Homework 4

T1

Recall the machine busy example from Example 2.11 in Section 2.6.7. Assuming the BUSYNESS bit vector is stored in R2, we can use the LC-3 instruction `0101 011 010 1 00001` (`AND R3, R2, #1`) to determine whether machine 0 is busy or not. If the result of this instruction is 0, then machine 0 is busy.

1. Write an LC-3 instruction that determines whether machine 2 is busy.
2. Write an LC-3 instruction that determines whether both machines 2 and 3 are busy.
3. Write an LC-3 instruction that determines whether all of the machines are busy.
4. Can you write an LC-3 instruction that determines whether machine 6 is busy? Is there a problem here?

T2

Suppose the following LC-3 program is loaded into memory starting at location x30FF.

Address	Value
x30FF	1110 0010 0000 0001
x3100	0110 0100 0100 0010
x3101	1111 0000 0010 0101
x3102	0001 0100 0100 0001
x3103	0001 0100 1000 0010

If the program is executed, what is the value in R2 at the end of execution?

T3

The LC-3 ISA contains the instruction `LDR DR, BaseR, offset`. After the instruction is decoded, the following operations (called microinstructions) are carried out to complete the processing of the `LDR` instruction:

```
MAR <- BaseR + SEXT(offset6) ; set up the memory address
MDR <- Memory[MAR] ; read mem at BaseR + offset
DR <- MDR ; load DR
```

Suppose that the architect of the LC-3 wanted to include an instruction `MOVE DR, SR` that would copy the memory location with address given by `SR` and store it into the memory location whose address is in `DR`.

1. The `MOVE` instruction is not really necessary since it can be accomplished with a sequence of existing LC-3 instructions. What sequence of existing LC-3 instructions implements (also called "emulates") `MOVE R0,R1`? (You may assume that no other registers store important values.)
2. If the `MOVE` instruction were added to the LC-3 ISA, what sequence of microinstructions, following the decode operation, would emulate `MOVE DR,SR`?

T4

The LC-3 does not have an opcode for **XOR**, so we're required to write instructions to implement the XOR operation by ourselves. Assume that the reserved opcode `1101` is implemented as OR instruction, which shares the same format as AND instruction.

The following instructions will store the value of (R1 XOR R2) to R3 (`XOR R3, R1, R2`). Fill in the two missing instructions to complete the program. You are only allowed to use the registers R1, R2, R3, and R4.

Address	Instruction
x3000	1001 100 001 111111
x3001	
x3002	1001 011 010 111111
x3003	
x3004	1101 011 011 000 100

T5

List five addressing modes in LC3. Given instructions ADD, NOT, LEA,LDR and JMP, categorize them into operate instructions, data movement instructions, or control instructions. For each instruction mentioned above, list addressing modes that can be used.

T6

1. Write a **single** LC3 assembly instruction that copies the content of `R5` to `R4`.
2. Write a **single** LC3 assembly instruction that clears the content of `R3`. (i.e. `R3 = 0`)
3. Write **3** LC3 assembly instructions that does `R1=R6-R7`.
   - You are ONLY allowed to change the value of `R1`.
   - You may assume that the initial value of `R1` is 0.
4. Write **3** LC3 assembly instructions that multiply the value at label `DATA` by 2. (`Mem[DATA] = Mem[DATA] * 2`)
   - You are ONLY allowed to change the value of `R1`.
   - You don't need to restore or clear the value of the register you used.
   - No need to consider overflow.
5. Set condition codes based on the value of `R1` using only **one** LC-3 instruction.
   - You are not allowed to change any value in the registers.

T7

If the current PC points to the address of an `JMP` instruction, how many memory accesses are required for the LC-3 to process that instruction? What about `ADD` and `LDI` instructions?

T8

The content in PC is x3010. The content of the following memory unit is as follows:

Address	Value
x304E	x70A4
x304F	x70A3
x3050	x70A2
x70A2	x70A4
x70A3	x70A3
x70A4	x70A2

1. After the execution of the following code, What is the value stored in `R6`?

Address	Value
x3010	1110 0110 0011 1110
x3011	0110 1000 1100 0001
x3012	0110 1111 0000 0001
x3013	0110 1101 1111 1111

2. Can you use one `LEA` instruction to do the same task as the three instructions above do? (Only consider loading value into `R6`.)

T9

After the execution of the following code, the value stored in `R0` is 12. Please speculate what the value stored in `R5` is like.

Address	Value
x3000	0101 0000 0010 0000
x3001	0101 1111 1110 0000
x3002	0001 1100 0010 0001
x3003	0001 1101 1000 0110
x3004	0101 1001 0100 0110
x3005	0000 0100 0000 0001
x3006	0001 0000 0010 0011
x3007	0001 1111 1110 0010
x3008	0001 0011 1111 0010
x3009	0000 1001 1111 1001
x300A	0101 1111 1110 0000

T10

`R0` and `R1` contain 16-bit bit vectors. The program below determines if rotating `R1` left by n bits produces the same bit vector that is in `R0`. If yes, the program stores the value n in

M[x3020]. If not, the program stores -1 to M[x3020].

Rotating left a bit vector by one bit consists of left shifting the bit vector one bit, and then loading into bit[0] the bit that was shifted out of bit[15].

For example, rotating left **111**1000011110000 by 3 bits produces 1000011110000**111**.

Your job: Complete the program below by supplying the missing instructions so it stores n in location M[x3020] if rotating left `R1` by n bits produces the bit vector in `R0`, and store -1 if it is not possible to produce the bit vector of `R0` by rotating left `R1`. You are required to only use four registers: `R0`, `R1`, `R2`, and `R3`.

> Hint: The highest bit determines whether a 2's complement is positive or negative.

Address	Value
x3000	1001 000 000 111111
x3001	0001 000 000 1 00001
x3002	0101 010 010 1 00000
x3003	0001 011 000 0 00 001
x3004	
x3005	0001 010 010 1 00001
x3006	
x3007	0000 010 000000111
x3008	0101 001 001 1 11111
x3009	0000 100 000000010
x300A	0001 001 001 0 00 001
x300B	0000 111 111110111
x300C	
x300D	
x300E	0000 111 111110100
x300F	0101 010 010 1 00000
x3010	0001 010 010 1 11111
x3011	
x3012	1111 0000 0010 0101