Машинно-зависимые языки программирования

Савельев Игорь Леонидович

План

- Вызов подпрограмм/функций
- Передача параметров в функцию
- Локальные переменные

gcc -S **-masm=intel** -Og **-fverbose-asm** ctest.c

```
int a = 1;
int b = 2;
int c = 0;

void func(void)
{
  c = a + b;
}

int main (void)
{
   func();
   return 0;
}
```

```
func:
.LFB0:
        .cfi_startproc
# ctest.c:9:    c = a + b;
        mov     eax, DWORD PTR b[rip]   # b, b
        add     eax, DWORD PTR a[rip]   # _3, a
# ctest.c:9:    c = a + b;
        mov     DWORD PTR c[rip], eax   # c, _3
# ctest.c:10: }
        ret
        .cfi_endproc
.LFE0:
        .size   func, .-func
        .globl  main
        .type   main, @function
```

```
func:
.LFB0:
        .file 1 "ctest.c"
        .loc 1 8 1
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        .loc 1 9 10
        movl    a(%rip), %edx
        movl    b(%rip), %eax
        addl    %edx, %eax
        .loc 1 9 6
        movl    %eax, c(%rip)
```

Инструменты

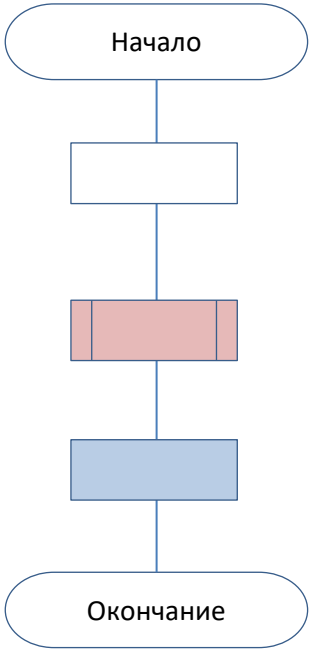
objdump –M intel -S **mytest**

objdump -d

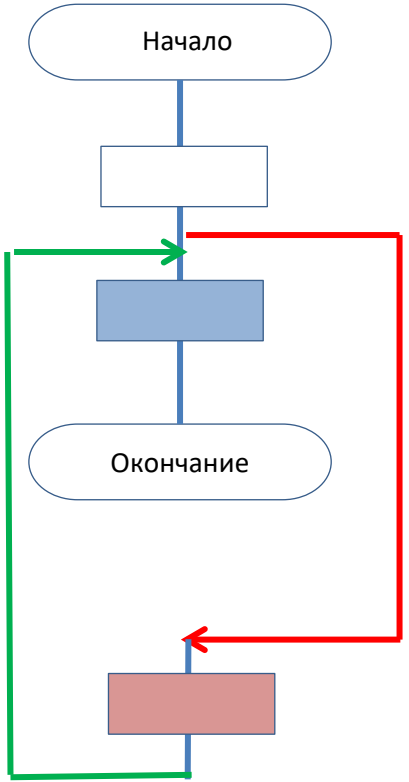
```
int a = 1;
int b = 2;
int c = 0;

void func(void)
{
  c = a + b;
}

int main (void)
{
   func();
   return 0;
}
```

```
0000000000401102 <func>:
int b = 2;
int c = 0;

void func(void)
{
 401102:      55                 push   rbp
 401103:      48 89 e5               mov    rbp,rsp
  c = a + b;
 401106:      8b 15 1c 2f 00 00      mov    edx,DWORD PTR [rip+0x2f1c]
# 404028 <a>
 40110c:      8b 05 1a 2f 00 00      mov    eax,DWORD PTR [rip+0x2f1a]
# 40402c <b>
 401112:      01 d0                add    eax,edx
 401114:      89 05 1a 2f 00 00      mov    DWORD PTR [rip+0x2f1a],eax
# 404034 <c>
}
 40111a:      90                 nop
 40111b:      5d                 pop    rbp
 40111c:      c3                 ret
```

1. Вызов подпрограмм/функций



Вызов некоторой подпрограммы выглядит как структура «следование». Но учитывая, что инструкции располагаются в памяти последовательно, вызов подпрограммы – это переход по какому-то адресу вне основной функции и возврат назад, после окончания ее работы.

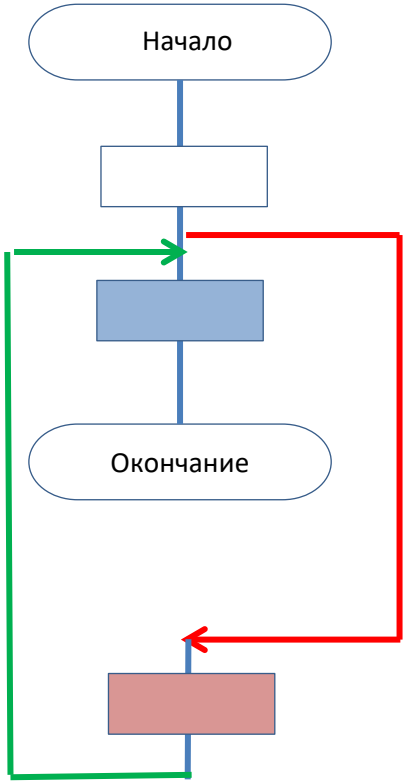


1. Вызов подпрограмм/функций

Для того, чтобы иметь возможность вернуться в точку вызова подпрограммы, следует при вызове подпрограммы запоминать этот адрес возврата.

