ICS Homework 3

Cao Shengcao

3.60

The assembly code:

```
loop:
    movl    %esi, %ecx
    movl    $1, %edx
    movl    $0, %eax
    jmp     .L2
.L3:
    movq    %rdi, %r8
    andq    %rdx, %r8
    orq     %r8, %rax
    salq    %cl, %rdx
.L2:
    testq   %rdx, %rdx
    jne     .L3
    rep; ret
```

- A. `%rdi` holds `x`, `%esi` and `%ecx` holds `n`, `%rax` holds `result`, and `%rdx` holds `mask`.
- B. Initially, `result = 0`, and `mask = 1`.
- C. `testq   %rdx, %rdx` and `jne     .L3` indicates the test condition `mask != 0`.
- D. `salq    %cl, %rdx` indicates the update `mask <<= n`. You may think that, to be more accurate, `%cl` is just the lowest byte of `%ecx`, so `mask <<= n & 0xFF` is better. Actually, I compiled the code below with arguments `gcc 3.60.c -m64 -O1 -S -o 3.60.s` and get almost the same assembly code. To conclude, writing `mask <<= n` is fine, because when `n` is too large the behavior is undefined.
- E. First, `movq    %rdi, %r8` and `andq    %rdx, %r8` calculate `x & mask`. Then, `orq     %r8, %rax` updates `result |= x & mask`.
- F. The C code is shown below.

```
long loop(long x, int n)
{
    long result = 0;
    long mask;
    for (mask = 1; mask != 0; mask <<= n)   // "mask <<= n & 0xFF" is better
    {
        result |= x & mask;
    }
    return result;
}
```

3.62

The assembly code:

```
.L8:            MODE_E
    movl    $27, %eax
    ret
.L3:            MODE_A
    movq    (%rsi), %rax
    movq    (%rdi), %rdx
    movq    %rdx, (%rsi)
    ret
.L5:            MODE_B
    movq    (%rdi), %rax
    addq    (%rsi), %rax
    movq    %rax, (%rdi)
    ret
.L6:            MODE_C
    movq    $59, (%rdi)
    movq    (%rsi), %rax
    ret
.L7:            MODE_D
    movq    (%rsi), %rax
    movq    %rax, (%rdi)
    movl    $27, %eax
    ret
.L9:            default
    movl    $12, %eax
    ret
```

The C code:

```c
long switch3(long* p1, long* p2, mode_t action)
{
    long result = 0;
    switch (action)
    {
    case MODE_A:
        result = *p2;
        *p2 = *p1;
        break;
    case MODE_B:
        result = *p1 + *p2;
        *p1 = result;
        break;
    case MODE_C:
        *p1 = 59;
        result = *p2;
        break;
    case MODE_D:
        *p1 = *p2;
    case MODE_E:
        result = 27;
        break;
    default:
        result = 12;
        break;
    }
    return result;
}
```

Don't forget to `break`! This exercise required us to reconstruct "one case that fell through to another" (which is `MODE_D`). Don't miss it like I did at first. Besides, it's unnecessary to write redundant statements like `action = *p1; *p2 = action;` in `MODE_A`.

3.64

- A. For the array `A[R][S][T]`, the element `A[i][j][k]` is located at

  $$\&A[i][j][k] = x_A + L(i \cdot S \cdot T + j \cdot T + k)$$

  where x_A is the starting address and L is the data type size.

- B. We can translate the assembly code line by line:

  `leaq (%rsi, %rsi, 2), %rax` $\rightarrow$ `t1 = j * 3`

  `leaq (%rsi, %rax, 4), %rax` $\rightarrow$ `t2 = j + t1 * 4`

  `movq %rdi, %rsi` $\rightarrow$ `t3 = i`

  `salq $6, %rsi` $\rightarrow$ `t4 = t3 * 64`

  `addq %rsi, %rdi` $\rightarrow$ `t5 = t3 + t4`

  `addq %rax, %rdi` $\rightarrow$ `t6 = t2 + t5`

  `addq %rdi, %rdx` $\rightarrow$ `t7 = t6 + k`

  `movq A(, %rdx, 8), %rax` $\rightarrow$ `t8 = *(long*)((char*)A + t7 * 8)` `movq %rax, (%rcx)` $\rightarrow$ `*dest = t8`

`movl $3640, %eax` $\rightarrow$ `t9 = 3640`

`ret` $\rightarrow$ `return t9`

Then we can infer that `t7` is actually `i * 65 + j * 13 + k`. Therefore we have $S = 5, T = 13$. Furthermore, $R \cdot S \cdot T \cdot L = sizeof(A)$, so $R = 3640/8/5/13 = 7$. The array is `long A[7][5][13]`.