Problem 1:

Please write down the HCL code for the following signals in the SEQ implementation.

1. need_valC
2. aluB
3. mem_write

1. bool need_valC = icode in {IIRMOVQ, IRMMOVQ, IMRMOVQ, IJXX, ICALL};

2. int aluB = [

    icode in {IRMMOVQ, IMRMOVQ, IOPQ, ICALL, IPUSHQ, IRET, IPOPQ} : valB;

    icode in {IRRMOVQ, IIRMOVQ} : 0;

] ;

3. bool mem_write = icode in {IRMMOVQ, IPUSHQ, ICALL};

Problem 2:

Please fill the blanks below. Assume the value of PC is 0x10.

Stage	Generic	Specific
	irmovq V, rB	Irmovq 0x123, %rbx
Fetch	icode:ifun <- M1[PC] rA:rB <- M1[PC+1] valC <- M8[PC+2] valP <- PC+10	icode:ifun<-M1[0x10]=3:0 rA:rB<-M1[0x11]=f:3 valC<-M8[0x12]=0x123 valP<-0x1a
Decode		
Execute	valE <- 0 + valC	valE <- 0 + 0x123
Memory		
Write back	R[rB] <- valE	R[%rbx] = 0x123
PC update	PC <- valP	PC = 0x1a

Please fill the blanks below. Assume the value of PC is 0x10 and the value of %rsp is 0x200. The 8-byte value stored in 0x200 is 0x4000.

Stage	Generic	Specific
	popq rA	popq %rax
Fetch	icode:ifun <- M1[PC] rA:rB <- M1[PC+1]	icode:ifun<-M1[0x10]=B:0 rA:rB<-M1[0x11]=0:f

	valP <- PC+2	valP<-0x12
Decode	valA <- R[%rsp] valB <- R[%rsp]	valA <- 0x200 valB <- 0x200
Execute	valE <- valB + 8	valE <- 0x208
Memory	valM <- M8[valA]	valM <- 0x4000
Write back	R[%rsp] <- valE R[rA] <- valM	R[%rsp] = 0x208 R[%rax] = 0x4000
PC update	PC <- valP	PC = 0x12

Problem 3:

Please explain the pros and cons of RISC compared with CISC.

Call and ret instructions access stack to save and load return addresses. Please consider a different set of these instructions:

callr Dest, which saves the return address in %rsi and jumps to the instruction at Dest;

retr, which jumps to the instruction with the address saved in %rsi.

1. Please fill in the blanks below to describe these two instructions which are executed in sequential Y86 implementation. You only need to write the generic logic for each instruction.

Stage	Generic	Specific
	callr Dest	retr
Fetch	icode:ifun <- M1[PC] valC <- M8[PC+1] valP <- PC+9	icode:ifun <- M1[PC] valP <- PC+1
Decode		valA <- R[%rsi]
Execute		
Memory		
Write back	R[%rsi] <- valP	
PC update	PC <- valC	PC <- valA

2. For any correctly implemented Y86 program, if we simply replace all call instructions with callr and all ret with retr, will the program run correctly? Why or why not?
   No. If function A calls function B, and then B calls C, %rsi will be filled with the return address from C to B, overwritten the return address from B to A