

МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

АССЕМБЛЕР, ЛАБОРАТОРНАЯ РАБОТА №1

Вставки и переходы

выполнил студент группы Б03-205

Овсянников Андрей

Долгопрудный, 2023 г.

1. Напишем на си такую программу:

```
#include <stdio.h>

int g = 1;

void func(int* a) {
    a += g;
}

int main() {
    int a = 0, b = 10;
    while (a != b) {
        func(&a);
    }
    printf("c, a: %d", a);
    return 0;
}
```

Листинг этой программы выглядит так с какого-то момента:

```
func:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp
    .cfi_def_cfa_register 6
    movq    %rdi, -8(%rbp)
    movq    -8(%rbp), %rax
    movl    (%rax), %edx
    movl    g(%rip), %eax
    addl    %eax, %edx
    movq    -8(%rbp), %rax
    movl    %edx, (%rax)
    nop
    popq    %rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size    func, .-func
    .section        .rodata
.LC0:
    .string  "c, a: %d"
    .text
    .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
    subq    $16, %rsp
    movq    %fs:40, %rax
    movq    %rax, -8(%rbp)
```

```

        xorl    %eax, %eax
        movl    $0, -16(%rbp)
        movl    $10, -12(%rbp)
        jmp     .L3
.L4:
        leaq    -16(%rbp), %rax
        movq    %rax, %rdi
        call    func
.L3:
        movl    -16(%rbp), %eax
        cmpl    %eax, -12(%rbp)
        jne     .L4
        movl    -16(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        movq    -8(%rbp), %rdx
        subq    %fs:40, %rdx
        je      .L6
        call    __stack_chk_fail@PLT
.L6:

```

Собственно, что мы тут можем увидеть, а увидеть мы можем появившиеся метки, кроме системных появилась функция `func` в листинге, более того, здесь мы можем рассмотреть как устроен цикл `while`, а устроил его компилятор простым `jne`. Теперь немного поиздеваемся над кодом, а именно напишем функцию `func` на ассемблере из плюсов (ахахахаха, а кто нам запретит). С определением локальной переменной проблем не должно возникнуть, так как он нам выделил стек, а мы уже сами решаем на какое место ее воткнуть, вот например он нам сейчас в `func` ничего не выделил.....(поверим, что он знает то делает). Тем не менее, давайте все операции, что отвечают за выполнение содержимого функции запишем через ассемблер:

```

#include <stdio.h>

int g = 1;

void func(int* a) {
    // a += g
    asm("movq    %rdi, -8(%rbp)\n"
        "movq    -8(%rbp), %rax\n"
        "movl    (%rax), %edx\n"
        "movl    g(%rip), %eax\n"
        "addl    %eax, %edx\n"
        "movq    -8(%rbp), %rax\n"
        "movl    %edx, (%rax)\n");
}

int main() {
    int a = 0, b = 10;
    while (a != b) {
        func(&a);
    }
    printf("c, a: %d", a);
    return 0;
}

```

Собственно в коде участок с `func` будет выглядеть так:

```

func:
.LFB0:

```

```

        .cfi_startproc
        endbr64
        pushq    %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq     %rsp, %rbp
        .cfi_def_cfa_register 6
        movq     %rdi, -8(%rbp)
#APP
# 7 "0.c" 1
        movq     %rdi, -8(%rbp)
        movq     -8(%rbp), %rax
        movl     (%rax), %edx
        movl     g(%rip), %eax
        addl     %eax, %edx
        movq     -8(%rbp), %rax
        movl     %edx, (%rax)

# 0 "" 2
#NO_APP
        nop
        popq     %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size    func, .-func
        .section          .rodata
.LC0:
        .string  "c, a: %d"
        .text
        .globl   main
        .type    main, @function

```

Осталось лишь проверить работает ли это чудо. До изменений и после программа вывела:

```
>> c, a: 10
```

Значит все работает корректно, код из плюсов просто вставился в листинг.