Вызов функции: CALL <метка>
По команде CALL **в стек сохраняется адрес** инструкции, куда следует вернуться после завершения работы подпрограммы.

Возврат: RET
Команда RET завершает работу подпрограммы, вычитывает из стека адрес возврата и осуществляет переход на него.



1. Вызов подпрограмм/функций

```
CMAIN:
l1:
  xor rax, rax
  xor rbx, rbx
l2:
  call myfunc
l3:
  inc rbx
  ret
myfunc:
  inc rax
  ret
```

```
> x /x l1
0x4014e3 <l1>: 0x48e58948
> x /x l2
0x4014ec <l2>: 0x000004e8
> x /x l3
0x4014f1 <l3>: 0xc3c3ff48
> x /x myfunc
0x4014f5 <myfunc>: 0xc3c0ff48
```

До CALL

```
> x /2x $rsp
0x23fe68: 0x004013a5 0x00000000
```

После CALL

```
> x /4x $rsp
0x23fe60: 0x004014f1 0x00000000
          0x004013a5 0x00000000
```

После RET

```
> x /2x $rsp
0x23fe68: 0x004013a5 0x00000000
```

1. Вызов подпрограмм/функций

```
section .text
        xor eax, eax
        call myfunc         ; в стек помещается адрес
                            ; следующей инструкции
        xor ebx, ebx

myfunc:             ; Начало подпрограммы
        …
        ret         ; возврат, очистка стека
```

Основное требование к подпрограмме: не портить регистры.
Это означает, что если в подпрограмме изменяются значения каких-то регистров, то их надо сохранить на входе в подпрограмму и восстановить на выходе

1. Вызов подпрограмм/функций

Основное требование к подпрограмме: не портить регистры.
Это означает, что если в подпрограмме изменяются значения каких-то регистров, то их надо сохранить на входе в подпрограмму и восстановить на выходе, используя push/pop команды

```
section .text
        xor eax, eax
        call myfunc         ; в стек помещается адрес
                            ; следующей инструкции
        xor ebx, ebx

myfunc:             ; Начало подпрограммы
        push ecx
        mov ecx, 3          ; Изменяем регистр
        pop ecx
        ret         ; возврат, очистка стека
```

2. Передача параметров в функцию

1. Глобальные переменные
2. Регистры
3. Стек

1. Глобальные переменные

Используем
- Переменные в секции .DATA и BSS
- Extern переменные при взаимодействии с другими модулями

2. Регистры

Для передачи параметров используем регистры
Пример: int 0x80 И syscall
Это самый быстрый вызов подпрограммы

Arch/ABI	Instruction	System call #	Ret val	Ret val2	Error	Notes
i386	**int $0x80**	**eax**	**eax**	**edx**	-	
x86-64	syscall	rax	rax	rdx	-	

Arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes
i386	ebx	ecx	edx	esi	edi	ebp	-	
x86-64	rdi	rsi	rdx	r10	r8	r9	-	

2. Передача параметров в функцию

2. Регистры

Для передачи параметров используем регистры
Пример: int 0x80 И syscall
Это самый быстрый вызов подпрограммы

Аргументы с 7 и далее – через стек

Аргументы вещественного типа – через регистры xmm0 - xmm7

При этом в регистр AL указывается количество таким переменных

2. Передача параметров в функцию

```
typedef struct {
  int a, b;
  double d;
} structparm;
structparm s;
int e, f, g, h, i, j, k;
long double ld;
double m, n;
__m256 y;

extern void func (int e, int f,
                  structparm s, int g, int h,
                  long double ld, double m,
                  __m256 y,
                  double n, int i, int j, int k);

func (e, f, s, g, h, ld, m, y, n, i, j, k);
```

General Purpose Registers	Floating Point Registers	Stack Frame Offset
%rdi: e	%xmm0: s.d	0: ld
%rsi: f	%xmm1: m	16: j
%rdx: s.a,s.b	%ymm2: y	24: k
%rcx: g	%xmm3: n	
%r8: h		
%r9: i		

2. Передача параметров в функцию

```
int c = 0;


int func(int _a, int _b)
{
  return (_a + _b );
}

int main (void)
{
   c = func(1, 2);
   return 0;
}
```

```
int func(int _a, int _b)
{
 401102:      55                 push   rbp
 401103:      48 89 e5             mov    rbp,rsp
 401106:      89 7d fc             mov    DWORD PTR [rbp-0x4], edi
 401109:      89 75 f8             mov    DWORD PTR [rbp-0x8], esi
  return (_a + _b );
 40110c:      8b 55 fc             mov    edx,DWORD PTR [rbp-0x4]
 40110f:      8b 45 f8             mov    eax,DWORD PTR [rbp-0x8]
 401112:      01 d0                add    eax, edx
}
 401114:      5d                 pop    rbp
 401115:      c3                 ret
0000000000401116 <main>:
int main (void)
{
 401116:      55                 push   rbp
 401117:      48 89 e5             mov    rbp,rsp
   c = func(1, 2);
 40111a:      be 02 00 00 00       mov    esi,0x2
 40111f:      bf 01 00 00 00       mov    edi,0x1
 401124:      e8 d9 ff ff ff       call   401102 <func>
 401129:      89 05 fd 2e 00 00    mov    DWORD PTR [rip+0x2efd],  eax       # 40402c <c>
```

