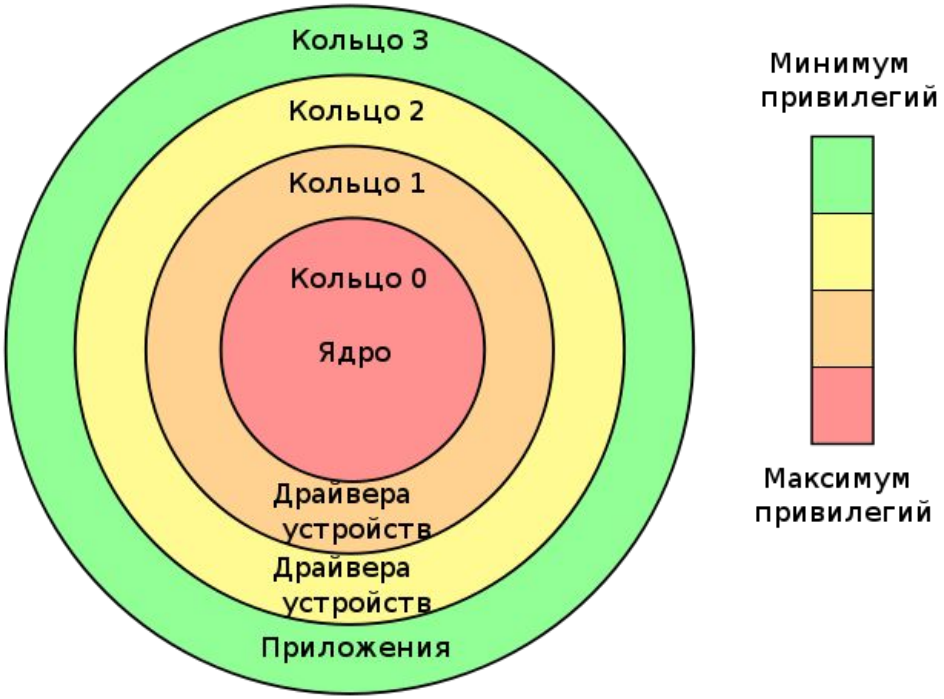
Режимы работы процессора

- Реальный режим (real mode)
- Защищенный режим (protected mode)
- Режим виртуального 8086
- Режим системного управления (SMM)

Кольца защиты (Ring)

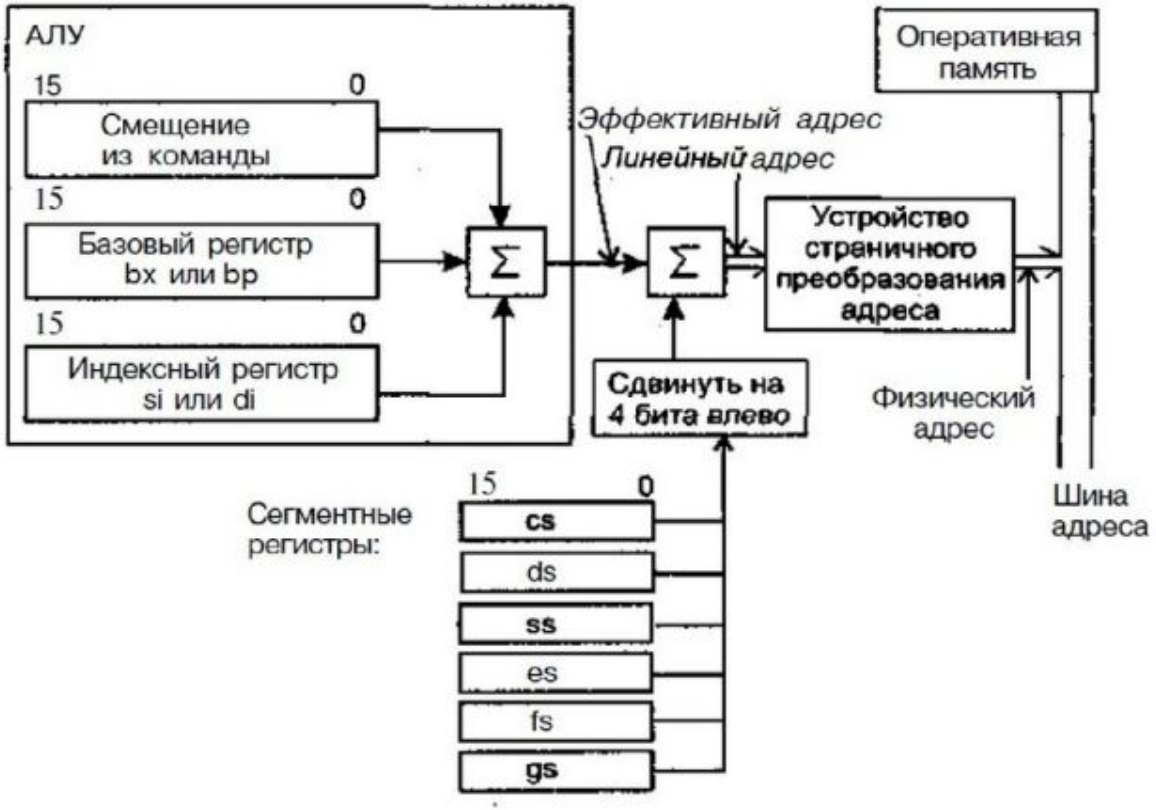
- Ring -2 (Режим системного управления, System Management Mode)
- Ring -1 (режим гипервизора, Hypervisor mode)
- Ring 0 (режим ядра, супервизора – ring mode, supervisor mode)
- Ring 1
- Ring 2
- Ring 3

(режим пользователя, user mode)



Реальный режим

- Физические адреса от 0 до 1 Мб
- Макс размер сегмента 64 Кб (16 разр)
- Использование сегментной адресации



