Лекция 5

22 февраля

Языки программирования (ЯП), базирующиеся на стеке вызовов

- ЯП с поддержкой рекурсии
  - C, Pascal, Java, …
  - Код функции можно вызывать повторно (“Reentrant”)
    - Одновременно могут выполняться несколько вызовов функции
  - Необходимо выделять память под сохранение состояния каждого работающего вызова
    - Аргументы
    - Локальные переменные
    - Адрес возврата
- Стек
  - Сохранять состояние вызова функции надо в ограниченный период времени: от момента вызова до момент выхода
  - Вызываемая функция всегда завершается до вызывающей
- Стек выделяется **Фреймами**
  - Состояние отдельного вызова функции

Порядок вызова функции

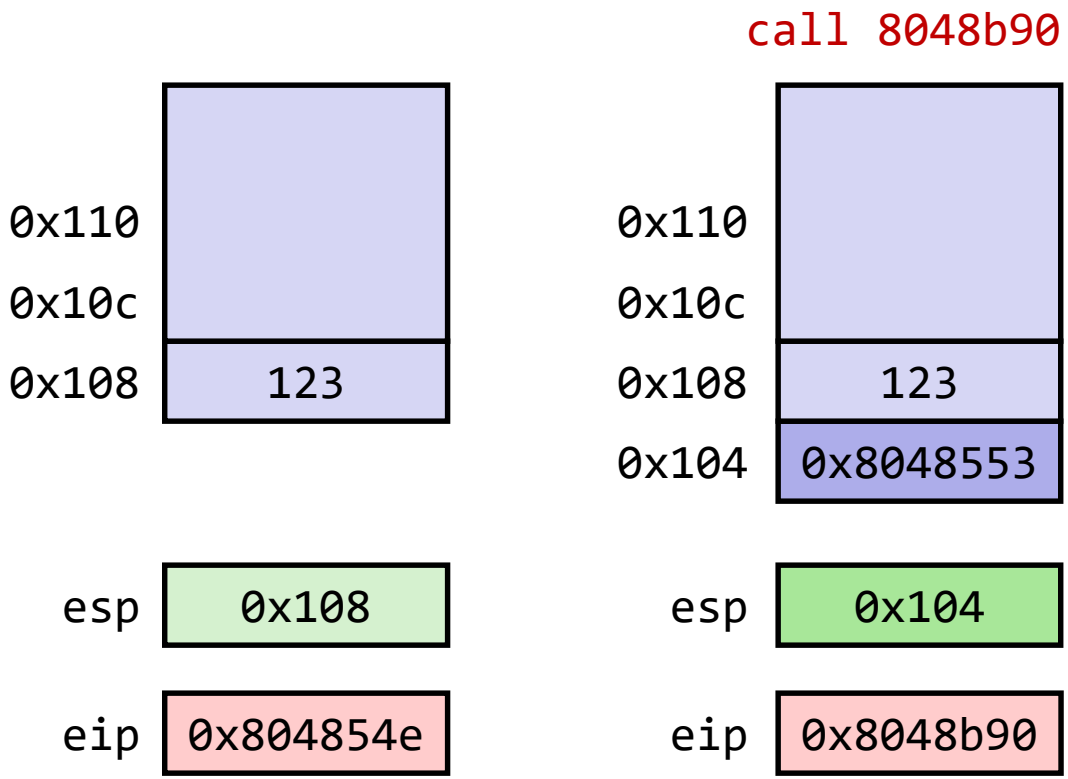
- Аппаратный стек используется для вызова функций и возврата из них
- Вызов функции: **`call label`**
  - На стек помещается адрес возврата
  - Выполняется прыжок на метку *label*
- Адрес возврата:
  - Адрес инструкции непосредственно расположенной за инструкцией call

```
804854e:   e8 3d 06 00 00      call  8048b90 <main>
8048553:   50                  push  eax
```

  - Адрес возврата = `0x8048553`
- Возврат из функции: **ret**
  - Выгрузка адреса из стека
  - Прыжок на этот адрес

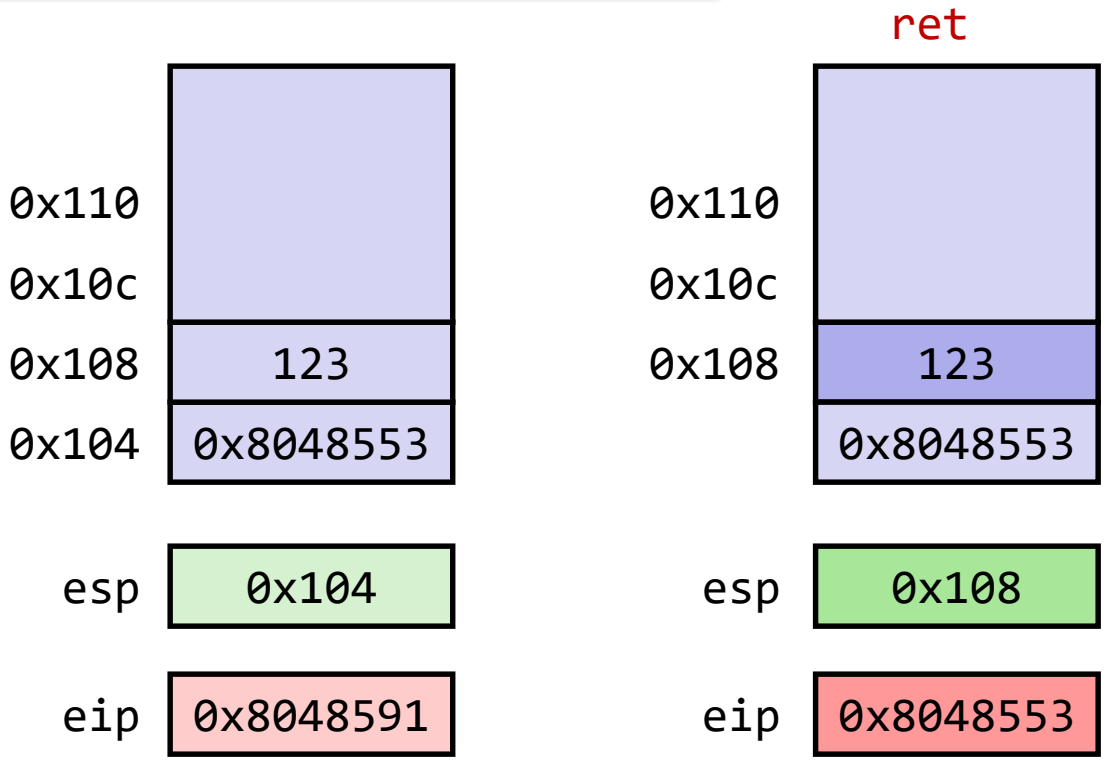
Вызов функции

```
804854e:    e8 3d 06 00 00    call  8048b90 <main>
8048553:    50                push  eax
```



Выход из функции

```
8048591:    c3                    ret
```

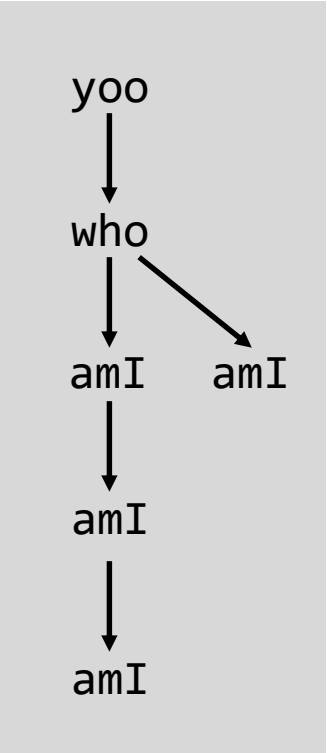


Пример цепочки вызовов

```
yoo(…)
{
  •
  •
  who();
  •
  •
}
```

```
who(…)
{
  • • •
  amI();
  • • •
  amI();
  • • •
}
```