2. Передача параметров в функцию

```
int func(int _a, int _b, int _c, int _d, int _e, int _f)
```

```
int main (void)
{
 401124:      55                  push   rbp
 401125:      48 89 e5                mov    rbp,rsp
   c = func(1, 2, 3, 4, 5, 6);
 401128:      41 b9 06 00 00 00       mov    r9d,0x6
 40112e:      41 b8 05 00 00 00       mov    r8d,0x5
 401134:      b9 04 00 00 00          mov    ecx,0x4
 401139:      ba 03 00 00 00          mov    edx,0x3
 40113e:      be 02 00 00 00          mov    esi,0x2
 401143:      bf 01 00 00 00          mov    edi,0x1
 401148:      e8 b5 ff ff ff          call   401102 <func>
 40114d:      89 05 d9 2e 00 00       mov    DWORD PTR [rip+0x2ed9], eax       # 40402c
<c>
```

2. Передача параметров в функцию

```
float func(int _a, int _b, int _c, int _d, int _e, int _f)
```

```
int main (void)
{
  40112c:      55                  push   rbp
  40112d:      48 89 e5               mov    rbp,rsp
    c = func(1, 2, 3, 4, 5, 6);
  401130:      41 b9 06 00 00 00      mov    r9d,0x6
  401136:      41 b8 05 00 00 00      mov    r8d,0x5
  40113c:      b9 04 00 00 00         mov    ecx,0x4
  401141:      ba 03 00 00 00         mov    edx,0x3
  401146:      be 02 00 00 00         mov    esi,0x2
  40114b:      bf 01 00 00 00         mov    edi,0x1
  401150:      e8 ad ff ff ff         call   401102 <func>
  401155:      66 0f 7e c0            movd   eax, xmm0       ; x32 – из st0
  401159:      89 05 cd 2e 00 00      mov    DWORD PTR [rip+0x2ecd], eax       # 40402c <c>
```

2. Передача параметров в функцию

```
float func (float _a, float _b)
```

```
int main (void)
{
  40111c:       55                      push   rbp
  40111d:       48 89 e5                mov    rbp,rsp
    c = func(1, 2);
  401120:       f3 0f 10 0d dc 0e 00    movss  xmm1, DWORD PTR [rip+0xedc]       # 402004
<_IO_stdin_used+0x4>
  401127:       00
  401128:       f3 0f 10 05 d8 0e 00    movss  xmm0, DWORD PTR [rip+0xed8]       # 402008
<_IO_stdin_used+0x8>
  40112f:       00
  401130:       e8 cd ff ff ff          call   401102 <func>
  401135:       66 0f 7e c0             movd   eax,  xmm0
  401139:       89 05 ed 2e 00 00       mov    DWORD PTR [rip+0x2eed],eax       # 40402c <c>
```

2. Передача параметров в функцию

```
float func(int _a, int _b, int _c, int _d, int _e, int _f, int _g)

int main (void)
{
 40112c:      55                  push   rbp
 40112d:      48 89 e5               mov    rbp,rsp
  c = func(1, 2, 3, 4, 5, 6, 7);
 401130:      6a 07                  push   0x7
 401132:      41 b9 06 00 00 00      mov    r9d,0x6
 401138:      41 b8 05 00 00 00      mov    r8d,0x5
 40113e:      b9 04 00 00 00         mov    ecx,0x4
 401143:      ba 03 00 00 00         mov    edx,0x3
 401148:      be 02 00 00 00         mov    esi,0x2
 40114d:      bf 01 00 00 00         mov    edi,0x1
 401152:      e8 ab ff ff ff         call   401102 <func>
 401157:      48 83 c4 08            add    rsp,0x8
 40115b:      66 0f 7e c0            movd   eax,xmm0
 40115f:      89 05 c7 2e 00 00      mov    DWORD PTR [rip+0x2ec7],eax        # 40402c <c>
```