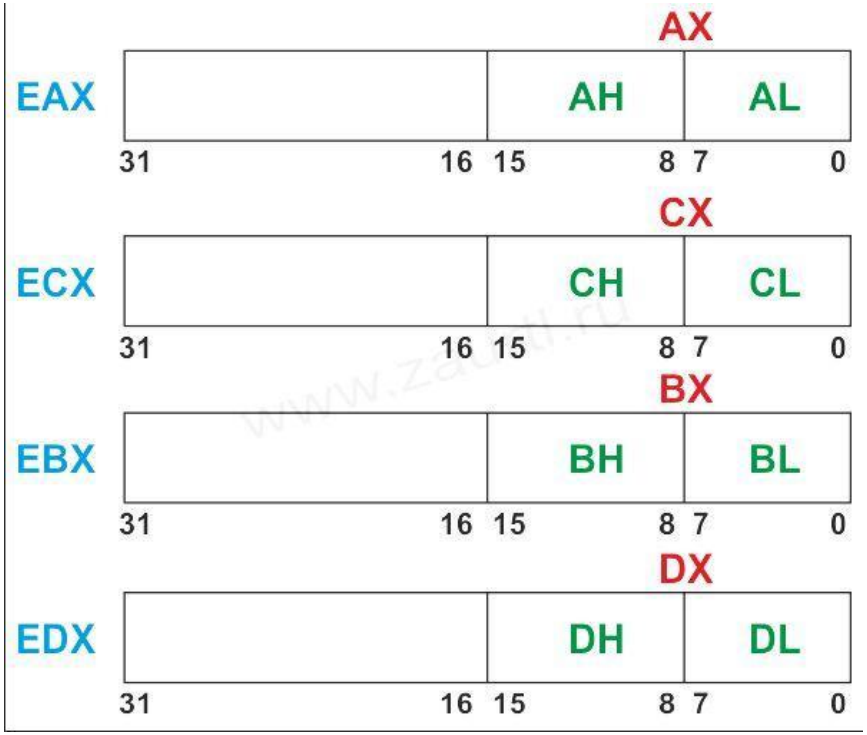
Типы данных

- Двоичные числа
- Двоично-десятичные числа
- Неупакованный BCD-формат
- Упакованный BCD-формат
- Числа с плавающей точкой
- Символьный тип

Регистры процессора

- Регистры общего назначения (EAX, EBX, ECX, EDX)
- Адресные регистры (ESI, EDI, EBP)
- Управляющие регистры (ESP, EIP, EFLAGS)
- Сегментные регистры (CS, DS, SS, ES, GS, FS)
- Регистры управления памятью (GDTR, LDTR, IDTR)
- Регистры управления (CR0-CR4)
- Отладочные регистры (DR0-DR7)
- Машинно-зависимые регистры (MSR)

Регистры общего назначения



Адресные регистры

- ESI – индекса источника
- EDI - регистр индекса результата
- EBP - регистр указатель стековой базы

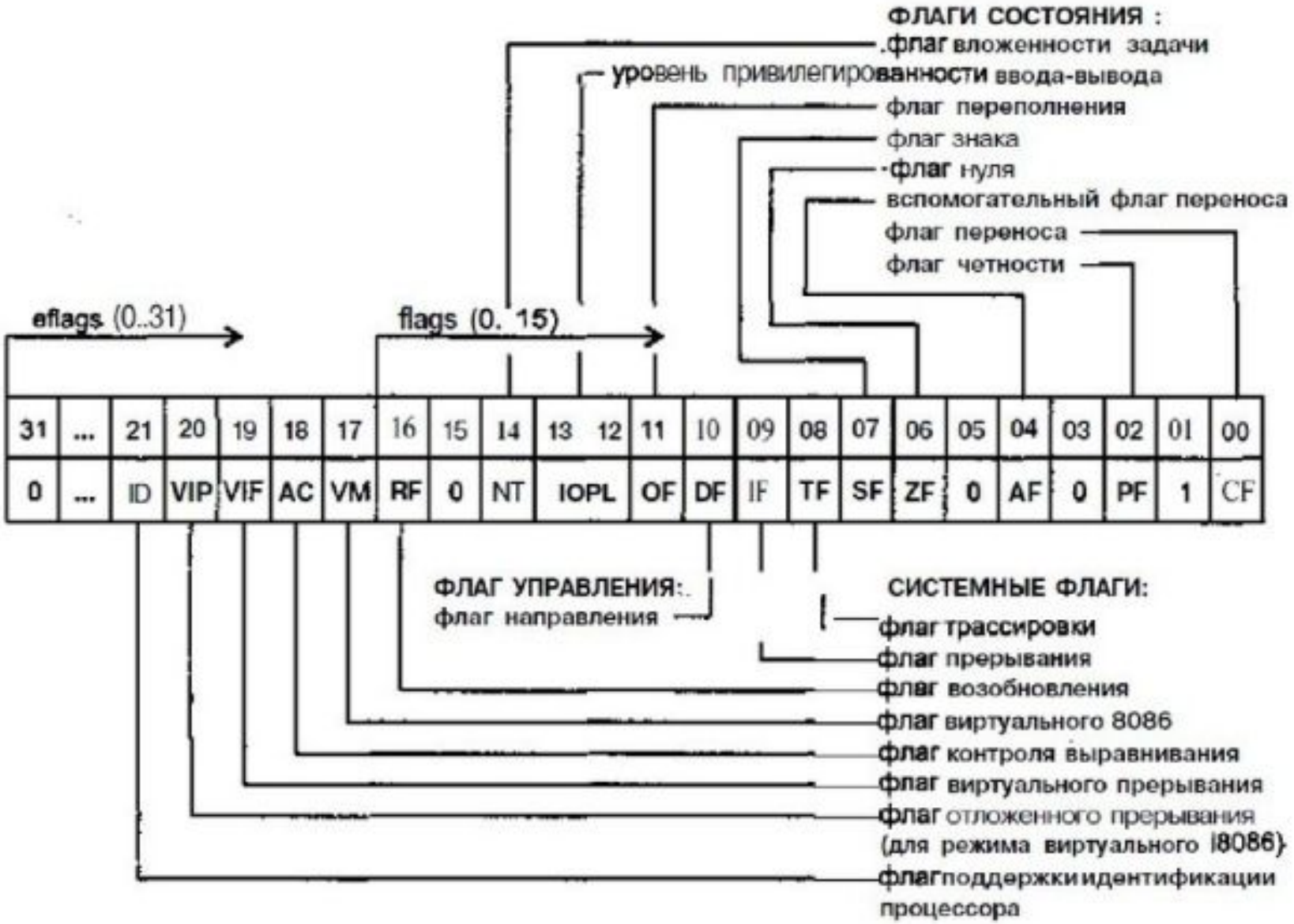
Регистры состояния

- ESP - Указатель на вершину стека
- EIP - Счетчик команд
- EFLAGS - Регистр флагов

