Simple Computer System with Memory-Mapped Input/Output

By Sidhardh Burre

CS/ECE 2330 Digital Logic Design

May 3, 2021

Table of Contents

Table of Figures

1 Problem Statement

Design a hierarchical schematic of a simple computer system as shown in Figure 1.

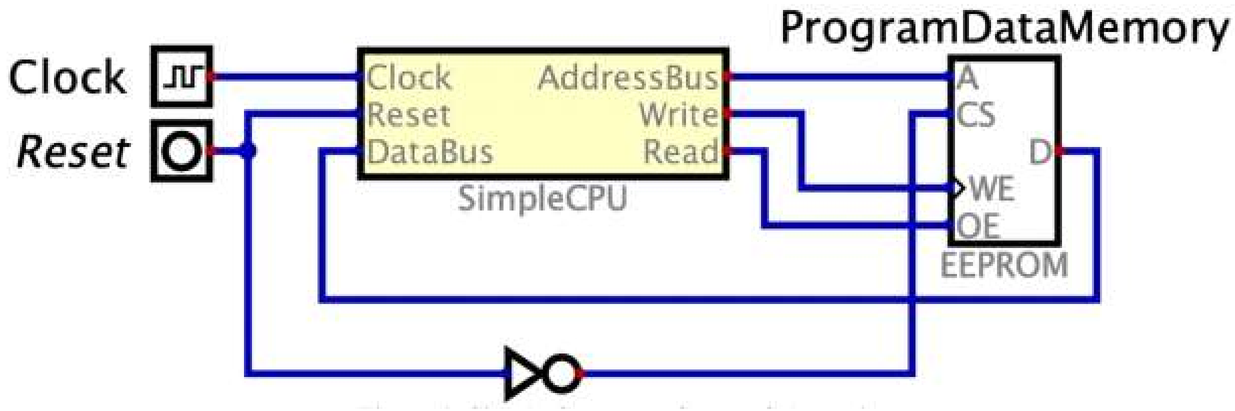
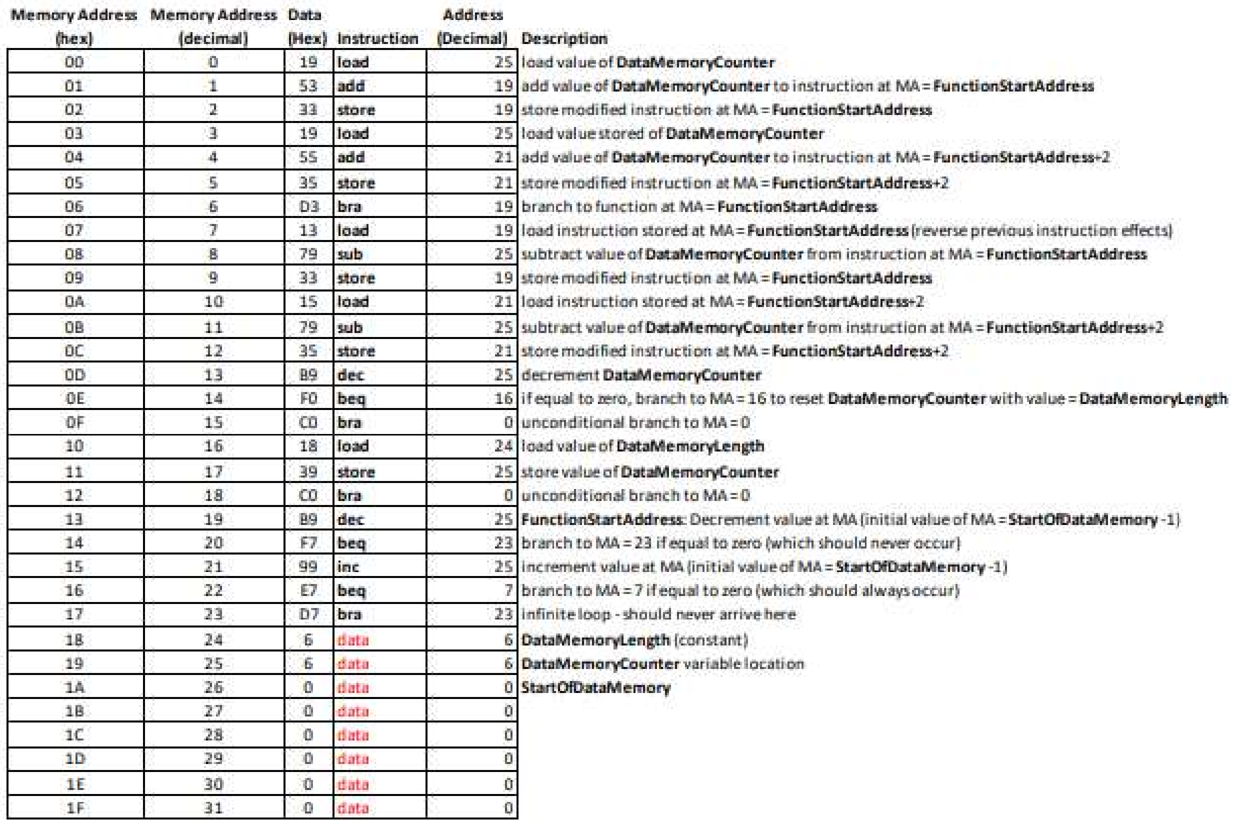


