Архитектура ЭВМ и язык ассемблера

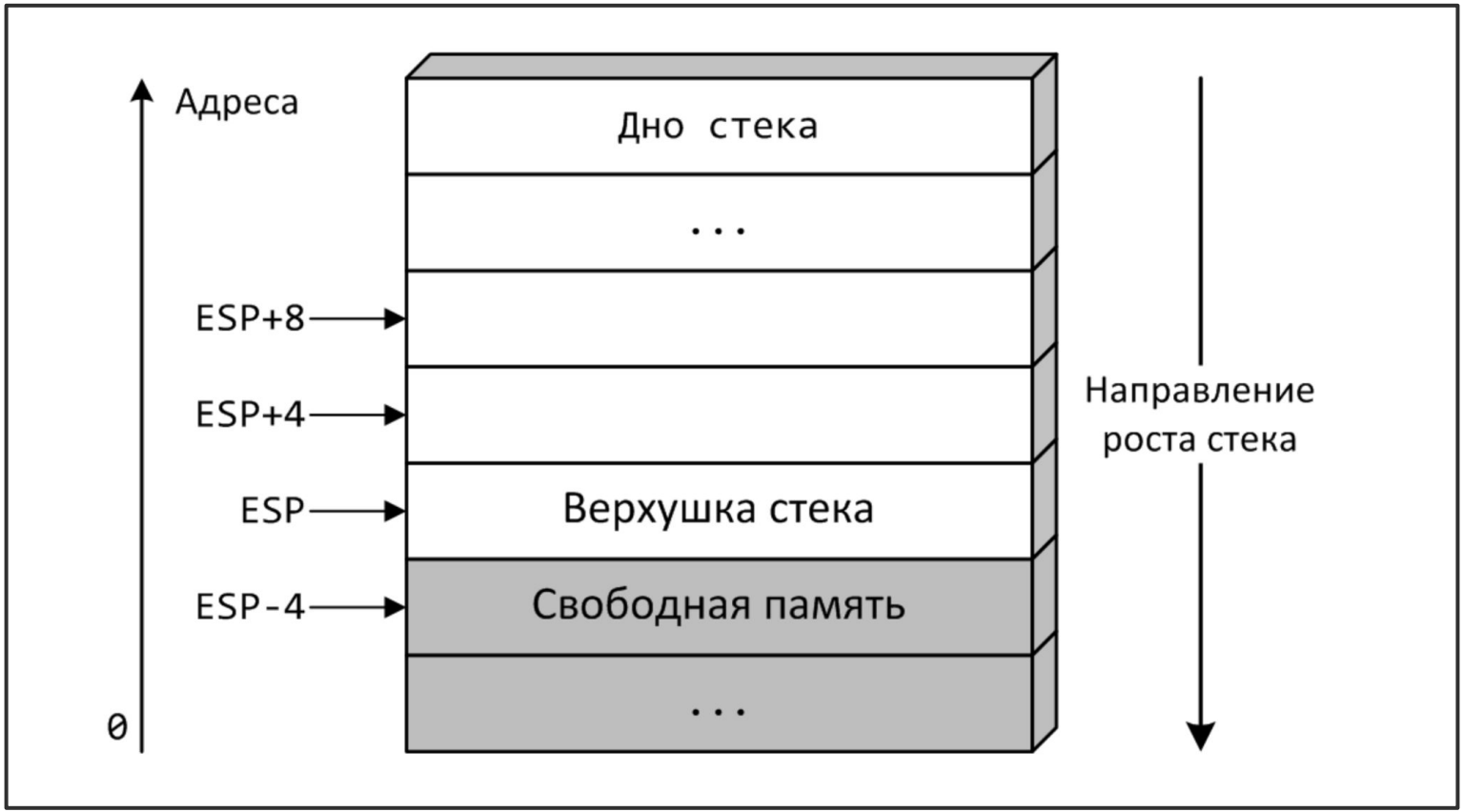
Семинар #34:

1. Аппаратный стек, операции push и pop.
2. Вызов функции по System V i386 ABI.
3. Вспомогательные макросы для вызова функций.
4. Примеры: перемножение матриц, quicksort.

13.05.2025

Аппаратный стек, инструкции PUSH и POP.

Аппаратный стек



Операции push и pop

```
IF OperandSize = 64
      THEN
            ESP := ESP – 8;
            Memory[SS:ESP] := SRC;
ELSE IF OperandSize = 32
      THEN
            ESP := ESP – 4;
            Memory[SS:ESP] := SRC;
      ELSE (* OperandSize = 16 *)
            ESP := ESP – 2;
            Memory[SS:ESP] := SRC;
```

