Kennesaw State University (KSU)
College of Computing and Software Engineering (CCSE)
CS-3503 Computer Organization & Architecture

Lab-2&3: A 5-bit Digital Memory & Computer Design

Pre-lab (Required)

- Getting ready – NI Multisim

Multisim is industry standard SPICE simulation and circuit design software for analog, digital, and power electronics in education and research.

1. Please use the web link below to **install it on your** Windows OS based computer (Mac OS and Linux distributions cannot install it).
   https://www.ni.com/en/support/downloads/software-products/download.multisim.html

Create a student account and download and install (needs Windows 10 64-bit or use a VM such as VMWare Horizon Client from KSU on Mac) from the link above. If the KSU does not provide a Multisim license (you can check that in your student profile of UITS Software OnTheHub WebStore or here: https://kennesaw.onthehub.com/WebStore/ProductsByMajorVersionList.aspx), then a trial version of 45 days is available for free from NI.

Any KSU student could use both the NI Multisim 14 and MS Visual Studio 2022 via:

2. the CCSE Tutoring Center at Room J-263 has **KSU computers** which have native installations, or
3. the CCSE Virtual Desktop on https://cseview.kennesaw.edu/ to install **VMWare** Horizon Client, the instructions to use this are provided in a separate document on the D2L.
4. You may also use NI Multisim® online at https://www.multisim.com . Do not forget to save your file/work locally or on their cloud, for which you need to make an account, for later D2L submission and screenshots creation for each lab.

Helpful Online Resources (Optional)

- Getting started – Multisim related videos (all links work when this document is posted, but a video may have been removed later on and hence, some link may not be functional later on):

Multisim: Brief Introduction: https://www.youtube.com/watch?v=B8EwjRDZa6k
Multisim: Basic Logic Gates: https://www.youtube.com/watch?v=rflzjtZrBec
A Digital Circuit: https://www.youtube.com/watch?v=jwvM5Zt9kkc
XOR: https://www.youtube.com/watch?v=dl8a1heSk1U
NAND: https://www.youtube.com/watch?v=lm0CdSSQiAQ
Decoder: https://www.youtube.com/watch?v=7rhQwNeEc4o
MUX: https://www.youtube.com/watch?v=JyB7R8ToW3Y
Half-Adder: https://www.youtube.com/watch?v=ijIDYFePnrw
Full Adder: https://www.youtube.com/watch?v=bgRvQas0ZHI&t=2s
DFF: https://www.youtube.com/watch?v=_hX2_q8qyTs
Shift Register: https://www.youtube.com/watch?v=BdliATa-uEQ
Counter: https://www.youtube.com/watch?v=XYVzg-BGD7M

Overall Specifications

In these two labs, you will be individually creating and simulating, utilizing an industry standard digital circuit simulator software Multisim® of the National Instruments (NI®), a stored-program computer (more specifically, von Neumann architecture where program instructions and data share the single memory space, hence the program can be modified during its execution; it would also use the same data and address buses) with the following specifications:

1. This computer will be based on the Load-Store hybridized with the Accumulator (AC register) machine architecture.
   a. The first operand (of any operation involving two input operands) will be copied into the AC by a memory read of appropriate location when a Load (see below) instruction is executed.
   b. An arithmetic or logical (see below) instruction's execution will involve the second operand (of any operation involving two input operands) to be copied into the Memory Buffer Register (MBR) first by a memory read of appropriate location.
   c. The result of an arithmetic or logical operation will be computed through ALU, and temporarily stored in the AC updated right afterwards.
   d. The result of an operation will be copied from the AC to a memory location by memory write operation when a Store (see below) instruction is executed.
2. It will have 5-bit machine architecture, i.e. each CPU instruction/data register and every memory location will be of 5-bit machine word size.
3. There will be 2-bit opcode in each instruction, hence there will be four instructions possible that could be executed by this computer. These instructions mnemonics along with their binary opcode will be as follows: Load=00, Store=01, Add=10, and AND=11.
4. It is going to have a single operand-based Instruction Set Architecture (ISA) that will use Direct Memory Addressing mode only. Hence, the 3-bit operand (could be a label just like MARIE assembly programming language) field in an instruction will be the memory address of the operand directly involved.
5. Please store the following two programs (one at a time and demonstrate it before switching to the other one) at the given binary addresses in the memory of your implementation:
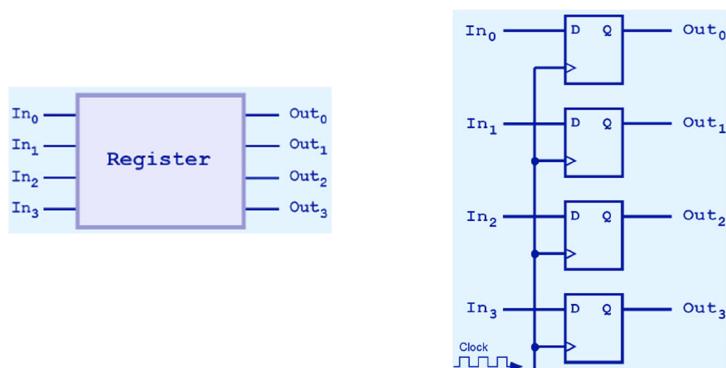
		Program-1	Program-2	
000		Load A	Load A	
001		Add B	AND B	
010		Store C	Store D	
011		00000	00000	(No Operation)
100	A	00111	00111	
101	B	00110	00110	
110	C	00000	00000	
111	D	00000	00000	

Lab-2: A 5-bit Digital Memory Design
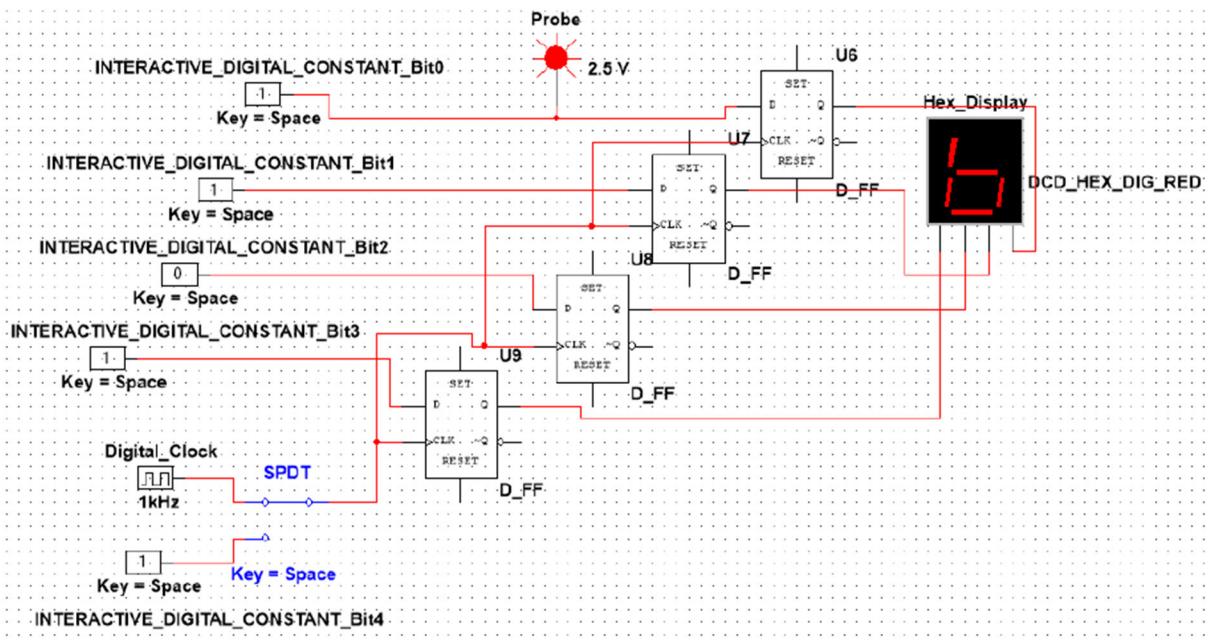
What and how to do?