Регистры управления

- CR0
- CR1
- CR2
- CR3
- CR4

Регистр EFLAGS / FLAGS



Сегментные регистры

16-битные регистры для хранения селекторов сегмента

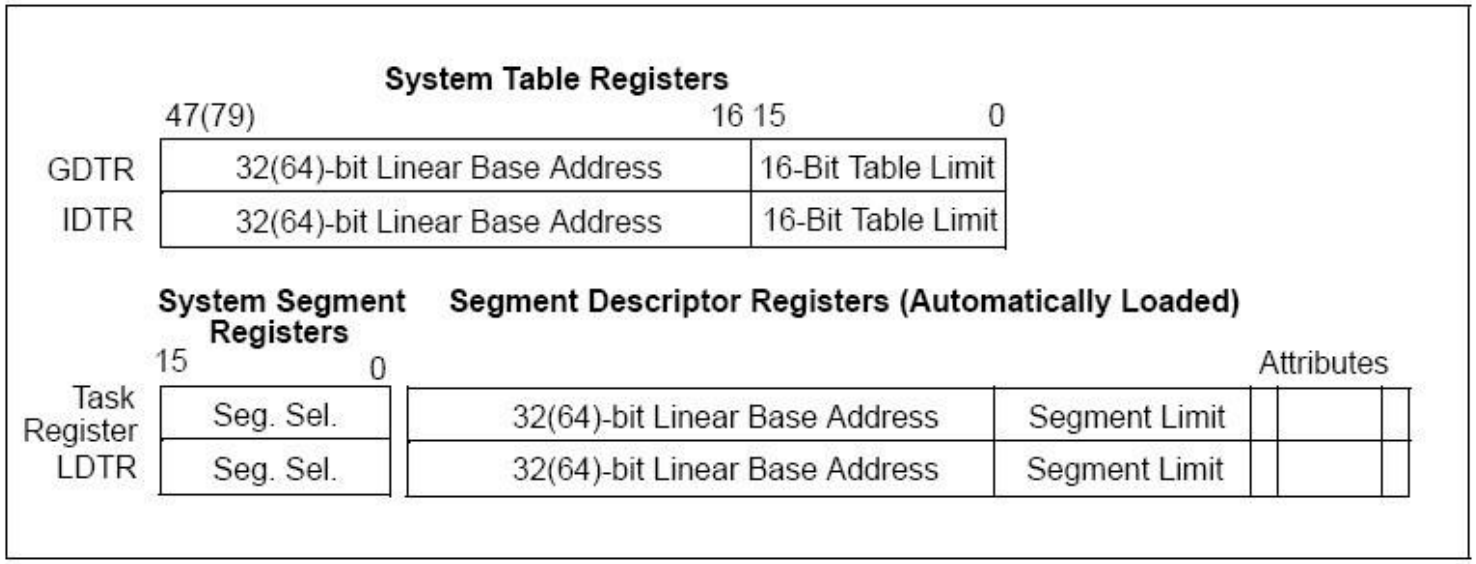
- CS – сегмент кода
- DS – сегмент данных
- SS – сегмент стека

Дополнительные сегменты:

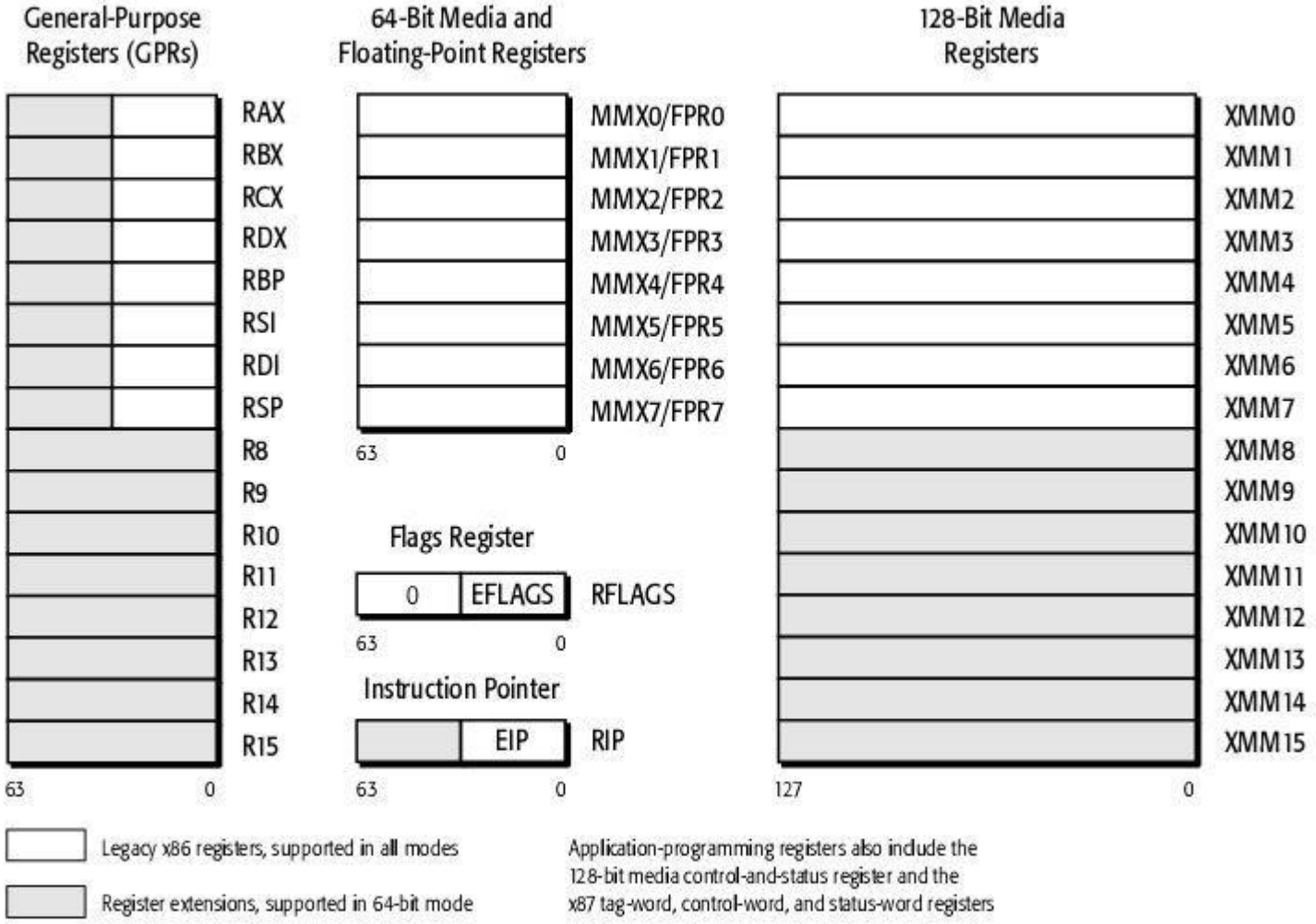
- ES
- GS
- FS

Системные регистры

- GDTR
- LDTR
- IDTR
- TR



Регистры x64



Порты ввода/вывода (I/O Ports)

Используются для взаимодействия с утройствами

- IN eax, port_num (DX) – чтение из порта
- OUT port_num(DX), eax- запись в порт

MSR–регистры (Model-Specific Registers)

- Зависят от модели процессора
- Вызываются только из режима ядра
- RDMSR – чтение, ECX –номер MSR

Результат - EDX:EAX

- WRMSR

Примеры:

RDTSC – читает MSR-регистр IA32_TIME_STAMP_COUNTER (0x10)

