Стек и подпрограммы

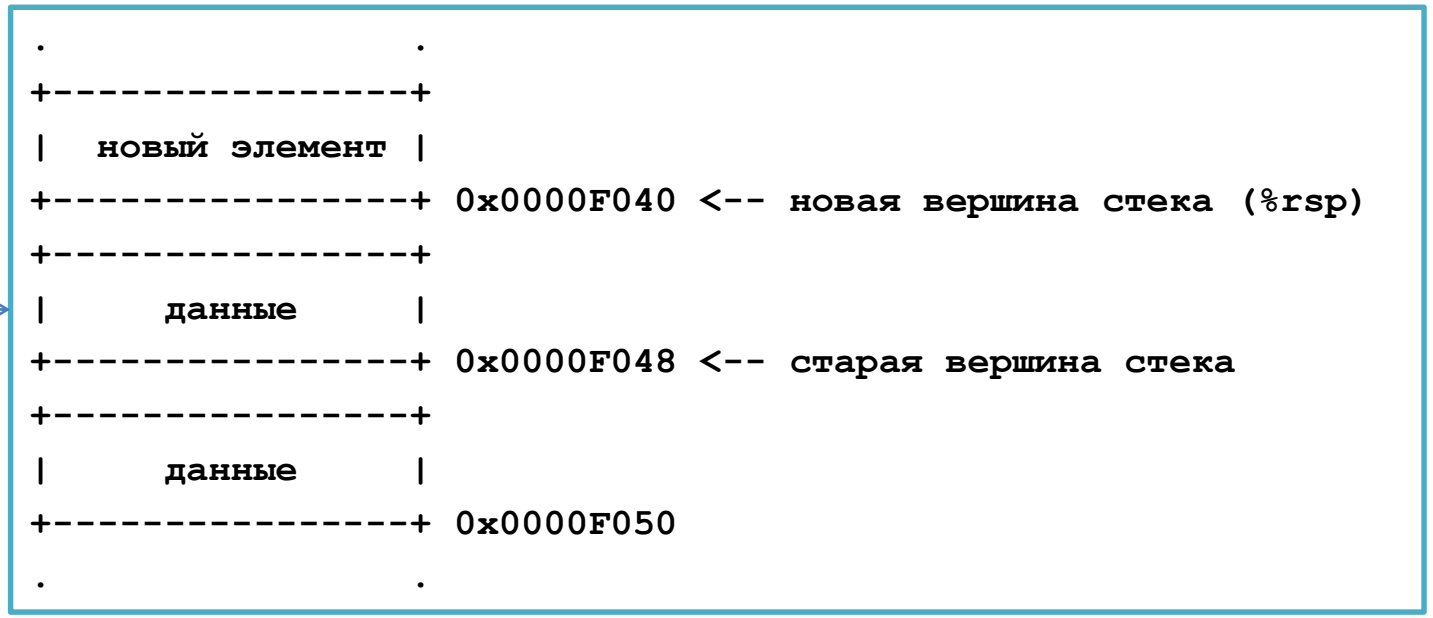
К лабе 2

Стек

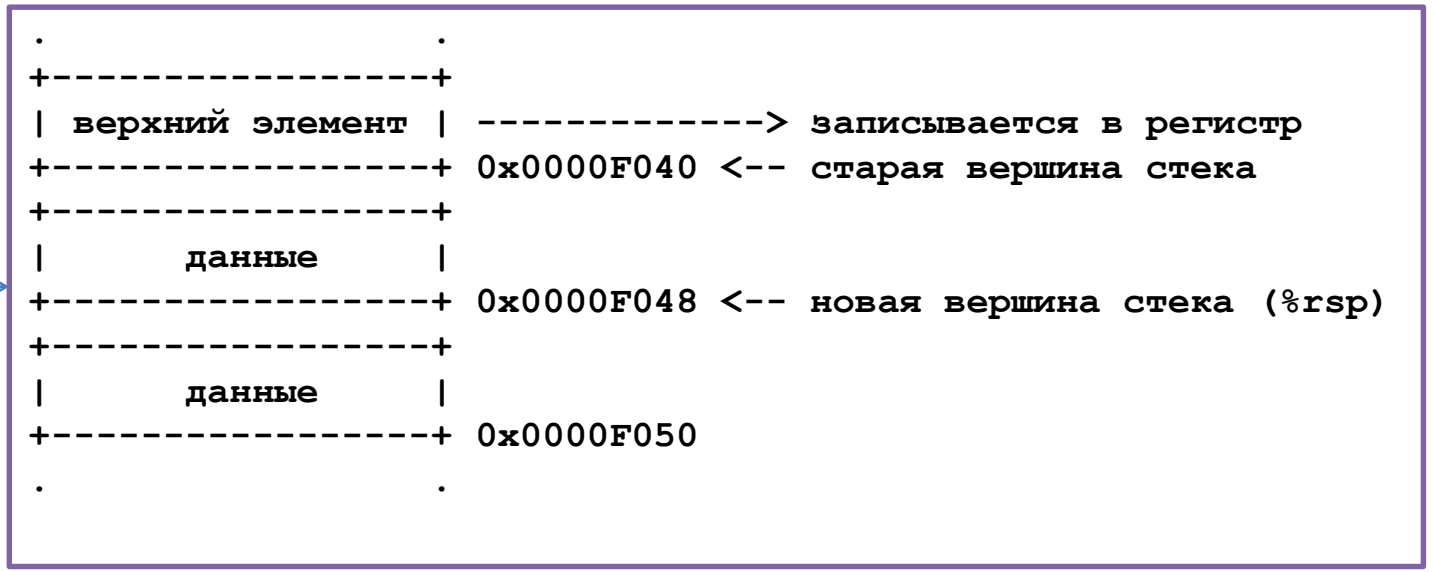

0x0000F040
0x0000F048
данные
0x0000F050 <-- вершина стека (%rsp)
данные
0x0000F058
0x0000FFF0
данные
0x0000FFF8
данные
0x00010000 <-- дно стека

Стек: *push* и *pop*

pushq



popq



Листинг функции (задача)

```
#include <stdio.h>
int x=2, y=5, z;

void sum()
{
  z = x+y;
  return;
}

int main()
{
  sum();
  return 0;
}
```

```
#include <stdio.h>
int x=2, y=5, z;

int sum()
{
  return x+y;
}

int main()
{
  z = sum();
  return 0;
}
```

```
#include <stdio.h>

int sum(int _x, int _y)
{
  return _x+_y;
}

int main()
{
  int x=2, y=5, z;
  z = sum(x,y);
  return 0;
}
```

- Как выглядит функция?
- Как передается возвращаемое значение?
- Как выглядят локальные переменные?

Вызов и возврат

```
call  метка
```

- Поместить в стек адрес следующей за call команды. Этот адрес называется адресом возврата.
- Передать управление на метку.

```
ret
ret   число
```