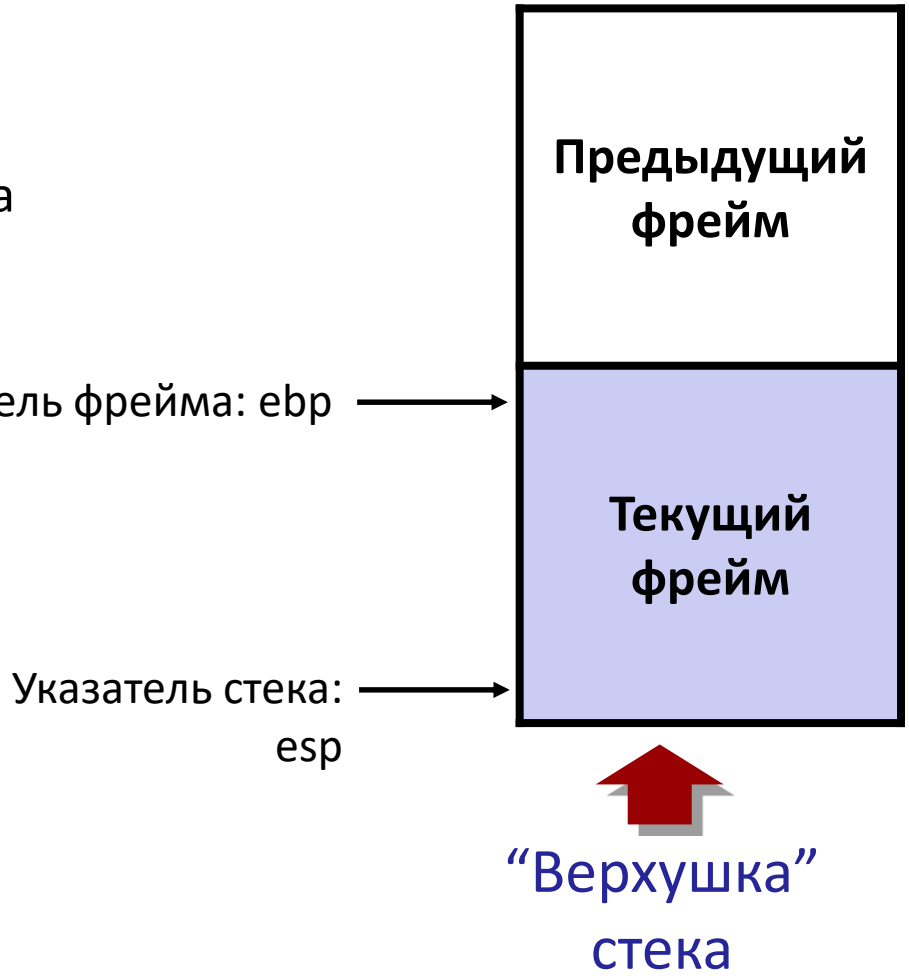
```
amI(…)
{
  •
  •
  amI();
  •
  •
}
```



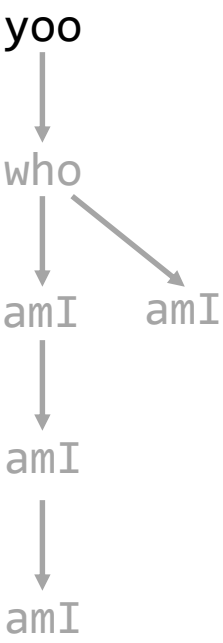
Функция amI() рекурсивная

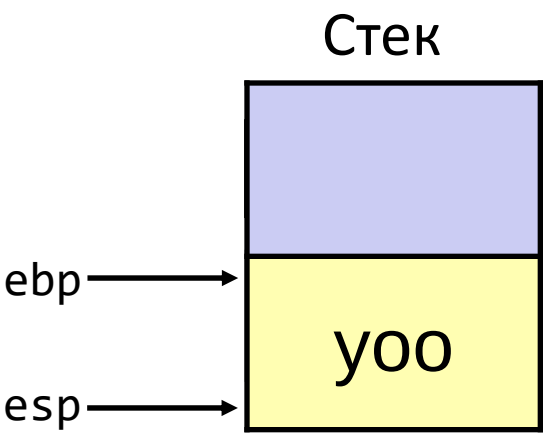
Стек фреймов

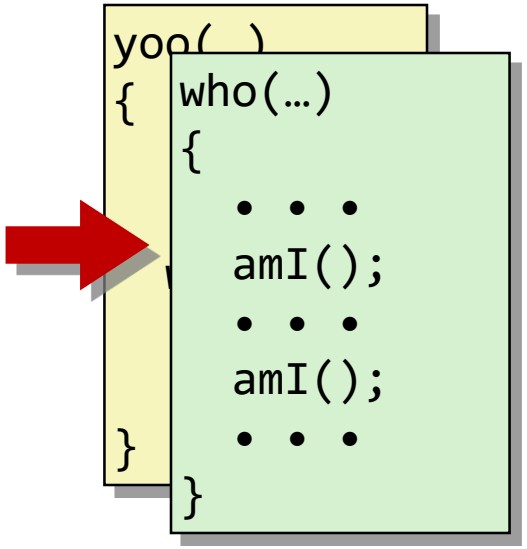
- Во фрейме размещаются
  - Локальные переменные
  - Данные, необходимые для возврата из функции
  - Временные переменные
- Управление фреймами
  - Пространство выделятся во время входа в функцию
    - «пролог» функции
  - Освобождается на выходе
    - «эпилог» функции

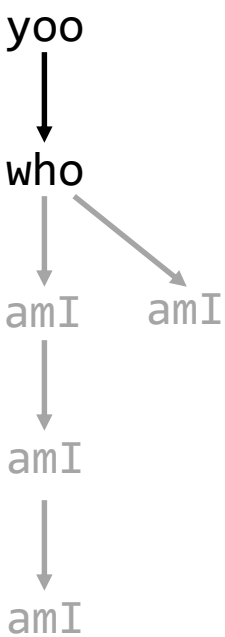


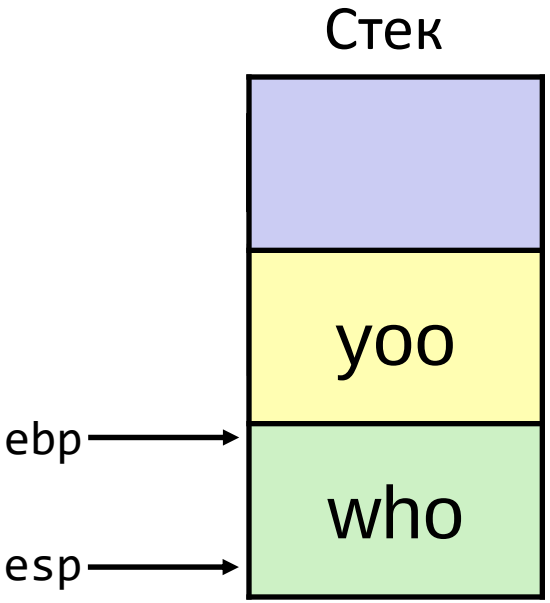
```
yoo(…)
{
   •
   •
   who();
   •
   •
}
```