Figure 1: Simple Computer System Schematic

The SimpleCPU is a hierarchical component that represents the Central Processing Unit (CPU) designed previously. The functionality of the design will be demonstrated by loading the program shown in Figure 2 into the EEPROM. The program tests all instructions, components, and memory address.



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

Additionally, circuits will be designed to add a memory-mapped I/O device, as shown in Figure 3. The memory-mapped device is an 8-bit port register that is accessed when MA = 0x1F. The AddressDecoder component decodes the address to determine which memory device to enable with the Chip Select (CS) signals, CS0 and CS1. When the port register is enabled, the decrement/increment operations will display in the two 7-segmenet displays; otherwise, the program operation is the same as described previously.

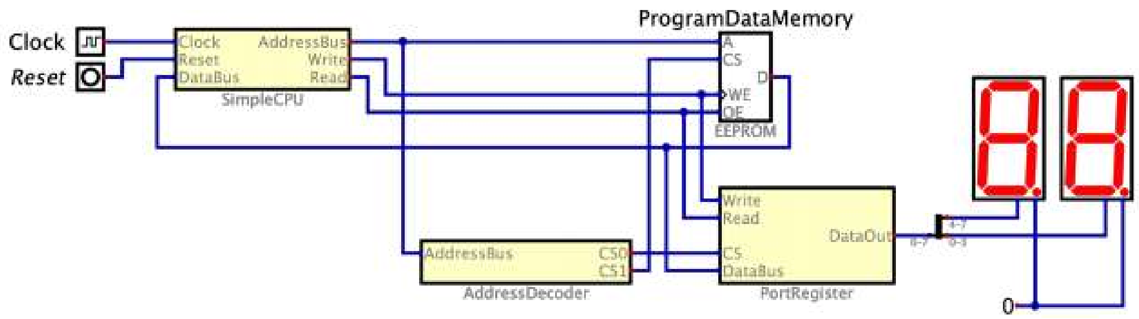


Figure 3: Simple CPU System with Memory-Mapped I/O

2 Analytical Design

Employing the project parameters as guideline, the Simple CPU System both with and without the Memory-Mapped I/O were constructed as shown in Figures 1 and 3. The design for the Simple CPU and its constituents is shown in Figures 4 and 5. The remaining circuits were left as described in previous reports.

