# ICS hw4

howlinggggg

October 29, 2022

**T1.** (5.1)List five addressing modes. Given instructions ADD, JMP, LEA, LDR and NOT, identify whether the instructions are operate instructions, data movement instructions, or control instructions. For each instruction, list the addressing modes that can be used with the instruction.

**answewer.**

1. immediate
2. register
3. PC-relative
4. indirect
5. Base+offset

- ADD: operate instruction, 1, 2.
- JMP: control instruction, 2.
- LEA: data movement, 1.
- LDR: data movement, 5.
- NOT: operate instruction, 2.

**T2.** (5.4)Say we have a memory consisting of 256 locations, and each location contains 16 bits.

a. How many bits are required for the address?

b. If we use the PC-relative addressing mode, and want to allow control transfer between instructions 20 locations away, how many bits of a branch instruction are needed to specify the PC-relative offset?

c. If a control instruction is in location 3, what is the PC-relative offset of address 10? Assume that the control transfer instructions work the same way as in the LC-3.

**answewer.**

a. $\lceil \log_2 256 \rceil = 8$ bits.

b. $\lceil \log_2 2 \times 20 \rceil = 6$ bits.

c. $10 - 3 - 1 = 6$, 000110.

**T3.** (5.6)Recall the machine busy example from 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.

a. Write an LC-3 instruction that determines whether machine 2 is busy.

b. Write an LC-3 instruction that determines whether both machines 2 and 3 are busy.

c. Write an LC-3 instruction that indicates none of the machines are busy.

d. Can you write an LC-3 instruction that determines whether machine 6 is busy? Is there a problem here?

**ansewer.**

a. 0101 011 010 1 00100 (`AND R3, R2, #4`).

$$\begin{array}{rr} & \text{xxxx xxx0} \\ \text{AND} & \text{0000 0001} \\ \hline & \text{0000 0000} \end{array}$$

b. 0101 011 010 1 01100 (`AND R3, R2, #12`).

$$\begin{array}{rr} & \text{xxxx 00xx} \\ \text{AND} & \text{0000 1100} \\ \hline & \text{0000 0000} \end{array}$$

c. 0101 011 010 1 11111 (`AND R3, R2, #-1`).

$$\begin{array}{rr} & \text{xxxx xxxx} \\ \text{AND} & \text{1111 1111} \\ \hline & \text{xxxx xxxx} \end{array}$$

d. Can't. Because the immediate can only present $-16 \sim 15$

$$\begin{array}{rr} & \text{x0xx xxxx} \\ \text{AND} & \text{0100 0000} \\ \hline & \text{0000 0000} \end{array}$$

**T4.** (5.9)We would like to have an instruction that does nothing. Many ISAs actually have an opcode devoted to doing nothing. It is usually called NOP, for NO OPERATION. The instruction is fetched, decoded, and executed. The execution phase is to do nothing! Which of the following three instructions could be used for NOP and have the program still work correctly?

a. 0001 001 001 1 00000

b. 0000 111 000000001

c. 0000 000 000000000

What does the ADD instruction do that the others do not do?

**answewer.** c could be used for NOP. The ADD instruction actually load the word from register and store to the register after execute "ADD".

**T5.** (5.13)

a. How might one use a single LC-3 instruction to move the value in R2 into R3?

b. The LC-3 has no subtract instruction. How could one perform the following operation using only three LC-3 instructions:
`R1 <- R2 - R3`

c. Using only one LC-3 instruction and without changing the contents of any register, how might one set the condition codes based on the value that resides in R1?

d. Is there a sequence of LC-3 instructions that will cause the condition codes at the end of the sequence to be N = 1, Z = 1, and P = 0? Explain.

e. Write an LC-3 instruction that clears the contents of R2.

**answewer.**

a. 0001 011 010 1 00000 `ADD, R3, R2, #0`

b.

```
1001  011  011  111111  ;NOT(R3)
0001  011  011  1  00001  ;R3 = R3 + 1
0001  001  010  0  00  011  ;R1 = R2 + R3
```

c. 0001 001 001 1 00000 `ADD, R1, R1, #0`

d. No. Only one bit will be set as 1, other bits set to 0.

e. 0101 010 010 1 00000 `AND, R2, R2, 0x00`

**T6.** (5.14)The LC-3 does not have an opcode for the logical function XOR. That is, there is no instruction in the LC-3 ISA that performs the XOR operation. However, we can write a sequence of instructions to implement the XOR operation. **Assume that the reserved instruction 1101 is OR instruction, its addressing mode is the same as AND.**

The following five-instruction sequence performs the XOR of the contents of register 1 and register 2 and puts the result in register 3.

Fill in the two missing instructions so that the five-instruction sequence will do the job.