2. Передача параметров в функцию

float func (int _a, int _b, int _c, int _d, int _e, int _f, int _g)

```
float func (int _a, int _b, int _c, int _d, int _e, int _f, int _g)
{
 401102:      55                 push   rbp
 401103:      48 89 e5             mov    rbp,rsp
 401106:      89 7d fc             mov    DWORD PTR [rbp-0x4], edi
 401109:      89 75 f8             mov    DWORD PTR [rbp-0x8],esi
 40110c:      89 55 f4             mov    DWORD PTR [rbp-0xc],edx
 40110f:      89 4d f0             mov    DWORD PTR [rbp-0x10],ecx
 401112:      44 89 45 ec            mov    DWORD PTR [rbp-0x14],r8d
 401116:      44 89 4d e8            mov    DWORD PTR [rbp-0x18],r9d
  return (_a + _g );
 40111a:      8b 55 fc             mov    edx,DWORD PTR [rbp-0x4]
 40111d:      8b 45 10             mov    eax,  DWORD PTR [rbp+0x10]
 401120:      01 d0                add    eax,edx
 401122:      66 0f ef c0            pxor   xmm0,xmm0
 401126:      f3 0f 2a c0            cvtsi2ss xmm0,eax
}
```

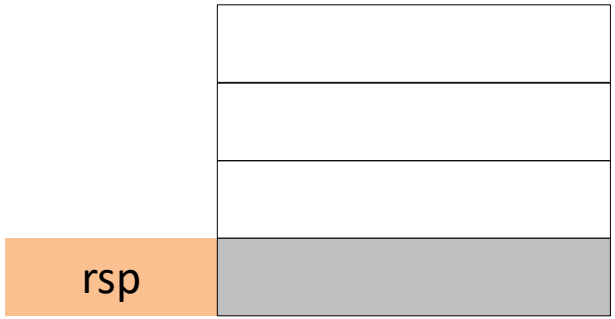
3. Стек

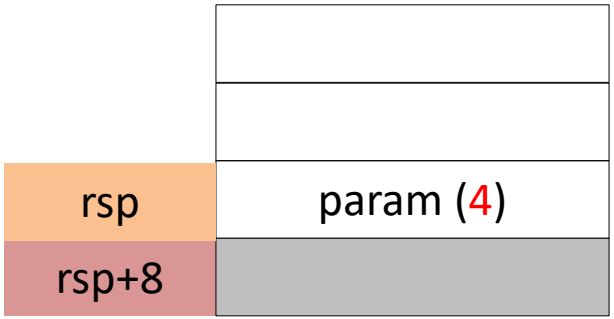
До вызова функции все параметры, используя push, загружаются в стек.

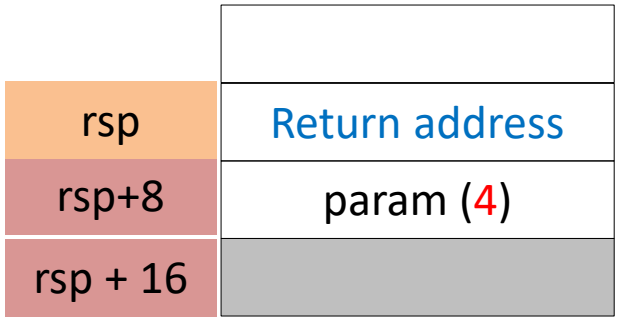
В функции из стека эти параметры выгружаются в регистр по мере надобности.

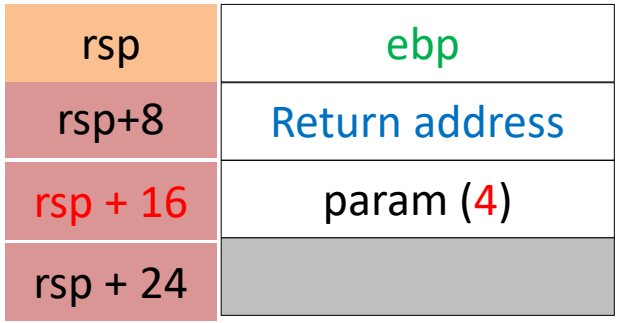
Основное требование – ничего не забыть и лишнего из стека не взять.

```
section .text
        mov eax, 4
        push eax
        call myfunc
myfunc:
        push ebp
        ??????
        pop ebp
        ret
```