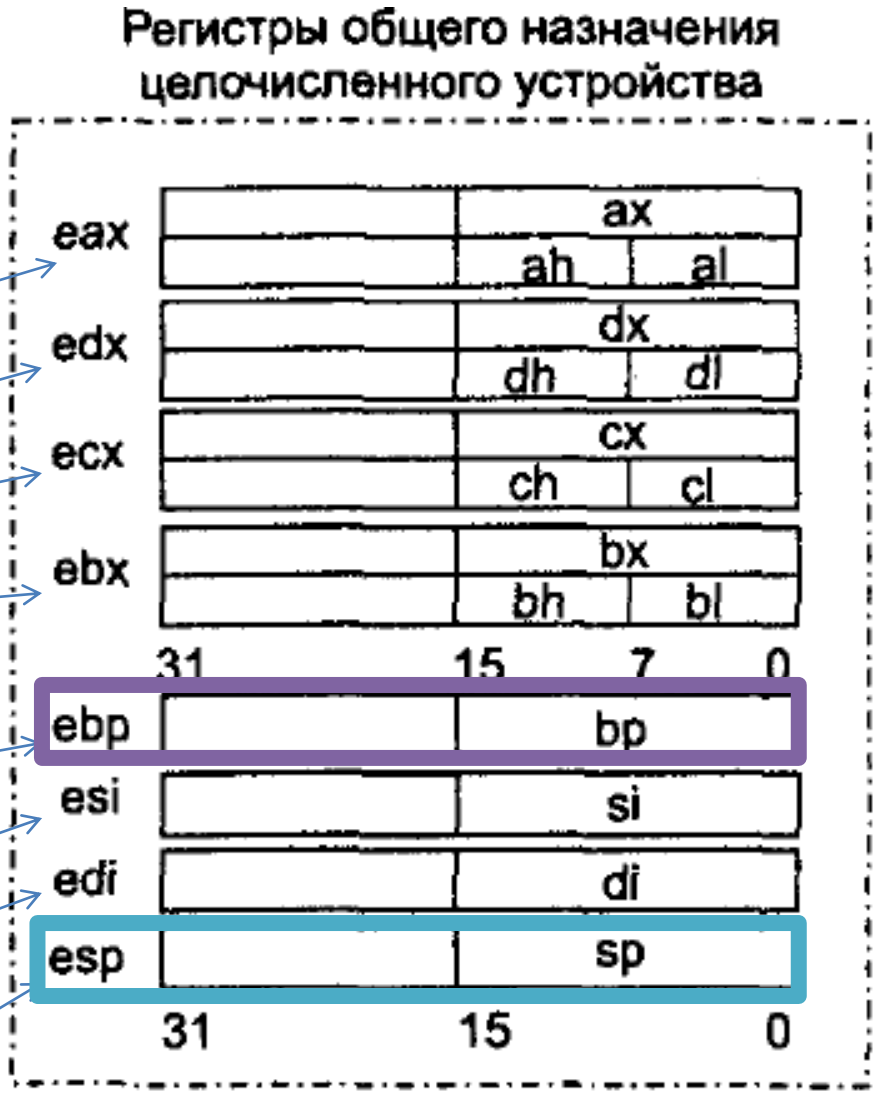
- Извлечь из стека новое значение регистра %eip (то есть передать управление на команду, расположенную по адресу из стека).
- Если команде передан операнд *число*, %esp увеличивается на это число. Это необходимо для того, чтобы подпрограмма могла убрать из стека свои параметры.

Регистры общего назначения

- операнды логических и арифметических операций
- компоненты адреса
- указатели на ячейки памяти

- Аккумулятор
- Регистр данных
- Регистр-счетчик
- Базовый регистр

- Регистр указателя базы кадра стека
- Индекс источника
- Индекс приемника
- Регистр указателя стека



Передача аргументов (x32)

- **При помощи регистров (x64).**
  - число регистров ограничено
  - может не хватить регистров для собственной работы
  + доступ к регистрам очень быстрый.

- **При помощи общей области памяти.**
  - не поддерживает многопоточное выполнение кода
  - невозможно сказать, какие глобальные переменные она изменяет и где ожидает свои параметры

  Не используйте без крайней необходимости.

- **При помощи стека**

```
sub:
        pushl %ebp

        movl  %esp, %ebp
```

