COMP.2030 Lab 9

1. Suppose that MIPS registers `$t0` and `$t1` are mapped to the x86 registers `%eax` and `%ebx`, respectively. Convert the MIPS instructions below left to the corresponding x86 instructions. Use the same label X in your x86 code.

```
      lw   $t1, X($zero)
      li   $t0, 1
      lw   $t1, X($t0)
      move $t0, $t1
      sw   $t0, ($t1)
      addi $t0, $t0, 1
```

2. Suppose that the MIPS registers `$t0`, `$t1` and `$t2` registers are mapped to the x86 registers `%rax`, `%rbx` and `%rcx`, respectively. Convert the MIPS instruction on the left below to the corresponding x86 instructions. Use the same labels in your x86 code as shown in the MIPS code.

```
     lw   $t2, X($zero)
     li   $t0, 1
rept:
     bge  $t0, $t2, exit  # $t0>=$t2
     sll  $t0, $t0, 2
     lw   $t1, X($t0)
     bgt  $t2, $t1, next  # $t2>$t1
     move $t2, $t1
     sw   $t0, ($t1)
next:
     sra  $t0, $t0, 2
     addi $t0, $t0, 1
```

1. The function `fun_a` has the overall structure shown below:

```
long fun_a(unsigned long x){
    long val = 0;
    while (            ){



    }


    return val;
}
```

```
# x in %rdi
fun_a:
        movl    $0, %eax
        jmp     .L5
.L6:
        xorq    %rdi, %rax
        shrq    %rdi  # shift right 1
.L5:
        testq   %rdi, %rdi
        jne     .L6
        andl    $1, %eax
        ret
```

The gcc C compiler generates the x86-64 assembly code on the right. Reverse engineer this assembly code and fill in the missing parts of the `fun_a` definition so the C code does the same thing.

2. The gcc C compiler generates the assembly code blow to the right. Reverse engineer the operation of this code and fill in the missing parts of the C code to the left so that it does the same thing.

```
long loop(long x, long n){

    long result =                 ;
    long mask;
    for (mask =                   ;

                                  ;

         mask =                   ){

        result |=                 ;
    }
    return result;
}
```

```
# x in %rdi, n in %esi
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
        ret
```