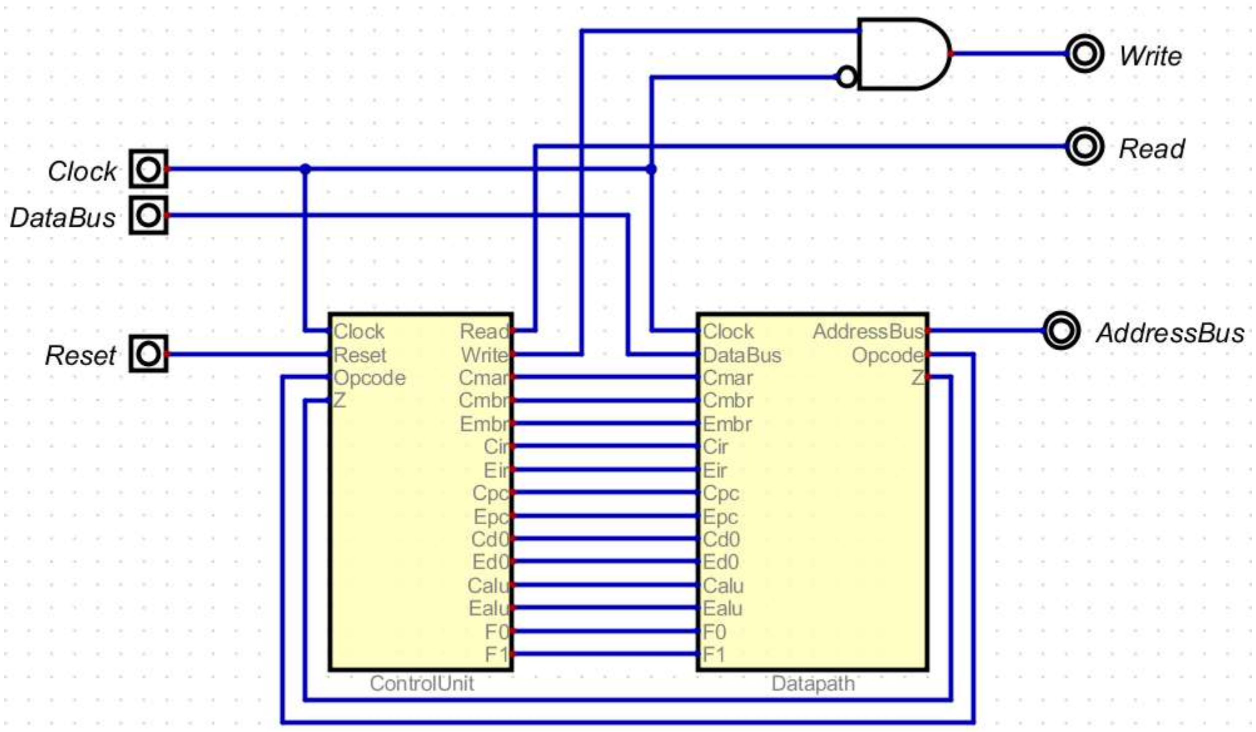


Figure 4: SimpleCPU

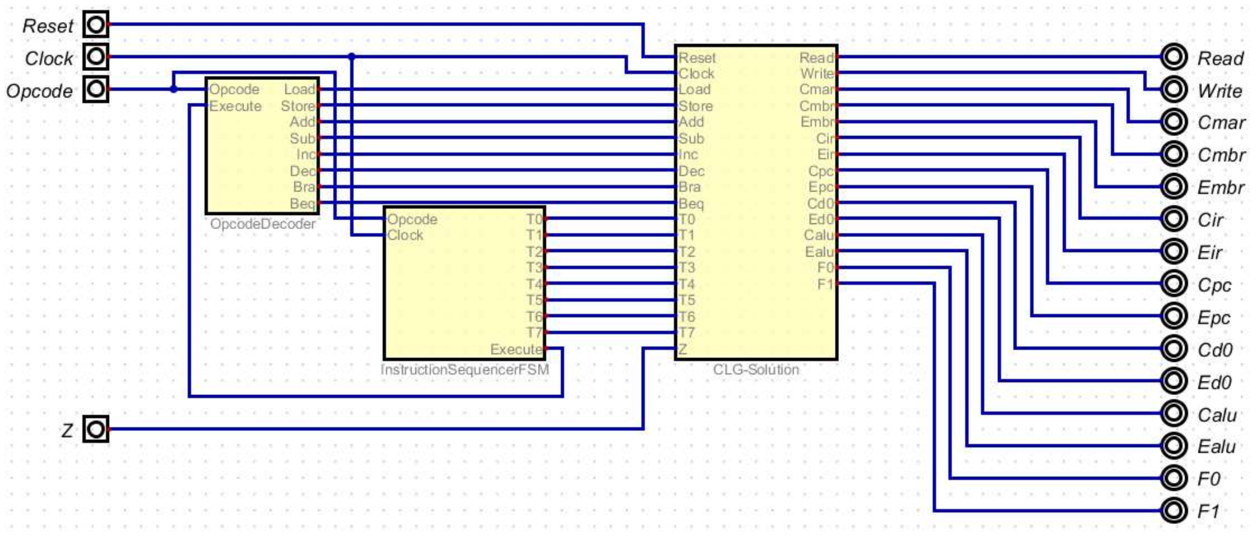


Figure 5: Control Unit

The Address Decoder and Port Register circuits were created exclusively for the Memory-Mapped I/O portion of the design and are pictured in Figures 6 and 7.