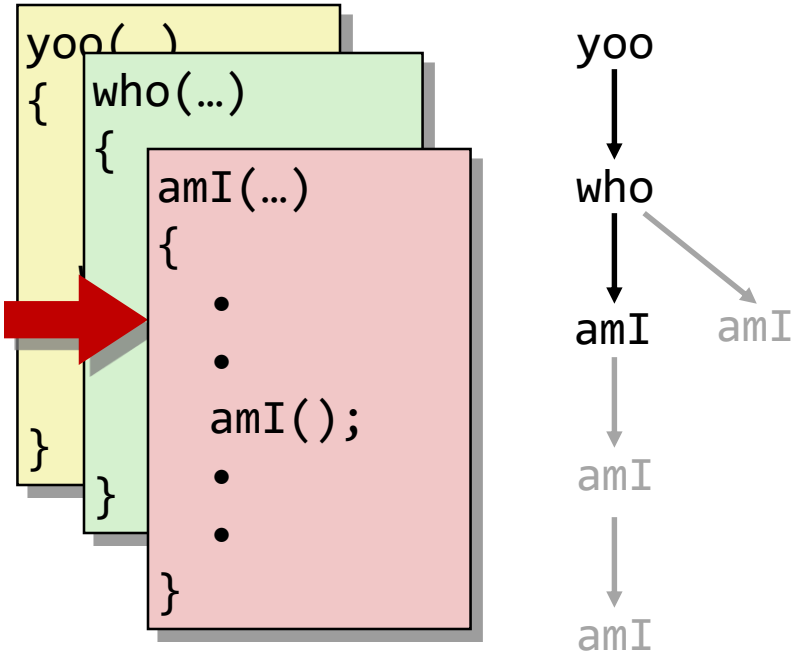


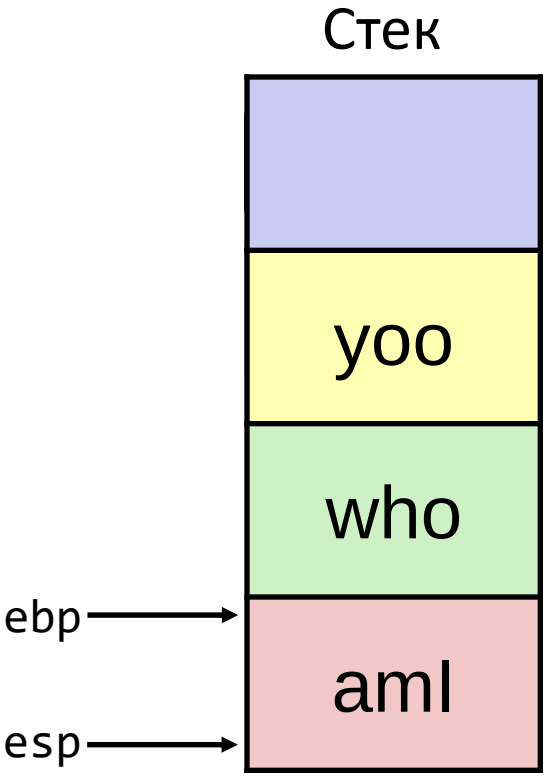


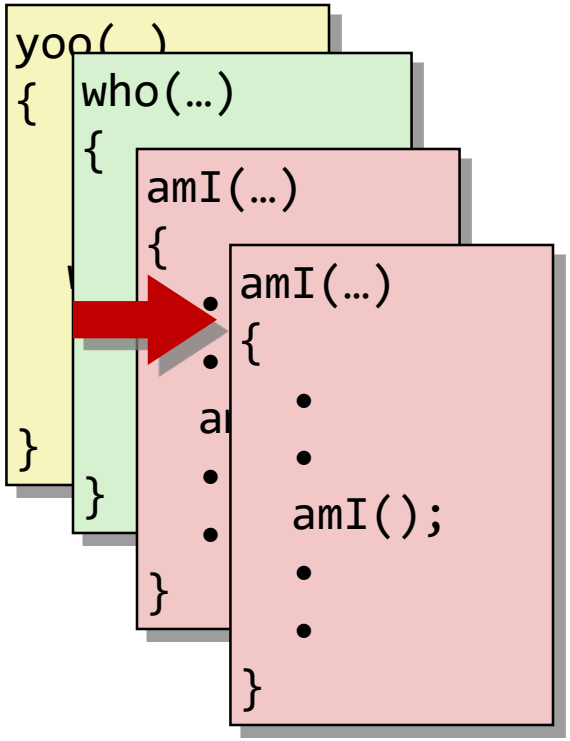


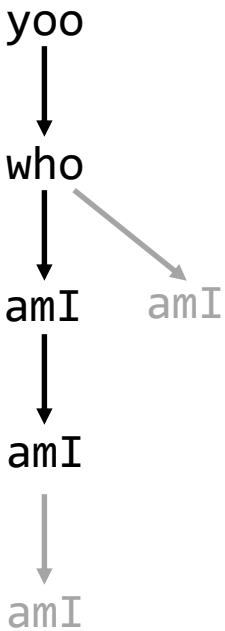


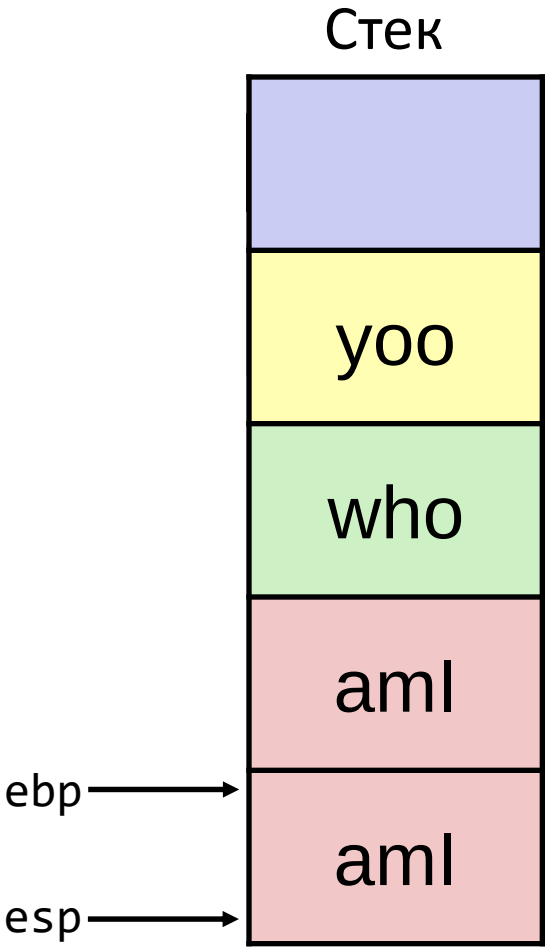


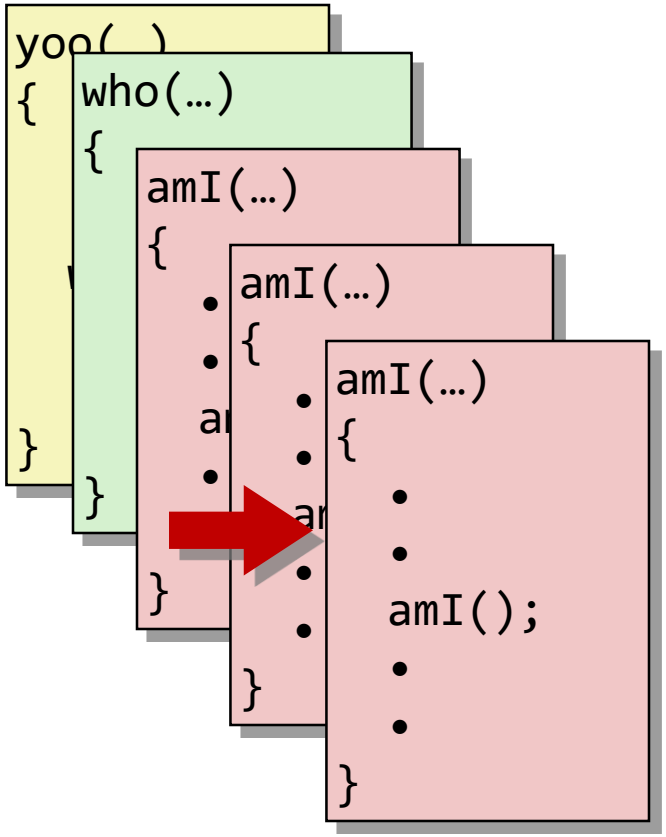


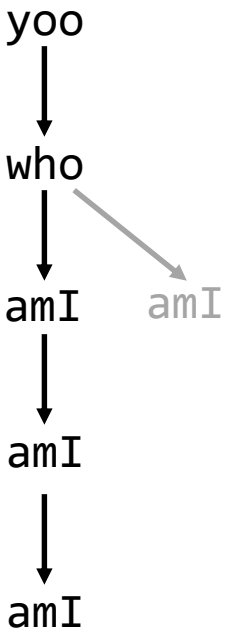


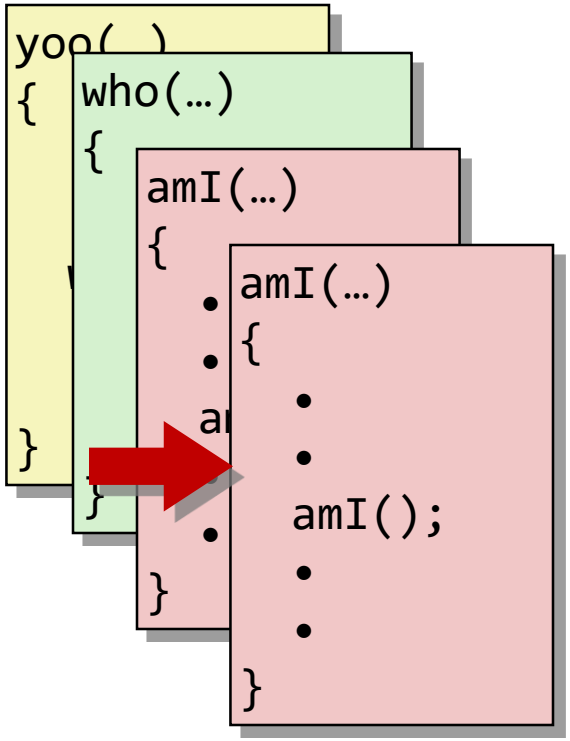


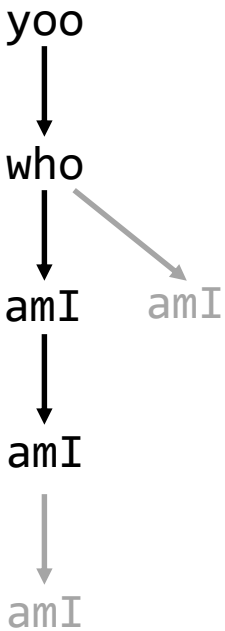


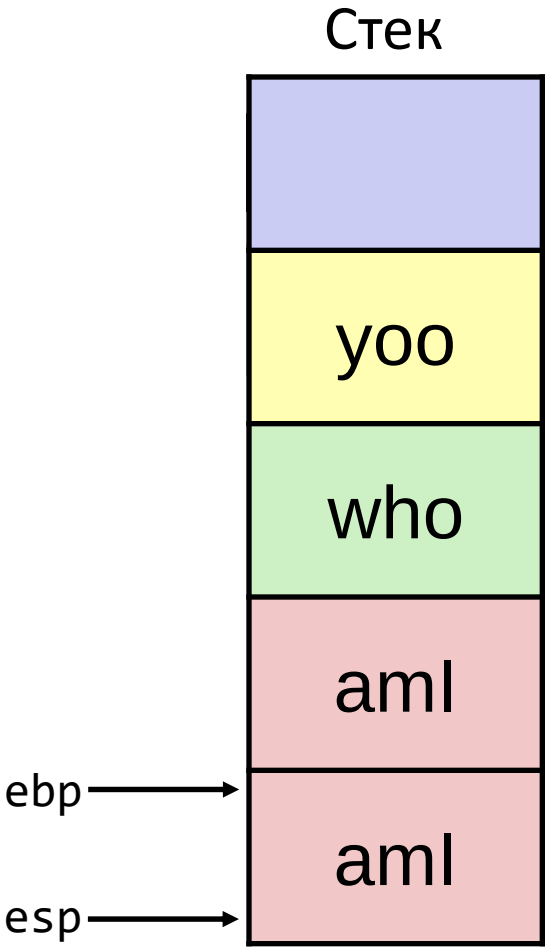


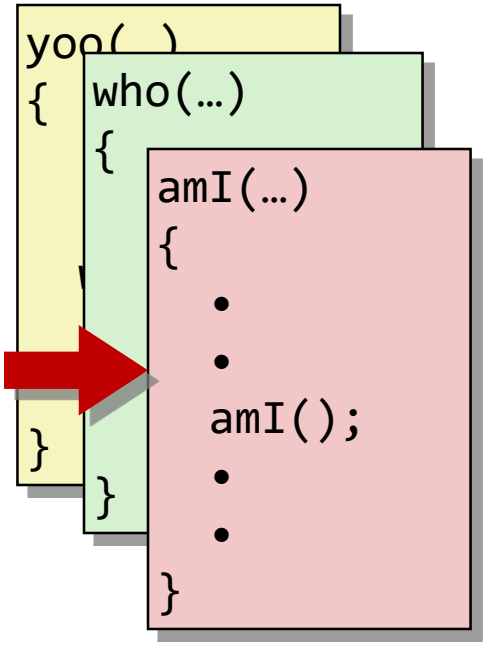


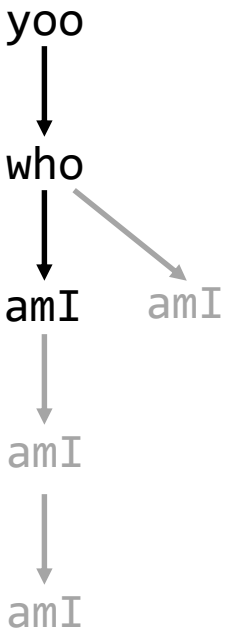


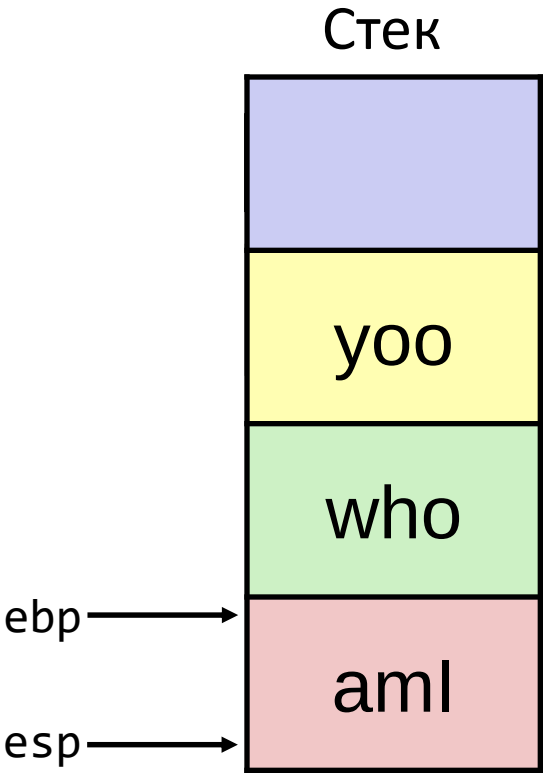


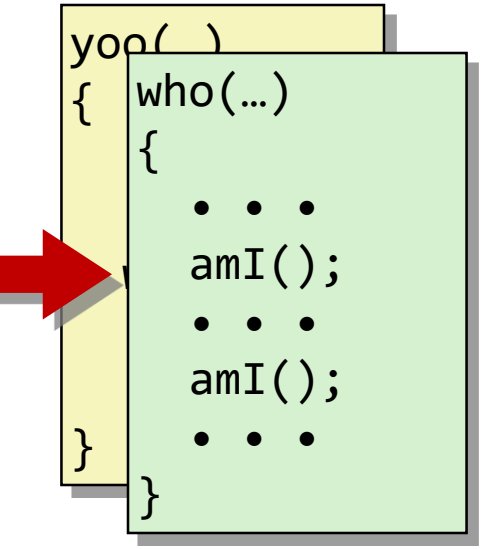


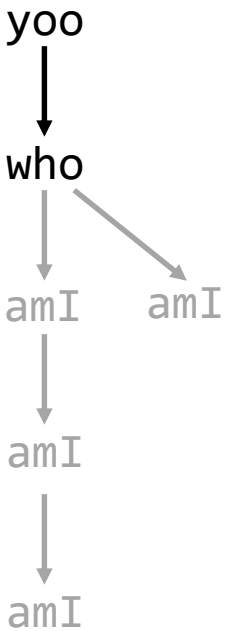


