# Exercise 2

## Byte Codes Translation

Please write down the byte codes of the following Y86-64 instructions.

| Y86-64 instructions | Byte codes (hex value) |
|---|---|
| `popq %rbx` | |
| `jne 0x1234567` | |
| `subq %rax, %rdx` | |
| `mrmovq 0x432(%rax), %rcx` | |
| `rrmovq %rdx, %rbx` | |
| `call 0x8765432` | |

## Instruction Design

Suppose we are going to implement **`cirmovxx V, rB`**, which conditionally moves value V to register rB, in our SEQ Y86_64 processor.

| Stage | `cirmovxx V, rB` |
|---|---|
| Fetch | |
| Decode | |
| Execute | |
| Memory | |
| Write back | |
| PC update | |

## Jump Table

In Section 3.6.8, we saw a common way to implement switch statements is to create a set of

code blocks and then index those blocks using a jump table. Consider the C code shown below for a function `switchv`.

```
long switchv(long idx) {
    long result = 0;
    switch(idx) {
        case 3:
            result = 0xa;
            break;
        case 1:
        case 5:
        Case 7:
            result = 0xb;
            break;
        case 4:
            result = 0xc;
            break;
        default:
            result = 0xd;
    }
    return result;
}
```

Alice wants to implement **`switchv`** in Y86-64 using jump table. Since Y86-64 instruction set does not include indirect jump instruction, she decides to get the same effect by combining several of them. Below is part of her solution.

```
jtable:
    .quad LD
    .quad L0
    .quad L1
    .quad L2
    .quad L3
    .quad L4
    .quad L5
    .quad L6
switchv:
    irmovq [1], %r8
    irmovq [2], %r10
    irmovq [3], %r11
    irmovq $0, %rax
    irmovq jtable, %rcx
    rrmovq %rdi, %rdx
    subq %r8, %rdx
```

```
jump:
    mrmovq (%rcx), %rdi
    # "Question 2"
L0:
    [4]
    ret
L1:
    [5]
L2:
    [6]
    ret
L3:
    irmovq $0xc, %rax
    ret
L4:
    jmp L0
L5:
```

```
    jg jump
    subq %r10, %rdi
    jl jump
locate:
    irmovq $0x8, %r8
    addq %r8, %rcx
    subq %r11, %rdi
    jl jump
    jmp locate
```

```
    jmp LD
L6:
     jmp L0
LD:
    irmovq $0xd, %rax
    ret
```

1. Please fill in the blanks.
2. The marks “Question 2” stands for indirect jump to *`%rdi`, please write down a combination of Y86-64 instructions to make that effect. (Hint: use two Y86-64 instructions)

# Y86-64 Assembly

Consider the following assembly.

```
0x000:                          |       .pos 0
0x000: 30f4 00100000 00000000   | Init: irmovq 0x1000, %rsp
0x00a: 30f5 00100000 00000000   |       irmovq 0x1000, %rbp
0x014: _[1]_                    |       call Main
0x01d: 00                       |       halt
                                |
_[2]_:                          |       .align 8
_[3]_: 33010000 00000000        | Array:  .quad 0x133
0x028: fc0d0000 00000000        |       .quad 0xdfc
0x030: 2f0f0000 00000000        |       .quad 0xf2f
0x038: 33020000 00000000        |       .quad 0x_[4]_
                                |
_[5]_: a05f                     | Main:  pushq %rbp
0x042: 2045                     |       rrmovq %rsp, %rbp
0x044: a03f                     |       ____[6]____
0x046: ____[7]____              |       irmovq Array, %rdx
0x050: 5002 00000000 00000000   |       mrmovq (%rdx), %rax
0x05a: 5032 08000000 00000000   |       mrmovq 8(%rdx), %rbx
0x064: 5012 10000000 00000000   |       mrmovq 0x10(%rdx), %rcx
0x06e: 5022 18000000 00000000   |       mrmovq 0x18(%rdx), %rdx
```

```
_[8]_: ____[9]____              |       addq %rax, %rbx
0x07a: 6112                     |       subq %rcx, %rdx
0x07c: 6131                     |       ____[10]____
0x07e: b03f                     |       popq %rbx
0x080: 70 55000000 00000000     |       jmp End
0x08a: 2054                     | End:  rrmovq %rbp, %rsp
0x08c: b05f                     |       popq %rbp
0x08e: 90                       |       ret
```

1. Please fill in the blanks within above Y86-64 binary and assembly code.
2. Suppose the entry point is 0x000, please calculate the value of below registers after the program HALT.

| | |
|---|---|
| %rdx | [1] |
| %rsp | [2] |
| CC.ZF | [3] |
| CC.SF | [4] |
| CC.OF | [5] |