



ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование



Меня хорошо видно && слышно?

Ставьте +, если все хорошо
Напишите в чат, если есть проблемы

Проверить, идет ли запись!



The background of the slide is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer. A network of thin, light blue lines connects various points across the blue area, creating a geometric pattern. The title text is centered in white.

Основы архитектуры x86 и языка ассемблера для языка C

Легкоступ Виктор

Преподаватель



Легкоступ Виктор

- Специализация: фильтрация данных, оценивание параметров систем, системы автоматического управления, обработка сигналов, численные методы.
- Рабочие инструменты: Matlab/Simulink, Mathematica, C++, Python
- Профессиональные интересы: БЛА, системы управления и измерения, моделирование на C++, Python, Matlab, программирование микроконтроллеров, IoT

Правила вебинара



Активно участвуем



Задаем вопросы в чат или голосом



Off-topic обсуждаем в Slack #канал группы или #general



Вопросы вижу в чате, могу ответить не сразу



Архитектура

Архитектура семейства x86

Процессоры данного семейства имеют фон-неймановскую архитектуру устройства памяти (адресное пространство кода и данных является единым), что дает определенную гибкость при манипуляциях с данными и кодом.

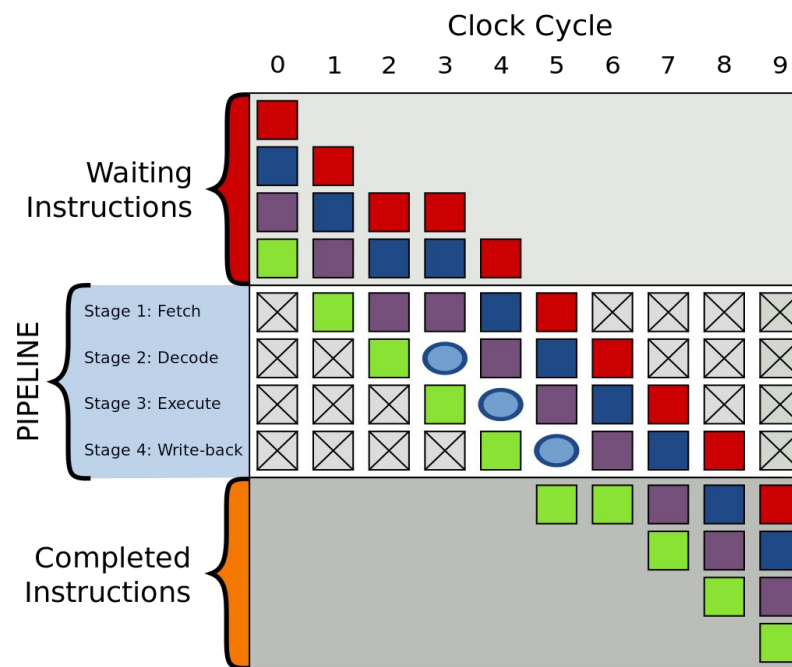
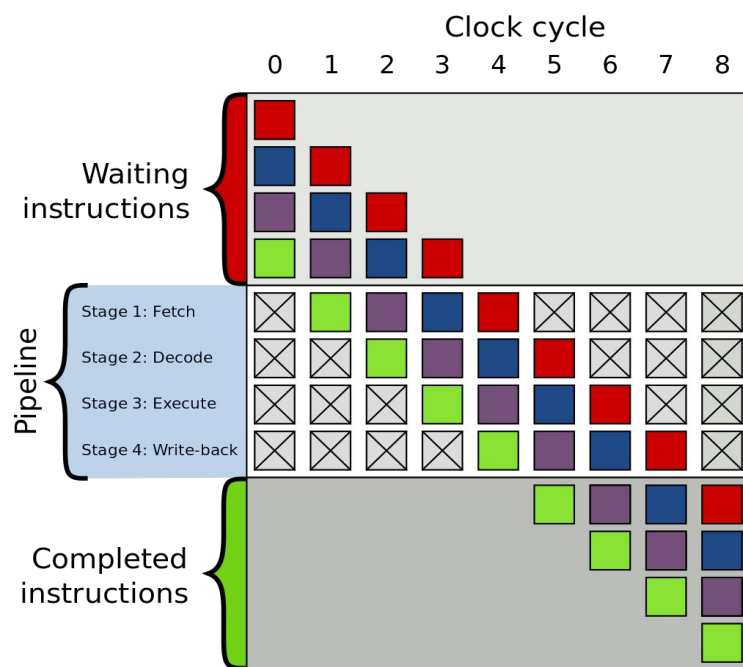
Система команд самого ядра является RISC (простая, то есть одна команда выполняет ровно одно действие), однако пользовательская система команд является CISC (комплексная, то есть одна машинная инструкция перед конвейером разбивается на несколько простейших). Фактически такая система команд является гибридной.

Традиционно в CISC процессорах имеется небольшое количество регистров общего назначения.

Простой вычислительный конвейер

```
int a, b;  
int c = a + b;
```

```
mov    edx, DWORD PTR [rbp-4] ; берем a  
mov    eax, DWORD PTR [rbp-8] ; берем b  
add    eax, edx                ; складываем и помещаем результат в eax  
mov    DWORD PTR [rbp-12], eax ; записываем результат в c
```