SYSENTER/SYSEXIT, SYSCALL/SYSRET

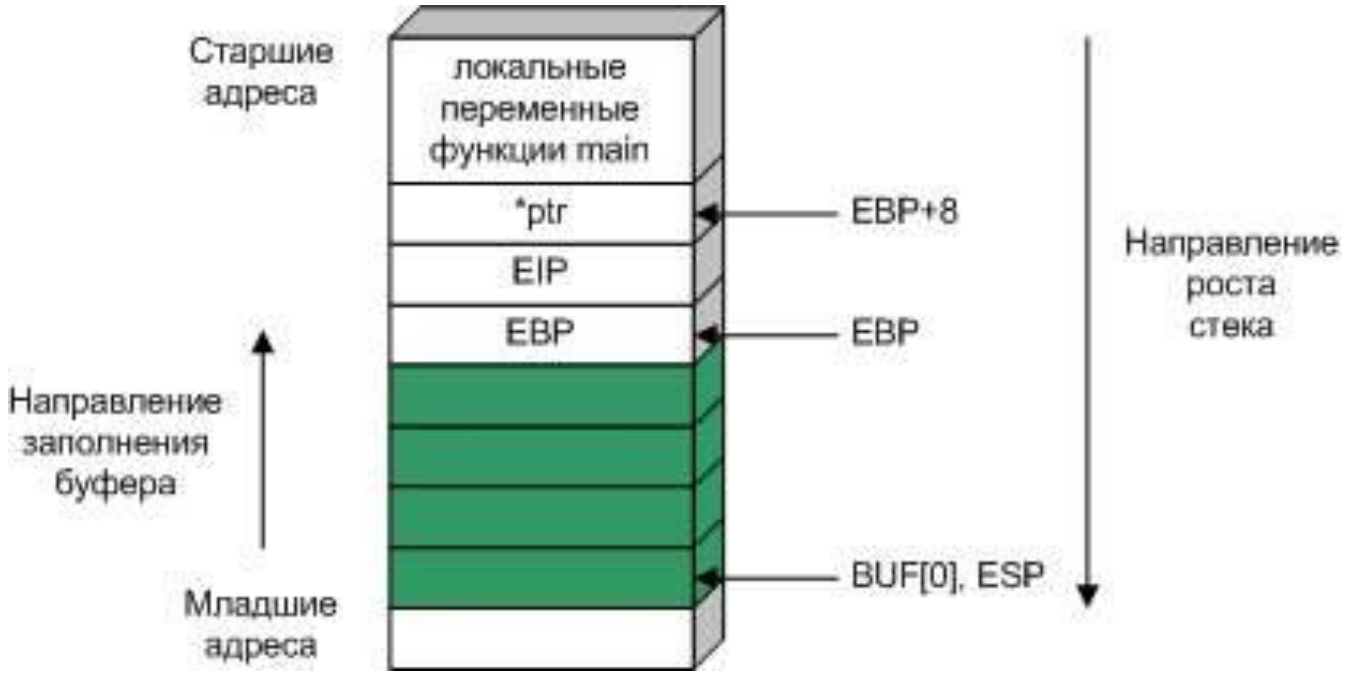
Расширения инструкций процессора

Работа с аудио- и видео-данными

- FPU / NPX
- MMX
- MMX Extended
- 3dNow!
- 3dNow! Extended
- SSE
- SSE2
- SSE3
- SSSE3
- SSE4
- AVX

Стек

- ESP – хранит адрес вершины стека
- EBP – хранит адрес начала стекового фрейма
- SS – регистр, хранит селектор стека
- Стек – растет от старших адресов к младшим



Языки ассемблера

Команды языка соответствуют инструкциям процессора

Синтаксисы:

- Intel
- AT&T

Ассемблеры:

- MASM
- NASM
- FASM
- TASM
- GAS

Типы команд

- Арифметические
- Логические
- Передачи данных
- Перехода
- Пропуска
- Вызова подпрограммы
- Возврата из подпрограммы
- Смешанные

Формат команды

- Поле префиксов
  - Замена сегмента
  - Изменение размерности адреса
  - Изменение размерности операнда
  - Необходимость повторения команды
- Поле кода операции
- Поле операндов (от 0 до 2)

[метка:]	мнемоника_команды	[операнд(ы)]	[;комментарий]

Пример

- Префикс
- Команда
- Операнды

```
.data
. . .
string1   db '0123456789',0ah,0dh,'$'
string2   db '0123406789','$'
. . .
.code
. . .
                cld
                lea    si,string1
                lea    di,string2
                mov    cx,10
cycl:
repe        cmps string1,string2
                jcxz   equal
                jne    not_match
```

Типы операндов

- Байт
- Слово
- Десятичный операнд
- Разряд
- Число
- Составной операнд

Способы адресации [1]

- Регистровая адресация

mov ax, bx

- Непосредственная адресация

mov ax, 2

- Прямая адресация

mov ax, es:0001

mov ax, ds:word_var (ds – по умолчанию)

- Косвенная адресация

mov ax, [bx]

- Адресация по базе со сдвигом

mov ax, [bx+2]

mov eax, [ebp]+2 / mov eax, 2[ebp]

Способы адресации [2]

- Косвенная адресация с масштабированием

mov eax, [esi*3]+2

- Адресация по базе с индексированием

mov ax, [bx+si+2]

mov ax, [bx][si]+2

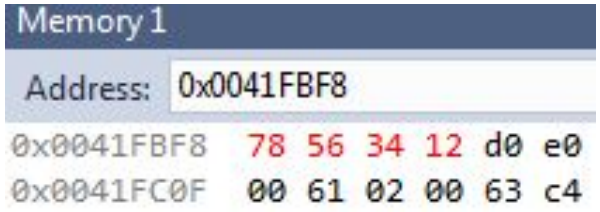
- Адресация по базе с индексированием и масштабированием
- mov edx, es:[eax+ecx*2+4]

Порядок байт

- big-endian, от старшего к младшему (SPARC, TCP/IP)
- little-endian, от младшего к старшему (x86)
- bi-endian – переключаемый порядок
- middle-endian – смешанный порядок

Формат хранения переменных

```
int data = 0x12345678;
void *addr_data = &data; // addr_data = 0x0041FBF8
```



ASCII

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

Команды пересылки

1. **MOV** DST,SRC; переслать (SRC) в (DST).
2. **PUSH** RP; поместить на вершину стека содержимое пары регистров RP (например push bx).
3. **POP** RP; снять с вершины стека два байта и поместить в пару RP (например pop ax).
4. **XCHG** DST, SRC; поменять местами содержимое (DST) и (SRC). Оба операнда не могут быть одновременно содержимым ячеек памяти.
5. **XLAT** SRC; извлечь из таблицы с начальным адресом SRC байт данных имеющий номер от начала таблицы = (AL), и поместить его в AL. Адрес SRC должен находиться в регистре BX. Другой вариант: XLATB.
6. **LEA** RP,M; загрузить в регистр RP эффективный адрес (смещение) ячейки памяти с символическим адресом M.

Арифметические команды

1. **ADD** DST, SRC; сложить содержимое SRC и DST и результат переслать в DST.

add al, [mem_byte]; mem_byte однобайтовая ячейка памяти

add [mem_word], dx; mem_word двухбайтовая ячейка памяти

add ch,10001010b;

2. **INC** DST; увеличить (DST) на 1 (инкремент (DST)).

3. **SUB** DST, SRC; вычесть (SRC) из (DST) и результат поместить в DST.

4. **DEC** DST; декремент (DST).