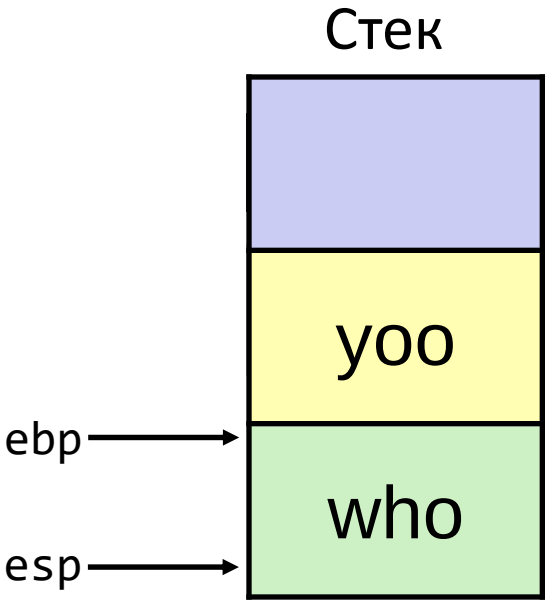


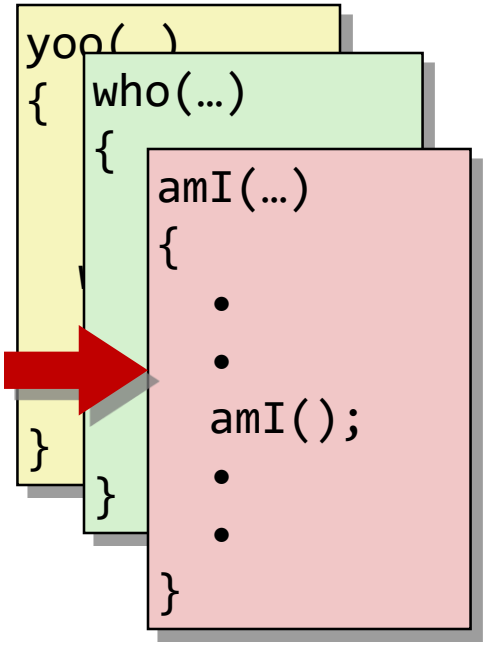


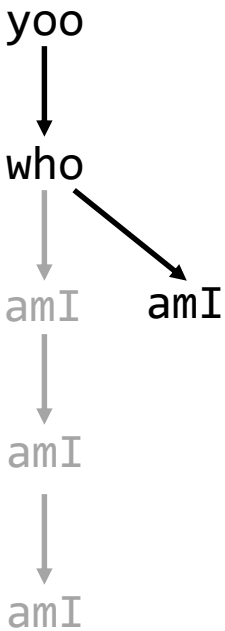


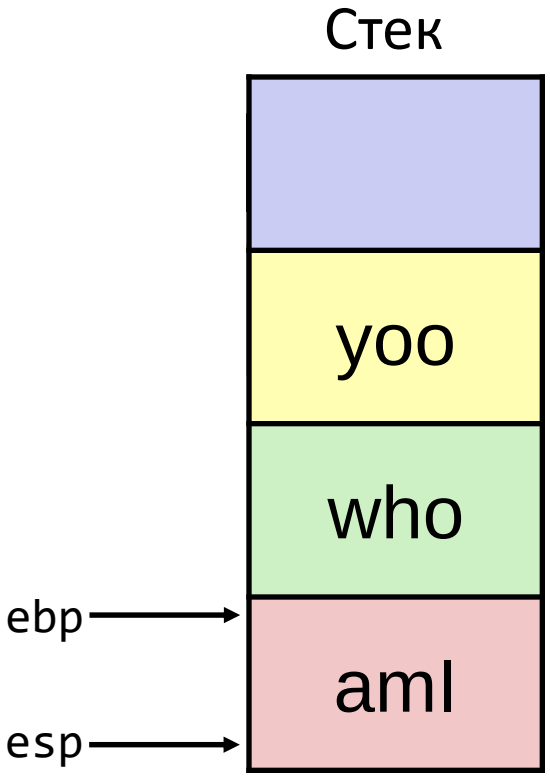


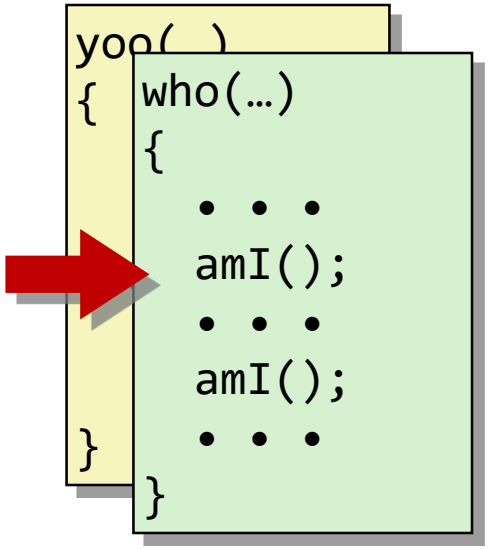


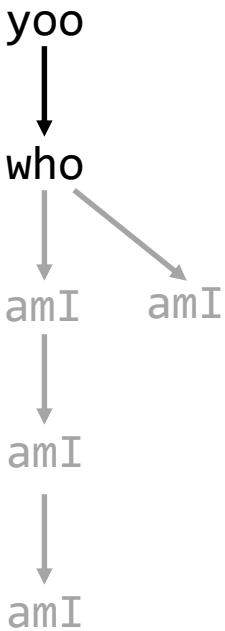


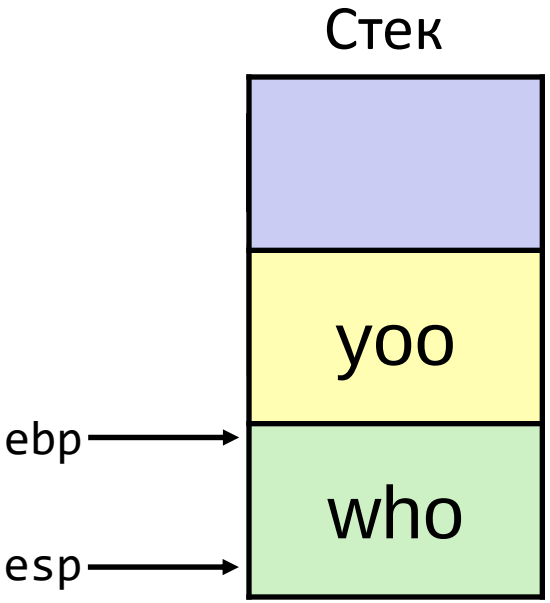




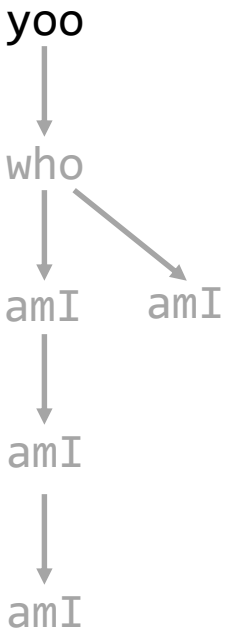


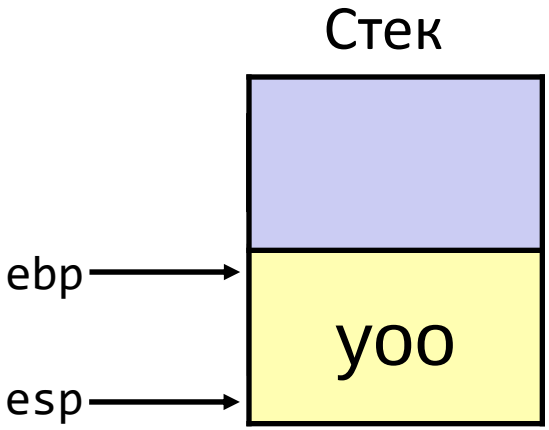






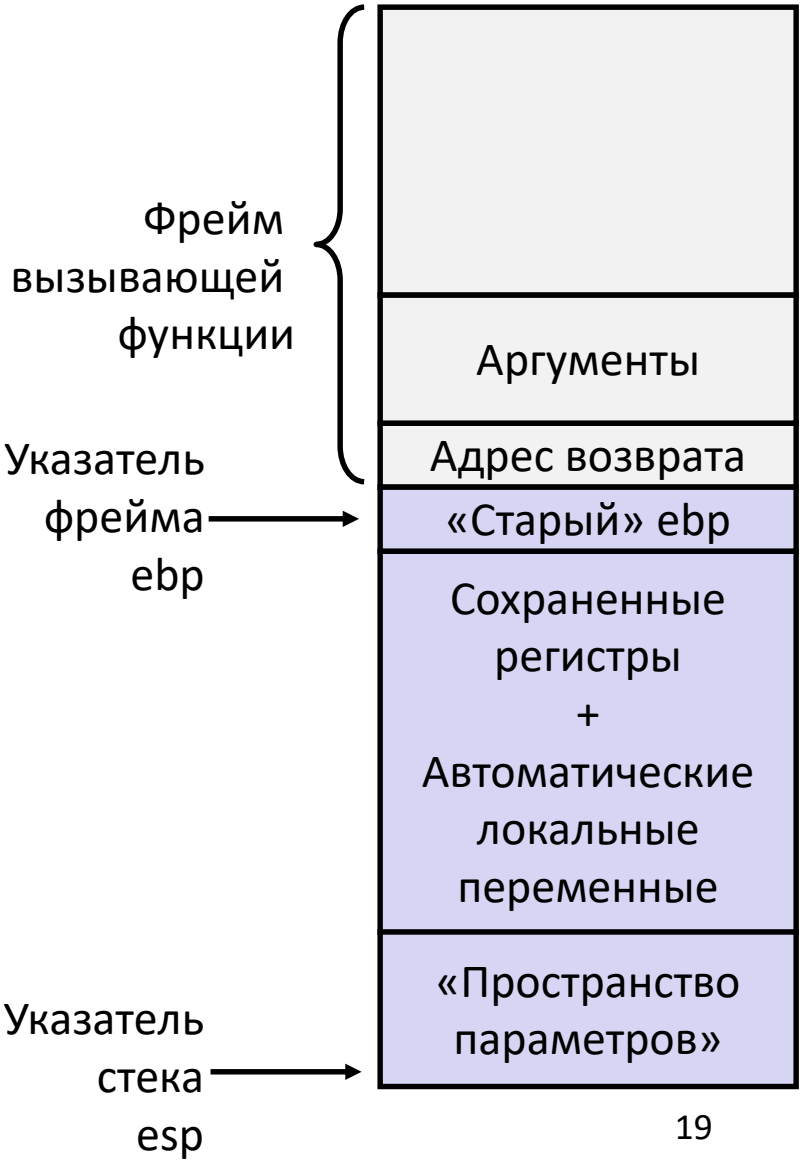
```
yoo(…)
{
   •
   •
   who();
   •
   •
}
```





Организация фрейма в IA-32/Linux

- Текущий фрейм (от “верхушки” ко «дну»)
  - “Пространство параметров”: фактические параметры вызываемых функций