2. Теперь нам нужно получить в листинге метки. Как мы можем заметить, компилятор уже организовал их нам в коде, но так как метки это просто метки, пока мы на них не делаем джампы, то нарисуем парочку, а именно `.st_loc` и `.movi`).

```

func:
.LFB0:
        .cfi_startproc
        endbr64
        pushq    %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
st_loc: movq     %rsp, %rbp
        .cfi_def_cfa_register 6
movi:   movq     %rdi, -8(%rbp)
        movq     -8(%rbp), %rax
        movl     (%rax), %edx
        movl     g(%rip), %eax
        addl     %eax, %edx
        movq     -8(%rbp), %rax
        movl     %edx, (%rax)
        nop
        popq     %rbp
        .cfi_def_cfa 7, 8

```

```
ret
.cfi_endproc
```

Вставка дополняется ими совершенно аналогично, будет просто:

```
...
asm("st_loc:    movq    %rdi, -8(%rbp)\n"
    "movq      -8(%rbp), %rax\n"
    "...");
...
```

- 3 + 4. Собственно, мы уже получили в предыдущих пунктах метки перехода и оператор сравнения. Сделано это было к примеру на вот таком кусочке:

```
...
        jmp      .L3
.L4:
        leaq     -16(%rbp), %rax
        movq     %rax, %rdi
        call     func
.L3:
        movl     -16(%rbp), %eax
        cmpl     %eax, -12(%rbp)
        jne      .L4
        movl     -16(%rbp), %eax
...
```

Здесь мы аккуратно проверяем условия цикла while в сишном коде. Вообще оператор прыжка(условия) может быть с вот такими окончаниями: Джамп тут происходит на метку .L4 при условии выполнения условия что содержимое %eax не равно локальной переменной -12(%rbp), это аккуратно локальные a в первом и b во втором регистрах.

5. Теперь пришло время изучить, как в листингах выглядят глобальные массивы. Напишем следующую программу:

```
#include <stdio.h>

int arr1[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
int arr2[3];

int main() {
    int a = 27;
    for (unsigned int i = 0; i < 3; i++) {
        arr2[i] = a % (i + 1);
        a = a / 3;
    }

    for (unsigned int i = 0; i < 3; i++) {
        printf("%d", arr1[arr2[i]]);
    }

    return 0;
}
```

Листинг этого прикола полностью выводить пока что не требуется, так как он достаточно длинный, будем изучать его по частям.

Начнем с задания массивов. Это происходит так вот:

```
        .align 32
        .type   arr1, @object
        .size   arr1, 40
arr1:
        .long   1
        .long   2
```

Операторы перехода

`cmp` *операнд_2, операнд_1*
`j**` *метка*

Мнемоника	Английское слово	Смысл	Тип операндов
e	equal	равенство	любые
n	not	инверсия условия	любые
g	greater	больше	со знаком
l	less	меньше	со знаком
a	above	больше	без знака
b	below	меньше	без знака

- `je — jne`: равно — не равно;
- `jg — jng`: больше — не больше.

Рис. 1: операторы перехода

```
.long    3
.long    4
.long    5
.long    6
.long    7
.long    8
.long    9
.long    0
.globl   arr2
.bss
.align   8
.type    arr2, @object
.size    arr2, 12

arr2:
.zero    12
```

Присвоение элемента осуществляется здесь следующими строками:

```
leaq     arr2(%rip), %rax
movl     %ecx, (%rdx,%rax)
```

А именно в `rax` присваивается указатель на первый элемент глобального `arr2`, после этого значение из `ecx` (хотим присвоить) ставится на смещенный на `rdx` регистр `rax`, то бишь аккуратно на `i` (оно в регистре `rdx`) место в коде.