5. **CMP** DST, SRC; сравнить содержимое DST и SRC. Эта команда выполняет вычитание (SRC) из (DST) но разность не помещает в DST и по результату операции воздействует на флаги.

Логические команды и команды сдвига

1. **AND** DST, SRC; поразрядное логическое "И".
2. **OR** DST, SRC; поразрядное логическое "ИЛИ".
4. **NOT** DST; инверсия всех битов приемника.
5. **TEST** DST, SRC; выполняет операцию AND над операндами, но воздействует только на флаги и не изменяет самих операндов.
6. **SHR** DST, CNT; логический сдвиг вправо, освобождающиеся слева биты заполняются нулем, крайний правый бит выталкивается во флаг CF. Операнд DST может быть ячейкой
7. **SHL** DST, CNT; логический сдвиг влево.
8. **RLC** DST, CNT; циклический сдвиг влево через перенос
9. **RRC** DST, CNT;циклический сдвиг вправо через перенос
10. **ROR** DST, CNT;циклический сдвиг влево
11. **ROL** DST, CNT;циклический сдвиг вправо

Использование сдвигов

- Умножение
- Деление
- Работа с 64 переменными

Команды передачи управления

1. **CALL** SUBR; вызов подпрограммы с адресом SUBR;
2. **RET**; возврат из подпрограммы к оператору следующему непосредственно за CALL, то есть в приведенном выше примере к MOV ..
3. **JMP** NAME; безусловный переход к команде с символическим адресом NAME.
4. **JA** NAME или **JNBE** NAME; условный переход, если, например, в результате сравнения CMP DST, SRC приемник по **абсолютной величине больше** источника, то перейти к метке name.
5. **JB** NAME или **JNAE** NAME; условный переход, если, например, в результате сравнения CMP DST, SRC приемник по **абсолютной величине меньше** источника, то перейти к метке name (команды п4 и п5 выполняются по результатам выполнения операций над **беззнаковыми** числами).
6. **JZ** NAME или **JE** NAME; перейти, если результат операции влияющей на флаг нуля - нулевой (переход по "нулю").
7. **JNZ** NAME или **JNE** NAME; переход по "не нулю". (команды п6 и п7 выполняются по результатам выполнения операций над числами со **знаком**).

2.6 Instruction types

Data transfer instructions

8086 instruction set

IN Input byte or word from port
LAHF Load AH from flags
LDS Load pointer using data segment
LEA Load effective address
LES Load pointer using extra segment
MOV Move to/from register/memory
OUT Output byte or word to port
POP Pop word off stack
POPF Pop flags off stack
PUSH Push word onto stack
PUSHF Push flags onto stack
SAHF Store AH into flags
XCHG Exchange byte or word
XLAT Translate byte

Additional 80286 instructions

INS Input string from port
OUTS Output string to port
POPA Pop all registers
PUSHA Push all registers

Additional 80386 instructions

LFS Load pointer using FS
LGS Load pointer using GS
LSS Load pointer using SS
MOVSX Move with sign extended
MOVZX Move with zero extended
POPAD Pop all double (32 bit) registers
POPD Pop double register
POPFD Pop double flag register
PUSHAD Push all double registers
PUSHD Push double register
PUSHFD Push double flag register

Additional 80486 instruction

BSWAP Byte swap

Additional Pentium instruction

MOV Move to/from control register

2.6 Instruction types

Arithmetic instructions

8086 instruction set

AAA ASCII adjust for addition
AAD ASCII adjust for division
AAM ASCII adjust for multiply
AAS ASCII adjust for subtraction
ADC Add byte or word plus carry
ADD Add byte or word
CBW Convert byte or word
CMP Compare byte or word
CWD Convert word to double-word
DAA Decimal adjust for addition
DAS Decimal adjust for subtraction
DEC Decrement byte or word by one
DIV Divide byte or word
IDIV Integer divide byte or word
IMUL Integer multiply byte or word
INC Increment byte or word by one
MUL Multiply byte or word (unsigned)
NEG Negate byte or word
SBB Subtract byte or word and carry (borrow)
SUB Subtract byte or word

Additional 80386 instructions

CDQ Convert double-word to quad-word
CWDE Convert word to double-word

Additional 80486 instructions

CMPXCHG Compare and exchange
XADD Exchange and add

Additional Pentium instruction

CMPXCHG8B Compare and exchange 8 bytes

2.6 Instruction types

Bit manipulation instructions

8086 instruction set

AND Logical AND of byte or word
NOT Logical NOT of byte or word
OR Logical OR of byte or word
RCL Rotate left trough carry byte or word
RCR Rotate right trough carry byte or word
ROL Rotate left byte or word
ROR Rotate right byte or word
SAL Arithmetic shift left byte or word
SAR Arithmetic shift right byte or word
SHL Logical shift left byte or word
SHR Logical shift right byte or word
TEST Test byte or word
XOR Logical exclusive-OR of byte or word

Additional 80386 instructions

BSF Bit scan forward
BSR Bit scan reverse
BT Bit test
BTC Bit test and complement
BTR Bit test and reset
BTS Bit test and set
SETcc Set byte on condition
SHLD Shift left double precision
SHRD Shift right double precision

2.6 Instruction types

String instructions

8086 instruction set

CMPS	Compare byte or word string
LODS	Load byte or word string
MOVS	Move byte or word string
MOVSB(MOVSW)	Move byte string (word string)
REP	Repeat
REPE (REPZ)	Repeat while equal (zero)
REPNE (REPNZ)	Repeat while not equal (not zero)
SCAS	Scan byte or word string
STOS	Store byte or word string

Основные команды

- CALL / RET
- JMP
- PUSH / POP
- JE / JNE
- XOR
- MOV
- CMP
- NOP

Команды условного перехода

Op Code	mnemonic	Description
74 cb	JE rel8	Jump short if equal (ZF=1).
74 cb	JZ rel8	Jump short if zero (ZF ← 1).
0F 84 cw	JE rel16	Jump near if equal (ZF=1). Not supported in 64-bit mode.
0F 84 cw	JZ rel16	Jump near if 0 (ZF=1). Not supported in 64-bit mode.
0F 84 cd	JE rel32	Jump near if equal (ZF=1).
0F 84 cd	JZ rel32	Jump near if 0 (ZF=1).
75 cb	JNE rel8	Jump short if not equal (ZF=0).
75 cb	JNZ rel8	Jump short if not zero (ZF=0).
0F 85 cd	JNE rel32	Jump near if not equal (ZF=0).
0F 85 cd	JNZ rel32	Jump near if not zero (ZF=0).

Безусловный переход (JMP)

Opcode	Mnemonic	Description
EB cb	JMP rel8	Jump short, relative, displacement relative to next instruction.
E9 cw	JMP rel16	Jump near, relative, displacement relative to next instruction.
E9 cd	JMP rel32	Jump near, relative, displacement relative to next instruction.
FF /4	JMP r/m16	Jump near, absolute indirect, address given in r/m16.
FF /4	JMP r/m32	Jump near, absolute indirect, address given in r/m32.
EA cd	JMP ptr16:16	Jump far, absolute, address given in operand.
EA cp	JMP ptr16:32	Jump far, absolute, address given in operand.
FF /5	JMP m16:16	Jump far, absolute indirect, address given in m16:16.
FF /5	JMP m16:32	Jump far, absolute indirect, address given in m16:32.

