Homework 4

Instruction:

- The submission deadline is **2024 Dec. 11 23:59**. Late submissions will not be accepted.
- The solutions should be submitted to Canvas. Email submissions (or other form of submissions) will not be accepted.
- Your submission should contain the following files:
    - A "*.circ" file that contains the processor design.
    - Four memory image file that contains the binary machine codes of the four testing programs, named as "Mem-1", "Mem-2", "Mem-3" and "Mem-4", respectively.
    - The completed answer sheet (either WORD or PDF file).
- If you choose to answer the bonus question, you should submit two extra files (i.e., they should be separated files besides your solutions to Question 1-4).
    - A "*.circ" file that contains the circuit for the bonus question.
    - The completed answer sheet of the bonus question (either WORD or PDF file).
- You can use all build-in components provided in Logisim.
- The weight of this HW 8% in the total grade (excluding the bonus question). The bonus question contributes at most 4% to the total grade (but the total CA grade is capped by 30% and the extra marks of the bonus question cannot be carried into the final exam grade).

Question:

Build a processor in Logisim. The processor should meet the following requirements:

- This is a 16-bit processor. In particular, all registers are 16-bit, and each instruction is also 16-bit.
- The processor has 8 general-purpose registers $r1 – $r8.
- The data and instruction are stored in two separated memory, each having its own address space. Both the data and instruction memory are addressed by 16-bits, i.e., the size of the memory block at each address is 16-bits. (Notice, this is different from the examples in our lecture, where each memory block is 8-bit.)
- The processor supports the following instructions.

Name	Format	Functionality
li	li $r1, x	load immediate number to one register ($r1)
add	add $r1, $r2, $r3	add two register numbers ($r2 and $r3) and store the result in one register ($r1)
and	and $r1, $r2, $r3	bit-wise logical AND of two register numbers ($r2 and $r3), and store the result in a register ($r1)
or	or $r1, $r2, $r3	bit-wise logical OR of two register numbers ($r2 and $r3), and store the result in a register ($r1)
load	ld $r1, $r2	load a 16-bit number from data memory to a register ($r1). The data memory address is the value in a register ($r2)
store	st $r1, $r2	load a 16-bit number from a register ($r1) to data memory. The data memory address is the value in a register ($r2)
move	move $r1, $r2	copy the value of one register ($r2) to another register ($r1)

addi	addi $r1, $r2, x	add a register number ($r2) and an immediate number (x), and store the result in one register ($r1)
andi	andi $r1, $r2, x	bit-wise logical AND of a register number ($r2) and an immediate number x, and store the result in a register ($r1)
ori	ori $r1, $r2, x	bit-wise logical OR of a register number ($r2) and an immediate number x, and store the result in a register ($r1)
ble	ble $r1, $r2, x	if the first register number ($r1) is smaller than or equal to the second register number ($r2), then jump to address **(PC + x)**.
bne	bne $r1, $r2, x	if the first register number ($r1) does not equal the second register number ($r2), then jump to address **(PC + x)**.
jump	jump x	Jump to address (PC + x).
call	call x	Jump to address (PC + x), where x is an immediate number, and at the same time the address of the next instruction to a special register $ra ($ra is not one of the 8 general registers $r1 -- r8).
rtn	rtn	Jump to address stored in $ra.
halt	halt	all registers (including PC, the 8 general purpose registers and all other registers) in the processor are disabled (so the processor halts).

You should design the instruction set for the processor to fulfill the following requirements:

- Let X = the sum of all the student ID members in your group
  - Example: if the group has three members, and the Student ID number are 58451111, 58502222, 58891234, then
    X = 58451111 + 58502222 + 58891234 = 175844567
    (If your group has only 2 members, then just X simply equals the sum of the 2 members. Similar for groups with only 1 member.)
- Let Y = X mod 7
  - Example: if X=175844567, then
    Y= 175844567 mod 7 = 3
- For **R-type** instructions, the format is defined as follows
  - Bits (Y~Y+2): write (destination) register
  - Bits (Y+3~Y+5): read register 1
  - Bits (Y+6~Y+8): read register 2
  - All the other bits are used for Opcode. (Note: in your design, the instructions should distinguish from each other only based on the Opcode, which is different from the MIPS R-type instructions that distinguish from each other by both Opcode and other bit fields such as *funct* and *shamt*).
- For other types of instructions, you can design the format freely as long as it works correctly.

Write the binary machine code of the following testing assembly programs, store the binaries in memory images, and execute the testing programs with your processor built in Logisim. **You are encouraged to test your programs with more testing programs designed by yourself to assure the correctness of your design.**

Test Program 1:

```
.text
        li $r1, 1
        li $r2, 2
        li $r3, 10
        add $r2, $r1, $r2
        ble $r2, $r3, -1 #jump to the previous instruction if the condition is satisfied
        halt
```

Test Program 2:

```
.text
        li $r1, 6
        li $r2, 5
        andi $r3, $r1, 3
        ori $r4, $r3, 8
        halt
```

Test Program 3:

```
.text
        li $r1, 6
        li $r2, 5
        and $r3, $r1, $r2
        li $r8, 0
        store $r3, $r8
        or $r4, $r1, $r2
        li $r8, 1
        store $r4, $r8
        li $r8, 1
        load $r7, $r8
        halt
```

Test Program 4:

```
.text
        li $r1, 6
        li $r2, 4
        call 7
        move $r4, $r3
        li $r1, 7
        li $r2, 8
        call 3
        move $r5, $r3
        jump 3
        add $r3, $r1, $r2
        rtn
        halt
```

Bonus Question:

Build a processor that fulfills all the requirements above, while using a five-stage pipeline and resolving the control hazards and data hazards properly:

1. *IF:* instruction fetch stage
2. *ID:* instruction decode & register file read stage
3. *EX:* execution stage
4. *MEM:* memory access stage
5. *WB:* write-back stage

Your pipelined processor should use the same instruction set as in the above question, and be able to execute the above testing programs using the same memory images.