# Assignment 4 for COMP 2280
## Due: 11:30 PM on Wednesday April 13th, 2022
**Total Marks: 46**

**Notes**:
1. For the written questions show all of your work and submit a single PDF file.
2. For programming questions hand in your source program (.asm file).
3. Name your programs LastnameFirstnameAxQy.asm, replace x with the assignment number and y with the question number.
4. Please comment appropriately for programming question. Please read the "programming standards" for COMP 2280. Marks are allocated for good documentation.
5. Hand in your assignment through the UMLearn

# Written Part

# Question 1 (Module 10) [8 marks]

The MAR, MDR, and ALU are structures written (and read) in various phases of the instruction processing cycle, depending on the instruction being executed.

- In each table cell, below, enter the instructions that result in **writes/loads** to the corresponding structure (row) during the corresponding phase (column) of the instruction processing cycle.
- In the case of the ALU, also indicate the phases where it performs a **calculation**, on top of **writes/loads** to it.
- **Ignore the reads** or loads from the MAR/MDR/ALU onto the **bus**.
- For example, place ST in the MAR row of the FETCH column if ST causes the MAR to be written during the FETCH phase.
- To make this simpler, let us only consider the following instructions:

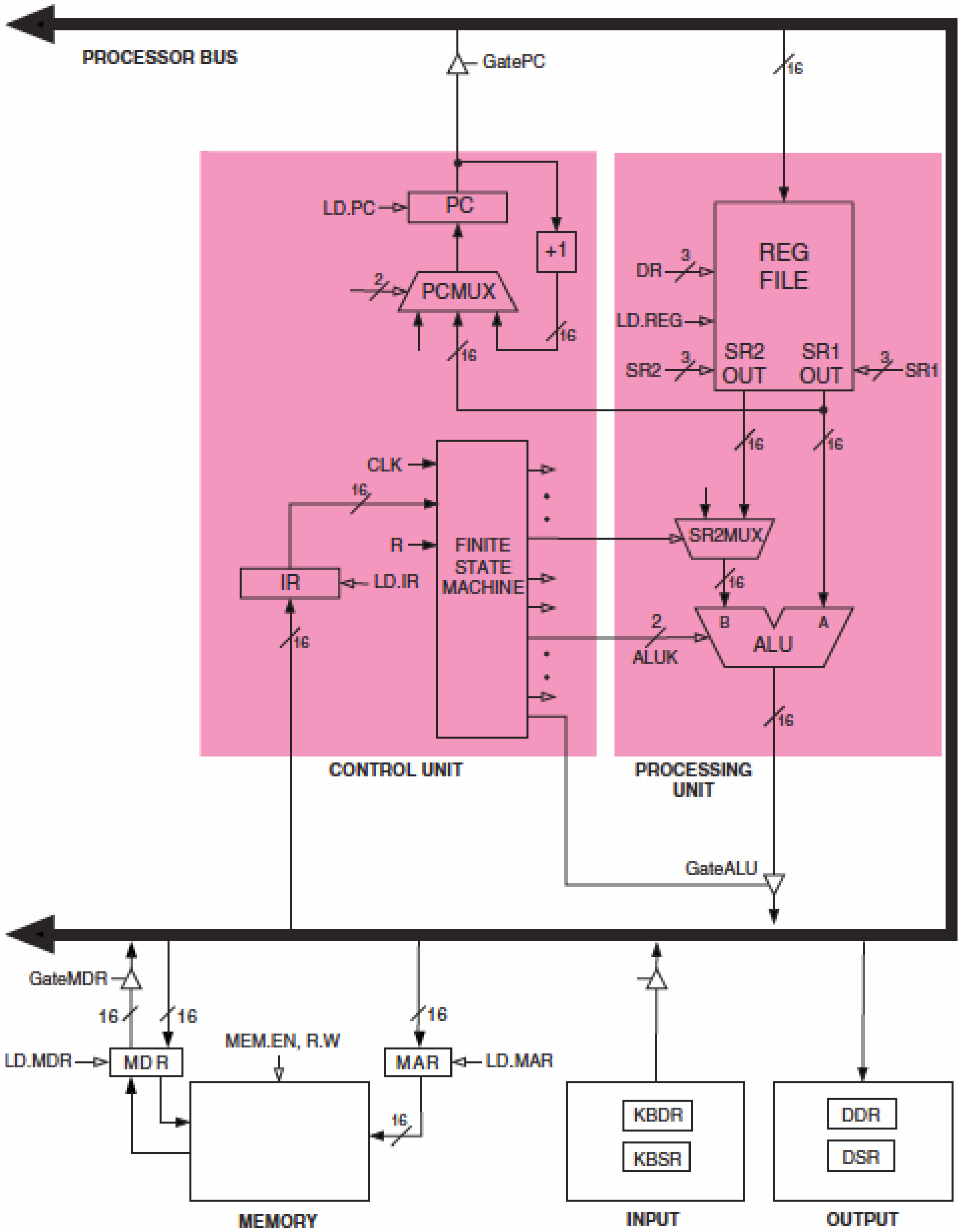
| AND | LDR | ST |
|---|---|---|

| | FETCH | DECODE | EVAL ADDR | FETCH OPR | EXECUTE | STORE |
|---|---|---|---|---|---|---|
| **MAR** | AND<br>LDR<br>ST | | LDR<br>ST | | | |
| **MDR** | AND<br>LDR<br>ST | | | LDR | | ST |
| **ALU** | | | LDR<br>ST | AND | AND | |

