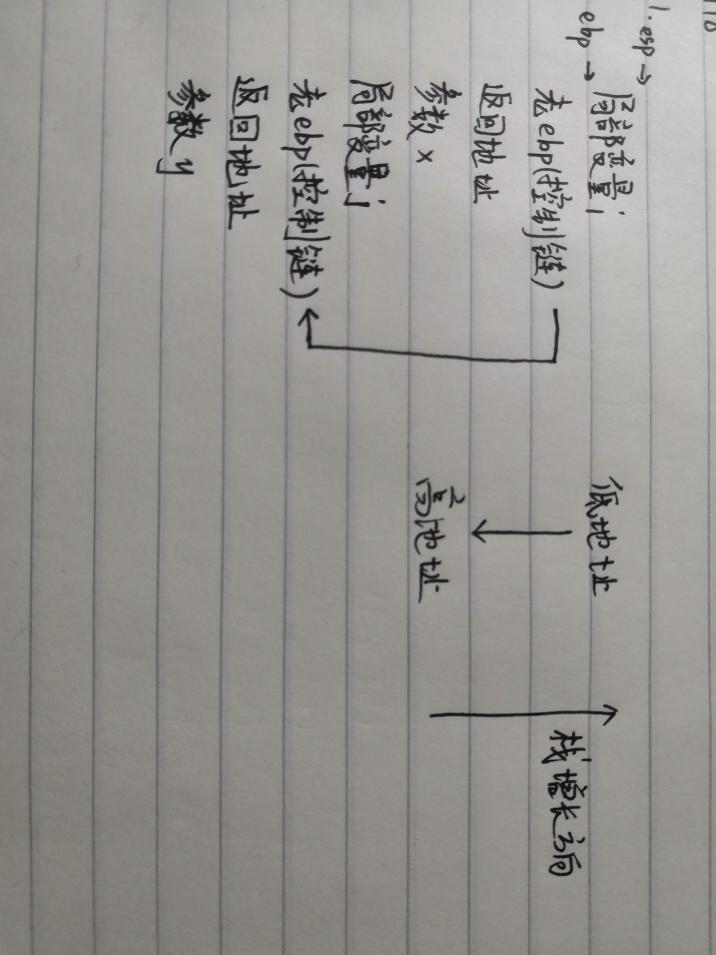
H8 PB18000290 胡毅翔

1.



2.

编译器版本：

    .ident  "GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0"

汇编代码：

main:

.LFB0:

    .cfi\_startproc

    pushq   %rbp

    .cfi\_def\_cfa\_offset 16

    .cfi\_offset 6, -16

    movq    %rsp, %rbp

    .cfi\_def\_cfa\_register 6

    subq    $32, %rsp

    movq    %fs:40, %rax

    movq    %rax, -8(%rbp)

    xorl    %eax, %eax

    movq    $4, -32(%rbp)

    movq    $8, -24(%rbp)

    movq    -16(%rbp), %rax

    movq    %rax, %rdx

    movl    $0, %esi

    leaq    .LC0(%rip), %rdi

    movl    $0, %eax

    call    printf@PLT

    movl    $0, %eax

    movq    -8(%rbp), %rcx

    xorq    %fs:40, %rcx

    je  .L3

    call    \_\_stack\_chk\_fail@PLT

1. sizeof(a)的值是多少，请说明理由。

Sizeof(a)的值为0。

由汇编代码中movl $0,%esi可知。

1. a[0][0]的值是多少，请说明理由。

是地址为-16(%rbp)，长度为64位所存储的值。因未初始化，所以每次输出的值是不确定的。

由汇编代码：

movq -16(%rbp),%rax

movq %rax,%rdx

可知。

3.