In this lab, you are individually creating in NI Multisim the following:

I.   A memory with 3-bit addressable (hence, eight locations in total) and 5-bit word size at each location. Each bit of a word is going to be a D-flip-flop. For example, a 4-bit memory location (a register, shown below as well, is nothing but a local memory in the CPU for temporary storage just before an operations execution and hence, is exactly like a memory location) implementation is shown next.
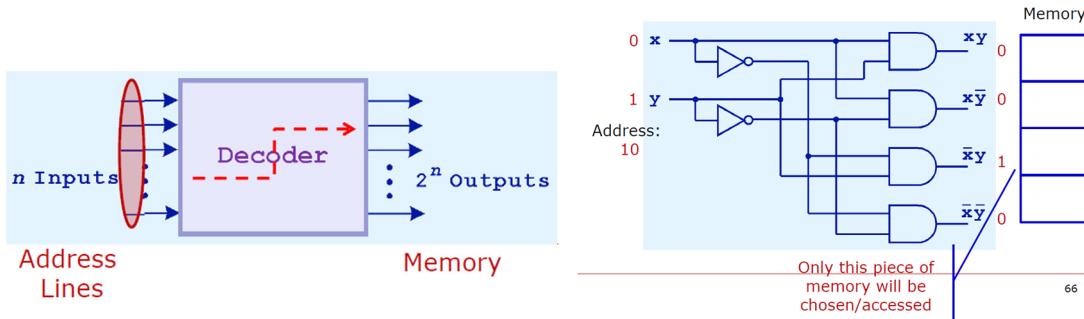
For reference, a Multisim implementation for a 4-bit word storage for a register or a memory location along with input Constants to store data into a memory location, and contents' display at a location using a Probe and DCD Hex Dig are shown next.

Please note the manually controllable switch (SPDT) to select between manually and automatically D-flip flops' update (recall that flip flop memory update happens on the low to high or rising edge signal) provision via Constant and Clock respectively. A bit data input for storage can be fed to a flip-flop using Constant. A nibble (4-bits combined) stored can be shown using a DCD_Hex_Dig_Red display. Hence, you'll need a "Probe" for MSb and "DCD Hex Dig RED" for the Lower nibble of a 5-bit memory-word/register.
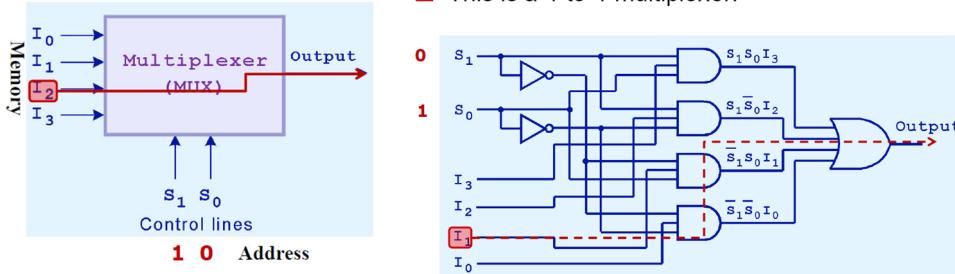
II. A 3-bit (hence, 3-to-8) **Decoder** (shown below to the left) with n=3 lines on the input side (which will be connected later to the Memory Address Register MAR in the CPU), and $2^n=8$ lines on the output of it. Each output line will be a switched-on Minterm at a time to address and **write** a particular memory location once the write-enabled clock signal arrives. An example 2-to-4 decoder circuit implementation is shown below to the right for reference.