(1): 1001 100 001 111111

(2):

(3): 1001 101 010 111111

(4):

(5): 1101 011 100 000 101

**ansewer.**

```
1001 100 001 111111 ;R4 = NOT(R1)
0101 100 100 0 00 010 ;R4 = R4 AND R2
1001 101 010 111111 ;R5 = NOT(R2)
0101 101 101 0 00 001 ;R5 = R5 AND R1
1101 011 100 000 101 ;R3 = R4 OR R5
```

**T7.** (5.15)State the contents of R1, R2, R3, and R4 after the program starting at location x3100 halts.

| Address | Data |
|---|---|
| 0011 0001 0000 0000 | 1110 001 000100000 |
| 0011 0001 0000 0001 | 0010 010 000100000 |
| 0011 0001 0000 0010 | 1010 011 000100000 |
| 0011 0001 0000 0011 | 0110 100 010 000001 |
| 0011 0001 0000 0100 | 1111 0000 0010 0101 |
| : | : |
| 0011 0001 0010 0010 | 0100 0101 0110 0110 |
| 0011 0001 0010 0011 | 0100 0101 0110 0111 |
| : | : |
| 0100 0101 0110 0111 | 1010 1011 1100 1101 |
| 0100 0101 0110 1000 | 1111 1110 1101 0011 |

**ansewer.** R1 $\leftarrow$ x3100+1+32 = 0011 0001 0010 0001 (x3121)
R2 $\leftarrow$ mem[x3101 + 1 + 32] = mem[x3122] = 0100 0101 0110 0110
R3 $\leftarrow$ mem[mem[x3102 + 1 + 32]] = mem[mem[x3123]] = 1010 1011 1100 1101
R4 $\leftarrow$ mem [R2 + 1] = mem[0100 0101 0110 0111] = 1010 1011 1100 1101

**T8.** (5.17)How many times does the LC-3 make a read or write request to memory during the processing of the LD instruction?
How many times during the processing of the LDI instruction? How many times during the processing of the LEA instruction?
Also, indicate what phases the instructions don't need. Processing includes all phases of the instruction cycle.

**ansewer.** LD: DR = mem[$PC^+$ + PCoffset9], 2 times, don't need the phase EXECUTE.
LDI: DR = mem[mem[$PC^+$ + PCoffset9]], 2 times, don't need the phase EXECUTE.
LEA: DR = $PC^+$ + PCoffset9, 0 time, don't need the phases FETCH OPERAND and EXECUTE.

**T9.** Suppose the following LC-3 program is loaded into memory starting at location `x30FF`:

```
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?

**ansewer.**

```
LEA R1, #1
LDR R2, R1, #2
TRAP
```

R1 = x30FF + 1 + 1 = x3101
R2 = mem[R1 + 2] = 0001 0100 1000 0010

**T10.** If the value stored in R0 is 5 at the end of the execution of the following instructions, what can be inferred about R5?

```
x2FFF  0101  0000  0010  0000
x3000  0101  1111  1110  0000
x3001  0001  1101  1110  0001
x3002  0101  1001  0100  0110
x3003  0000  0100  0000  0001
x3004  0001  0000  0010  0001
x3005  0001  1101  1000  0110
x3006  0001  1111  1110  0001
x3007  0001  0011  1111  1000
x3008  0000  1001  1111  1001
x3009  0101  1111  1110  0000
```

**ansewer.** R5 = xxxx xxxx 1111 1000