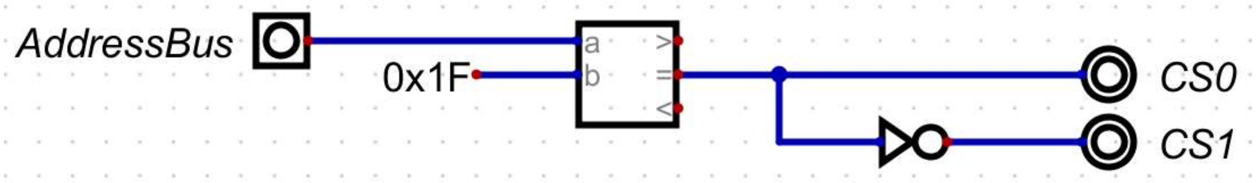


Figure 6: Address Decoder

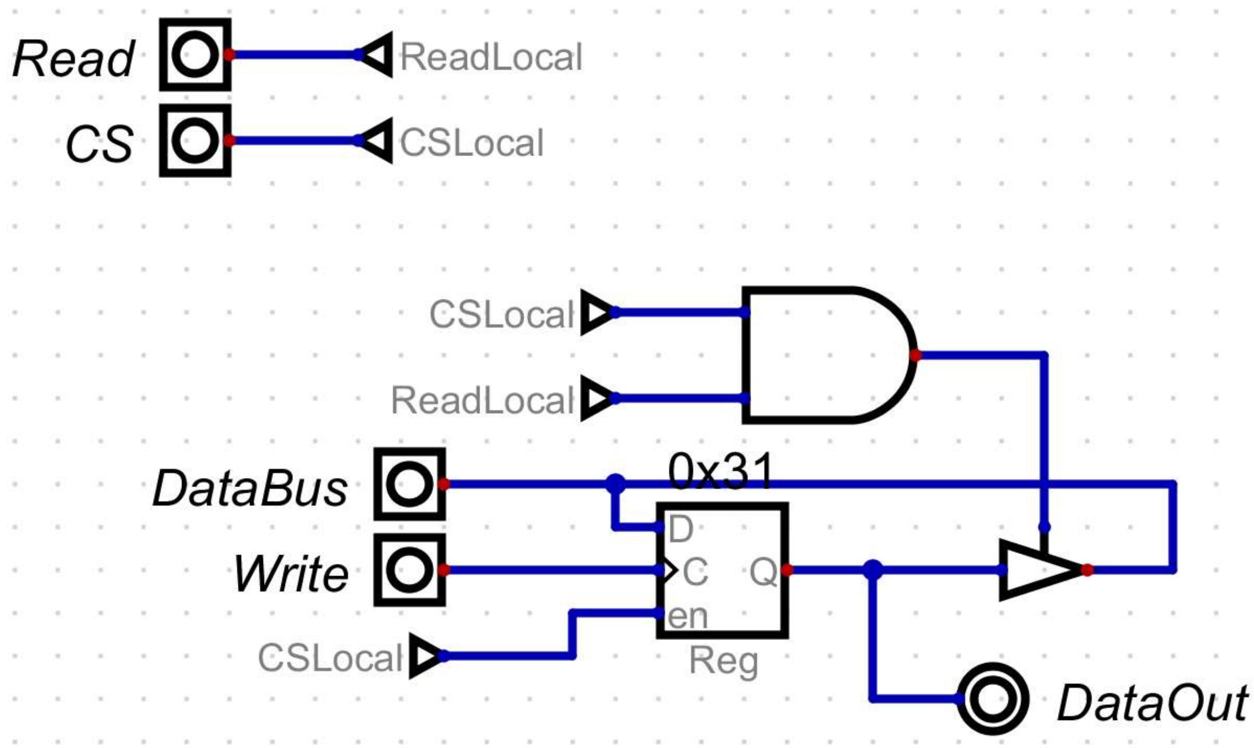


Figure 7: Port Register

3 Numerical Verification

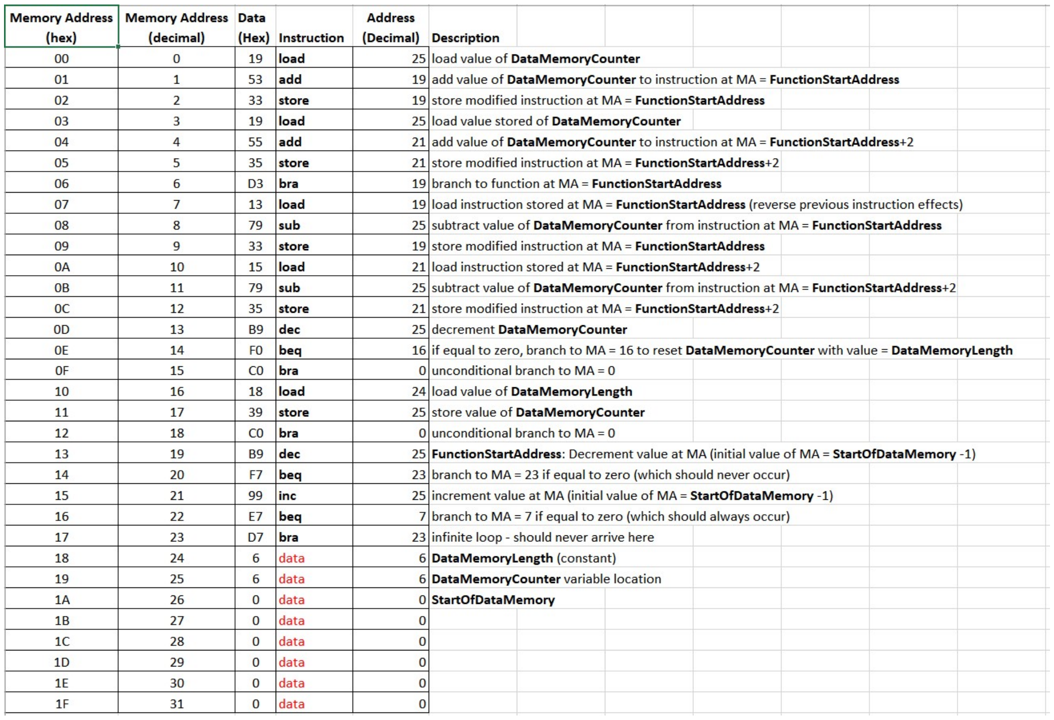
The simplest way to test this circuit would be to run a program that exercises all the normal commands within the CPU as well as the additional Address Decoder and the Port Register. To do so, a program that incorporates all CPU operations as well as accesses the Port Register was created shown in Figure 8. This program was then executed.