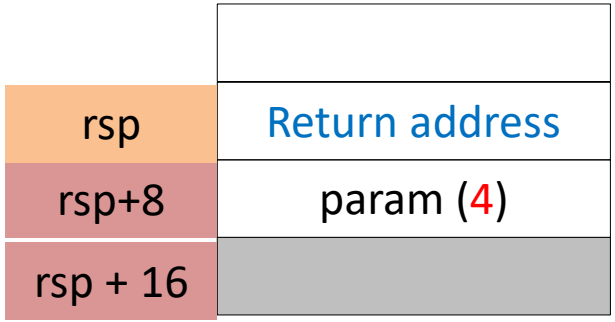




3. Стек

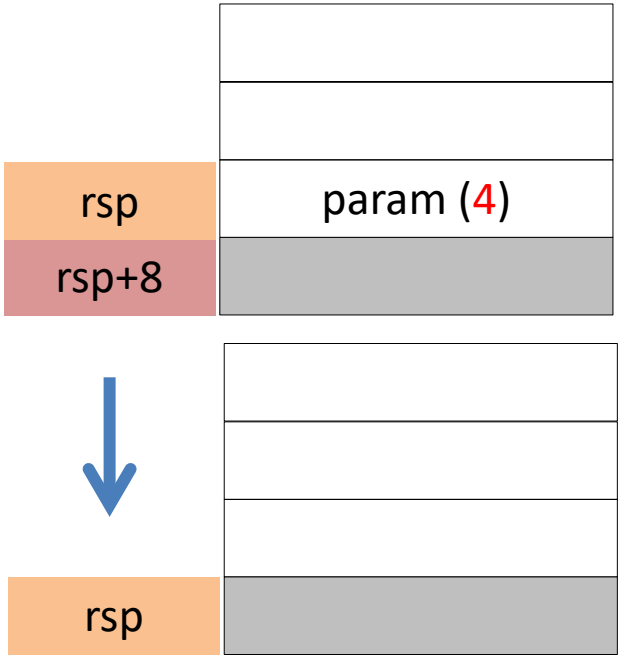
До вызова функции все параметры, используя push, загружаются в стек.
В функции из стека эти параметры выгружаются в регистр по мере надобности.
Основное требование – ничего не забыть и лишнего из стека не взять.

```
section .text
        mov eax, 4
        push eax
        call myfunc
        ?????
myfunc:
        push ebp
        …
        pop ebp
        ret
```



1) pop eax

2) add rsp, 8



2. Передача параметров в функцию

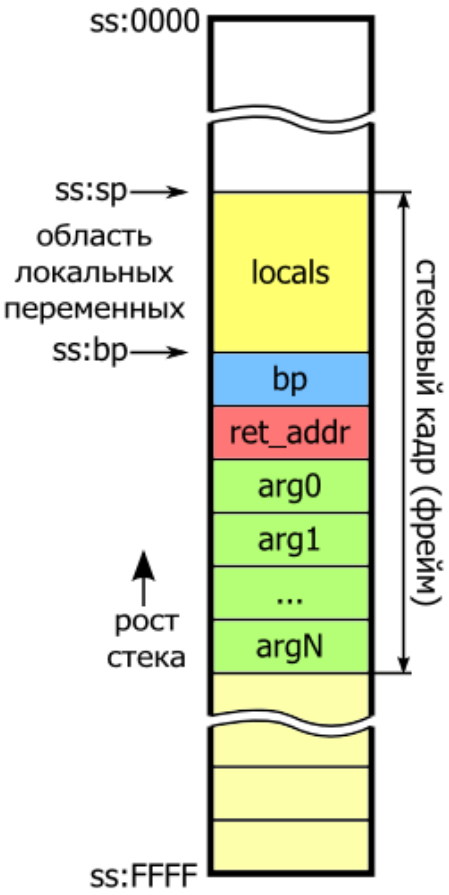
```
section .text
        mov eax, 4
        push eax
        call myfunc
        add rsp, 8
myfunc:
        push ebp
        …
        pop ebp
        ret
```

```
section .text
        mov eax, 4
        push eax
        call myfunc
        ;
myfunc:
        push ebp
        …
        pop ebp
        ret 8
```

Opcode*	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
C3	RET	ZO	Valid	Valid	Near return to calling procedure.
CB	RET	ZO	Valid	Valid	Far return to calling procedure.
C2 iw	RET imm16	I	Valid	Valid	Near return to calling procedure and pop imm16 bytes from stack.
CA iw	RET imm16	I	Valid	Valid	Far return to calling procedure and pop imm16 bytes from stack.

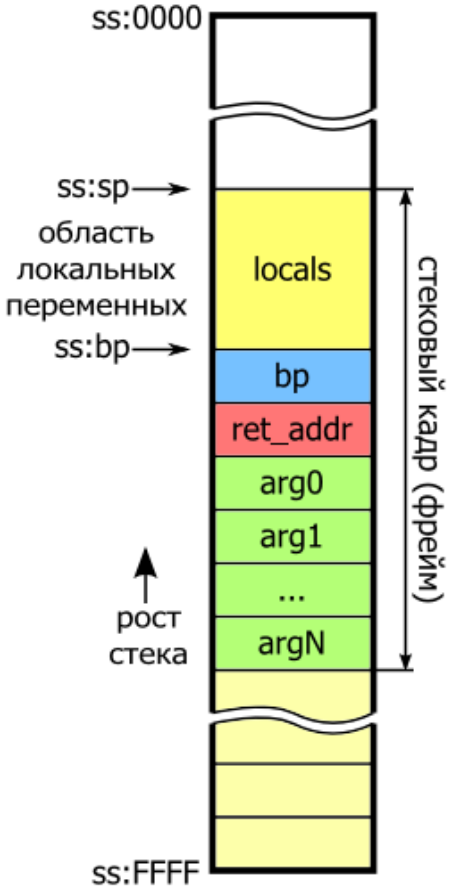
3. Локальные переменные

В отличие от глобальных переменных, размещаемых в секциях DATA и BSS, локальные переменные располагаются в стеке.
Выделение памяти под переменную производится смещением значения регистра ESP в меньшую сторону (стек растет в сторону уменьшения адресов) на размер локальных переменных.
Так как регистр ESP «занят» при использовании команд push/pop, то для навигации внутри выделенного пространства на стеке используется вспомогательный регистр EBP.