This is a 2-to-4 decoder :
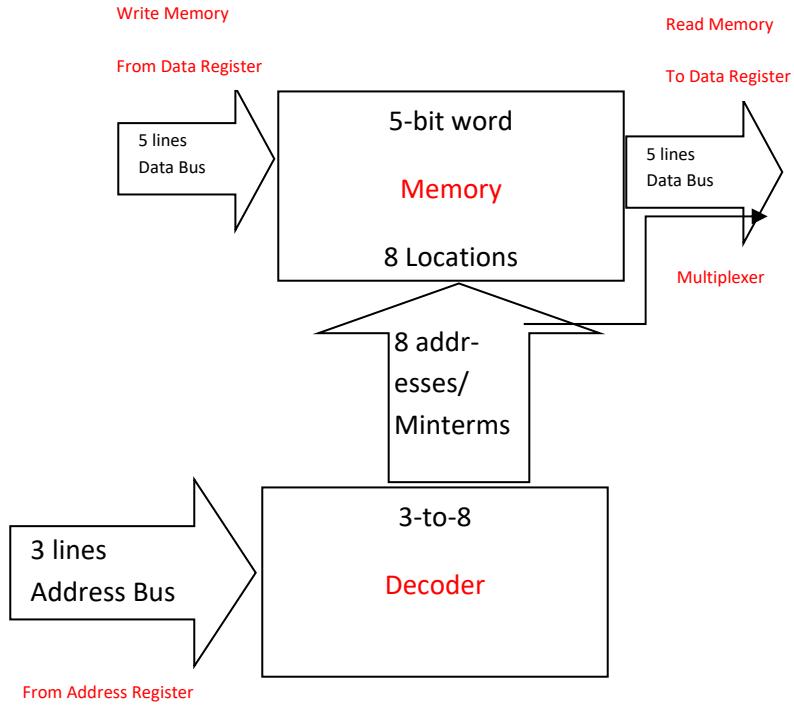
III. A 3-bit (hence, 8-to-1) **Multiplexer** (shown below to the left is a 2-bit 4-to-1 Multiplexer) with n=3 lines on the address/control side (which will be connected later to the Memory Address Register MAR in the CPU), and $2^n=8$ lines on the input of it which will be connected to each memory location one-to-one from first to last memory location. The only output line is a bus (the parallel lines connecting a full word at a memory address to a full data register in the CPU LSb-to-LSb, MSb-to-MSb, and all other bits in the same fashion) to **read** a memory location's contents. An example 4-to-1 multiplexer circuit implementation is shown below to the right for reference.

☐ This is a 4-to-1 multiplexer.
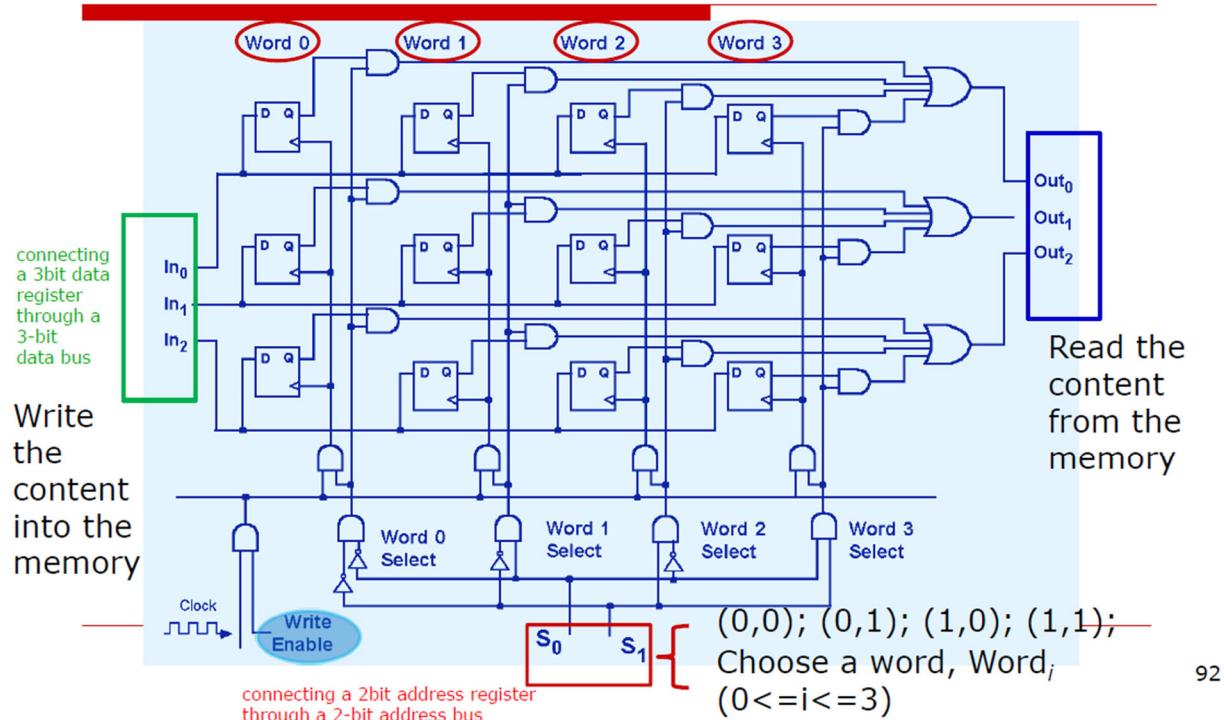
At this time, you should have a completely connected, functional, addressable Memory unit (see below a reference block diagram showing this) which would be capable of letting you read or write a 5-bit data if 3-bit address is given.