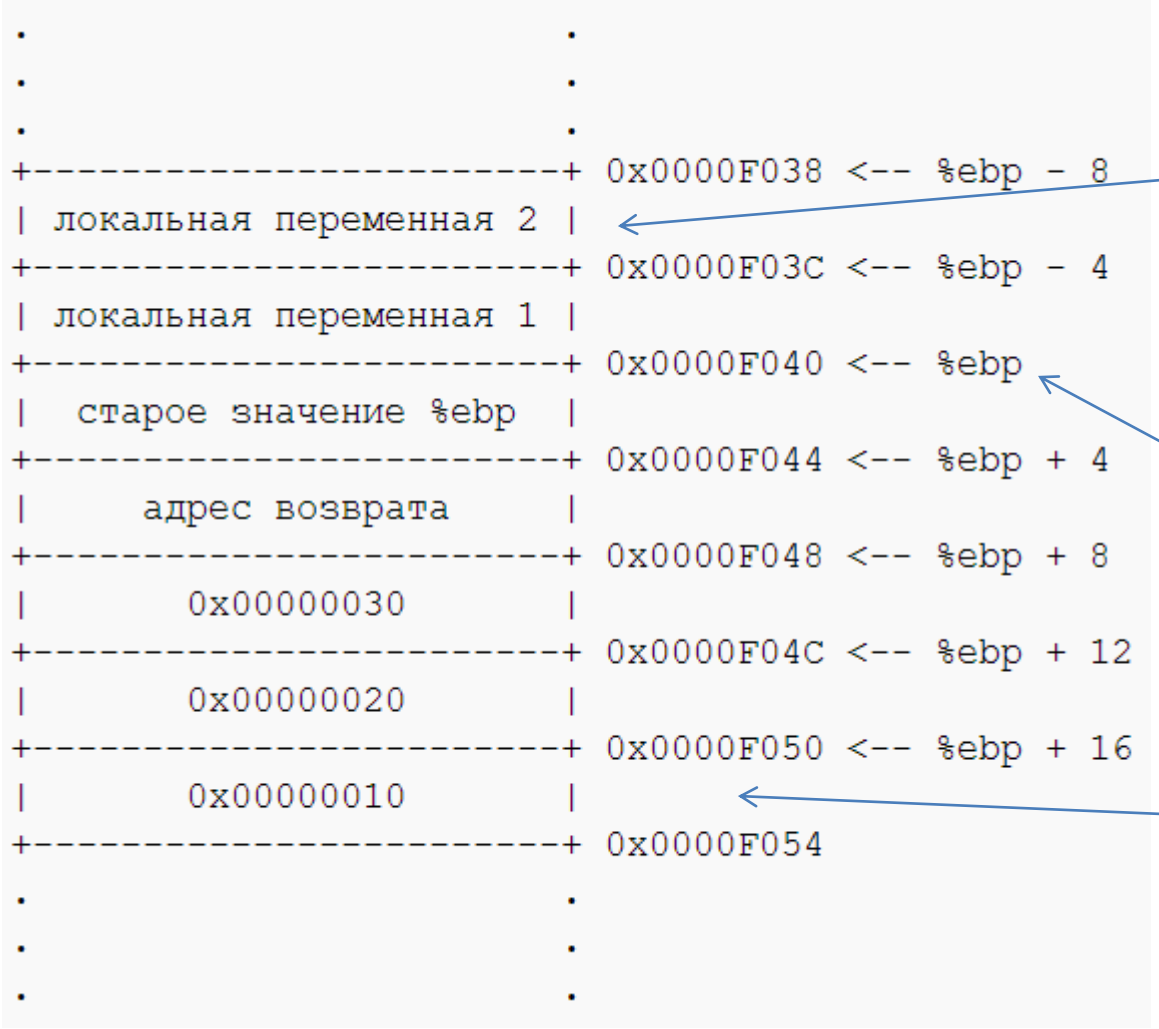
пролог sub

```
main:
        pushl $0x00000010
        pushl $0x00000020
        pushl $0x00000030
        call  sub
        addl  $12, %esp
```

вызов sub

```
.                       .
.                       .
.                       .
+-----------------------+ 0x0000F040 <-- новое значение %ebp
| старое значение %ebp  |
+-----------------------+ 0x0000F044 <-- %ebp + 4
|    адрес возврата     |
+-----------------------+ 0x0000F048 <-- %ebp + 8
|      0x00000030       |
+-----------------------+ 0x0000F04C <-- %ebp + 12
|      0x00000020       |
+-----------------------+ 0x0000F050 <-- %ebp + 16
|      0x00000010       |
+-----------------------+ 0x0000F054
.                       .                  8(%ebp) = 0x00000030
.                       .                 12(%ebp) = 0x00000020
.                       .                 16(%ebp) = 0x00000010
```

Локальные переменные



память не инициализирована

«якорь» функции

обратный порядок аргументов

Локальные переменные:

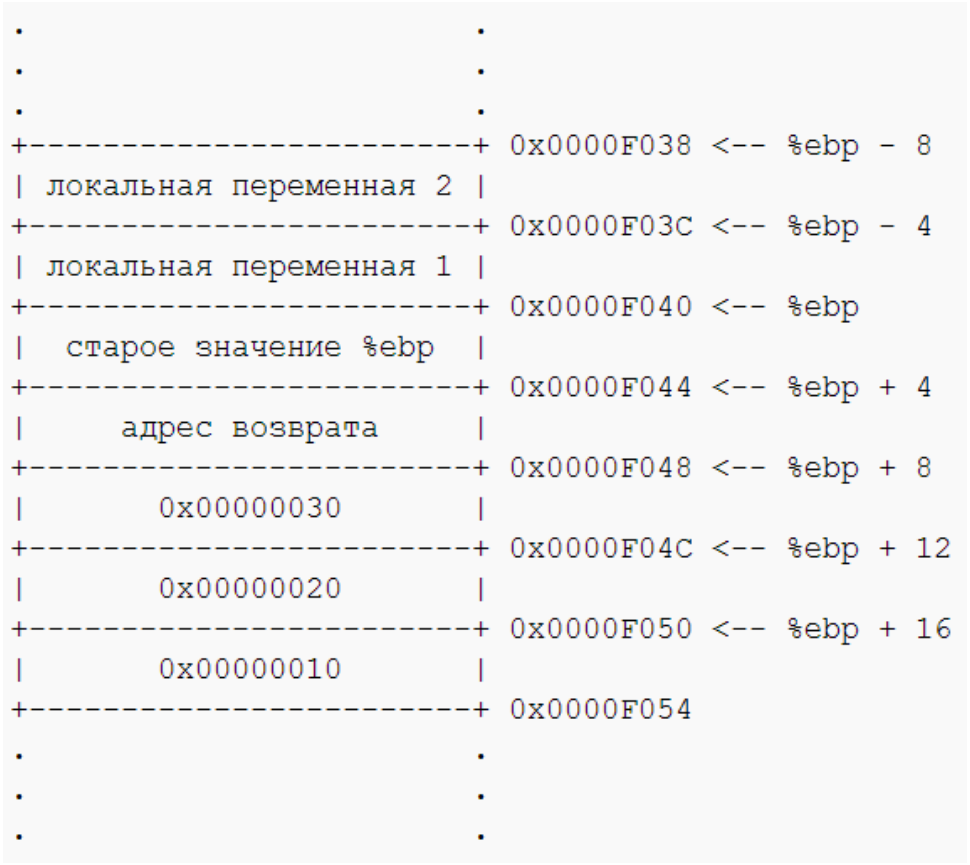
выделение памяти

```
subq    $16,    %rsp            //int x,y;
movl    $5,     -8(%rbp)        //x = 5;
movl    $10,    -16(%rbp)       //y = 10;
```

```
subq    $8,     %rsp            //int x,y;
movl    $5,     -4(%rbp)        //x = 5;
movl    $10,    -8(%rbp)        //y = 10;
```

```
subq    $80,    %rsp            //int x[10];
movl    $5,     -8(%rbp)        //x[0] = 5;
movl    $10,    -56(%rbp)       //x[6] = 10;
```

Эпилог подпрограммы



эпилог sub

```
movl   %ebp, %esp

popl   %ebp

ret
```

- использовать команду ret с аргументом;
- использовать необходимое число раз команду pop и выбросить результат;
- увеличить %esp на размер всех помещенных в стек параметров.

вызов sub

```
main:
        pushl $0x00000010
        pushl $0x00000020
        pushl $0x00000030
        call  sub
        addl  $12, %esp
```

«Базовая» программа на Assembler

```
.data
format_string:
        .string “HW %d\n”

.text
        .globl main
main:
        pushq   %rbp
        movq    %rsp,   %rbp

        movl    $format_string,         %edi
        movl    $0x00000010,            %esi
        call    printf

        movl    $0,     %eax
        movq    %rpb,   %rsp
        popq    %rbp
        ret
```

«Базовая» программа с функцией на Assembler

```
.data
x:                  .long           10
y:                  .long           20
format_string:      .string  “HW %d\n”
.text
sum:
        pushq   %rbp
        movq    %rsp,   %rbp
        movl    x,      %eax
        addl    y,      %eax
        movq    %rpb,   %rsp
        popq    %rbp
        ret
        .globl main
main:
        pushq   %rbp
        movq    %rsp,   %rbp
        call    sum
        movl    $format_string,%edi
        movl    %eax,   %esi
        call    printf
        movl    $0,     %eax
        movq    %rpb,   %rsp
        popq    %rbp
        ret
```