3. Локальные переменные

```
int func(int _a, int _b)
{
 401102:      55                 push   rbp
 401103:      48 89 e5              mov    rbp, rsp
 401106:      89 7d fc            mov    DWORD PTR [rbp-0x4], edi
 401109:      89 75 f8            mov    DWORD PTR [rbp-0x8], esi
  return (_a + _b );
 40110c:      8b 55 fc            mov    edx, DWORD PTR [rbp-0x4]
 40110f:      8b 45 f8            mov    eax, DWORD PTR [rbp-0x8]
 401112:      01 d0               add    eax, edx
}
 401114:      5d                 pop    rbp
 401115:      c3                 ret
```

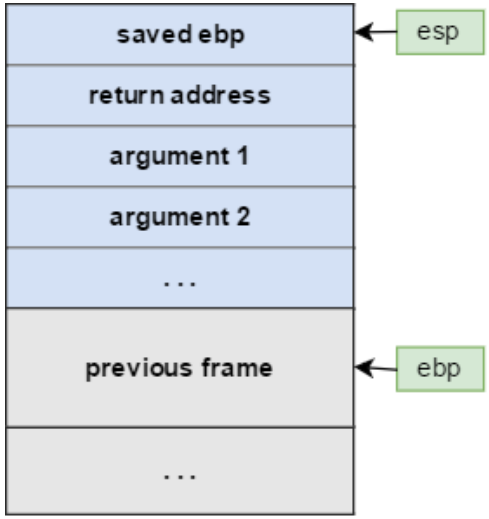


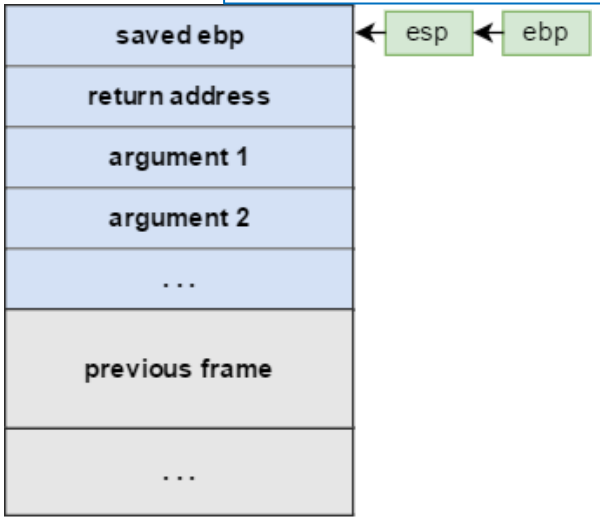
3. Локальные переменные

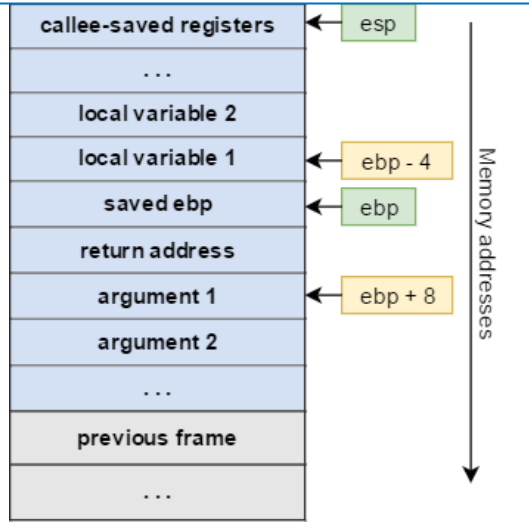
! Пример для 32 бит (4 байта) !

```
push 2 ; пушим аргументы в обратном порядке
push 10

call _subtract ;

add esp, 8 ; удаляем 8 байт со стека (два аргумента по 4 байта)
```

```
_subtract:
push ebp ; сохранение указателя базы предыдущего фрейма
mov ebp, esp ; настройка ebp
sub esp, 8 ; В стеке выделяем место под две переменные
mov eax, [ebp+8] ; копирование первого аргумента функции
mov  [ebp-4] , eax; копирование в локальную переменную 1
mov ebx  [ebp+12] ; копирование второго аргумента
mov  [ebp-8] , eax; копирование в локальную переменную  2
…… работа с регистрами
add esp, 8   ;   или mov esp, ebp -  «удаление» локальных переменных
pop ebp ; восстановление указателя базы предыдущего фрейма
ret
```







3. Локальные переменные

Вход в функцию (пролог)

```
push ebp          ; сохраняем изменяемое
mov ebp, esp      ; запоминаем границу стека
sub esp, X        ; выделили X байта
```

ENTER X, 0

Выход из функции (эпилог)

```
mov esp, epb      ; удаляем лок.переменные
pop ebp           ; восстанавливаем регистр
```

LEAVE

Зачем изменять ESP перед RET?