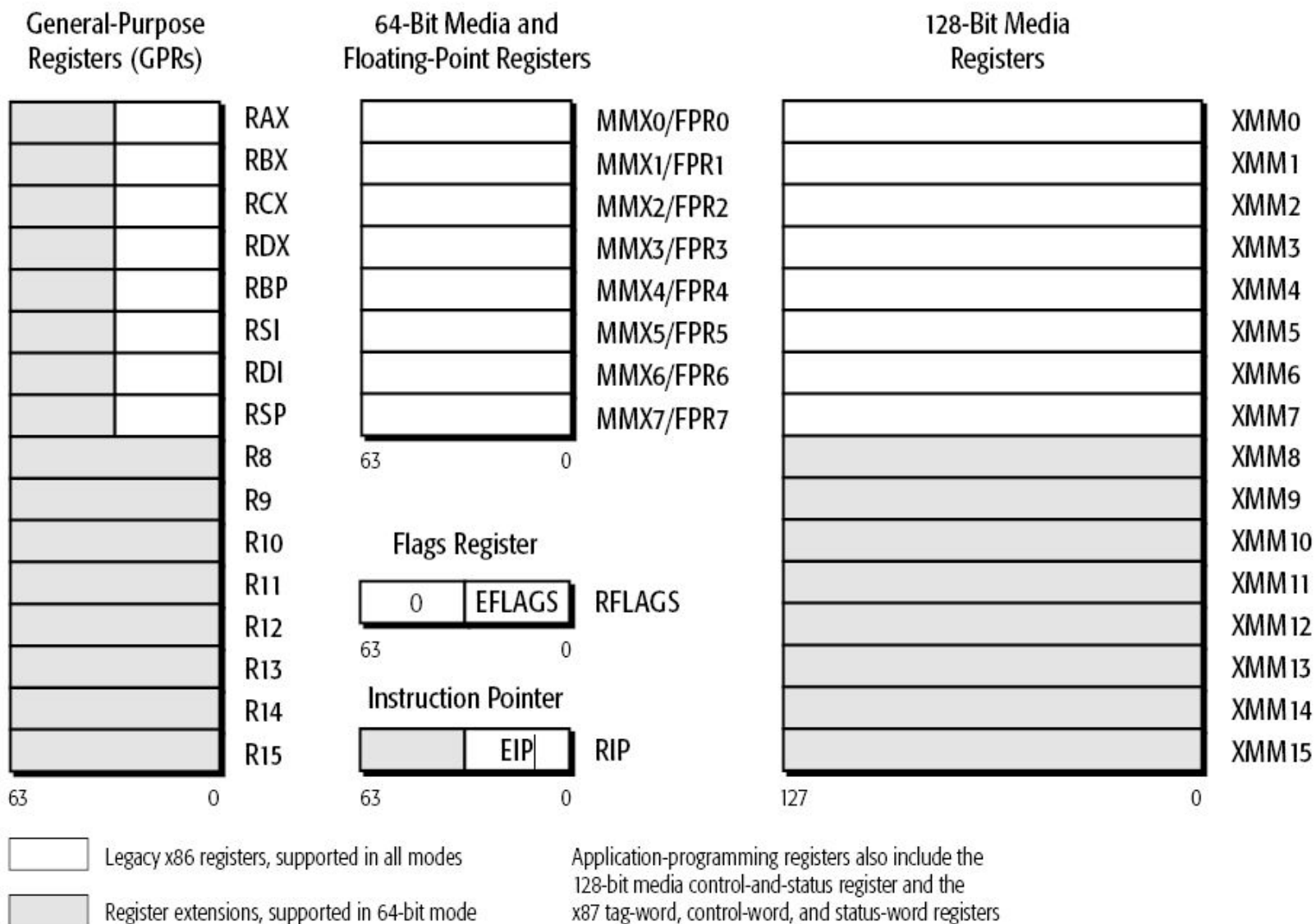


Что будет, если в конвейер попадет оператор условного перехода?

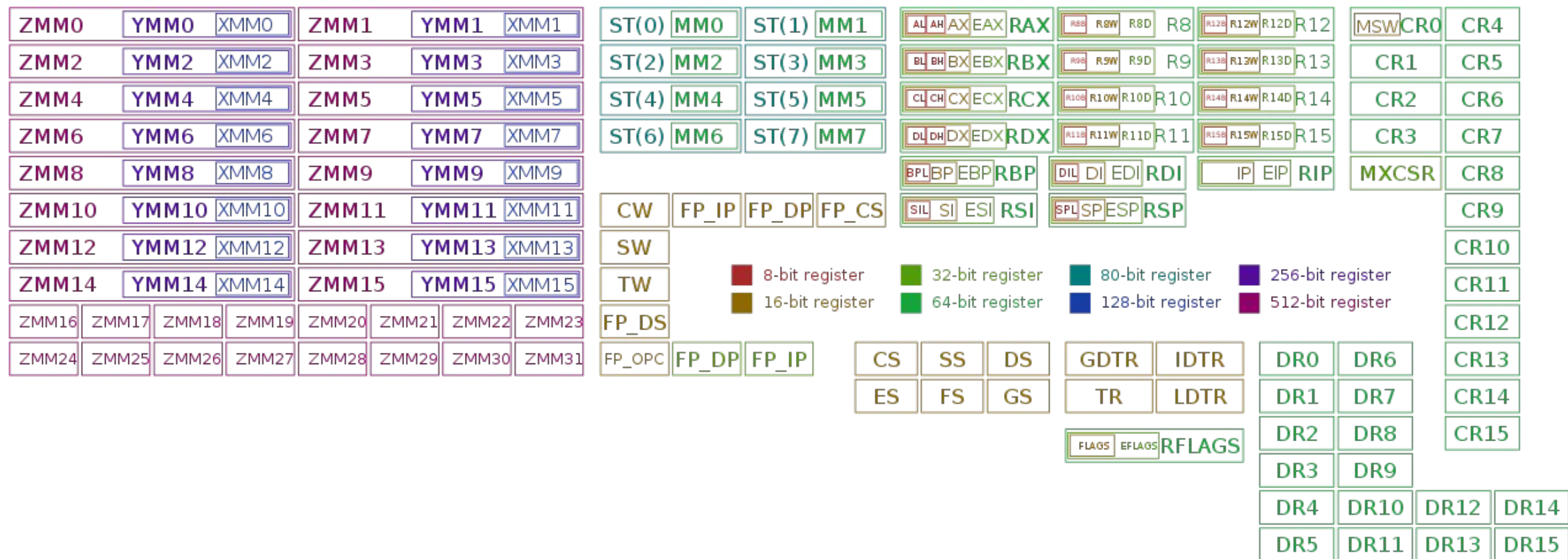
Регистры процессора общего назначения

		q (8 bytes)		l (4 bytes)	w (2 bytes)	b (1 byte)	
%rax	%eax				%ax		accumulate
%rbx	%ebx				%bx		base
%rcx	%ecx				%cx		counter
%rdx	%edx				%dx		data
%rsi	%esi				%si		source index
%rdi	%edi				%di		destination index
%rsp	%esp				%sp		stack pointer
%rbp	%ebp				%bp		base pointer

