ECE2330 – Digital Logic Design

Simple Computer System

Learning Objective

After this activity, you will be able to

- Use a schematic capture tool to create a hierarchical design of a simple computer system,
- Verify that the implementation you specify satisfies the requirements, and
- Develop assembly programs to further verify your design

Problem Statement

For this learning activity, you are **required** to develop a hierarchical schematic of a simple computer system, as shown in Figure 1.

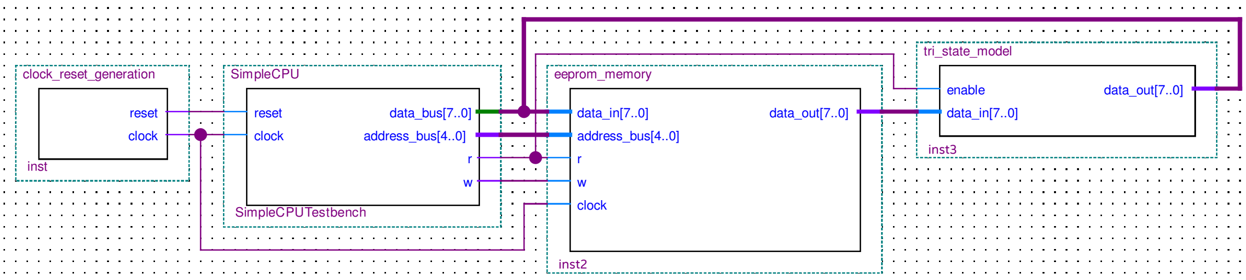

Figure 1: Simple Computer System Schematic

For this design, the **SimpleCPU** component is a hierarchical component that represents the Central Processing Unit (CPU) you have designed. You will verify the correct functionality of your design by loading the program shown in Figure 2 into the EEPROM. The program tests all 8 instructions, tests all components of **SimpleCPU**, and accesses almost every location in program/data memory.

The program is a self-modifying program, in that it modifies the instructions located at Memory Address (MA) 0x13 (dec 25) and 0x15 (inc 25) based on the value of **DataMemoryCounter** = 6 located at MA = 0x19.

The addresses of the two instructions that are modified are initialized with an address equal to **StartOfMemoryAddress** – 1 = 25 (decimal), so that after modification the two instructions will decrement and increment the data at MA = 0x1F. Since this location is initialized with 0x00, the contents will change from 0x00 to 0xFF when decremented and then back to 0x00 when incremented.

Then, the modifications to the two instructions are reversed, **DataMemoryCounter** is decremented and tested for equality to zero.

If it is not equal to zero, the program branches to the beginning and repeats the process for the remaining data locations at MA = 0x1E down to 0x1B. If it is equal to zero, it is re-initialized with the value of **DataMemoryLength** (a constant), branches back to the beginning, and repeats the process.

You will be able to verify the correct design of your simple CPU if the program/data memory is essentially restored to its original state after all 6 data locations have been decremented/incremented and **DataMemoryCounter** has been re-initialized.

Memory Address (hex)	Memory Address (decimal)	Data (Hex)	Instruction	Address (Decimal)	Description
00	0	19	**load**	25	load value of **DataMemoryCounter**
01	1	53	**add**	19	add value of **DataMemoryCounter** to instruction at MA = **FunctionStartAddress**
02	2	33	**store**	19	store modified instruction at MA = **FunctionStartAddress**
03	3	19	**load**	25	load value stored of **DataMemoryCounter**
04	4	55	**add**	21	add value of **DataMemoryCounter** to instruction at MA = **FunctionStartAddress**+2
05	5	35	**store**	21	store modified instruction at MA = **FunctionStartAddress**+2
06	6	D3	**bra**	19	branch to function at MA = **FunctionStartAddress**
07	7	13	**load**	19	load instruction stored at MA = **FunctionStartAddress** (reverse previous instruction effects)
08	8	79	**sub**	25	subtract value of **DataMemoryCounter** from instruction at MA = **FunctionStartAddress**
09	9	33	**store**	19	store modified instruction at MA = **FunctionStartAddress**
0A	10	15	**load**	21	load instruction stored at MA = **FunctionStartAddress**+2
0B	11	79	**sub**	25	subtract value of **DataMemoryCounter** from instruction at MA = **FunctionStartAddress**+2
0C	12	35	**store**	21	store modified instruction at MA = **FunctionStartAddress**+2
0D	13	B9	**dec**	25	decrement **DataMemoryCounter**
0E	14	F0	**beq**	16	if equal to zero, branch to MA = 16 to reset **DataMemoryCounter** with value = **DataMemoryLength**
0F	15	C0	**bra**	0	unconditional branch to MA = 0
10	16	18	**load**	24	load value of **DataMemoryLength**
11	17	39	**store**	25	store value of **DataMemoryCounter**
12	18	C0	**bra**	0	unconditional branch to MA = 0
13	19	B9	**dec**	25	**FunctionStartAddress**: Decrement value at MA (initial value of MA = **StartOfDataMemory** -1)
14	20	F7	**beq**	23	branch to MA = 23 if equal to zero (which should never occur)
15	21	99	**inc**	25	increment value at MA (initial value of MA = **StartOfDataMemory** -1)
16	22	E7	**beq**	7	branch to MA = 7 if equal to zero (which should always occur)
17	23	D7	**bra**	23	infinite loop - should never arrive here
18	24	6	data	6	**DataMemoryLength** (constant)
19	25	6	data	6	**DataMemoryCounter** variable location
1A	26	0	data	0	**StartOfDataMemory**
1B	27	0	data	0	
1C	28	0	data	0	
1D	29	0	data	0	
1E	30	0	data	0	
1F	31	0	data	0	

Figure 2: Simple CPU Test Program

Finally, you are **required** to develop a program to further test your design. The main program calls 4 functions sequentially, and then loops back to the beginning, looping indefinitely. The first function increments memory location 31 (hex = 1F) from 0 to **Limit** using the increment instruction, where **Limit** is a constant value stored as part of your program. The second function decrements memory location 31 from **Limit** to 0 using the decrement instruction.