Команда RET берет адрес возврата из стека. Так как мы подвинули вершину стека при создании локальной переменной, нам надо вернуть значение ESP, чтобы RET взял корректный адрес из стека

4. Соглашение о вызовах

https://en.wikipedia.org/wiki/X86_calling_conventions

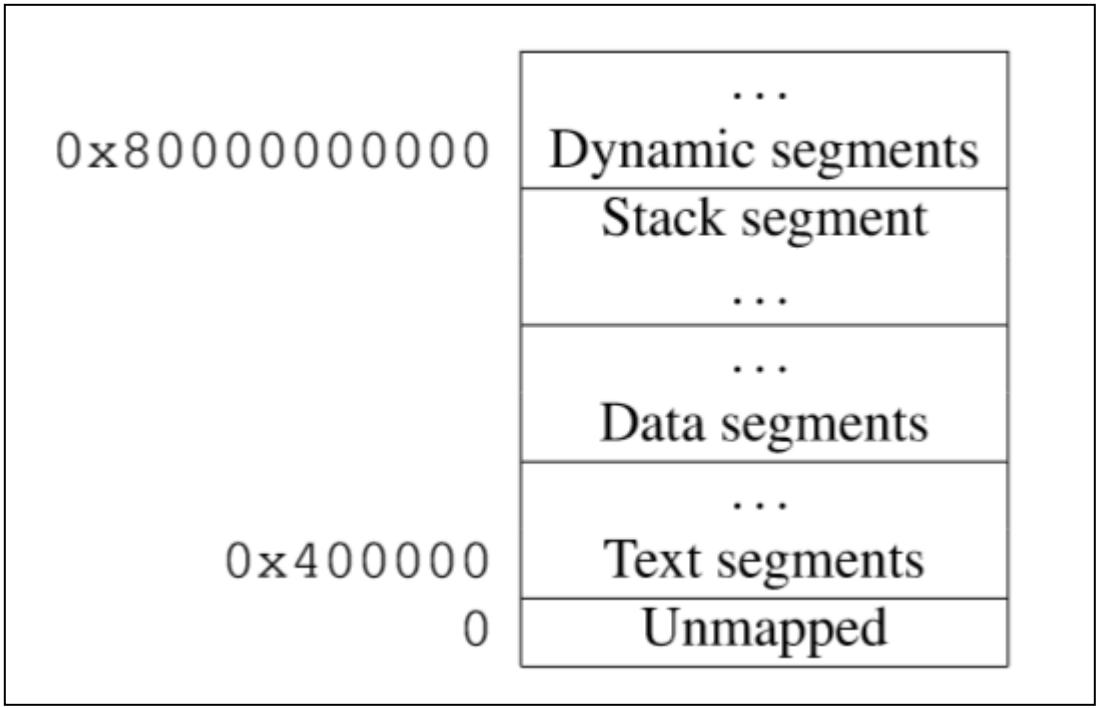
Architecture	Name	Operating system, compiler	Parameters		Stack cleanup	Notes
			Registers	Stack order		
8086	cdecl			RTL (C)	Caller	
	Pascal			LTR (Pascal)	Callee	
	fastcall (non-member)	Microsoft	AX, DX, BX	LTR (Pascal)	Callee	Return pointer in BX.
	fastcall (member function)	Microsoft	AX, DX	LTR (Pascal)	Callee	this on stack low address. Return pointer in AX.
	fastcall	Turbo C[29]	AX, DX, BX	LTR (Pascal)	Callee	this on stack low address. Return pointer on stack high address.
		Watcom	AX, DX, BX, CX	RTL (C)	Callee	Return pointer in SI.

4. Соглашение о вызовах

https://en.wikipedia.org/wiki/X86_calling_conventions

Architecture	Name	Operating system, compiler	Parameters		Stack cleanup	Notes
			Registers	Stack order		
x86-64	Microsoft x64 calling convention[21]	Windows (Microsoft Visual C++, GCC, Intel C++ Compiler, Delphi), UEFI	RCX/XMM0, RDX/XMM1, R8/XMM2, R9/XMM3	RTL (C)	Caller	Stack aligned on 16 bytes. 32 bytes shadow space on stack. The specified 8 registers can only be used for parameters 1 through 4. For C++ classes, the hidden this parameter is the first parameter, and is passed in RCX.[30]
	vectorcall	Windows (Microsoft Visual C++, Clang, ICC)	RCX/[XY]MM0, RDX/[XY]MM1, R8/[XY]MM2, R9/[XY]MM3 + [XY]MM4–5	RTL (C)	Caller	Extended from MS x64.[11]
	System V AMD64 ABI[28]	Solaris, Linux, BSD, mac OS, OpenVMS (GCC, Intel C++ Compiler, Clang, Delphi)	RDI, RSI, RDX, RCX, R8, R9, [XYZ]MM0–7	RTL (C)	Caller	Stack aligned on 16 bytes boundary. 128 bytes red zone below stack. The kernel interface uses RDI, RSI, RDX, R10, R8 and R9. In C++, this is the first parameter.

