Problem 1: HCL

Please write down the HCL expressions for the following signals (HINT: you can refer to the Section 4.2.2 in the CSAPP book).

EXAMPLE: Show if the two input signals a and b are equal

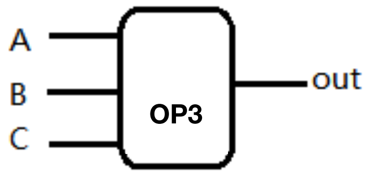
```
bool eq = (a&&b) || (!a && !b);
```

1. The HCL expression for a signal **NAND**, which is equal to **NAND** of inputs **a** and **b**, the truth table is given, and you should **only** use **NOT** (**!**) and **OR** (**||**) operators.

NAND	0	1
0	1	1
1	1	0

```
bool NAND = !a || !b;
```

2. A HCL expression called OP3: If and only if all the inputs are the same, output will be true (1). Each input and output is one-bit wise. (Hints: You can use boolean expressions or case expressions.)



```
Sol1: bool op3 = (A && B && C) || (!A && !B && !C);
Sol2: bool op3 = [A && B && C : 1;
                  A ^ B       : 0;
                  A ^ C       : 0;
                  B ^ C       : 0;
                  1           : 1;
                 ];
Or other solutions satisfied the truth table.
```

Problem 2: SEQ

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

1. Try to fill the table.

Stage	cirmovxx V, rB

Fetch	icode:ifun <- M1[PC]
	rA:rB <- M1[PC+1]
	valC <- M8[PC+2]
	valP <- PC+10

Decode	–

Execute	Cnd <- Cond(CC,ifun)
	valE <- valC + 0

Memory	–
Write back	R[rB] <- Cnd?valE:-
PC update	PC <- valP

2. Which of following logics should be modified, please give the HCL code. { aluA, aluB, new_pc, dstE }

```
word aluA = [
    icode in { IRRMOVQ, IOPQ } : valA
    icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, ICIRMOVXX} : valC;
    icode in { ICALL, IPUSHQ } : -8;
    icode in { IRET, IPOPQ } : 8;
];

word aluB = [
    icode  in  {  IRMMOVQ,  IMRMOVQ,  IOPQ,  ICALL,  IPUSHQ,  IRET,  IPOPQ  }  :
valB;
    icode in { IRRMOVQ, IIRMOVQ, ICIRMOVXX} : 0;
];

word dstE = [
    icode in { IRRMOVQ, IIRMOVQ, IOPQ } : rB;
    icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
    icode == ICIRMOVXX && Cnd : rB;
];
```

Problem 3: Y86-64

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 0:
        result = 0xaaa;
        break;
    case 2:
    case 5:
        result = 0xbbb;
        break;
    case 3:
        result = 0xccc;
        break;
    default:
        result = 0xddd;
    }
    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. Here is part of her solution.

```
jtable:
    .quad LD
    .quad L0
    .quad L1
    .quad L2
    .quad L3
    .quad L4
    .quad L5

switchv:
    irmovq [1], %r8
    irmovq [2], %r10
    irmovq [3], %r11

    irmovq $0, %rax
    irmovq jtable, %rcx
    rrmovq %rdi, %rdx
    subq %r8, %rdx
    jg dflt
    subq %r10, %rdi
    jl dflt
mul:
    irmovq $0x8, %r8
    subq %r10, %rdi
    je addr
    addq %r8, %rcx
    subq %r11, %rdi
    jmp mul
```

```
addr:
    addq %r8, %rcx
    mrmovq (%rcx), %rdi
    # "Question 2"
dflt:
    irmovq jtable, %rcx
    mrmovq (%rcx), %rdi
    # "Question 2"
L0:
    [4]
    ret
L1:
    [5]
L2:
    jmp L5
L3:
    irmovq $0xccc, %rax
    [6]
L4:
    jmp LD
L5:
    irmovq $0xbbb, %rax
    ret
LD:
    irmovq $0xddd, %rax
    ret
```

1. Please fill in the blanks.
    1. `$5`
    2. `$0`
    3. `$1`
    4. `irmovq $0xaaa, %rax`
    5. `jmp LD`
    6. `ret`
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).
    1. `pushq %rdi`
    2. `ret`