6 + 7. Вот просто приведу здесь полный листинг программы, описанной выше. Компилятор очень хорошо организовал мне циклы.

```

...
.LC0:
    .string      "%d"
    .text
    .globl       main
    .type        main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq        %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset  6, -16
    movq         %rsp, %rbp
    .cfi_def_cfa_register 6
    subq         $16, %rsp
    movl         $27, -12(%rbp)
    movl         $0, -8(%rbp)
    jmp          .L2
.L3:
    movl         -12(%rbp), %eax
    movl         -8(%rbp), %edx
    leal         1(%rdx), %ecx
    movl         $0, %edx
    divl         %ecx
    movl         %edx, %eax
    movl         %eax, %ecx
    movl         -8(%rbp), %eax
    leaq         0(%rax,4), %rdx
    leaq         arr2(%rip), %rax
    movl         %ecx, (%rdx,%rax)
    movl         -12(%rbp), %eax
    movslq       %eax, %rdx
    imulq        $1431655766, %rdx, %rdx
    shrq         $32, %rdx
    sarl         $31, %eax
    movl         %eax, %ecx
    movl         %edx, %eax
    subl         %ecx, %eax
    movl         %eax, -12(%rbp)
    addl         $1, -8(%rbp)
.L2:
    cmpl         $2, -8(%rbp)
    jbe         .L3
    movl         $0, -4(%rbp)
    jmp         .L4
.L5:
    movl         -4(%rbp), %eax
    leaq         0(%rax,4), %rdx
    leaq         arr2(%rip), %rax
    movl         (%rdx,%rax), %eax
    cltq
    leaq         0(%rax,4), %rdx
    leaq         arr1(%rip), %rax
    movl         (%rdx,%rax), %eax
    movl         %eax, %esi
    leaq         .LC0(%rip), %rax
    movq         %rax, %rdi
    movl         $0, %eax
    call        printf@PLT
    addl         $1, -4(%rbp)

```

```

.L4:
    cmpl    $2, -4(%rbp)
    jbe     .L5
    movl    $0, %eax
    leave
...

```

Тут можно очень не сложно различить, где циклы и условия. При этом с циклом while различия почти не будет.

8. Оператор цикла loop работает вполне нормально и понятно, думаю, что на его проверке не стоит заострять внимание.
9. Найдем максимум из 2 чисел так:

```

.file      "allmy.s"
.text
.section   .rodata
.LC0:
.string    "...: %d"
.text
.globl     main
.type      main, @function
main:
.LFB0:
.cfi_startproc
endbr64
pushq     %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq      %rsp, %rbp
.cfi_def_cfa_register 6
subq      $16, %rsp ;# then we have 16 bytes of local stack

    movl    $1, -12(%rbp)
    movl    $2, -8(%rbp)
    movl    -12(%rbp), %eax
    cmpl    %eax, -8(%rbp)
    ja      .L3
    jmp     .L4

.L3:
    movl    -8(%rbp), %esi
    jmp     exit_out
.L4:
    movl    -12(%rbp), %esi

exit_out:
    leaq     .LC0(%rip), %rax
    movq     %rax, %rdi
    movl     $0, %eax
    call     printf@PLT

    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
.size      main, .-main
.ident     "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
.section   .note.GNU-stack,"",@progbits
.section   .note.gnu.property,"a"

```



```

        .align 8
        .long 1f - 0f
        .long 4f - 1f
        .long 5
0:
        .string "GNU"
1:
        .align 8
        .long 0xc0000002
        .long 3f - 2f
2:
        .long 0x3
3:
        .align 8
4:

```

10. Просуммируем массив.

Рубрика забавный факт: если освободить для локальных переменных 4 или 8 байт в начале кода, то мы получим ошибку принтф-а.

```

        .file "allmy.s"
        .text
        .globl arr
        .data
        .align 32
        .type arr, @object
        .size arr, 40
arr:
        .long 1
        .long 2
        .long 3
        .long 4
        .long 5
        .long 6
        .long 7
        .long 8
        .long 9
        .long 0
        .section .rodata
.LC0:
        .string "...: %d"
        .text
        .globl main
        .type main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6
        subq $16, %rsp ;# then we have 16 bytes of local stack
start:
        movl $0, %eax
        movl $0, -4(%rbp)
        jmp for
infor:
        movl -4(%rbp), %ecx
        leaq 0(, %ecx, 4), %rdx

```

```

        leaq arr(%rip), %rbx
        addq (%rdx, %rbx), %rax
        addl $1, -4(%rbp)
for:
        cmpl $10, -4(%rbp)
        jb infor
exit_out:
        movl %eax, %esi
        leaq .LC0(%rip), %rax
        movq %rax, %rdi
        movl $0, %eax
        call printf@PLT
...