Write Memory

From Data Register

5 lines
Data Bus

5-bit word

Memory

8 Locations

Read Memory

To Data Register

5 lines
Data Bus

Multiplexer

8 addr-
esses/
Minterms

3-to-8

Decoder

3 lines
Address Bus

From Address Register

An example 2-bit address and 3-bit Word based Memory unit circuit implementation is shown below for reference.

3.6 4X3 Memory

Word 0   Word 1   Word 2   Word 3

connecting
a 3bit data
register
through a
3-bit
data bus

In$_0$
In$_1$
In$_2$

Out$_0$
Out$_1$
Out$_2$

Read the
content
from the
memory

Write
the
content
into the
memory

Clock

Write
Enable

Word 0
Select

Word 1
Select

Word 2
Select

Word 3
Select

S$_0$ S$_1$

connecting a 2bit address register
through a 2-bit address bus

(0,0); (0,1); (1,0); (1,1);
Choose a word, Word$_i$
($0 <= i <= 3$)

92

WHAT TO SUBMIT?

You will submit (by the deadline on the D2L) **one Multisim file** and **a MS Word/PDF file** that will document **four images** in the order below.
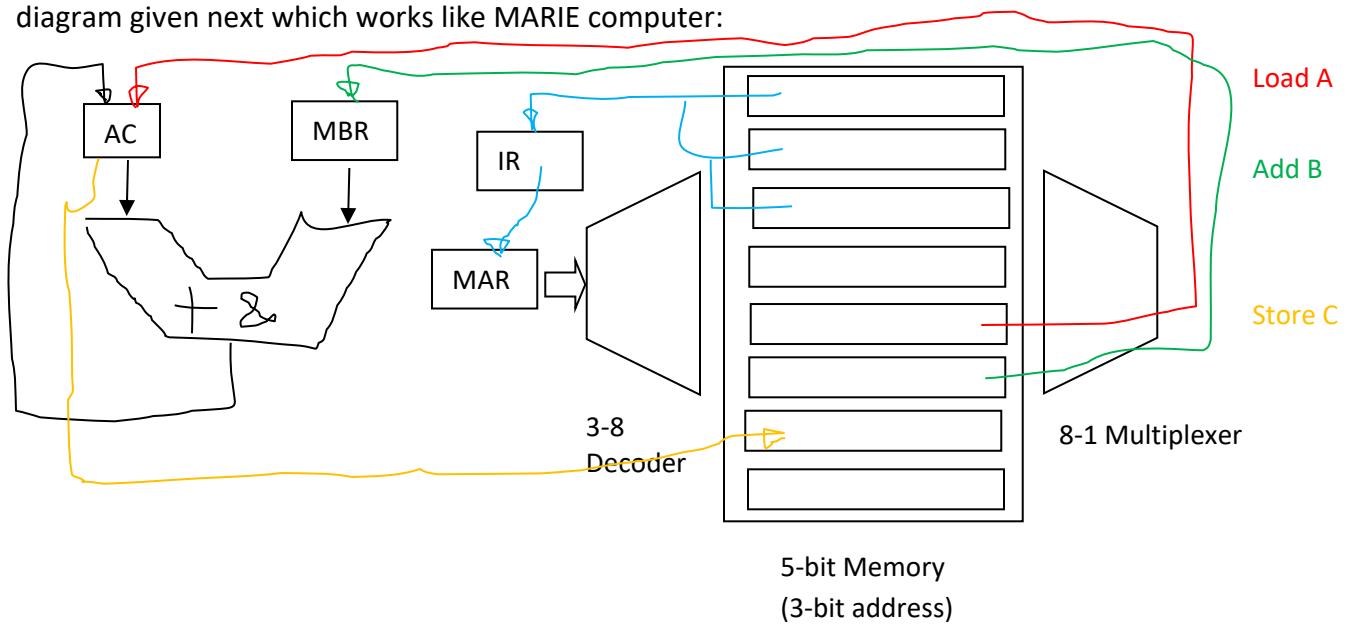
i. A full screenshot of Multisim canvas on which decoder, memory and multiplexer are present as per design specifications above.