NOP

- **N**o **OP**eration

- 0x90

Пример

```
.text:00401266                 lea     ecx, [ebp+pass]
.text:0040126C                 push    ecx             ; pass
.text:0040126D                 call    ?check_pass@@YAHPAD@Z ;
.text:00401272                 add     esp, 4
.text:00401275                 push    8Ch             ; size
```

```
0040125D  00 E8 5D FE FF FF 83 C4  0C 8D 8D FC FE FF FF 51
0040126D  E8 8E FE FF FF 83 C4 04  68 8C 00 00 00 8B 95 F8
0040127D  FE FF FF 52 E8 0A FF FF  FF 83 C4 08 33 C0 8B 4D
0040128D  FC 33 CD E8 10 00 00 00  8B E5 5D C3 CC CC CC CC
```

- E8 – опкод call
- 8E FE FF FF – аргумент, little-endian
- ff ff fe 8e = -0x172 (-370)
- check_pass = 0x40126D - 0x172 + 5 (размер инструкции) = 0x401100

```
.text:00401100 ; int __cdecl check_pass(char *pass)
.text:00401100 ?check_pass@@YAHPAD@Z proc near         ; CODE XREF: _main
.text:00401100                                         ; DATA XREF: _main
.text:00401100
.text:00401100 var_18          = dword ptr -18h
.text:00401100 ok              = dword ptr -14h
.text:00401100 var_10          = dword ptr -10h
.text:00401100 var_C           = dword ptr -0Ch
.text:00401100 var_8           = dword ptr -8
.text:00401100 var_2           = byte ptr -2
.text:00401100 var_1           = byte ptr -1
.text:00401100 pass            = dword ptr  8
.text:00401100
.text:00401100                 push    ebp
.text:00401101                 mov     ebp, esp
.text:00401103                 sub     esp, 18h
.text:00401106                 mov     [ebp+var_C], offset aPass ; "pass"
.text:0040110D                 mov     eax, [ebp+pass]
.text:00401110                 mov     [ebp+var_8], eax
```

Опкоды инструкций

- Принцип построения

Основные инструкции

Инструкция	
Call	E8,
Jmp	EB r0, E9 o0 o1, EA o0 o1 s0 s1
JE / JZ	0F 84 r0 r1, 74 r0
Jnz	
Push	50 – 57 , 68
Pop	58-5F,
Ret	C3 / C2 _ _ / RETF CB / CA _ _
Int 3	cc

Управляющие структуры: IF-ELSE

- if <A!=B> then <C=3> else <C=5>

```
mov eax, A
cmp eax, B
jne then
mov C, 5
 jmp end
then:
mov C, 3
end:
```

Управляющие структуры:switch-case

```
        mov eax, I
        shl bx, 1
        jmp cs:jump_table[bx]
jump_table dw foo0, foo1, foo2
foo0: call case0
        jmp endcase
foo1:
        call case1
        jmp endcase
foo2:
        call case2
        jmp endcase
```

Передача параметров: механизм

- По значению
- По ссылке
- По возвращаемому значению
- По результату
- По имени
- Отложенным вычислением

Передача параметров: место хранения

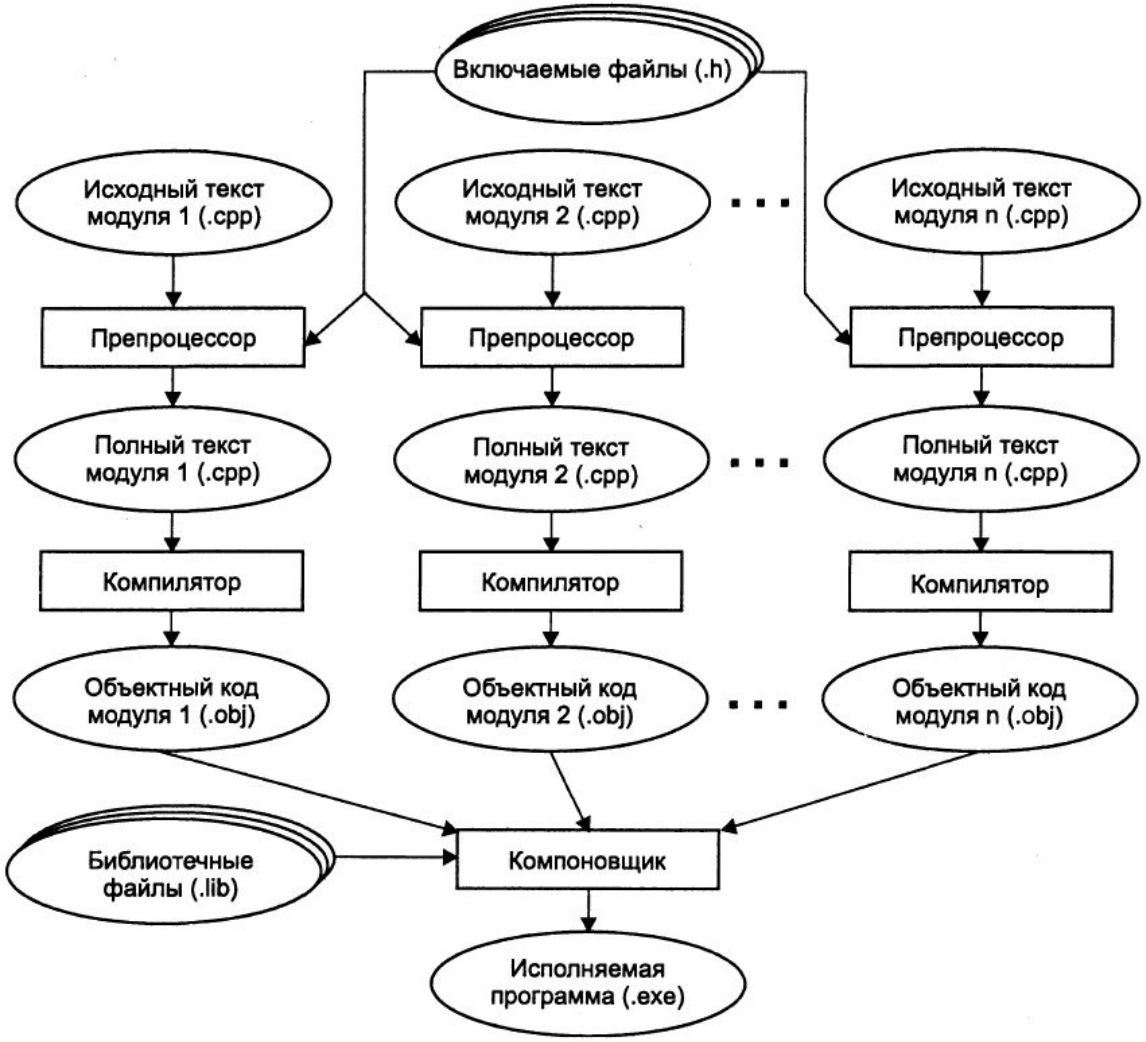
- В регистрах
- В глобальных переменных
- В стеке
- В потоке кода
- В блоке параметров

