Лабораторная работа 3 Стек и локальные переменные
Во всех таблицах первый столбик 64, второй 32

1.a Без аргументов и значений

```
nope:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    x(%rip), %edx
        movl    y(%rip), %eax
        addl    %edx, %eax
        movl    %eax, res(%rip)
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   nope, .-nope
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $0, %eax
        call    nope
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

```
nope:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    x@GOT(%eax), %edx
        movl    (%edx), %ecx
        movl    y@GOT(%eax), %edx
        movl    (%edx), %edx
        addl    %ecx, %edx
        movl    res@GOT(%eax), %eax
        movl    %edx, (%eax)
        nop
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   nope, .-nope
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        call    nope
        movl    $0, %eax
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

1.b Без аргументов и с возвращаемым значением (int/char);

```
nope:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    x(%rip), %edx
        movl    y(%rip), %eax
        addl    %edx, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   nope, .-nope
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $0, %eax
        call    nope
        movl    %eax, res(%rip)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
```

```
nope:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    x@GOT(%eax), %edx
        movl    (%edx), %edx
        movl    y@GOT(%eax), %eax
        movl    (%eax), %eax
        addl    %edx, %eax
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   nope, .-nope
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        pushl   %ebx
        .cfi_offset 3, -12
        call    __x86.get_pc_thunk.bx
        addl    $_GLOBAL_OFFSET_TABLE_, %ebx
        call    nope
        movl    res@GOT(%ebx), %edx
        movl    %eax, (%edx)
        movl    $0, %eax
        popl    %ebx
        .cfi_restore 3
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
```

2.b С одним локальным аргументом

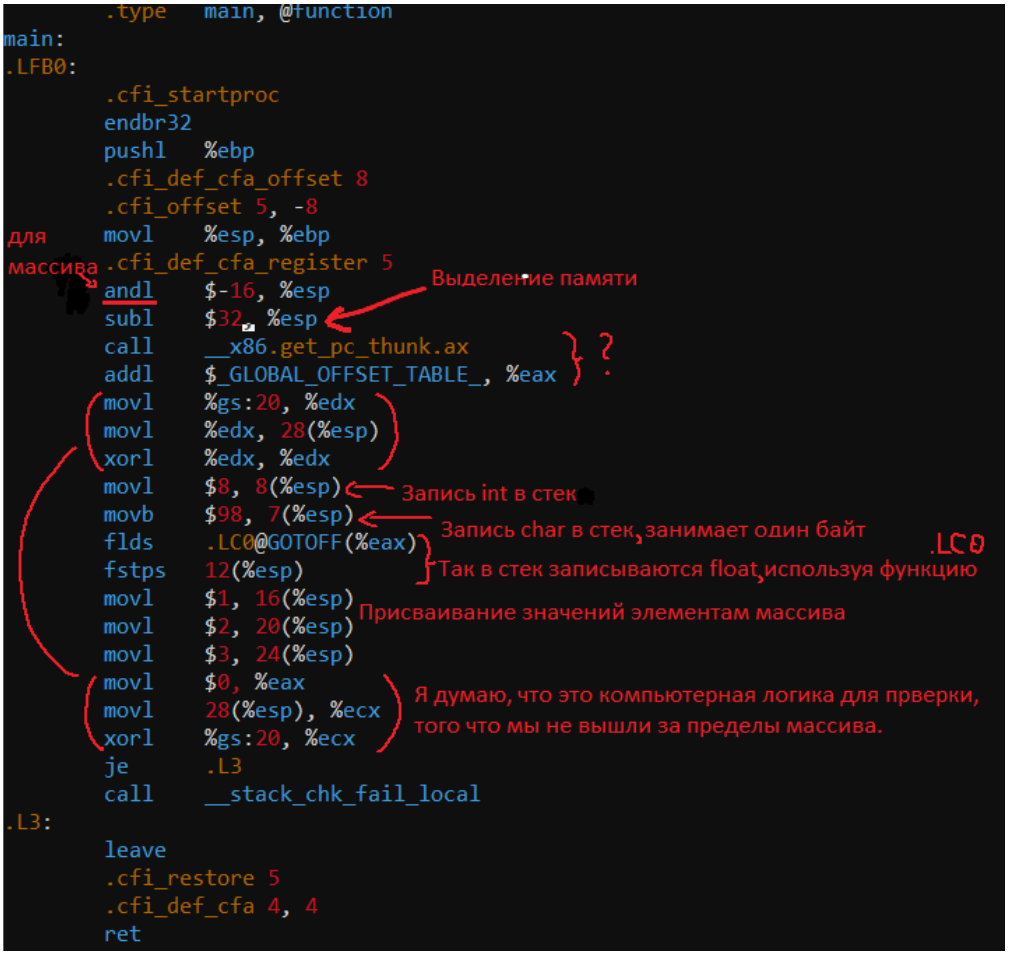
```
nope:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    x(%rip), %edx
        movl    y(%rip), %eax
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movl    -4(%rbp), %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   nope, .-nope
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $0, %eax
        call    nope
        movl    %eax, res(%rip)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
```

```
nope:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    x@GOTOFF(%eax), %edx
        movl    y@GOTOFF(%eax), %eax
        addl    %edx, %eax
        movl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   nope, .-nope
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        pushl   %ebx
        .cfi_offset 3, -12
        call    __x86.get_pc_thunk.bx
        addl    $_GLOBAL_OFFSET_TABLE_, %ebx
        call    nope
        movl    res@GOT(%ebx), %edx
        movl    %eax, (%edx)
        movl    $0, %eax
        popl    %ebx
        .cfi_restore 3
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
```