The program is a self-modifying program that modifies the instructions located at Memory Address (MA) 0x13 and 0x15 based on the value of the Data Memory Counter at 0x19. The addresses of the two instructions that are modified (0x13 and 0x15) are initialized with an address equal to Start of Memory Address – 1 = 25 (decimal), so that after modification, the two instructions will decrement and increment the Data at MA = 0x1F.

The modifications to the two instructions are reversed, Data Memory Counter is decremented and tested for equality to zero.

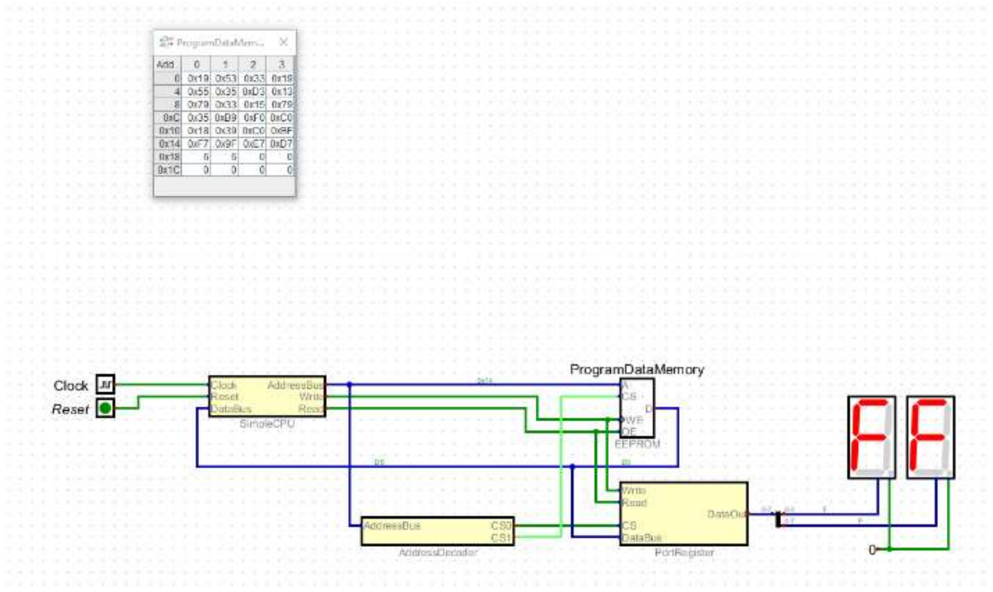
If it is not equal to zero, the program branches to the beginning and repeats the process for the remaining data locations at MA = 0x1E down to 0x1A. If it is equal to zero, it is re-initialized with the value of Data Memory Length (a constant, 6), branches back to the beginning, and repeats the process.

Because the Port Register is being used, instead of 0x1F being used, the Port Register's contents are displayed on the red seven-segment hex display. This action can be seen through Figures 9 and 10. Throughout the course of the program, each of the 6 values (0x1A-0x1E and the Port Register) are flipped to 0xFF and then to 0x00 as specified by the location value in address 0x19. This action can be seen in Figures 11 and 12. This value in address 0x19 decrements itself by 1 every time an address is incremented and then decremented until it reaches 0. Upon reaching 0, the program resets itself and changes the value at address 0x19 to 6, restarting the program from the beginning. This action can be seen in Figures 13 and 14.