  - Локальные переменные
  - Сохраненные регистры
  - Прежнее значение указателя фрейма
- Фрейм вызывающей функции
  - Адрес возврата
    - Помещается на стек инструкцией call
  - Значения фактических аргументов для текущего вызова

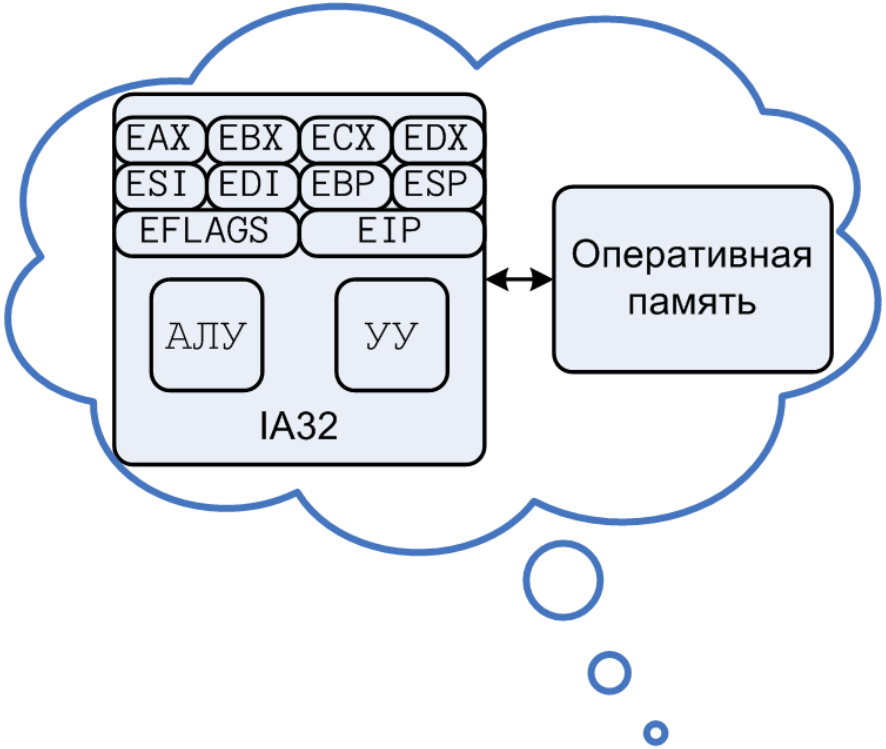


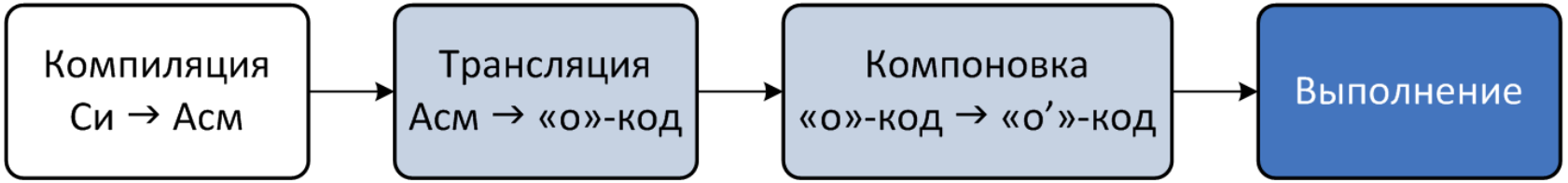
```c
int main() {
   int a = 1, b = 2, c;
   c = sum(a, b);
   return 0;
}

int sum(int x, int y) {
   int t = x + y;
   return t;
}
```

```
%include 'io.inc'
section .text
global CMAIN
CMAIN:
...
   mov    dword [ebp-16], 0x1  ; (1)
   mov    dword [ebp-12], 0x2  ; (2)
   mov    eax, dword [ebp-12]  ; (3)
   mov    dword [esp+4], eax   ; (4)
   mov    eax, dword [ebp-16]  ; (5)
   mov    dword [esp], eax     ; (6)
   call   sum                  ; (7)
   mov    dword [ebp-8], eax   ; (8)
...
global sum
sum:
   push   ebp                  ; (9)
   mov    ebp, esp             ; (10)
   sub    esp, 0x10            ; (11)
   mov    edx, dword [ebp+12]  ; (12)
   mov    eax, dword [ebp+8]   ; (13)
   add    eax, edx             ; (14)
   mov    dword [ebp-4], eax   ; (15)
   mov    eax, dword [ebp-4]   ; (16)
   mov    esp, ebp             ; (17)
   pop    ebp                  ; (18)
   ret                         ; (19)
```