Структура исполняемого файла

Секции:

- Кода
- Данных
- Стека
- Кучи
- Неинициализированных переменных (bss)

Процесс компиляции



Этапы создания исполняемой программы

Средства отладки

- Linux: ktrace, gdb, ddd, readelf, nm

Средства разработки

- GCC \ CL
- VS

Адресное пространство процесса: Windows

Адресное пространство процесса: Linux

Структура стека

Стековый кадр

- Вызов функции

call = { “push eip”, jmp func }

ret = { “pop eip”, “jmp eip” }

- Пролог функции

push ebp

mov ebp, esp

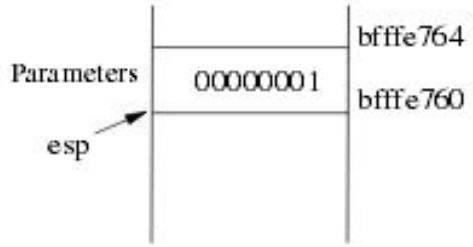
- Эпилог функции

mov esp, ebp

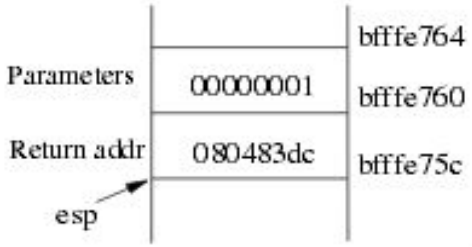
pop ebp

enter / leave

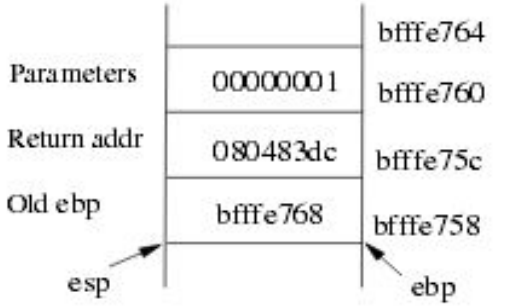
Стековый кадр



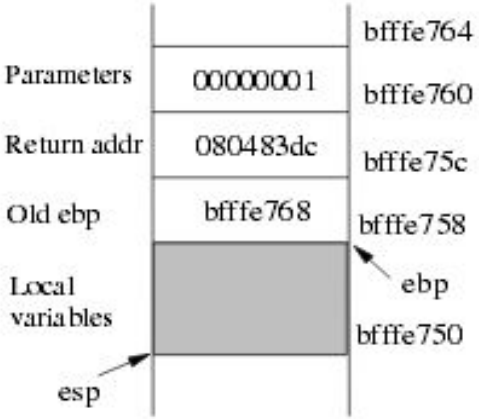
(a) Line 28: subl $4, %esp
Line 29: movl $1, (%esp)



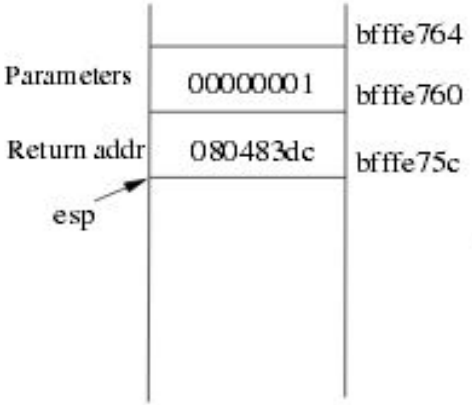
(b) Line 30: call foo



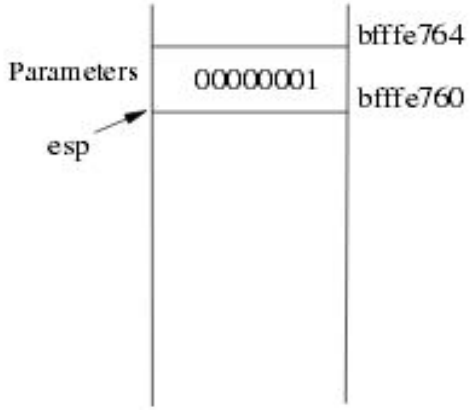
(c) Line 9: push %ebp
Line 10: movl %esp, %ebp



(d) Line 11: subl $8, %esp



(e) Line 16: leave



(f) Line 17: ret

Стековый кадр: пример

```
int callex(char *buffer, int int1, int int2) {
  printf("%s %d %d\n", buffer, int1, int2); /* выводит входные переменные */
  return 1;
}
```

```
        ; функция callex
.text 08048328          public callex
.text 08048328  callex proc near
.text 08048328  buffer = dword ptr 8
.text 08048328  int1   = dword ptr 0ch
.text 08048328  int2   = dword ptr 10h
        ; пролог функции
.text 08048328          push  ebp
.text 08048329          mov   ebp, esp
.text 0804832b          sub   esp, 8
        ; помещение в стек параметров int2, int1, buffer для функции printf
.text 0804832e          push  [ebp+int2]
.text 08048331          push  [ebp+int1]
.text 08048334          push  [ebp+buffer]
        ; помещение в стек адреса форматной строки "%s %d %d\n
.text 08048337          push  offset aSDD
        ; вызов функции printf
.text 0804833c          call  _printf
        ; очистка стека после возврата из функции printf
```

Соглашения о вызове (calling convention)

cdecl	Вызывающая функция	Параметры помещаются в стек в обратном порядке (справа налево)
clrcall	Н/Д	Параметры загружаются в стек выражений CLR по-порядку (слева направо).
stdcall / winapi	Вызываемая функция	Параметры помещаются в стек в обратном порядке (справа налево)
fastcall	Вызываемая функция	Хранятся в регистрах, затем помещаются в стек (RCX, RDX, R8, R9)
thiscall	Вызываемая функция	Помещаются в стек; указатель this хранится в регистре ECX
vectorcall	Вызываемая функция	Хранятся в регистрах, затем помещаются в стек в обратном порядке (справа налево)

Thread Environment Block (TEB)

Wow64 процессы в Windows имеют два PEB и два TEB. TEB создается функцией MmCreateTeb, PEB создается функцией MmCreatePeb

TEB — структура которая используется для хранения информации о потоках в текущем процессе, каждый поток имеет свой TEB.