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	

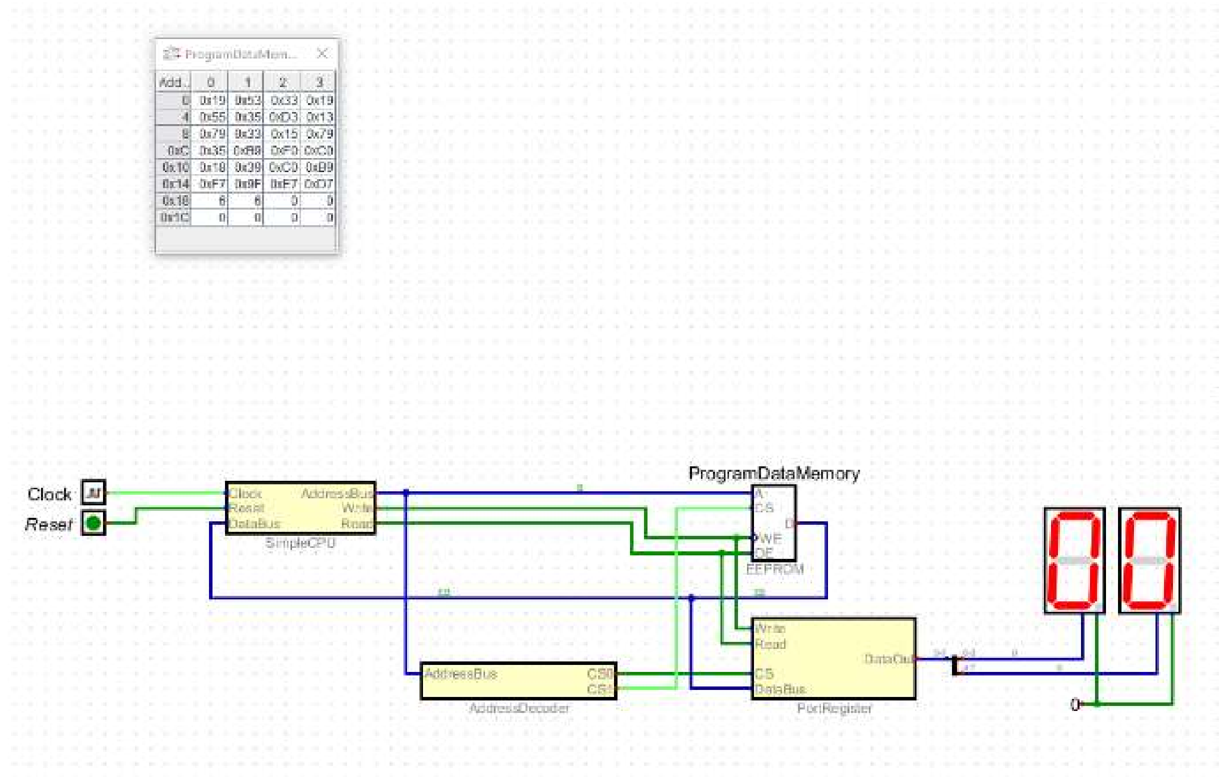
Figure 8: Test Program



ProgramDataMem...