# Assignment 4 for COMP 2280

## Due: 11:30 PM on Wednesday April 13th, 2022

**Total Marks: 46**



# Assignment 4 for COMP 2280

## Due: 11:30 PM on Wednesday April 13th, 2022

**Total Marks: 46**

## Question 2 [8 marks]

## a) Make a State Transition Table

- Using 3 state bits, design a finite state machine that will count out the sequence 0,2,5,7,4,6,3,1,0 … on each rising clock edge.
- Your FSM will have a clock input and 3 state/output bits $Q_2Q_1Q_0$ which is the binary representation of the current number counted. **Aka: The state is also the output**
- Your solution must include the truth tables for the next state given the current state (state transition table)
- Your solution must include the state equations (in sum-of-products form) for the state variables $Q_{2+}$, $Q_{1+}$, and $Q_{0+}$ based on the state transition table (see the two examples from the Module 9).
- Simplify the expressions in the equations where possible. Use a K-maps for this.
- **Do NOT draw a circuit diagram**

| $Q_2$ | $Q_1$ | $Q_0$ | $Q_2+$ | $Q_1+$ | $Q_0+$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

| $Q_2 \backslash Q_1 Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

$Q_0+ = Q_1Q_2' + Q_0'Q_1 + Q_0Q_1'Q_2$

Highlighting is not required

| $Q_2 \backslash Q_1 Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

$Q_1+ = Q_0'Q_2 + Q_0'Q_1' + Q_1'Q_2$

Highlighting is not required

| $Q_2 \backslash Q_1 Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

$Q_2+ = Q_1'Q_2 + Q_0Q_2 + Q_0'Q_1Q_2'$

Highlighting is not required

# Assignment 4 for COMP 2280

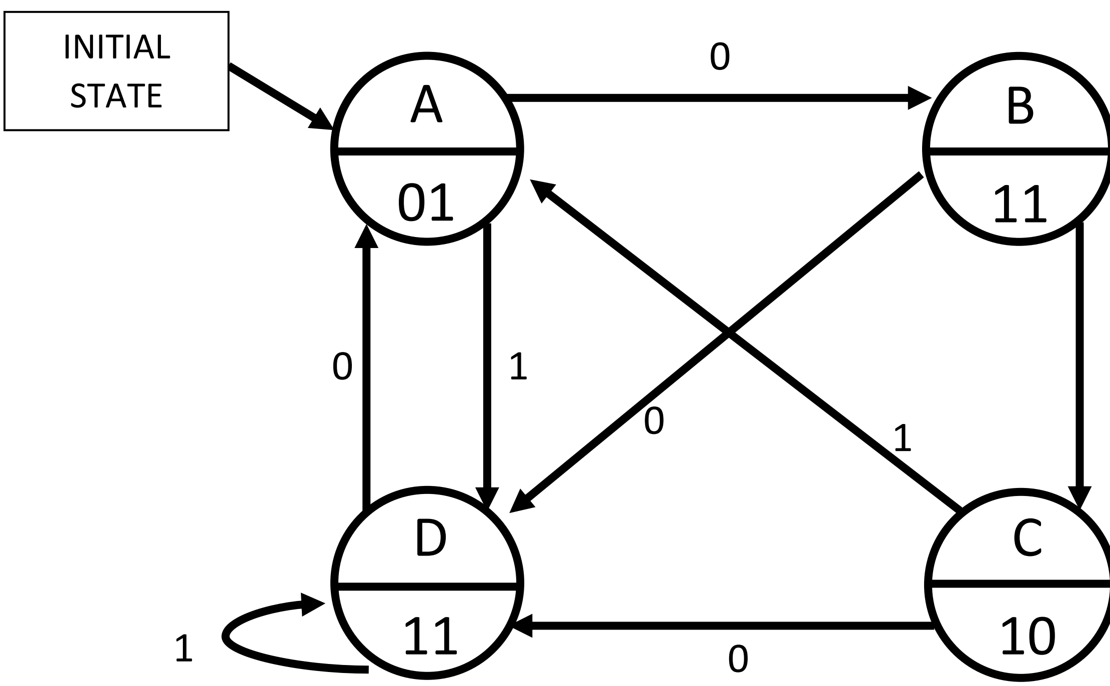
## Due: 11:30 PM on Wednesday April 13th, 2022