Промежуточные итоги





Дальнейший материал

- Взаимосвязь языка Си, языка ассемблера и особенностей архитектуры IA32
  - Операции над целыми числами и битовыми векторами
    - «быстрая» арифметика и обработка 64 разрядных чисел
    - побитовые операции, сдвиги, вращения
  - реализация управляющих конструкций языка Си
  - адресная арифметика
  - массивы
  - структуры и объединения, выравнивание данных
  - соглашение вызова
    - cdecl, stdcall, fastcall
    - выравнивание стека
    - ускорение вызова функций
  - числа с плавающей точкой
  - …

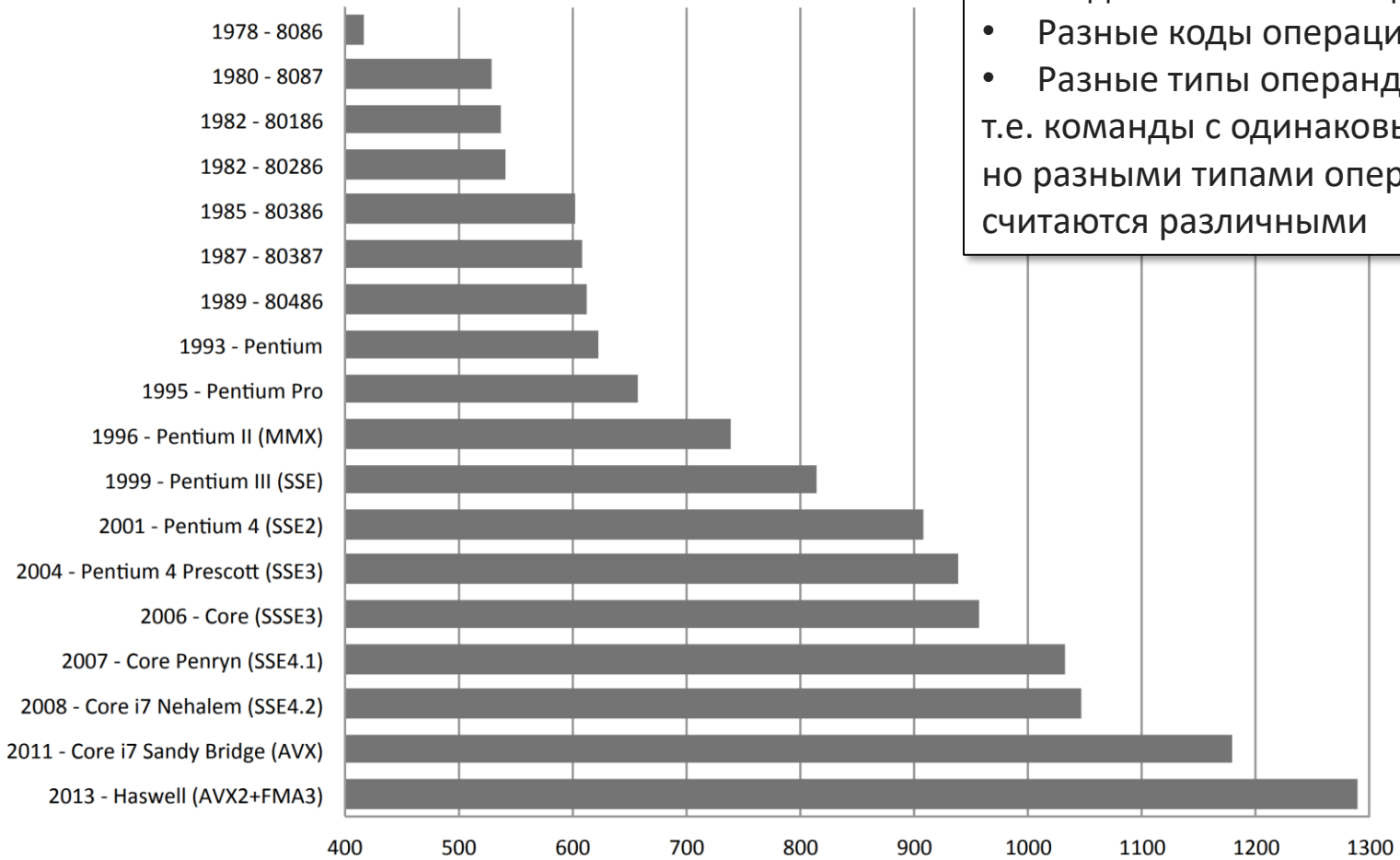
Группы команд

- Общего назначения
- x87 FPU
- MMX
- SSE
- IA-32e команды 64-разрядного режима работы
- Системные команды
- Аппаратная виртуализация
- …

- Пересылка данных
- Команды двоичной арифметики
- Команды двоично-десятичной арифметики
- Логические команды
- Сдвиги и вращения
- Битовые и байтовые команды
- Передача управления
- Строковые
- Ввод/Вывод
- Явное управление EFLAGS
- Вспомогательные

Цветом выделены группы команд, которые рассматриваются в курсе, на лекциях и семинарах.

Рост числа команд в архитектуре x86



Как подсчитывать команды?

- Разные коды операций
- Разные типы операндов

т.е. команды с одинаковым КОП, но разными типами операндов считаются различными

Суммарное число команд

Lopes B. et al. ISA aging: A X86 case study. – 2013.