Основные регистры процессора

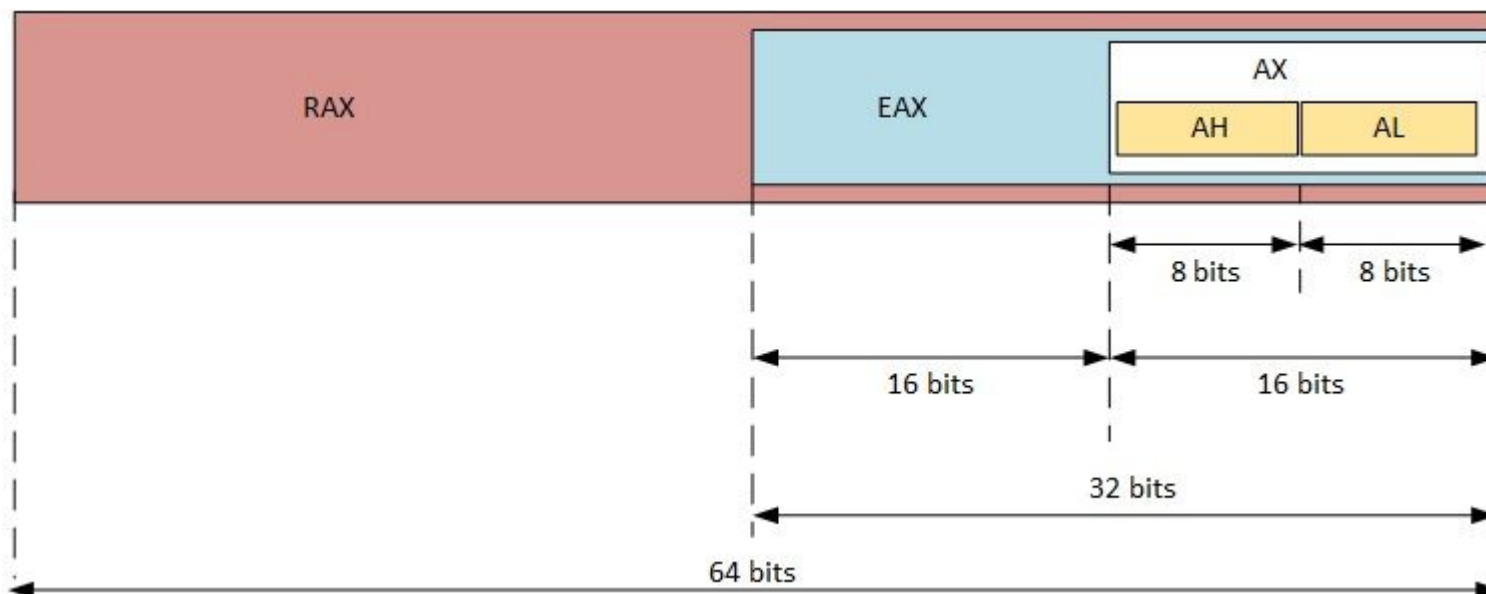


Все регистры (почти)