Командная строка




```
extern int main ( int argc , char *argv[ ] , char* envp[ ] );
```

Purpose	Start Address	Length
Unspecified	High Addresses	
Information block, including argument strings, environment strings, auxiliary information ...		varies
Unspecified		
Null auxiliary vector entry		1 eightbyte
Auxiliary vector entries ...		2 eightbytes each
0		eightbyte
Environment pointers ...		1 eightbyte each
0	`8+8*argc+%rsp`	eightbyte
Argument pointers	`8+%rsp`	argc eightbytes
Argument count	`%rsp`	eightbyte
Undefined	Low Addresses	

Командная строка

```
extern int main ( int argc , char *argv[ ] , char* envp[ ] );
```

Для stdlib (gcc) параметры в main() передаются через регистры (rdi, rsi, rdx)

Как работает передача переменного числа параметров?

printf(char * format, …)

Вставка ASM-инструкций

```
__asm__ (вставка
 : список_выходных_операндов
 : список_входных_операндов
: список_разрушаемых_регистров );
```

Операнд имеет следующий вид:

ограничение_типа (имя_переменной)
имя_переменной — ни что иное, как имя С-переменой, значение которой вы хотите использовать в ассемблерном коде.

ограничение_типа — строковая константа, описывает допустимый тип операнда.

Для выходных операндов строка ограничения типа должна начинаться с символа =.

Вставка ASM-инструкций

Параметры нумеруются последовательно, с 0, используются с %
Константа без % - абсолютный адрес

Ограничения типа (некоторые)
r – регистр общего назначения
a - eax, ax, al
b - ebx, bx, bl
c - ecx, cx, cl
d - edx, dx, dl
S - esi, si
D - edi, di
i – числовая константа
m - размещение в памяти

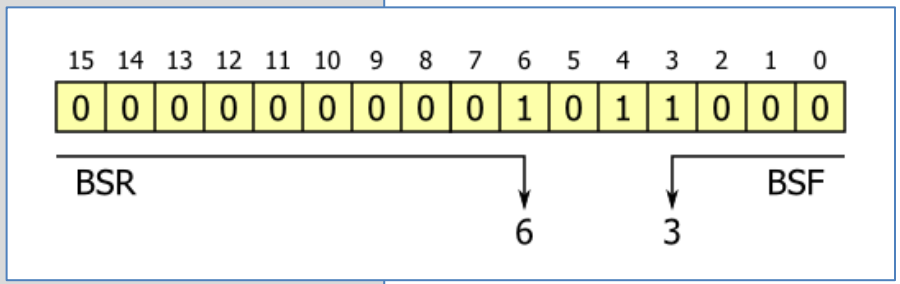
```
__asm__ __volatile__ ( "cld\n\t"
                        "rep movsl\n\t”
                       :
                       : "S" (src), "D" (dest), "c" (numwords)
                       : "ecx", "esi", "edi" )
```

Вставка ASM-инструкций

```
int main() {
    int a = 1;
    int b = 2;
    int c;

    asm(".intel_syntax noprefix\n\t" // директива GAS, включаем Intel синтаксис.
        "mov eax, %1\n\t"          // перемещаем в eax значение переменной a.
        "add eax, %2\n\t"          // прибавляем значение переменной b к eax.
        "mov %0, eax\n\t"          // перемещаем в переменную с значение eax.
        :"=r"(c)                   // список выходных параметров.
        :"r"(a), "r"(b)            // список входных параметров.
        : "eax"                    // список разрушаемых регистров.
    );

    printf("%d + %d = %d\n", a, b, c);
    return 0;
}
```

Задача – найти номер наиболее старшего бита в числе unsigned int

```
/*! \brief Find the bit position of the highest set bit in a word
   \param bits The word to be searched
   \return The bit number of the highest set bit, or -1 if the word is zero. */
static __inline__ int top_bit(unsigned int bits)
{
   int res;

#if defined(__i386__)  ||  defined(__x86_64__)
   __asm__ (" xorl %[res],%[res];\n"
            " decl %[res];\n"
            " bsrl %[bits],%[res]\n"
            : [res] "=&r" (res)
            : [bits] "rm" (bits));
   return res;
#elif defined(__ppc__)  ||   defined(__powerpc__)
   __asm__ ("cntlzw %[res],%[bits];\n"
         : [res] "=&r" (res)
         : [bits] "r" (bits));
   return 31 - res;
```



Пример

```
__inline long int  float2int(float flt)
{      int intgr;

       _asm
       {      fld flt
              fistp intgr
       } ;

       return intgr ;
}
```

```
#if defined(__x86_64__)
    __asm__ __volatile__(
       " emms;\n"
       " pxor %%mm0,%%mm0;\n"
       " leal -32(%%rsi,%%eax,2),%%edx;\n"     /* edx = top - 32 */

       " cmpl %%rdx,%%rsi;\n"
       " ja 1f;\n"

       /* Work in blocks of 16 int16_t's until we are near the end */
       " .p2align 2;\n"
       "2:\n"
       " movq (%%rdi),%%mm1;\n"
       " movq (%%rsi),%%mm2;\n"
       " pmaddwd %%mm2,%%mm1;\n"
       " paddd %%mm1,%%mm0;\n"
       " movq 8(%%rdi),%%mm1;\n"
       " movq 8(%%rsi),%%mm2;\n"
       " pmaddwd %%mm2,%%mm1;\n"
       " paddd %%mm1,%%mm0;\n"
```

Спасибо