Homework5

邢添瑆
2024202862

Q1

Consider the following program:

```
#define LEN 10
int a[LEN][LEN];
void f(void)
{
  int i, j;
  for (i = 0; i < LEN; i++)
    for (j = 0; j < LEN; j++)
    {
      a[i][j] = i * LEN + j;
    }
}
```

Suppose the address of a is 0x10000000. After the function f() finished, fill the following table (if you don't know the value, please write NONE):

Answer1

Place	Value
%eax	0x10000000
%ecx	22
$0x10000004	1
0x10000012	NONE
0xFFFFFFF8	NONE
(%eax,%ecx,8)	44

Q2

Fill the blanks of the C program:

```
int dw_loop(int x, int y, int n)
{
  do
  {
    /* */
  } while (/* */);
  return x;
}
```

The assembly code is as follows:

```
  x@%ebp+8, y@%ebp+12, n@%ebp+16

  movl 8(%ebp), %eax // x
  movl 12(%ebp), %ecx // y
  movl 16(%ebp), %edx // n
.L2:
  addl %edx, %eax
  imull %edx, %ecx
  subl $1, %edx
  testl %edx, %edx
  jle  .L5
  cmpl %edx, %ecx
  jl .L2
.L5:
```

Answer2

```
int dw_loop(int x, int y, int n)
{
  do
  {
    x += n;
    y *=  n;
    n -= 1;
  } while (n > 0 && y < n);
  return x;
}
```

Q3

After ICS class, Barathrum has written a function like below:

```
int cmov_complex(int x, int y)
{
  return x < y ? x * y : (x + y) * y;
}
```

(1). Please write down the corresponding assembly code by using conditional move operations.

Answer3(1)

```
cmov_complex:
  movl 8(%ebp), %eax
  movl 12(%ebp), %ecx
  movl %eax, %edx
  imull %ecx, %eax
  addl %ecx, %edx
  imull %ecx, %edx
  cmp  8(%ebp), %ecx
  cmovge %edx, %eax
  ret
```

(2). When Barathrum compiles it with gcc, he finds that there's no cmov at all in the assembly code! Please explain why gcc doesn't use conditional move operations in this case.

Answer3(2)

如果要使用 cmov ，那么就必须进行两次乘法运算；但是如果使用这样的分支结构：

```
0000000000000000 <cmov_complex>:
   0:   endbr64
   4:   push   %rbp
   5:   mov    %rsp,%rbp
   8:   mov    %edi,-0x4(%rbp)
   b:   mov    %esi,-0x8(%rbp)
   e:   mov    -0x4(%rbp),%eax
  11:   cmp    -0x8(%rbp),%eax
  14:   jge    1f <cmov_complex+0x1f>
  16:   mov    -0x4(%rbp),%eax
  19:   imul   -0x8(%rbp),%eax
  1d:   jmp    2b <cmov_complex+0x2b>
  1f:   mov    -0x4(%rbp),%edx
  22:   mov    -0x8(%rbp),%eax
  25:   add    %edx,%eax
  27:   imul   -0x8(%rbp),%eax
  2b:   pop    %rbp
  2c:   ret
```

可以只进行一次乘法运算，效率比 cmov 更高。
在看了 stack overflow 上几个关于 GCC 优化的帖子以后，发现似乎是因为 x < y 是数据相关的、可预测的，即使加了 cmov 跳转也不一定更快，并且还会占用更多寄存器，所以 GCC 选择使用分支结构。

Q4

Translate the following switch statements into assembly using jump table.

```
int x = <some value>;
int result = 0;
switch (x)
{
case 24:
  result = x + x;
  break;
case 27:
case 28:
  result = x + 10;
  break;
case 26:
  result = x * 2;
// Notice: there is no break here!
case 29:
case 30:
  result = result + 5;
  break;
default:
  result = 3;
  break;
}
```

Answer4

```
        .section .rodata
.L9:
        .quad   .L1
        .quad   .L8
        .quad   .L2
        .quad   .L5
        .quad   .L5
        .quad   .L7
        .quad   .L7


        # x in %edi
        movl    %edi, %eax
        subl    $24, %eax
        cmpl    $6, %eax
        ja      .L8
        jmp     *.L9(,%eax,8)

.L1:
        lea     (%rdi,%rdi), %eax
        jmp     .L10

.L6:
        lea     (%rdi,%rdi), %eax
        addl    $5, %eax
        jmp     .L10

.L7:
.L8:
        leal    10(%rdi), %eax
        jmp     .L10

.L9:
.L3:
        movl    $5, %eax
        jmp     .L10

.L2:
        movl    $3, %eax
```

```
.L10
        ret
```