在该汇编代码中有关的指令后加注释，将源程序中的操作和生成的汇编代码对应起来，以判断确实是用短路计算方式来完成布尔表达式计算的。

     .file   "ex7-9.c"

     .text

     .globl  main

     .type   main, @function

main:

.LFB0:

     pushq   %rbp

     movq    %rsp, %rbp

     jmp     .L2

.L5:                            ;while语句的内容

     movl    -4(%rbp), %eax     ;取j

     movl    %eax, -8(%rbp)     ;将i的值赋为j

.L2:

     cmpl    $0, -8(%rbp)       ;判断i是否为0

     jne     .L3                ;若i为0，则跳转到j>5的判断

     cmpl    $0, -4(%rbp)       ;继续判断j是否为0

     je      .L4                ;若j为0则不继续判断（转.L4）

.L3:

     cmpl    $5, -4(%rbp)       ;继续判断j>5

     jg      .L5                ;若大于5

.L4:

     movl    $0, %eax           ;返回值置为0？

     popq    %rbp

     ret

.LFE0:

     .size   main, .-main

     .ident  "GCC: (Ubuntu 7.5.0-3ubuntu1~16.04) 7.5.0"

由上述代码可知，确实是按短路计算方式来完成布尔表达式的计算的。

4.

1)

因为cp1和cp2所指向的字符串保存在连续的地址空间。指针cp1指向的地址为a，指针cp2指向的地址为a+6。在执行指令strcpy(cp1,cp2)时，将cp2所指向的字符串的内容写到cp1大于的地址。写完后cp2所指向的地址空间对应的字符串便为”ghij”。即字符串”abcdefghij”第6个字符开始到结束对应的内容。

2)

因为在输出cp1时，发生了越界。

5.

请解释为什么会有这样的输出，画出各个形参和局部变量的内存布局。你可以结合你的实验环境产生的汇编码和运行结果进一步详细分析。

GCC: (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

下的运行结果：

0x7ffff924155c, 0x7ffff9241558, 0x7ffff9241554, 0x7ffff9241550

0x7ffff924156c, 0x7ffff924156e, 0x7ffff9241570, 0x7ffff9241574

0x7ffff924155c, 0x7ffff9241558, 0x7ffff9241550, 0x7ffff9241548

0x7ffff924156c, 0x7ffff924156e, 0x7ffff9241570, 0x7ffff9241574

|  |  |  |
| --- | --- | --- |
| func | 低地址 |  |
| 形参e | 0x7ffff9241550 | 4字节 |
| 形参f | 0x7ffff9241554 | 4字节 |
| 形参j | 0x7ffff9241558 | 4字节 |
| 形参i | 0x7ffff924155c | 4字节 |
|  |  |  |
| 局部变量i1 | 0x7ffff924156c | 2字节 |
| 局部变量j1 | 0x7ffff924156e | 2字节 |
| 局部变量f1 | 0x7ffff9241570 | 4字节 |
| 局部变量e1 | 0x7ffff9241574 | 4字节 |
|  | 高地址 |  |
| funcOld | 低地址 |  |
| 形参e | 0x7ffff9241548 | 8字节 |
| 形参f | 0x7ffff9241550 | 8字节 |
| 形参j | 0x7ffff9241558 | 4字节 |
| 形参i | 0x7ffff924155c | 4字节 |
|  |  |  |
| 局部变量i1 | 0x7ffff924156c | 2字节 |
| 局部变量j1 | 0x7ffff924156e | 2字节 |
| 局部变量f1 | 0x7ffff9241570 | 4字节 |
| 局部变量e1 | 0x7ffff9241574 | 4字节 |
|  | 高地址 |  |

funcOld节选

    leaq    -56(%rbp), %rsi

    leaq    -48(%rbp), %rcx

    leaq    -40(%rbp), %rdx

    leaq    -36(%rbp), %rax

func节选

    leaq    -48(%rbp), %rsi

    leaq    -44(%rbp), %rcx

    leaq    -40(%rbp), %rdx

    leaq    -36(%rbp), %rax

可以看出两个函数因声明方式不同对形参分配的空间大小不同。