Thread Environment Block (TEB)

- [TEB+0] Указатель на первый SEH на стэке.
- [TEB+4] Указатель на конец области памяти, выделенных на стеке.
- [TEB+8] Указатель на начало области памяти выделенных на стеке, для контроля исключений переполнения стека.
- [TEB+18] Адрес текущей TEB.
- [TEB+30] Адрес PEB.

```
typedef struct _CLIENT_ID {
    DWORD UniqueProcess;
    DWORD UniqueThread;
} CLIENT_ID, *PCLIENT_ID;

typedef struct _THREAD_BASIC_INFORMATION {
    typedef PVOID KPRIORITY;
    NTSTATUS ExitStatus;
    PVOID TebBaseAddress;
    CLIENT_ID ClientId;
    KAFFINITY AffinityMask;
    KPRIORITY Priority;
    KPRIORITY BasePriority;
} THREAD_BASIC_INFORMATION, *PTHREAD_BASIC_INFORMATION;
```

PEB

PEB содержит все параметры пользовательского процесса:

- местоположение главной выполняемой программы
- указатель/загрузчик данных (может использоваться, для перечисления всех dll/модулей, которые были/могут быть загруженными в процесс)
- указатель на информацию о динамической памяти (heap - куче)

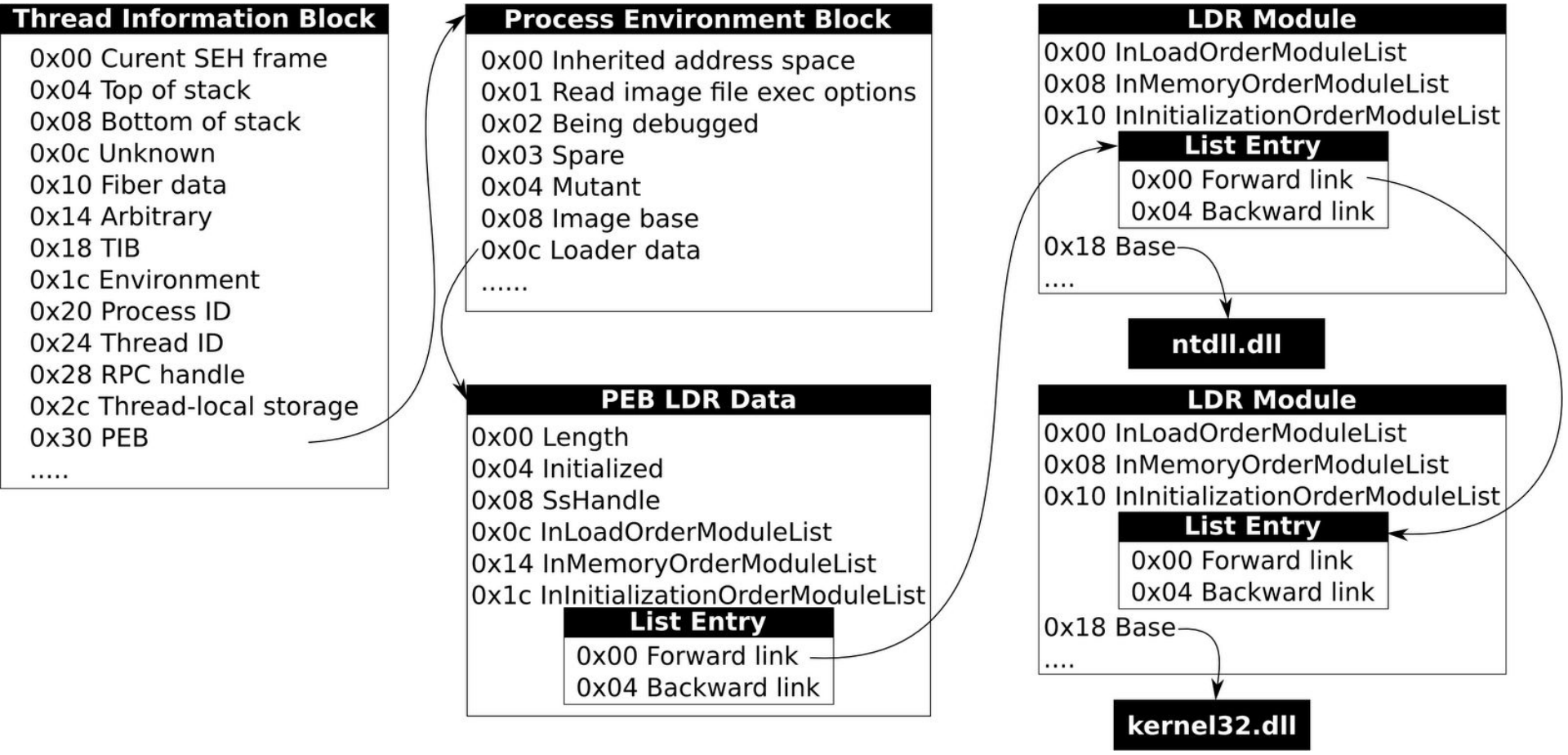
PEB

- Находится в TIB[0x30], fs:[0x30]

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  BYTE                          Reserved4[104];
  PVOID                         Reserved5[52];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved6[128];
  PVOID                         Reserved7[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

Для x64:

```
typedef struct _PEB {
    BYTE Reserved1[2];
    BYTE BeingDebugged;
    BYTE Reserved2[21];
    PPEB_LDR_DATA LoaderData;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    BYTE Reserved3[520];
    PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
    BYTE Reserved4[136];
    ULONG SessionId;
} PEB;
```


Thread Information Block
0x00 Curent SEH frame
0x04 Top of stack
0x08 Bottom of stack
0x0c Unknown
0x10 Fiber data
0x14 Arbitrary
0x18 TIB
0x1c Environment
0x20 Process ID
0x24 Thread ID
0x28 RPC handle
0x2c Thread-local storage
0x30 PEB
.....
Process Environment Block
0x00 Inherited address space
0x01 Read image file exec options
0x02 Being debugged
0x03 Spare
0x04 Mutant
0x08 Image base
0x0c Loader data
.....
PEB LDR Data
0x00 Length
0x04 Initialized
0x08 SsHandle
0x0c InLoadOrderModuleList
0x14 InMemoryOrderModuleList
0x1c InInitializationOrderModuleList
List Entry
0x00 Forward link
0x04 Backward link
LDR Module
0x00 InLoadOrderModuleList
0x08 InMemoryOrderModuleList
0x10 InInitializationOrderModuleList
List Entry
0x00 Forward link
0x04 Backward link
0x18 Base
....
ntdll.dll
LDR Module
0x00 InLoadOrderModuleList
0x08 InMemoryOrderModuleList
0x10 InInitializationOrderModuleList
List Entry
0x00 Forward link
0x04 Backward link
0x18 Base
....
kernel32.dll

Системные таблицы

- SSDT
- IDT
- LDT

Системные вызовы

- INT 2e
- SYSENTER/SYSEXIT, SYSCALL/SYSRET

ntdll_KiFastSystemCall

Номер в таблице SSDT

- CreateFile->NtCreateFile->ZwCreateFile

Литература

- Зубков С.В. Assembler для DOS, Windows и Unix
- Касперски К., Рокко Е. Искусство дизассемблирования
- Юричев Д. Reverse Engineering для начинающих