ii. A focused screenshot of Multisim canvas on which the decoder is present as per the design specifications above.

iii. A focused screenshot of Multisim canvas on which the multiplexer is present as per the design specifications above.

iv. A focused screenshot of Multisim canvas on which the memory unit is present as per the design specifications above and all memory locations are setup such that they contain Program-1.

Lab-3: A 5-bit Digital Computer Design

What and how to do?
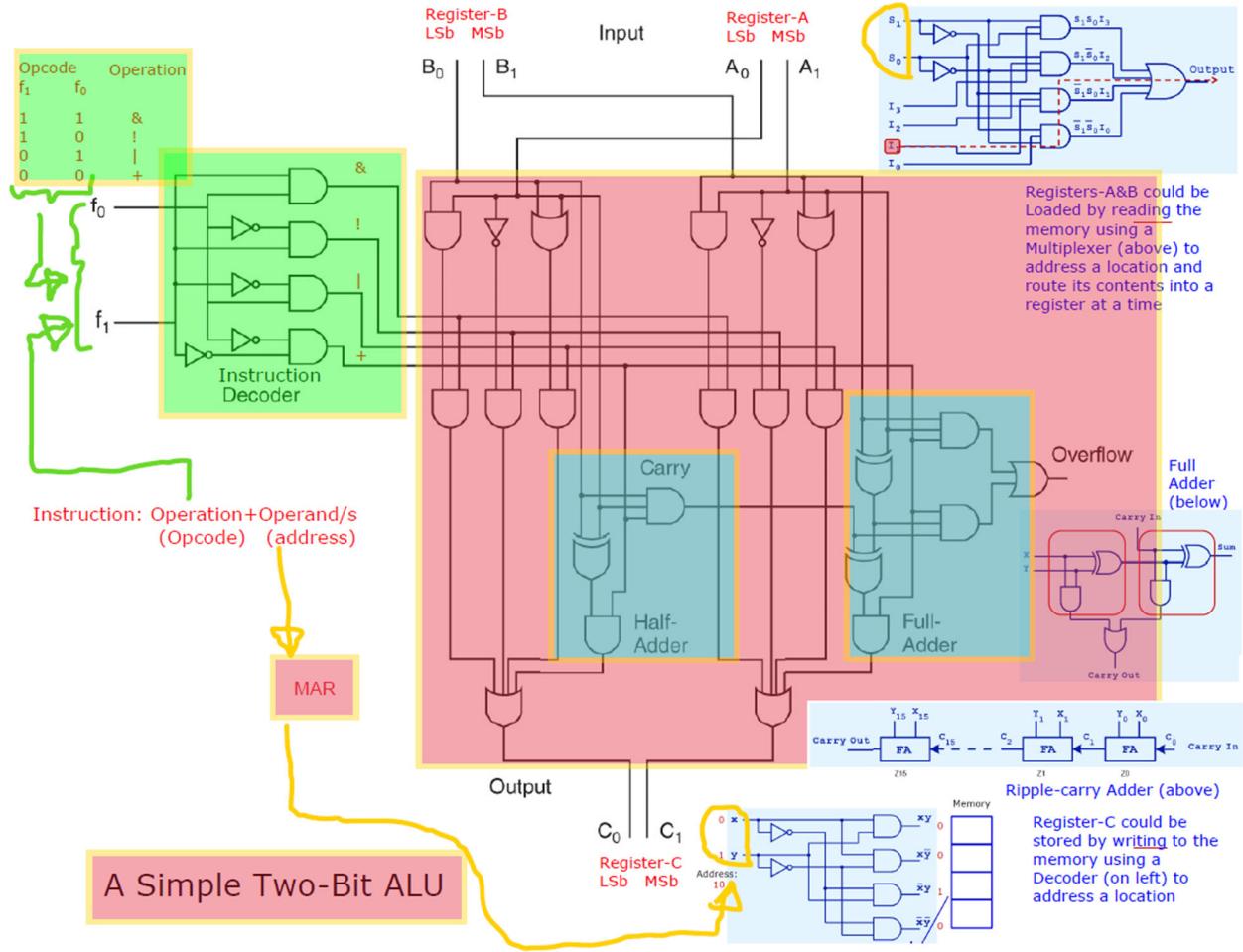
In this lab, you are individually creating in NI Multisim the following:

I.   Please implement the computer specified in the "Overall Specifications" section above and execute Program-1, and then Program-2. You will manually produce the control signals, which the CU will produce automatically otherwise and if it were a complete computer. You will save screenshots of the NI Multisim canvas in the following order: before each program's execution begins, and after each instruction's execution in the order of the instructions of a program.

II.   You will be implementing ALU and Registers of the CPU of this computer only. Your ALU will be capable of performing two instructions, namely Add as an arithmetic instruction and AND as a logical instruction. Your computer will be capable of performing two more instructions as Load and Store. This computer's CPU will have four registers: AC and MBR as 5-bit data registers and IR as a 5-bit instruction register, and a 3-bit Memory Address Register (MAR). The configuration of this computer is shown in the block diagram given next which works like MARIE computer:

Make sure that each register output will be connected to a "Probe" for Most-Significant-bit (MSb) and DCD_Hex _Dig_RED display for the Lower Nibble to show 5-bit contents just like how we did the same in showing each memory location's contents in Lab-2.

The next figure could be helpful in the detailed implementation of the ALU above.

A Simple Two-Bit ALU

Opcode		Operation
f_1	f_0	
1	1	&
1	0	!
0	1		
0	0	+

Instruction Decoder

Register-B
LSb MSb

Input

Register-A
LSb MSb

B_0 B_1 A_0 A_1

f_0
f_1

Registers-A&B could be Loaded by reading the memory using a Multiplexer (above) to address a location and route its contents into a register at a time

Instruction: Operation+Operand/s
(Opcode)   (address)

MAR

Carry
Half-Adder

Full-Adder

Overflow

Full Adder (below)

Carry In
X
Y
Sum
Carry Out

Ripple-carry Adder (above)

Output

C_0 C_1
Register-C
LSb MSb
Address: 10

Memory

Register-C could be stored by writing to the memory using a Decoder (on left) to address a location

WHAT TO SUBMIT?

You will submit (by the deadline on the D2L) **one Multisim file** and **an MS Word/PDF file** that will document **four images** as follows:

You will save screenshots in Word/PDF file of the NI Multisim canvas in the following order: before each program's execution begins, and after each instruction's execution in the order of the instructions of a program.