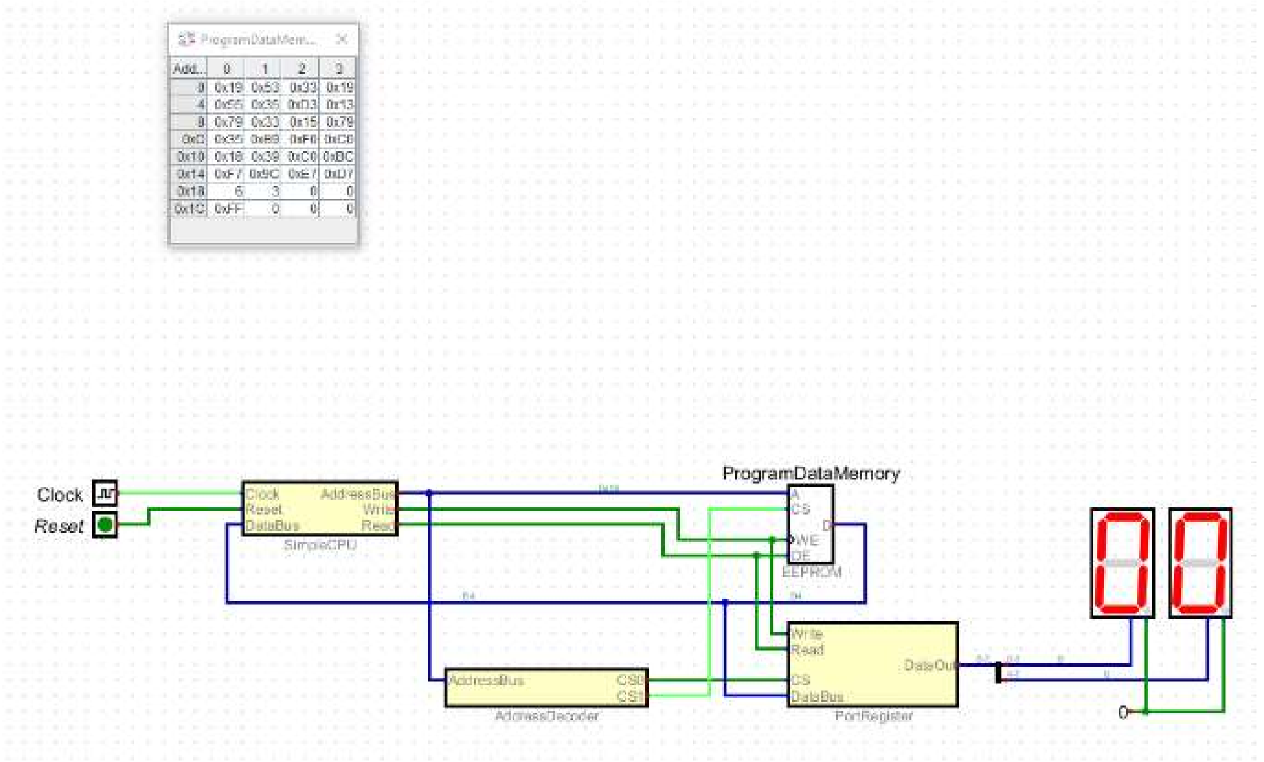
Add	0	1	2	3
0	0x19	0x53	0x33	0x19
4	0x55	0x35	0xD3	0x13
8	0x79	0x33	0x15	0x79
0xC	0x35	0xB9	0xF0	0xC0
0x10	0x18	0x39	0xC0	0x8F
0x14	0xF7	0x9F	0xE7	0xD7
0x18	6	6	0	0
0x1C	0	0	0	0

Figure 9: 0xFF in Port Register



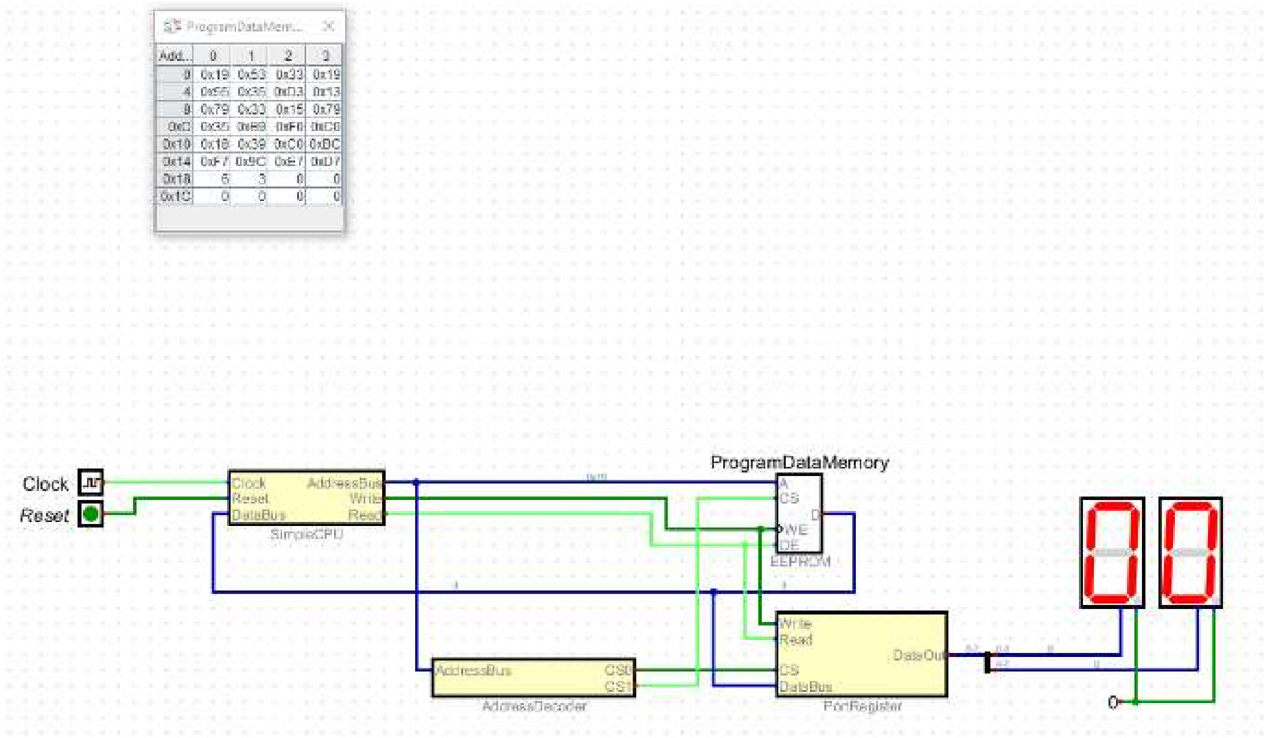
Add...	0	1	2	3
0	0x19	0x53	0x33	0x19
4	0x55	0x35	0xD3	0x13
8	0x79	0x33	0x15	0x79
0xC	0x35	0xB8	0xF0	0xC0
0x10	0x18	0x39	0xC0	0xB9
0x14	0xF7	0x9F	0xE7	0xD7
0x18	6	6	0	0
0x1C	0	0	0	0