2a,b,c Посмотрим на локальные переменные:

```c
#include <stdio.h>
int nope()
{
        int a;
        a =  15;
        return a;

}
int main()
{
        int a = 8;
        char b = 'b';
        float p = 3.14;
        int arr[] = {1, 2, 3};
        return 0;
}
~
~
```

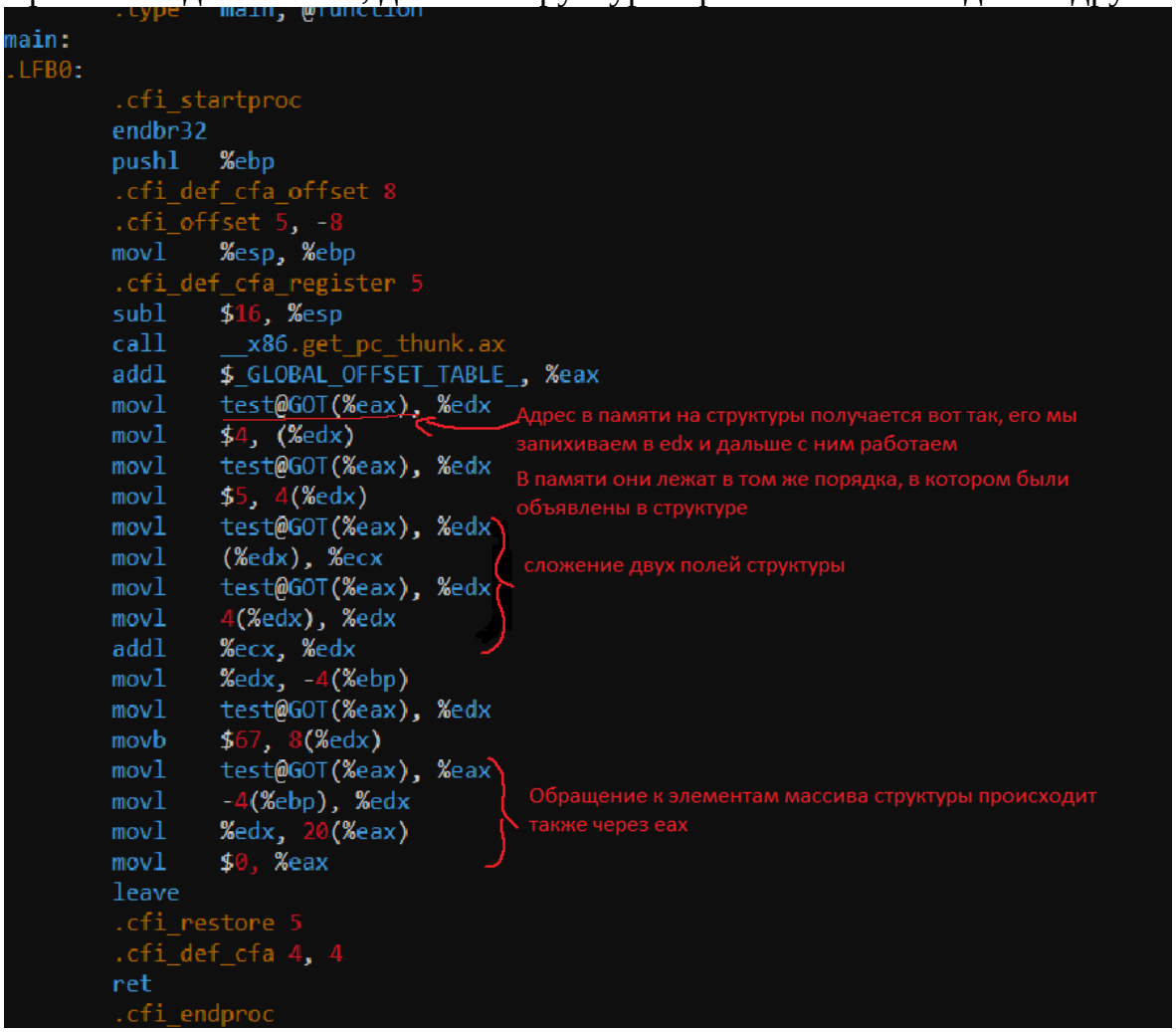


В 64-битной все примерно также

3 а-е Структуры:

```c
#include <stdio.h>
struct Exp
{
        int a;
        int b;
        char c;
        int arr[3];
};
struct Exp test;
int main()
{
        test.a = 4;
        test.b = 5;
        int z = test.a + test.b;
        test.c = 'C';
        test.arr[2] = z;
        return 0;
}
```

Есть разница в обращении к элементам в 32 и 64 битных системах(см скриншоты ниже). Но идея хранения одна и та же, данные структуры хранятся вместе одно за другим.



```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $5, 4+test(%rip)
        movl    $4, test(%rip)
        movl    test(%rip), %edx
        movl    4+test(%rip), %eax
        addl    %edx, %eax
        movl    %eax, -4(%rbp)
        movb    $67, 8+test(%rip)
        movl    -4(%rbp), %eax
        movl    %eax, 20+test(%rip)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

О, а вот тут обращение к элементам структуры намного проще. Все понятно, нет ни каких шаманств с eax

Сложение