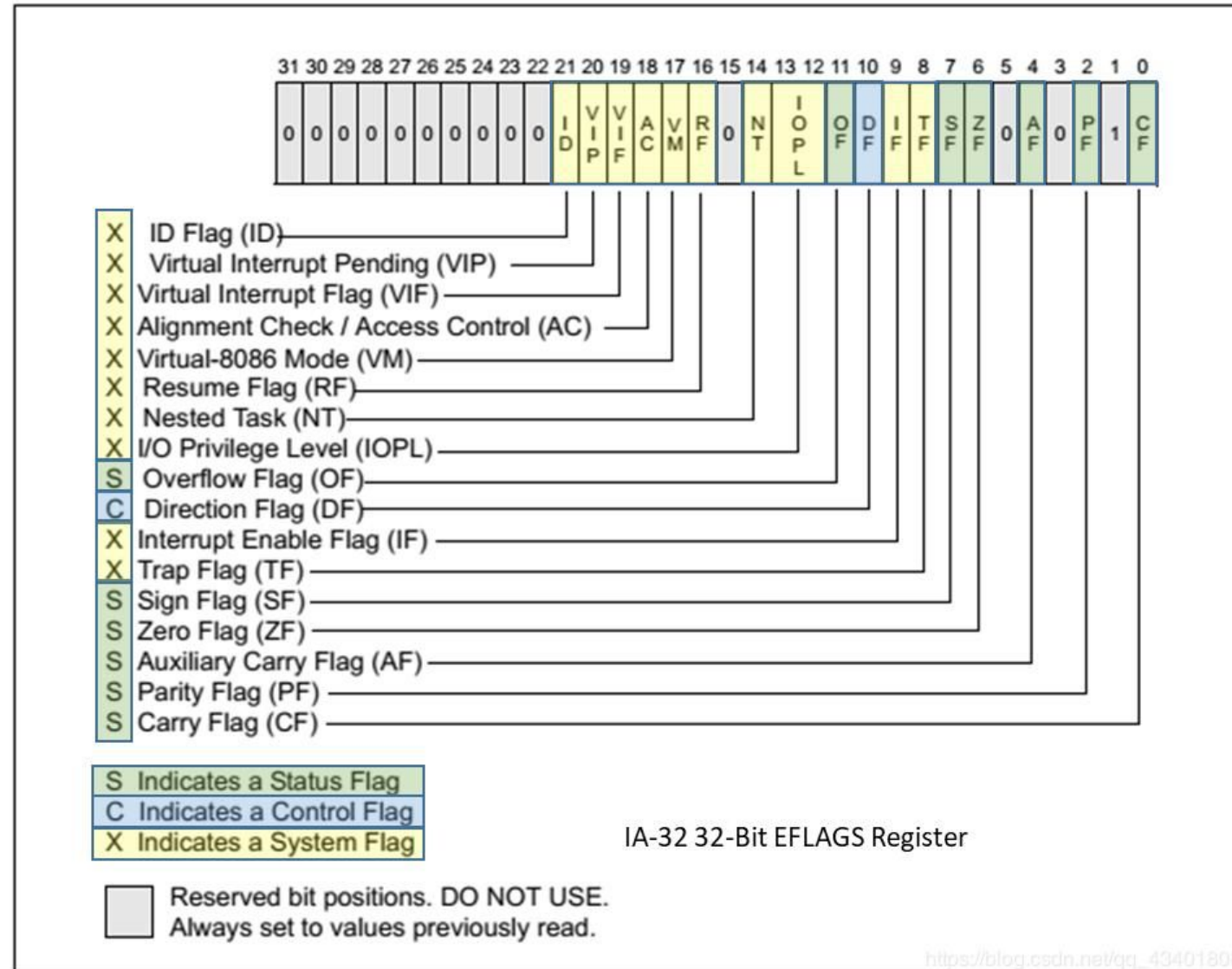
Структура регистра общего назначения

Расширение регистра RAX появилось в 64-битной версии процессоров архитектуры x86



Регистр флагов

Что такое флаги и для чего они нужны?



Сегментные регистры

Pentium Registers (cont'd)

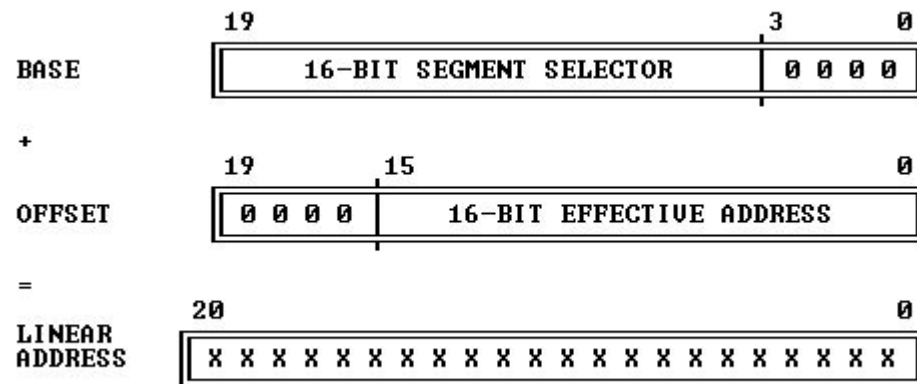
- **Segment register**

- * **Six 16-bit registers**
- * **Support segmented memory architecture**
- * **At any time, only six segments are accessible**
- * **Segments contain distinct contents**
 - » **Code**
 - » **Data**
 - » **Stack**

15		0
CS		Code segment
DS		Data segment
SS		Stack segment
ES		Extra segment
FS		Extra segment
GS		Extra segment

Сегментная адресация памяти

Figure 14-1. Real-Address Mode Address Formation



Реальный режим адресации процессора Intel 8086 и выше



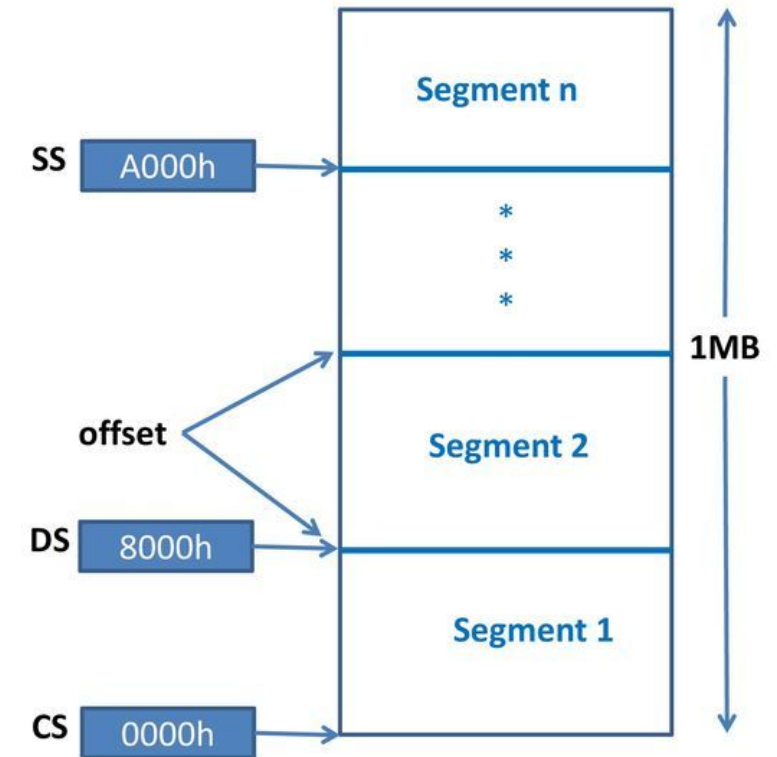
Сегментная адресация памяти

Real Mode Memory Addressing

- The first 1MB memory is Real memory or the Conventional memory

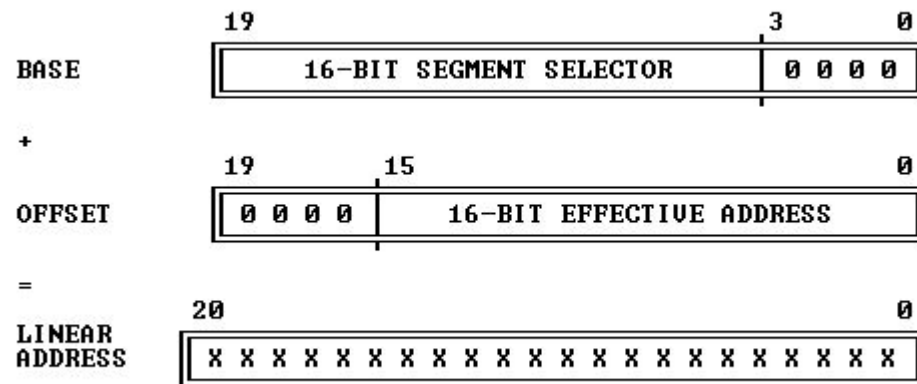
CS	Code
DS	Data
ES	Extra
SS	Stack
FS	
GS	