The third function once again increments memory location 31 from 0 to **Limit**, but the addition instruction is used, and memory location 31 changes by **Delta**, where **Delta** is a constant value stored as part of your program. The fourth function once again decrements from **Limit** to 0 using the subtraction instruction and **Delta**. A flowchart of the required functionality is shown in Figure 3, and simulation results are shown in Figure 4, where **Limit** = 10 and **Delta** = 2.

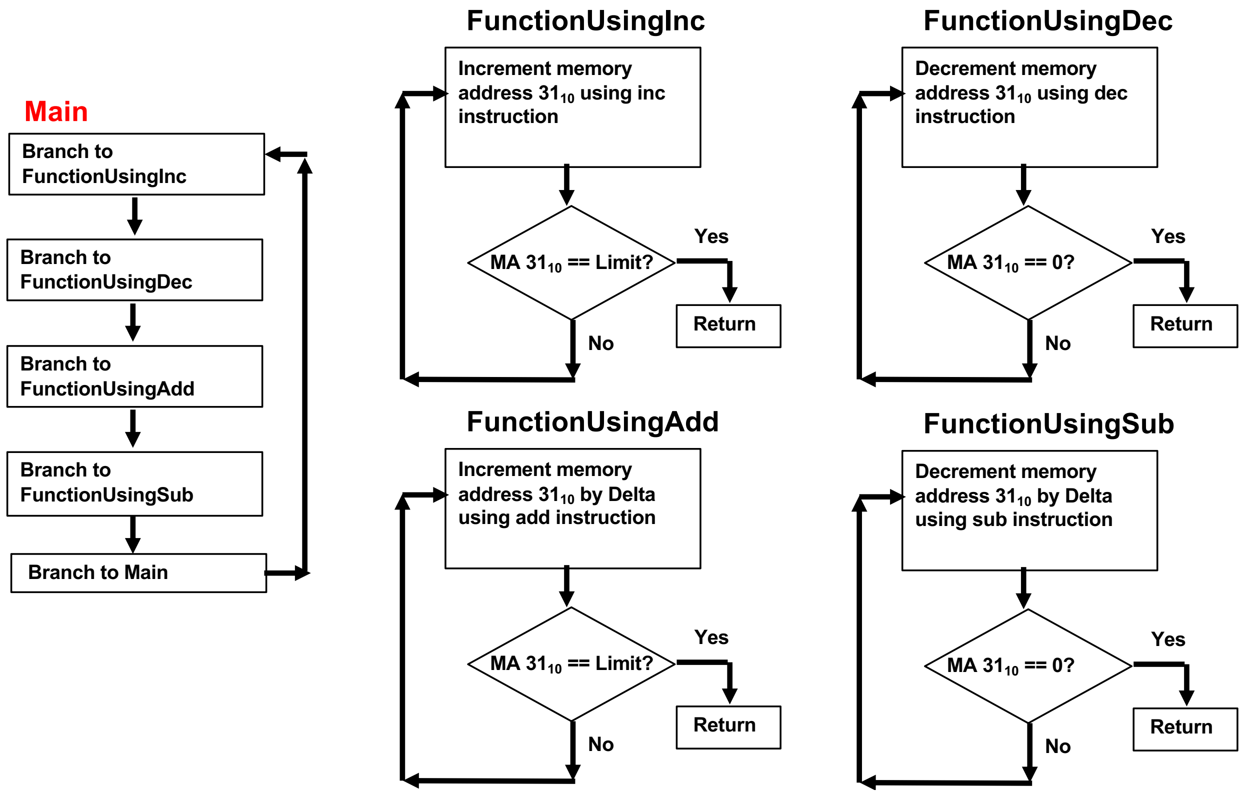

Figure 3: Flowchart of required program.

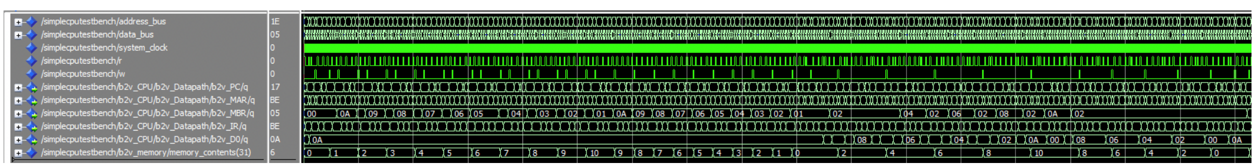

Figure 4: Simulation results.

What To Submit

You will submit **all** VHDL files associated with the CPU design you created for this project. You will also submit an explanation of the program you wrote and simulation results for verification (as a PDF file, annotated appropriately where necessary).

Grading Rubric

This assignment is worth a total of 20 points:

- VHDL files of CPU design
- Required assembly program
- Numerical Verification