```

Собственно, выведет все это чудо ..: 45, что говорит о правильно посчитанной сумме.

11. Найдем максимум в массиве.

Код теперь будет выглядеть как-то так. Он верно находит максимум при различных вариациях массива.

```

        .file      "allmy_if_in_for.s"

        .text
        .globl  arr
        .data
        .align 32
        .type   arr, @object
        .size   arr, 40
arr:
        .long   1
        .long   2
        .long   3
        .long   9
        .long   5
        .long   6
        .long   7
        .long   5
        .long   1
        .long   0

        .section      .rodata
.LC0:
        .string  "...: %d"

        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp ;# then we have 16 bytes of local stack

start:
        movl $0, %eax
        movl $0, -4(%rbp)
        jmp for

```

```

gg:
    movq (%rdx, %rbx), %rax
    jmp ei_f

infor:
    movl -4(%rbp), %ecx
    leaq 0(, %ecx, 4), %rdx
    leaq arr(%rip), %rbx
    cmpl (%rdx, %rbx), %eax
    jb gg
ei_f:    addl $1, -4(%rbp)

for:
    cmpl $10, -4(%rbp)
    jb infor

exit_out:
    movl %eax, %esi
    leaq .LC0(%rip), %rax
    movq %rax, %rdi
    movl $0, %eax
    call printf@PLT

...

```

12. Вывод текущего максимума.

!!!!!! ЗАМЕЧАНИЕ. printf та еще скотина, долго с ним боролся, но в итоге пооял, при вызове он кладет мусор во все используемые им регистры, по крайней во все "его" регистры, используемые нами ранее, хотя предположу, что он счищает вообще все таковые. К ним относятся напомним как минимум esi, edx, ecx, rdx... rdi. Так же оно сметает и eax!!!!!!

Теперь зная этот прикол напишем нормальный код.

```

.file "max_in_arr.s"
.text
.globl arr
.data
.align 32
.type arr, @object
.size arr, 40
arr:
    .long 1
    .long 2
    .long 3
    .long 9
    .long 5
    .long 6
    .long 7
    .long 5
    .long 1
    .long 0

.section .rodata
.LC0:
.string "global array maximum: %d"

.section .rodata
.LC1:
.string "current max: %d \n"

.text

```

```

        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp ;# then we have 16 bytes of local stack

start:
        movl    $0, %eax
        movl    $0, -4(%rbp)
        jmp     for

gg:
        movl    (%rdx, %rax), %ebx
        jmp     ei_f

infor:
        leaq    .LC1(%rip), %rdx
        movq    %rdx, %rdi
        movl    %ebx, %esi
        call    printf@PLT

        movl    -4(%rbp), %ecx
        leaq    0(, %ecx, 4), %rdx
        leaq    arr(%rip), %rax

        cmpl    (%rdx, %rax), %ebx
        jb      gg
ei_f:
        addl    $1, -4(%rbp)

for:
        cmpl    $10, -4(%rbp)
        jb      infor

exit_out:
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        movl    %ebx, %esi
        call    printf@PLT
        ...

```

Вывод этого будет:

```

current max: 0
current max: 1
current max: 2
current max: 3
current max: 9
current max: 9
current max: 9
current max: 9
current max: 9
current max: 9
global array maximum: 9

```

Чего мы и ожидали.

13. Сортировка пузырьком.

