CS 252 Homework 5 Solutions

1) The addressability of a memory is the number of bits in each location. So if a memory's addressability is 32 bits, then each location is 32 bits long. The MDR has the same size as the memory locations, so the MDR is **32 bits long**. The size of the MAR is determined by the number of locations, which is not mentioned here, so we cannot determine the size of the MAR.
2)
    a. We need a unique address for each memory location, so we nee $\log_2 256$ = **8 bits** of address.
    b. Since PC-offsets are 2's complement, to specify a location which is +/- 15 locations away from current instruction, we need **5 bits**.
    c. When the current instruction is at location 3, the PC would have value 4 (since PC points to the next instruction to execute). So the PC-relative offset of location 12 would be (12-4) = **8**.
3) If there are 32 registers, each of DR,SR1,SR2 field in the add instruction would be 5 bits long, so the operands would take 5 + 5 + 5 = 15 bits to specify. Combined with 4 bits for the opcode, the total length of the add instruction would be 4 + 15 = 19 bits long, but LC3 instructions cannot be more than16 bits long.
4)
    a. There are several ways of doing it. In all the examples below we want to move contents of R3 to R1
        i. Using ***A* AND *A* = *A***, we can write **R1 <- R3 AND R3**
        ii. Using add-immediate, we can write **R1 <- R3 + 0**
    b. Using and-immediate, we can write **R3 <- R3 AND 0**
    c. Consider the add-immediate instruction **R1 <- R1 + 0**. The contents of R1 are not changed. But the condition codes 'n','z','p' are set according to the result of the add instruction which is nothing but the contents of R1 itself.
    d. As seen earlier, subtraction can be converted to addition by negating the second operand. So first we calculate the 2's complement of R2 and then add it to R3.

i. **R2 <- NOT R2**

ii. **R2 <- R2 + 1**

iii. **R1 <- R3 + R2**

e. **No**. Every instruction which changes the condition codes changes all the three bits. And at any one time, only one of the three bits can be '1'. So you cannot have two bits set to '1' at the same time.

5) Let us decode the instructions

Location	Instruction	Decoded instruction
0x4400	1001 011 001 111111	R3 <- NOT R1
0x4401	0001 011 011 100001	R3 <- R3 + 1
0x4402	0001 011 010 000011	R3 <- R2 + R3
0x4403	0000 100 001 000000	BR[n=1] to PC + 0x40

So if the sequence of operations caused the last branch statement to execute, the 'n' condition code must have been '1' , which means that the result of the 3rd instruction was negative. Since R3 is the 2's complement of R1, the 3rd instruction effectively was R3 <- R2 – R1. So if R2 – R1 was negative, we can conclude **R2 was less than R1**.

6) Consider each instruction

a. 0001 001 001 1 00000 => R1 <- R1 + 0. No register content is modified by this instruction, but the condition codes are modified. So this **cannot be used as NOP**.

b. 0000 000 000000010 => This is the branch instruction, but it branches only when all 'n','z','p' are 0, which cannot happen. So this is a branch which never executes so **can be used as a NOP**.

c. 0000 111 000000001 => This is the branch instruction, which branches when either of 'n','z','p' are 1, which is always the case. So this is an unconditional branch. But this **cannot be used as NOP** because the branch would skip over one instruction (PC already points to next instruction and this would move PC by one more).
The only thing ADD does different is that it modifies the condition codes.

7) This is by demorgan's law: A OR B = NOT(NOT A AND NOT B)

a. 1001 011 001 111111 => R3 <- NOT R1

b. **1001 101 010 111111 => R5 <- NOT R2**

c. 0101 110 011 000 101 => R6 <- R3 AND R5

d. **1001 100 110 111111 => R4 <- NOT R6**

8) Lets consider each instruction one by one
   a. 1110 0110 0011 1111 => LEA: R3 <- PC + offset. Here offset is 0x3F. Since PC not points to 0x3011, R3 will have value 0x3011 + 0x3F = 0x3050.
   b. 0110 1000 1100 0000 => LDR: R4 <- M[R3+0]. So R4 will get memory value at address 0x3050 which is 0x70A4.
   c. 0110 1011 0000 0000 => LDR: R5 <- M[R4+0]. So R5 will get memory value at address 0x70A4 which is 0xABCD.

   So the final value at R5 is **0xABCD**.

   We can replace this by a single instruction located at 0x3010 by using indirect addressing. The operation which we carried out was to load M[M[PC + offset]] into R5, which can be done using indirect addressing in one instruction.

   LDI R5, 0x3F => 1010 1010 0011 1111

9)