**Total Marks: 46**

## Question 3

**a) [9 marks]** Consider the following state machine transition diagram.

- It has a one-bit input X and a two-bit output YZ.
- Fill out the state transition table below.
- Your solution must include the truth tables for the next state and the output, given the current state and the current value of the input (state transition table)
- Your solution must include the equations for the state variables and the output based on the state transition table.
- Simplify the expressions in the equations where possible, XOR/XNOR included.
- Use the "standard" assignment of the state variables, that is, the initial state A = 00, B = 01, etc., this will give the simplest equations.
- In the state diagram the value under the line is the value of the output Y.
- **Remember: the output is based on the current state.**



# Assignment 4 for COMP 2280
## Due: 11:30 PM on Wednesday April 13th, 2022
**Total Marks: 46**

| In: X | Current State | $Q_1$ | $Q_0$ | Out: Y | Out: Z | Next State | $Q_1$+ | $Q_0$+ |
|---|---|---|---|---|---|---|---|---|
| 0 | A | 0 | 0 | 0 | 1 | B | 0 | 1 |
| 0 | B | 0 | 1 | 1 | 1 | D | 1 | 1 |
| 0 | C | 1 | 0 | 1 | 0 | D | 1 | 1 |
| 0 | D | 1 | 1 | 1 | 1 | A | 0 | 0 |
| 1 | A | 0 | 0 | 0 | 1 | D | 1 | 1 |
| 1 | B | 0 | 1 | 1 | 1 | C | 1 | 0 |
| 1 | C | 1 | 0 | 1 | 0 | A | 0 | 0 |
| 1 | D | 1 | 1 | 1 | 1 | D | 1 | 1 |

| X \ $Q_1 Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

$Q_1$'($Q_0$ xor X) + $Q_1$($Q_0$ xnor X) can be simplified to ($Q_0$ xor X xor $Q_1$)
($Q_0$ xnor X) is also equivalent to ($Q_0$' xor X) or ($Q_0$ xnor X')
Second term can be $Q_0$'X' or $Q_0$'$Q_1$' or $Q_1$'X'
EDIT: Do not take off points if they didn't do the XOR

$Q_0$+ = ($Q_0$ xor X xor $Q_1$) + ($Q_0$'X' or $Q_0$'$Q_1$' or $Q_1$'X')

| X \ $Q_1 Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

$Q_1$'($Q_0$ xor X) + $Q_1$($Q_0$ xnor X) can be simplified to ($Q_0$ xor X xor $Q_1$)
($Q_0$ xnor X) is also equivalent to ($Q_0$' xor X) or ($Q_0$ xnor X')
Second term can be $Q_0$X or $Q_0$'$Q_1$ or $Q_1$'X
EDIT: Do not take off points if they didn't do the XOR

$Q_1$+ = ($Q_0$ xor X xor $Q_1$)+ ($Q_0$X or $Q_0$'$Q_1$ or $Q_1$'X)

| X \ $Q_1 Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |

Y = $Q_0$ + $Q_1$

| X \ $Q_1 Q_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

Z = $Q_0$ + $Q_1$'