16 bit Segment registers

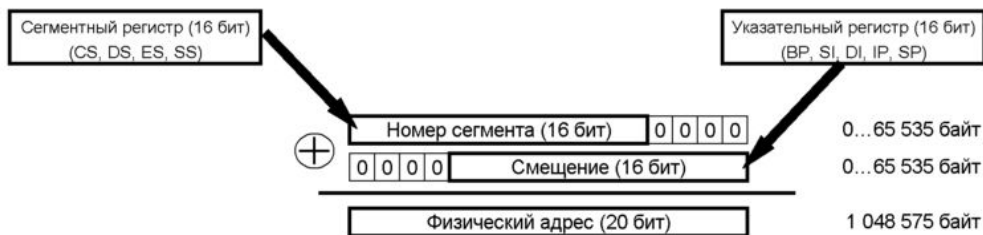


- 1 MB requires 20 bit address
- Each segment is 64 KB
- Offset address is 16 bit or 2 byte
- Actual address = segment address + offset address

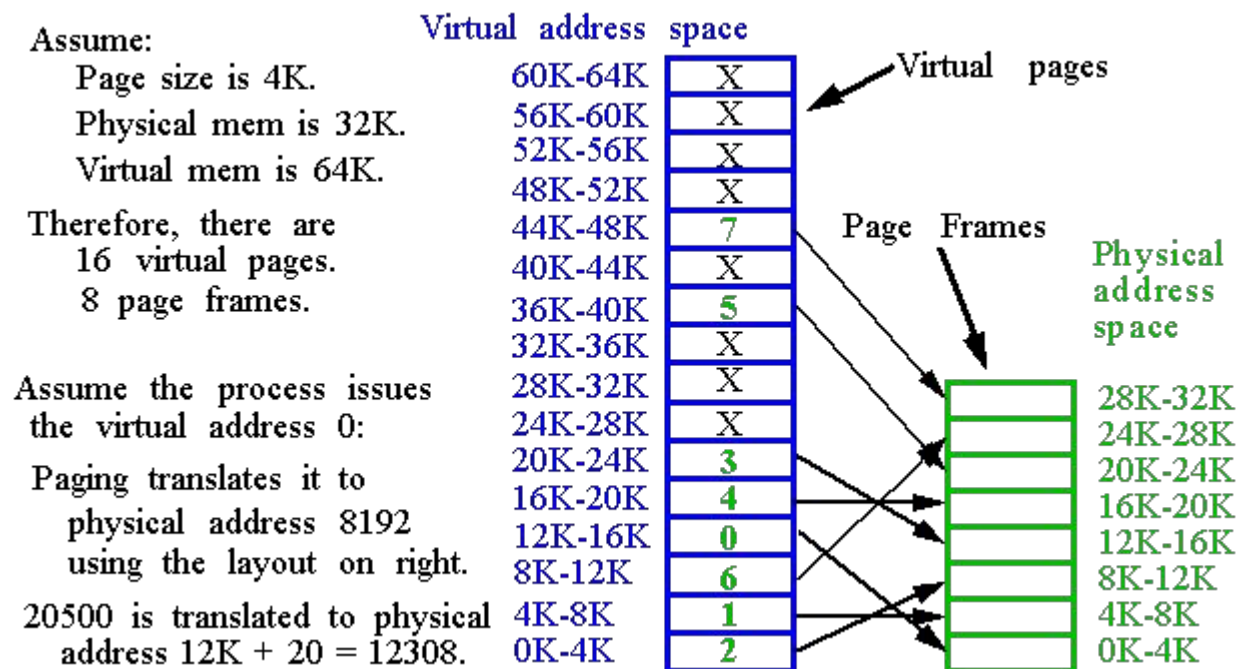
Figure 14-1. Real-Address Mode Address Formation



Реальный режим адресации процессора Intel 8086 и выше



Страничная адресация памяти



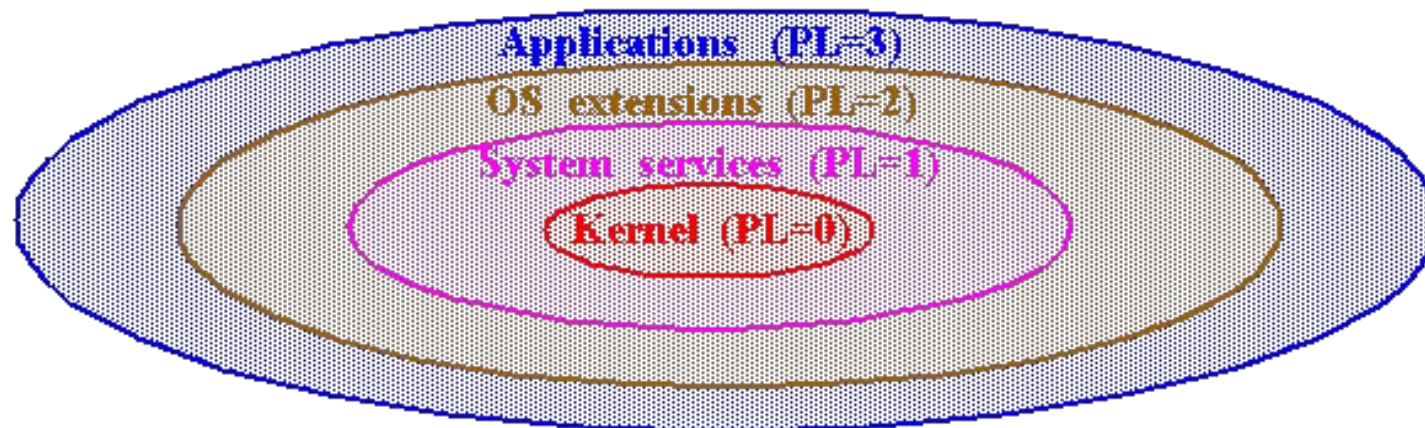
Режимы работы процессора

Старые процессоры работали в реальном режиме работы (real mode).