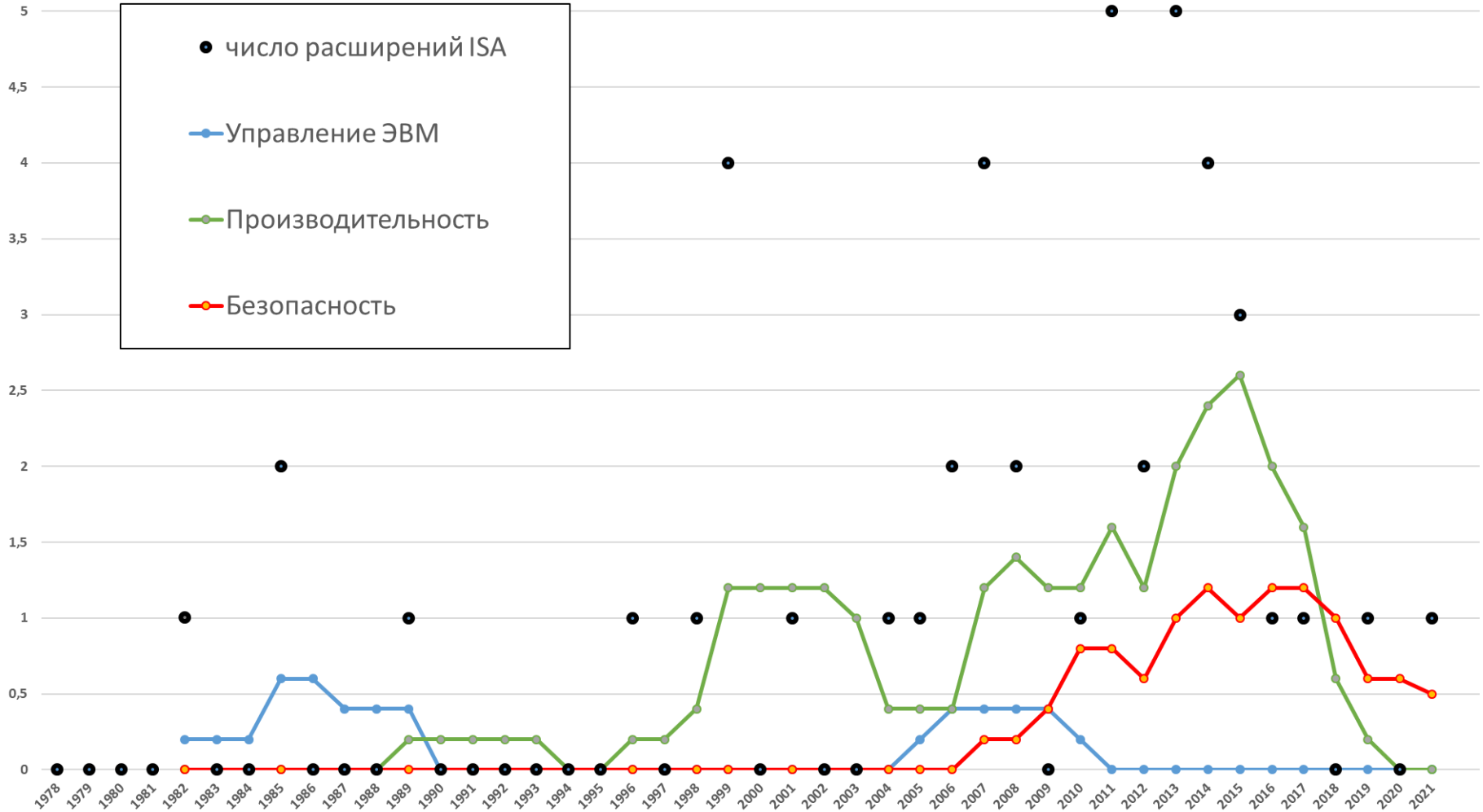
Год	Расширение	µархитектура	Описание
1978	Real	8086	Исходный набор команд реального режима
1982	Protected v1	80286	Первая версия защищенного режима
1985	Protected v2	80386	Защищенный режим
1985	SMM	80386	Режим управления системой (System Management Mode)
1989	FPU	80486	Интеграция x87 FPU в основной кристалл
1996	MMX	P5	Первая реализация SIMD-инструкций
1998	3DNow!	K6-2	SIMD-инструкции для работы с FP одинарной точности
1999	SSE	P5	SIMD-инструкции для работы с FP одинарной точности
1999	E3DNow!	K7	Расширение 3DNow! (новые DSP-инструкции + несколько MMX-инструкций)
1999	Professional 3DNow!		Маркетинговое название от AMD для E3DNow! + SSE
1999	EMMX	P6	Расширение MMX-инструкций
2001	SSE2	P6	Попытка заменить MMX-инструкции для более широких XMM регистров
2004	SSE3	Core	Третье расширение потоковых SIMD-инструкций
2005	VT-x	Pentium 4	Аппаратная виртуализация (Intel)
2006	AMD-V	Athlon 64	Аппаратная виртуализация (AMD)
2006	SSSE3	Core	Дополнение к расширению SSE3, инструкции для работы с запакованными целыми
2007	SSE4.1	Penryn	Расширение 4.1 для потоковых SIMD-инструкций
2007	SSE4a	K10	4 SIMD-инструкции, не относящиеся 4.1 или 4.2
2007	ABM	K10	Манипуляции отдельными битами
2008	SSE4.2	Nehalem	Расширение 4.2 для потоковых SIMD-инструкций
2007	SSE5		Предложено AMD в 2007, но никогда не было реализовано
2008	SSE4		Расширение 4 потоковых SIMD-инструкций, маркетинговое название для SSE4.1 + SSE4.2
2010	CLMUL	Westmere	Умножение без переноса для блочного шифрования
2011	AVX	Sandy Bridge	Операции над 256-разрядными целыми (Advanced Vector Extensions)
2012	F16C	Ivy Bridge	Приведение типов FP одинарной и половинной точности
2011	XOP	Bulldozer	Операции над векторами
2011	FMA4	Bulldozer	Умножение-сложение с однократным округлением над 4 операндами
2011	LWP	Bulldozer	Легковесное профилирование
2011	SMX	Nehalem	Контроль работы приложений
2011	AES	Westmere	Аппаратная реализация шифрования AES
2012	TBM	Piledriver	Манипуляции отдельными битами
2013	BMI1	Jaguar	Манипуляции отдельными битами
2013	BMI2	Haswell	Манипуляции отдельными битами
2013	FMA3	Haswell	Умножение-сложение с однократным округлением над 3 операндами
2013	TSX	Haswell	Транзакционная память
2013	AVX2	Haswell	Расширение AVX
2014	ADX	Broadwell	Сложение целых произвольной точности
2014	RdRand	Broadwell	Генерация случайных чисел
2014	PREFETCH	Broadwell	Упреждающая загрузка в кэш (ранее часть 3DNow!)
2015	AVX-512	Airmont	Операции над 512-разрядными регистрами
2015	MPX	Skylake	Защита памяти
2015	SGX	Skylake	Управление анклавами - области памяти, которые шифруются/расшифровываются на лету
2016	SHA	Goldmont	Вычисление криптографического хэша SHA
2017	SME	Zen	Шифрование/расшифровывание страниц памяти на лету
2019	TME	Ice Lake	Тотальное шифрование памяти
2021	CET	Tiger Lake	Контроль целостности потока управления

Справочная информация
Использовались материалы с сайта
https://en.wikichip.org/wiki/WikiChip

Как оценить тенденции в развитии ISA?

Простое скользящее среднее $SMA_t = \frac{1}{n}\sum_{i=0}^{n-1} p_{t-i}$

p_{t-i} — значение исходной функции в момент времени t, отдалённый от текущего на i интервалов.



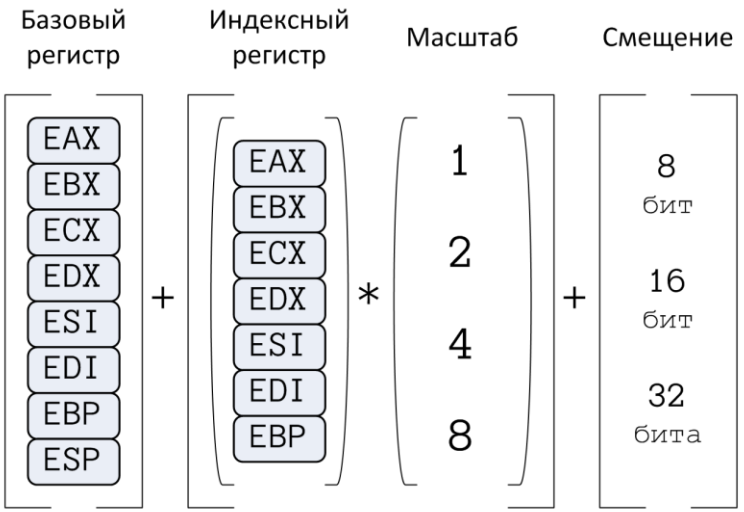
«Основные» команды IA-32

Пересылка данных	Двоичная арифметика ✓	Передача управления	Логические ✓	Сдвиги и вращения	Битовые и байтовые	Прочее
• MOV	• ADD	• JMP	• **AND**	• **SAR**	• **SETcc**	• **LEA**
• **XCHG**	• **ADC**	• Jcc	• **OR**	• **SHR**	• TEST	• NOP
• **BSWAP**	• SUB	• CALL	• **XOR**	• **SAL, SHL**		
• MOVSX	• **SBB**	• RET	• **NOT**	• **ROR**		
• MOVZX	• NEG			• **ROL**		
• CDQ	• IMUL			• **RCR**		
• CWD	• MUL			• **RCL**		
• CBW	• IDIV					
• PUSH	• DIV					
• POP	• INC					
• **CMOVcc**	• DEC					
	• CMP					

Красным выделены не упоминавшиеся ранее на лекциях ассемблерные инструкции.
Команды двоичной арифметики и логические команды представлены в полном составе.

Общий вид адресного кода при обращении к памяти

- LEA
  - r32, m



Исполнительный адрес = База + (Масштаб * Индекс) + Смещение

Виды (способы) косвенной адресации	Выражение	Пример
Относительная	База + Смещение	**[ebp-16]**
Базово-индексная	База + Индекс	**[eax+ecx]**
Базово-индексная с масштабом	База + Масштаб * Индекс	**[eax+4*ecx]**
Базово-индексная с масштабом и смещением	База + Масштаб * Индекс + Смещение	**[eax+4*ecx+10]**

Пример
Реализация стрелки Пирса

```
unsigned pierce_arrow(unsigned a, unsigned b) {
   int t = ~(a | b);
   return t;
}
```

```
section .text
global pierce_arrow
pierce_arrow:
   push    ebp
   mov     ebp, esp
   mov     eax, dword [ebp+12] ; (1)
   or      eax, dword [ebp+8]  ; (2)
   not     eax                 ; (3)
   pop     ebp
   ret
```