Присваивание локальной переменной элменту массива тоже выглядит супер просто

Структуры в функциях

```c
#include <stdio.h>
struct Exp
{
        int a;
        int b;
        char c;
        int arr[3];
};
struct Exp test;

struct Exp nope(struct Exp obj)
{
        struct Exp res;
        res.a = obj.a + obj.b;
        res.b = obj.a - obj.b;
        return res;
}
int main()
{
        test.a = 5;
        test.b = 4;
        test.c = 'C';
        struct Exp res = nope(test);
        return 0;
}
```


main:
.LFB1:
.cfi_startproc
endbr32
leal 4(%esp), %ecx
.cfi_def_cfa 1, 0
andl $-16, %esp
pushl -4(%ecx)
pushl %ebp
.cfi_escape 0x10,0x5,0x2,0x75,0
movl %esp, %ebp
pushl %ecx
.cfi_escape 0xf,0x3,0x75,0x7c,0x6
subl $36, %esp
call __x86.get_pc_thunk.ax
addl $_GLOBAL_OFFSET_TABLE_, %eax
movl %gs:20, %ecx
movl %ecx, -12(%ebp)
xorl %ecx, %ecx
movl test@GOT(%eax), %edx
movl $5, (%edx)
movl test@GOT(%eax), %edx
movl $4, 4(%edx)
movl test@GOT(%eax), %edx
movb $67, 8(%edx)
leal -36(%ebp), %edx
subl $4, %esp
movl test@GOT(%eax), %eax
pushl 20(%eax)
pushl 16(%eax)
pushl 12(%eax)
pushl 8(%eax)
pushl 4(%eax)
pushl (%eax)
pushl %edx
call nope
addl $28, %esp
movl $0, %eax
movl -12(%ebp), %ecx
xorl %gs:20, %ecx
je .L6
call __stack_chk_fail_local
Присваивание значений test
Закидываем в стек поля test, для того, чтобы использовать их в функции nope
Вызываем функцию