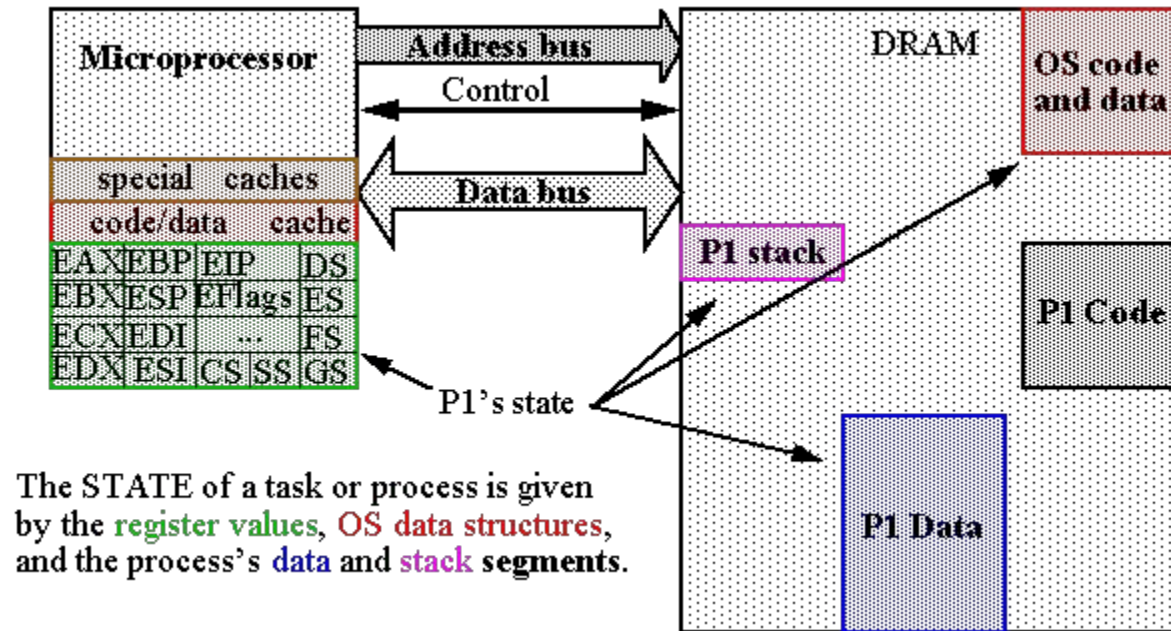
Современные процессоры работают в защищенном (protected mode).

Что это значит?

Уровни привилегий исполняемого кода



Взаимодействие процесса с ОС



The background of the image is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. The image is divided into three horizontal sections. The top and bottom sections show the city buildings. The middle section is a solid blue band with a white, glowing network pattern of lines and dots. The word 'Ассемблер' is centered in this band in a large, white, sans-serif font.

Ассемблер

Типы представления

AT&T



Ltmp2:

```
.cfi_def_cfa_register %rbp
movslq %edi, %rax
imulq $1759218605, %rax,
movq %rsi, %rax
shrq $63, %rax
sarq $44, %rsi
addl %eax, %esi
leaq L_.str(%rip), %rdi
xorl %eax, %eax
callq _printf
xorl %eax, %eax
popq %rbp
retq
.cfi_endproc
```

Intel



Ltmp2:

```
.cfi_def_cfa_register rbp
movsxd rax, edi
imul rsi, rax, 175921860
mov rax, rsi
shr rax, 63
sar rsi, 44
add esi, eax
lea rdi, [rip + L_.str]
xor eax, eax
call _printf
xor eax, eax
```

Типы представления

	Intel/Microsoft	AT&T/GNU as
Operand order: op a,b	a = a op b (dst first)	b = a op b (dst last)
Memory address	[baseregister+offset]	offset(baseregister)
Instruction mnemonics	mov, add, push, ...	movl, addl, pushl [operand size is added to end]
Register names	eax, ebx, ebp, esp, ...	%eax, %ebx, %ebp, %esp, ...
Constants	17, 42	\$17, \$42
Comments	; to end of line	# to end of line or /* ... */

Основные команды

Перемещение данных:

```
mov eax, ebx
```

```
mov ebp, byte ptr [var]
```

```
lea edx, [edx*2 + eax - 48]
```

Основные команды

Перемещение данных:

```
mov eax, ebx
```

```
mov ebp, byte ptr [var]
```

```
lea edx, [edx*2 + eax - 48]
```

Работа со стэком:

```
push eax
```

```
pop edi
```

```
pushf
```

```
popf
```


Основные команды

Перемещение данных:

```
mov eax, ebx
```

```
mov ebp, byte ptr [var]
```

```
lea edx, [edx*2 + eax - 48]
```

Работа со стэком:

```
push eax
```

```
pop edi
```

```
pushf
```

```
popf
```

Вызов подпрограмм:

```
call label
```

```
ret
```

```
int number
```

Основные команды

Перемещение данных:

```
mov eax, ebx
```

```
mov ebp, byte ptr [var]
```

```
lea edx, [edx*2 + eax - 48]
```

Работа со стэком:

```
push eax
```

```
pop edi
```

```
pushf
```

```
popf
```

Вызов подпрограмм:

```
call label
```

```
ret
```

```
int number
```

Арифметические операции:

```
add rcx, rax
```

```
sub rcx, rax
```

```
inc edx
```

```
dec ebp
```

```
neg ax
```

```
imul rax, rcx
```

```
idiv rcx, rdx
```


Основные команды

Перемещение данных:

```
mov eax, ebx
```

```
mov ebp, byte ptr [var]
```

```
lea edx, [edx*2 + eax - 48]
```

Работа со стэком:

```
push eax
```

```
pop edi
```

```
pushf
```

```
popf
```