PUSH

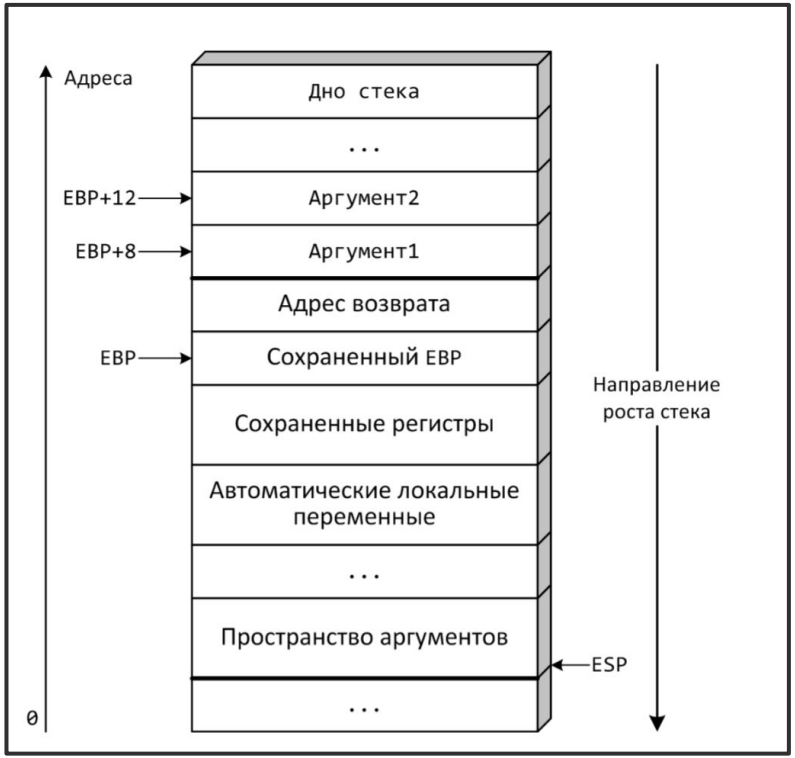
```
IF OperandSize = 32
      THEN
            DEST := SS:ESP; (* Copy a doubleword *)
            ESP := ESP + 4;
      ELSE (* OperandSize = 16*)
            DEST := SS:ESP; (* Copy a word *)
            ESP := ESP + 2;
FI;
```

POP

Вызов функций по System V i386 ABI

Формат стекового фрейма

Position	Contents	Frame
`4n+8(%ebp)`	memory argument fourbyte n	Previous
	...	
`8(%ebp)`	memory argument fourbyte 0	
`4(%ebp)`	return address	Current
`0(%ebp)`	previous `%ebp` value	
`-4(%ebp)`	unspecified	
	...	
`0(%esp)`	variable size	



Сохранение регистров на стеке

Register	Usage	Preserved across function calls
`%eax`	scratch register; also used to return integer and pointer values from functions; also stores the address of a returned struct or union	No
`%ebx`	callee-saved register; also used to hold the GOT pointer when making function calls via the PLT	Yes
`%ecx`	scratch register	No
`%edx`	scratch register; also used to return the upper 32bits of some 64bit return types	No
`%esp`	stack pointer	Yes
`%ebp`	callee-saved register; optionally used as frame pointer	Yes
`%esi`	callee-saved register	yes
`%edi`	callee-saved register	yes

Выравнивание стека

Position	Contents	Frame
`4n+8(%ebp)`	memory argument fourbyte n	Previous
	...	
`8(%ebp)`	memory argument fourbyte 0	
`4(%ebp)`	return address	Current
`0(%ebp)`	previous `%ebp` value	
`-4(%ebp)`	unspecified	
	...	
`0(%esp)`	variable size	

The end of the input argument area shall be aligned on a 16 (32 or 64, if `__m256` or `__m512` is passed on stack) **byte boundary**. In other words, the value `(%esp + 4)` is always a multiple of 16 (32 or 64) when control is transferred to the function entry point. The stack pointer, `%esp`, always points to the end of the latest allocated stack frame.

Вспомогательные макросы для вызова функций

Подготовка к вызову функции

```
%macro ALIGN_STACK 1.nolist
    sub     esp, %1
    and     esp, 0xfffffff0
    add     esp, %1
%endmacro

%macro UNALIGN_STACK 1.nolist
    add     esp, %1
%endmacro
```

```
; Инициализируем матрицу A.
ALIGN_STACK 12
push MATRIX_SIZE ; size_y
push MATRIX_SIZE ; size_x
push mat_A       ; matrix_base
call matrix_init
UNALIGN_STACK 12
```

Пролог и эпилог функции

```
%macro FUNCTION_PROLOGUE 1.nolist
    push  ebp
    mov   ebp, esp
    sub   esp, %1
%endmacro

%macro FUNCTION_EPILOGUE 0.nolist
    mov   esp, ebp
    pop   ebp
%endmacro
```

```
%macro FUNCTION_PROLOGUE 1.nolist
    enter   %1, 0
%endmacro

%macro FUNCTION_EPILOGUE 0.nolist
    leave
%endmacro
```

Пролог функции

```
global matrix_init
%define size_y          dword [ebp + 16]
%define size_x          dword [ebp + 12]
%define matrix_base     dword [ebp +  8]
%define tmp_ebx         dword [ebp -  4]
%define tmp_edi         dword [ebp -  8]
%define tmp_esi         dword [ebp - 12]
matrix_init:
    ; Инициализируем стековый фремй
    FUNCTION_PROLOGUE 12
    ; Сохраняем callee-preserved регистры.
    mov     tmp_ebx, ebx
    mov     tmp_edi, edi
    mov     tmp_esi, esi
```

Эпилог функции

```
    ; Восстанавливаем callee-preserved регистры.
    mov     ebx, tmp_ebx
    mov     edi, tmp_edi
    mov     esi, tmp_esi

    ; Восстанавливаем стековый фрейм.
    FUNCTION_EPILOGUE
    ; Возвращаемся из функции.
    ret

; Удаляем макросы во избежание ошибок.
%undef size_y
%undef size_x
%undef matrix_base
%undef tmp_ebx
%undef tmp_edi
%undef tmp_esi
```

Вопросы?

Красивые иконки взяты с сайта handdrawngoods.com