```
nope:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $56, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    8(%ebp), %eax
        movl    %eax, -44(%ebp)
        movl    %gs:20, %eax
        movl    %eax, -12(%ebp)
        xorl    %eax, %eax
        movl    12(%ebp), %edx
        movl    16(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -36(%ebp)
        movl    12(%ebp), %edx
        movl    16(%ebp), %eax
        subl    %eax, %edx
        movl    %edx, %eax
        movl    %eax, -32(%ebp)
        movl    -44(%ebp), %eax
        movl    -36(%ebp), %edx
        movl    %edx, (%eax)
        movl    -32(%ebp), %edx
        movl    %edx, 4(%eax)
        movl    -28(%ebp), %edx
        movl    %edx, 8(%eax)
        movl    -24(%ebp), %edx
        movl    %edx, 12(%eax)
        movl    -20(%ebp), %edx
        movl    %edx, 16(%eax)
        movl    -16(%ebp), %edx
        movl    %edx, 20(%eax)
        movl    -12(%ebp), %eax
        xorl    %gs:20, %eax
        je      .L3
        call    __stack_chk_fail_local
.L3:
        movl    -44(%ebp), %eax
        leave
```

Он как-то сам заранее вычисляет, сколько памяти требуется выделить для функции nope

что-то непоянтное

присваивание значений полю res.a локальной структуры res

присваивание значения полю res.b локальной структуры res

Это вот мы локальные данные функции nope переносим в локальные данные функции main. Получается что, так работает return res

Опять проверки того, что ничего не сломалось

4. Рекурсивная функция и оценка размера стека:

```c
#include <stdio.h>
int recursion(int a)
{
    int arr[1024];
    printf( _Format: "%d ", a);
    recursion( a: a+1);
}
int main()
{
    int a = 0;
    a = recursion(a);
    return 0;
}
```

Для различных длин массивов получим количество раз срабатывания, умножим на размер одного инта и количество элементов в массиве, получим размер стека в байтах.

arr[x]	2	16	128	256	512	1000	1024	1048	8192
Байт	259056	1184192	1893888	1978368	2021376	2040000	2043904	2041504	2031616
Кб	252,9	1156	1849,5	1932	1974	1992.2	1996	1993	1984

Как видно из таблицы зависимость кол-ва раз срабатывания нелинейно зависит от количества элементов в массиве, которым забиваем память. Наибольшее значение находится при 1024 элементах, т. е. Размер стека = 1996 Кб ≈ 2 Мб. Как-то мало, учитывая что сам Clion использует 1300 Мб, чтобы жить

Имя	Состояние	15% ЦП	51% Память	2% Диск	0% Сеть
CLion (5)		0%	1 348,2 МБ	1,7 МБ/с	0 Мбит/с

Аналогично для Ubuntu размер стека = 7828 Кб = 7.7 Мб ≈ 8 Мб
Листинги рекурсивных функций

```
recursion:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $4096, %rsp
        orq     $0, (%rsp)
        subq    $32, %rsp
        movl    %edi, -4116(%rbp)
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        movl    -4116(%rbp), %eax
        addl    $1, %eax
        movl    %eax, %edi
        call    recursion
        nop
        movq    -8(%rbp), %rdx
        xorq    %fs:40, %rdx
        je      .L2
        call    __stack_chk_fail@PLT
.L2:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

```
recursion:
.LFB0:
        .cfi_startproc
        endbr32
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        subl    $4096, %esp
        orl     $0, (%esp)
        subl    $24, %esp
        call    __x86.get_pc_thunk.ax
        addl    $_GLOBAL_OFFSET_TABLE_, %eax
        movl    %gs:20, %eax
        movl    %eax, -12(%ebp)
        xorl    %eax, %eax
        movl    8(%ebp), %eax
        addl    $1, %eax
        subl    $12, %esp
        pushl   %eax
        call    recursion
        addl    $16, %esp
        nop
        movl    -12(%ebp), %edx
        xorl    %gs:20, %edx
        je      .L2
        call    __stack_chk_fail_local
.L2:
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
```

В 64 битной помимо 4096 требующихся на массив используется дополнительно 32 байта на каждом вызове функции, значит нужно сделать поправку на размер стека N*32, где N – кол-во циклов рекурсии. N*32 ≈ 16 Кб. Эти дополнительные 32 байта, нужны в стеке для значений регистров rbp, edi, eax, fs:40 и тд.
В 32 битной память тоже выделяется, но 36 байт. Думаю, что разница возникает из-за того значение счетчика передается через регистр edi в 64-битной, а не стек.