Код сортировки пузырьком:

```
.file          "max_in_arr.s"

.text
.globl  arr
.data
.align 32
.type   arr, @object
.size   arr, 40
arr:
.long   1
.long   2
.long   3
.long   9
.long   5
.long   6
.long   7
.long   5
.long   1
.long   0

.section       .rodata
.LC0:
.string "%d "

.text
.globl  main
.type   main, @function
main:
.LFB0:
.cfi_startproc
endbr64
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $16, %rsp ;# then we have 16 bytes of local stack

start:
movl    $0, %eax
movl    $0, -4(%rbp) ;# i
jmp     for_1

infor_1:
movl    $0, -8(%rbp) ;# j
jmp     for_2

infor_2:
movl    -8(%rbp), %eax
leaq    0(, %eax, 4), %rbx ;# +j -> rbx
movl    -4(%rbp), %ecx
leaq    0(, %ecx, 4), %rdx ;# +i -> rdx
leaq    arr(%rip), %rax

movl    (%rbx, %rax), %ecx
cmpl    (%rdx, %rax), %ecx
ja      swap
```

```

        jmp ch_pl_2
swap:
    movl (%rbx, %rax), %ecx
    movl (%rdx, %rax), %esi ;# normal registers have run out((
    movl %esi, (%rbx, %rax)
    movl %ecx, (%rdx, %rax)

ch_pl_2:    addl $1, -8(%rbp)

for_2:
    cmpl $10, -8(%rbp)
    jnb infor_2

    addl $1, -4(%rbp)

for_1:
    cmpl $10, -4(%rbp)
    jnb infor_1

    movl $0, -4(%rbp)
    jmp for_p

infor_p:
    movl -4(%rbp), %ecx
    leaq 0(, %ecx, 4), %rbx ;# +i -> rbx
    leaq arr(%rip), %rax
    movl (%rbx, %rax), %esi
    leaq .LC0(%rip), %rax
    movq %rax, %rdi
    call printf@PLT

    addl $1, -4(%rbp)

for_p:
    cmpl $10, -4(%rbp)
    jnb infor_p

exit_out:
    ...

```

Выведет такая штука мой массив по возрастанию, что говорит о корректности работы.

14. Bobble profiler.

Впишем теперь с помощью ассемблерной вставки внутренности функции sort. Тут я представлю ее содержимое.

```

void sort() {
    asm("start:\n"
        "    movl $0, -4(%rbp) # i\n"
        "    jmp for_1\n"
        "\n"
        "\n"
        "infor_1:\n"
        "    movl $0, -8(%rbp) # j\n"
        "    jmp for_2\n"
        "\n"
        "infor_2:\n"
        "    movl -8(%rbp), %eax\n"
        "    leaq 0(, %eax, 4), %rbx # +j -> rbx\n"
        "    movl -4(%rbp), %ecx\n"
        "    leaq 0(, %ecx, 4), %rdx # +i -> rdx\n"
        "    leaq a(%rip), %rax\n"

```

```

"\n"
"    movl (%rbx, %rax), %ecx\n"
"    cmpl (%rdx, %rax), %ecx\n"
"    ja  swap\n"
"    jmp ch_pl_2\n"
"swap: \n"
"    movl (%rbx, %rax), %ecx\n"
"    movl (%rdx, %rax), %esi # normal registers have run out((\n"
"    movl %esi, (%rbx, %rax)\n"
"    movl %ecx, (%rdx, %rax)\n"
"\n"
"ch_pl_2:    addl $1, -8(%rbp)\n"
"\n"
"for_2:\n"
"    movl n(%rip), %ecx\n"
"    cmpl %ecx, -8(%rbp)\n"
"    jb  infor_2\n"
"\n"
"    addl $1, -4(%rbp)\n"
"\n"
"for_1:\n"
"    movl n(%rip), %ecx\n"
"    cmpl %ecx, -4(%rbp)\n"
"    jb  infor_1\n");
}

```

Представим тут так же результаты профайлера:

```

4.78815e-07
5.92416e-07
7.34575e-07
1.16575e-06
1.54906e-06
2.44013e-06
3.81466e-06
5.98682e-06
9.73436e-06
1.591e-05
2.67367e-05
4.33805e-05
7.3669e-05
0.000121372
0.000204171
0.000354647
0.000585164
0.000982711
0.00164693
0.00278774
0.00474385
0.00935751
0.0169847
0.0236952
0.0379802
0.0747821
0.137974
0.206431
0.317607
0.521221
0.954548
1.67532

```

Конец.