# Assignment 4 for COMP 2280

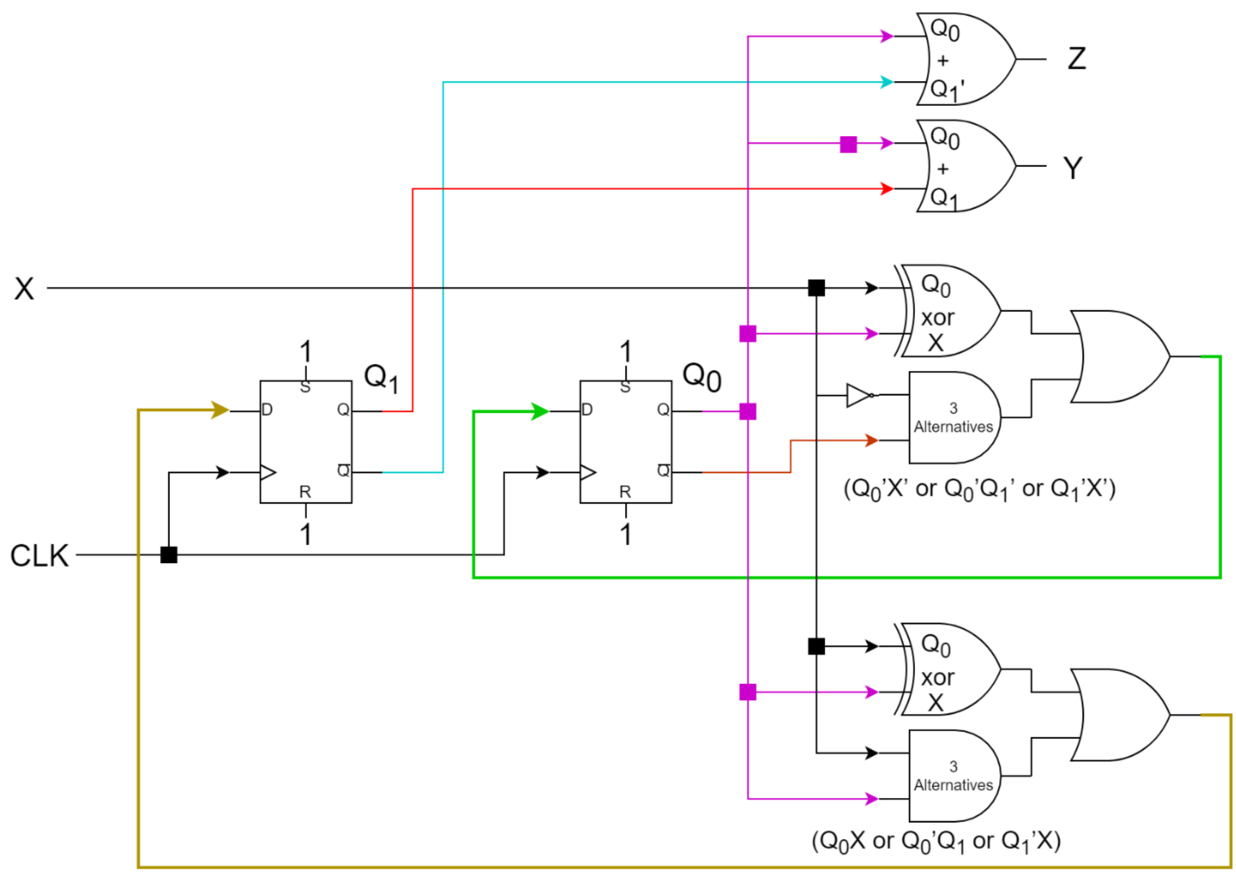
## Due: 11:30 PM on Wednesday April 13th, 2022

**Total Marks: 46**

## 3.b) [5 marks]

Design and Draw a circuit for the state machine from 3.a.