Вызов подпрограмм:

```
call label
```

```
ret
```

```
int number
```

Арифметические операции:

```
add rcx, rax
```

```
sub rcx, rax
```

```
inc edx
```

```
dec ebp
```

```
neg ax
```

```
imul rax, rcx
```

```
idiv rcx, rdx
```

Логические операции

```
and rbx, rdx
```

```
not rdx
```

```
or rax, rdx
```

```
xor ax, ax
```

```
cmp ax, bx
```

```
test ax, bx:
```

Основные команды

Переходы:

`je label` ;перейти, если равны (equal)

`jl label` ;перейти, если знаково меньше (less)

`jb label` ;перейти, если беззнаково меньше (below)

`jg label` ;перейти, если знаково больше (greater)

`ja label` ;перейти, если беззнаково больше (above)

`jne, jnl, jnb, jng, jna, ...` - инвертированные версии

ПРИМЕР:

`test rax, rdx`

`jz a` ; перейти, если `rax == 0`

`js b` ; перейти, если `rax < 0`

`a:`

`; какой-то код`

`b:`

`; какой-то еще код`

“Hello world!” для транслятора NASM

```
global _start
```

```
section .rodata
```

```
hello_world: db "Hello world!", 0x0
```

```
section .text
```

```
_start:
```

```
mov eax, 0x04
```

```
mov ebx, 0x1
```

```
mov ecx, hello_world
```

```
mov edx, 14
```

```
int 0x80
```

```
mov eax, 0x01
```

```
mov ebx, 0
```

```
int 0x80
```

Передача параметров системному вызову int
0x80

eax	ebx	ecx	edx
Номер системного вызова	arg1	arg2	arg3

<https://habr.com/ru/post/423077/>

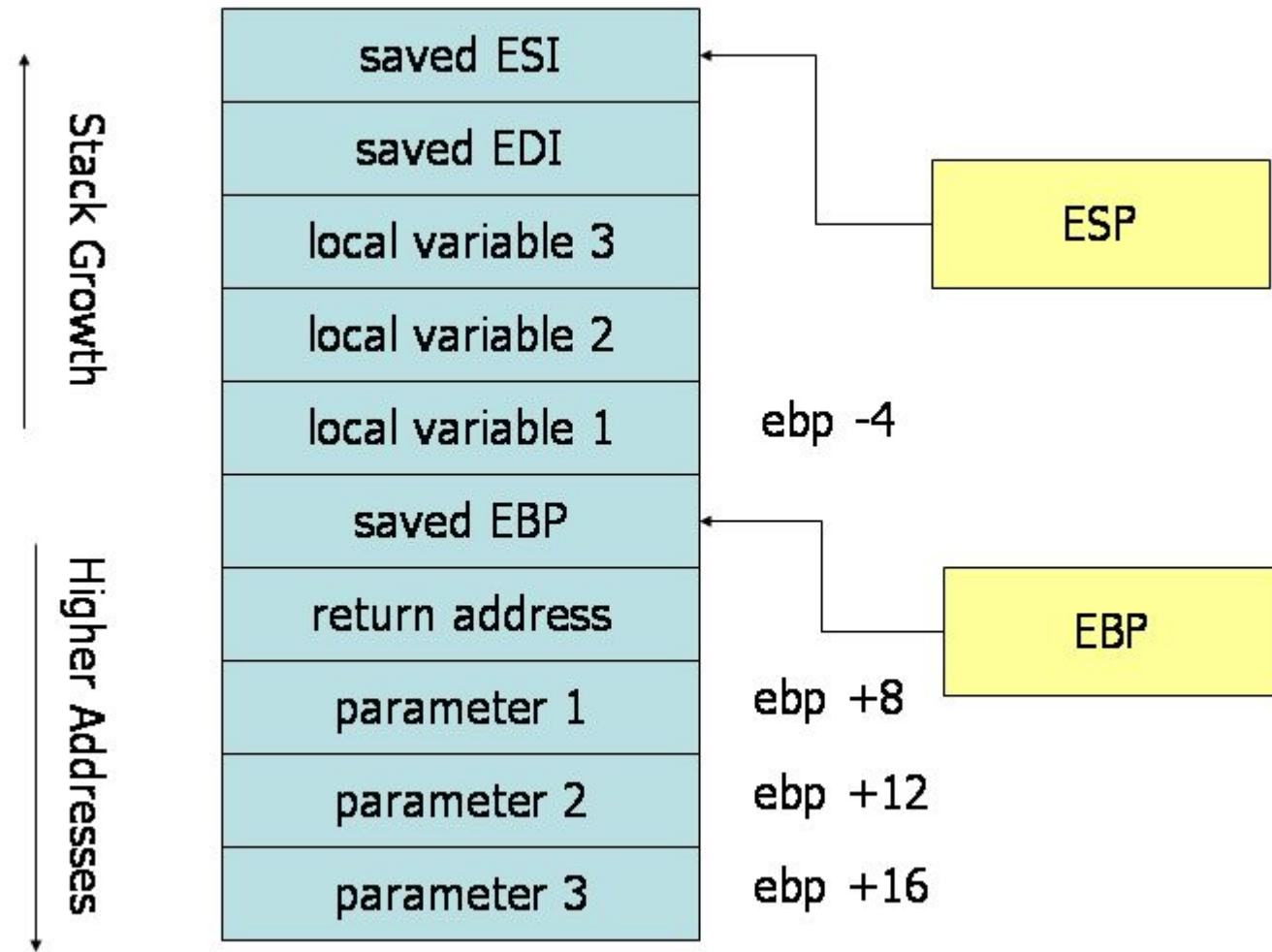
Передача параметров при cdecl-соглашении

```
_cdecl int MyFunction1(int a, int b)
{
    return a + b;
}
```

```
push 3
push 2
call _MyFunction1
add esp, 8

_MyFunction1:
push ebp
mov ebp, esp
mov eax, [ebp + 8]
mov edx, [ebp + 12]
add eax, edx
pop ebp
ret
```