```
x2FFF AND R0, R0, #0   ;R0 = 0
x3000 AND R7, R7, #0   ;R7 = 0
x3001 ADD R6, R7, #1   ;R6 = R7 + 1 = 1
x3002 AND R4, R5, R6   ;R4 = R5 AND R6
x3003 BR Z, #1
x3004 ADD R0, R0, #1   ;R0 += 1
x3005 ADD R6, R6, R6   ;R6 += R6
x3006 ADD R7, R7, #1   ;R7 += 1
x3007 ADD R1, R7, #-8  ;R1 = R7 - 8
x3008 BR N, #-7        ;PC = PC+1 +(-7) = x3002
x3009 AND R7, R7, #0   ;R7 = 0
```

**T11.** The LC-3 macho-company has decided to use opcode 1101 to implement a new instruction. They need your help to pick the most useful one from the following:

a. MOVE Ri, Rj; The contents of Rj are copied into Ri.

b. NAND Ri, Rj, Rk; Ri is the bit-wise NAND of Rj, Rk

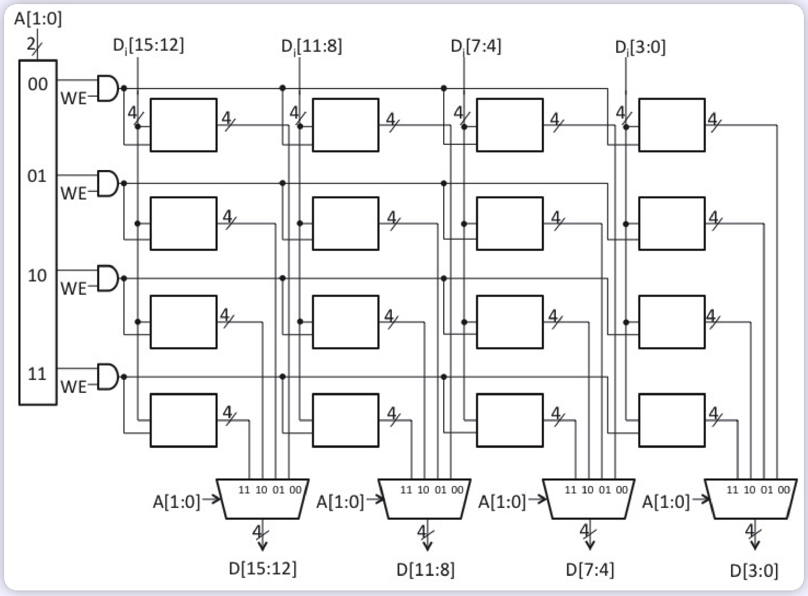
c. SHFL Ri, Rj, #2; The contents of Rj are shifted left 2 bits and stored into Ri.

d. MUL Ri, Rj, Rk; Ri is the product of 2's complement integers in Rj, Rk.

Justify your answer.

**ansewer.** c.
Instruction b. can be implemented by composed of AND and NOT. Instruction a. and d. can be implemented by c.

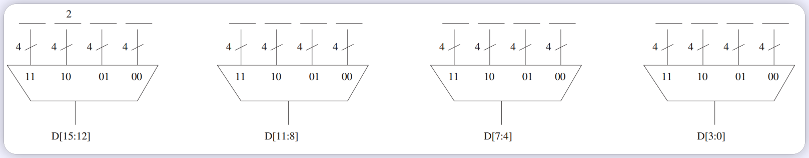
**T12.** The following diagram describes a 22 by 16-bit memory. Each of the four muxes has four-bit input sources and a four-bit output, and each four-bit source is the output of a single four-bit memory cell.



a. Unfortunately, the memory was wired by a student, and he got the inputs to some of the muxes mixed up. That is, instead of the four bits from a memory cell going to the correct four-bit input of the mux, the four bits all went to one of the other four-bit sources of that mux. The result was, as you can imagine, a mess. To figure out the mix-up in the wiring, the following sequence of memory accesses was performed:

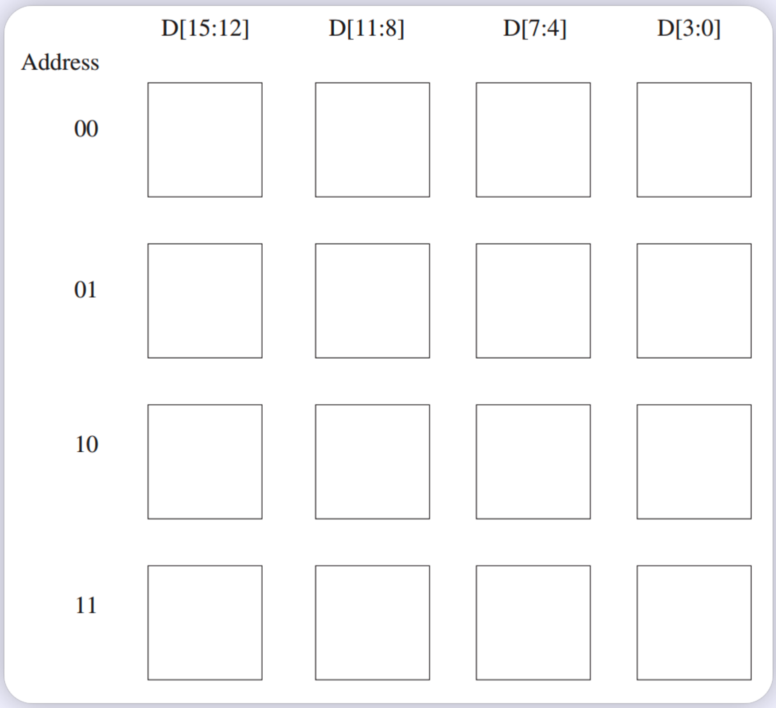
| Read/Write | MDR | MAR |
|---|---|---|
| Write | x134B | 01 |
| Write | xFCA2 | 10 |
| Write | xBEEF | 11 |
| Write | x072A | 00 |
| Read | xF34F | 10 |
| Read | x1CAB | 01 |
| Read | x0E2A | 00 |

Note: On a write, MDR is loaded before the access. On a read, MDR is loaded as a result of the access. Your job is to identify the mix-up in the wiring. Show which memory cells were wired to which mux inputs by filling in their corresponding addresses in the blanks provided. Note that one address has already been supplied for you.



b. After rewiring the muxes correctly and initializing all memory cells to **xF**, the following sequence of accesses was performed. Note that some of the information about each access has been left out. Your job: Fill in the blanks.

Show the contents of the memory cells by putting the hex digit that is stored in each after all the accesses have been performed.



| Address | D[15:12] | D[11:8] | D[7:4] | D[3:0] |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 10 | | | | |
| 11 | | | | |

**ansewer.**