**Note:** Make sure it is clear and readable, using Draw.io can be useful for this.



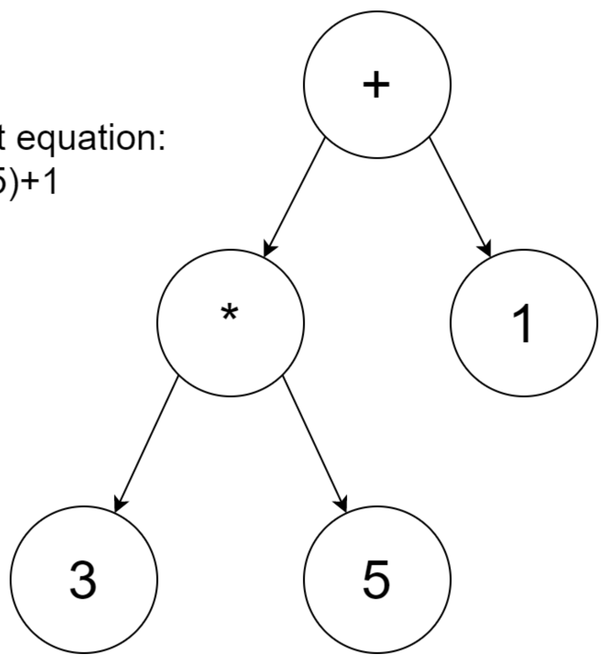
# 4) [16 marks]

Write a complete assembly language program for the LC-3 computer that evaluates an expression tree.

- An expression tree is a binary tree, with a data field and left and right pointers referred to as the left and right address fields.
- Therefore a node in the expression tree consists of three fields. Each field occupies one word of memory.
  - The first word/field is either an operator code or a 16-bit 2's complement integer.
  - The second word/field is the address of the left sub-tree.
  - The third word/field is the address of the right sub-tree.

| Operator Code | Operation |
|---|---|
| 0 | Multiply the value of right sub-tree by the value of the left sub-tree. |
| 1 | Add the value of right sub-tree to the value of the left sub-tree. |
| -1 | Subtract the value of right sub-tree from the value of the left subtree. |

Equivalent equation:
(3*5)+1




In memory:

```
root .fill 1 ; +
     .fill op1
     .fill L3
op1 .fill 0 ; *
     .fill L1
     .fill L2
L1   .fill 3 ; operand
     .fill 0
     .fill 0
L2   .fill 5 ; operand
     .fill 0
     .fill 0
L3   .fill 1 ; operand
     .fill 0
     .fill 0
```

Write a recursive subroutine named ***evaluate*** that is passed the address of an expression tree, evaluates the expression tree and returns the value of the expression tree.

- To evaluate an expression tree:
  - if the address of the tree is null (0)
    - return zero
  - if the address of the tree points to an operand (leaf node)
    - return the value in the node (first word / field of the node)

  - if the address of the tree points to node for an operator
    - evaluate the left sub-tree (recursive call)
    - evaluate the right sub-tree (recursive call)
    - apply the operator to the values of the two sub-trees
    - return the result of applying the operator to the operands
- To evaluate a sub-tree ***evaluate*** must call itself, this is the recursive part of the program.
  - A leaf node has null left and right address fields.
  - An operator node has left and right address fields that are not null.
  - Checking one of the address fields is sufficient to determine the type of a node.
- The main program must:
  - Establish the stack pointer,
  - Call evaluate with the address of root defined below
  - and store the result returned by evaluate at a memory location named result.
- End your program by displaying a termination message that includes your name.
- Test your program with the expression tree defined on the next page.

**Note:**
One of the operations to be performed is multiplication, but the LC-3 computer does not have a multiply instruction.

- You could write a subprogram to do the multiplication or you may download the subroutine named ***multiply8bits*** from Desire2Learn (UM Learn).
- ***multiply8bits*** is given two 8-bit 2's complement numbers and returns the product of those numbers.
- See the documentation in the subroutine for more details on using ***multiply8bits***.
- This subroutine gives the correct answer for operands that can be represented as 8-bit 2's complement numbers or, if the operands are 16-bits, any product that can be represented as a 16-bit 2's complement number.
- This is sufficient for this program.

```
root  .fill 0 ; *
      .fill op1
      .fill op3
op1   .fill 1 ; +
      .fill L1
      .fill op2
L1    .fill 20 ; operand
      .fill 0
      .fill 0
op2   .fill 0 ; *
      .fill L2
      .fill L3
L2    .fill 3 ; operand
      .fill 0
      .fill 0
L3    .fill -4 ; operand
      .fill 0
      .fill 0
op3   .fill -1 ; -
      .fill op4
      .fill L4
L4    .fill 5 ; operand
      .fill 0
      .fill 0
op4   .fill 1 ; +
      .fill L5
      .fill L6
L5    .fill 4 ; operand
      .fill 0
      .fill 0
L6    .fill 3 ; operand
      .fill 0
      .fill 0
```