Кадр стэка



Пролог и эпилог функций

```
push  ebp          ; Save the stack-frame base pointer (of the calling function).  
mov   ebp, esp     ; Set the stack-frame base pointer to be the current  
                        ; location on the stack.  
sub   esp, N       ; Grow the stack by N bytes to reserve space for local variables
```

```
mov   esp, ebp     ; Put the stack pointer back where it was when this function  
                        ; was called.  
pop   ebp          ; Restore the calling function's stack frame.  
ret                     ; Return to the calling function.
```

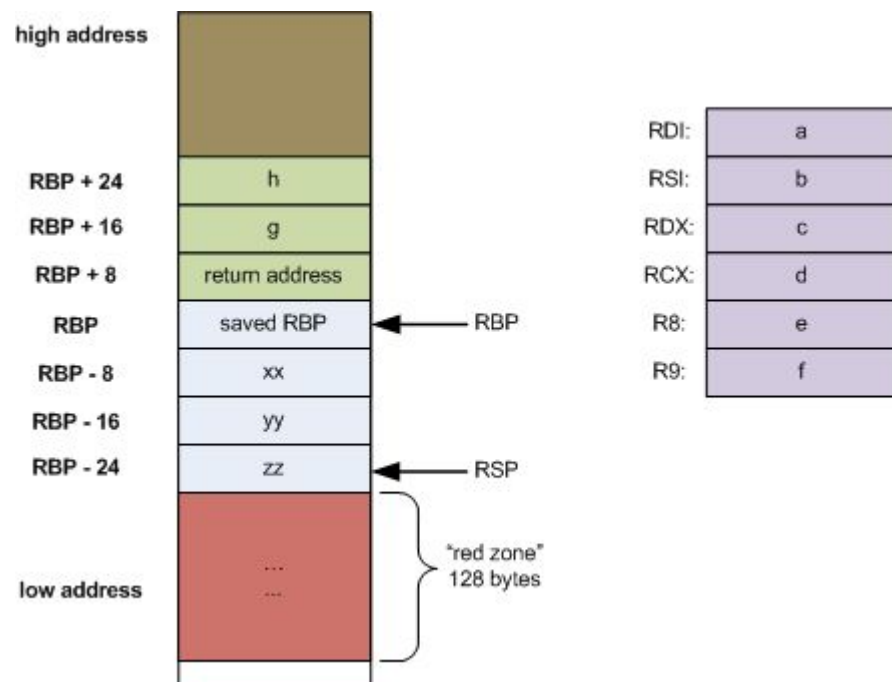

Структура стэка (AMD64 ABI)

```
long myfunc(long a, long b, long c, long d, long e, long f, long g, long h)
{
    long xx = a * b * c * d * e * f * g * h;
    long yy = a + b + c + d + e + f + g + h;
    long zz = utilfunc(xx, yy, xx % yy);
    return zz + 20;
}
```

Как по вашему мнению будут передаваться параметры в функцию?

Структура стэка (AMD64 ABI)

```
long myfunc(long a, long b, long c, long d, long e, long f, long g, long h)
{
    long xx = a * b * c * d * e * f * g * h;
    long yy = a + b + c + d + e + f + g + h;
    long zz = utilfunc(xx, yy, xx % yy);
    return zz + 20;
}
```

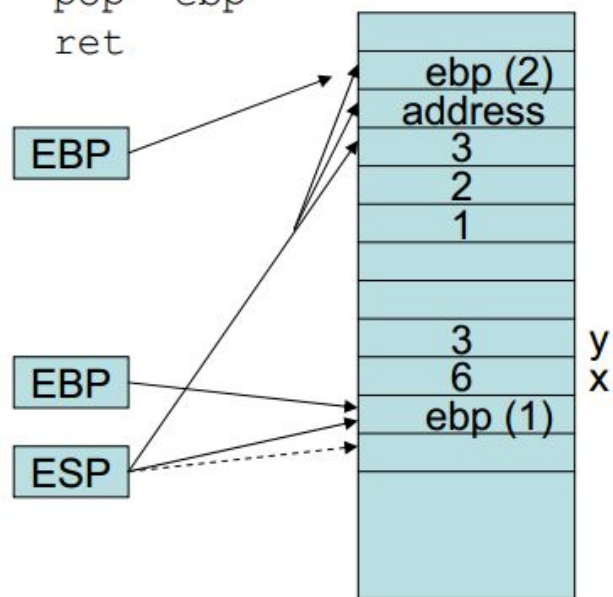


Passing parameters to functions

```
func(int a, int b, int c)
{
    return a+b+c;
}
main()
{
    int x, y=3;
    x=func(y,2,1);
}
```

```
func:
    push ebp
    mov  ebp, esp
    mov  eax, DWORD PTR [ebp+8]
    add  eax, DWORD PTR [ebp+12]
    add  eax, DWORD PTR [ebp+16]
    pop  ebp
    ret
```

```
main:
    push ebp
    mov  ebp, esp
    sub  esp, 28
    mov  DWORD PTR [ebp-8], 3
    mov  DWORD PTR [esp+8], 1
    mov  DWORD PTR [esp+4], 2
    mov  eax, DWORD PTR [ebp-8]
    mov  DWORD PTR [esp], eax
    call func
    mov  DWORD PTR [ebp-4], eax
    leave
    ret
```



Как выглядит C код в ассемблере

Онлайн транслятор C в ассемблер: <https://godbolt.org/>

Домашнее задание

1 Домашнее задание приведено на сайте Отус



Срок:



Пройдите, пожалуйста, опрос в Чате с преподавателем после приёмки вашего ДЗ

Рефлексия



С какими основными мыслями и инсайтами уходите с вебинара



Каких целей вебинара не удалось достичь


Список материалов для изучения

<https://eax.me/assembler-basics/> - хороший гайд по основам

Столяров - Азы программирования

Отладка Windows приложений

<https://ravesli.com/uroki-assemblera/>

The background of the image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a white geometric network pattern of dots and lines. The text is centered within this blue layer.

Заполните, пожалуйста,
опрос о занятии по ссылке в чате



Спасибо за внимание!

Приходите на следующие вебинары

Напишите вашу Фамилию и Имя,
укажите контакты