Figure 10: Port Register Returning to 0x00



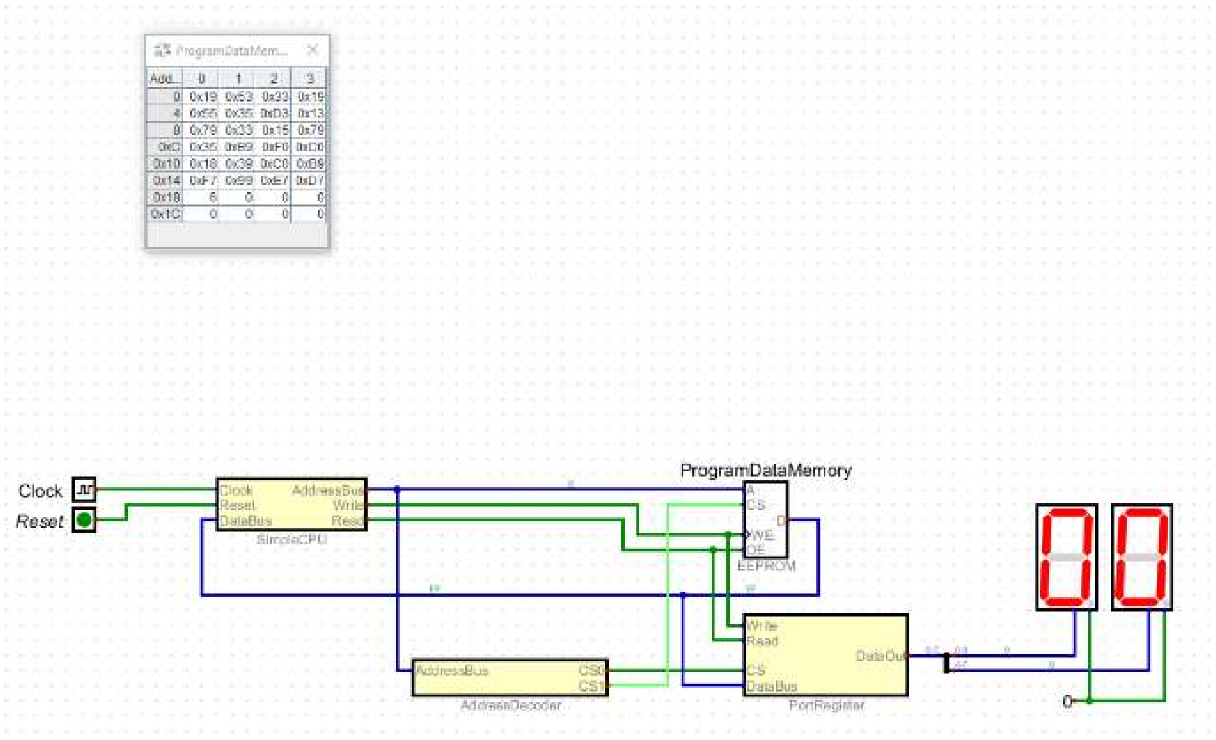
Add...	0	1	2	3
0	0x19	0x53	0x33	0x19
4	0x55	0x35	0xD3	0x13
8	0x79	0x33	0x15	0x79
0xC	0x35	0xB8	0xF0	0xC0
0x10	0x18	0x39	0xC0	0xBC
0x14	0xF7	0x9C	0xE7	0xD7
0x18	6	3	0	0
0x1C	0xFF	0	0	0

Figure 11: Third Address (0x1C) Turning to 0xFF


ProgramDataMem...

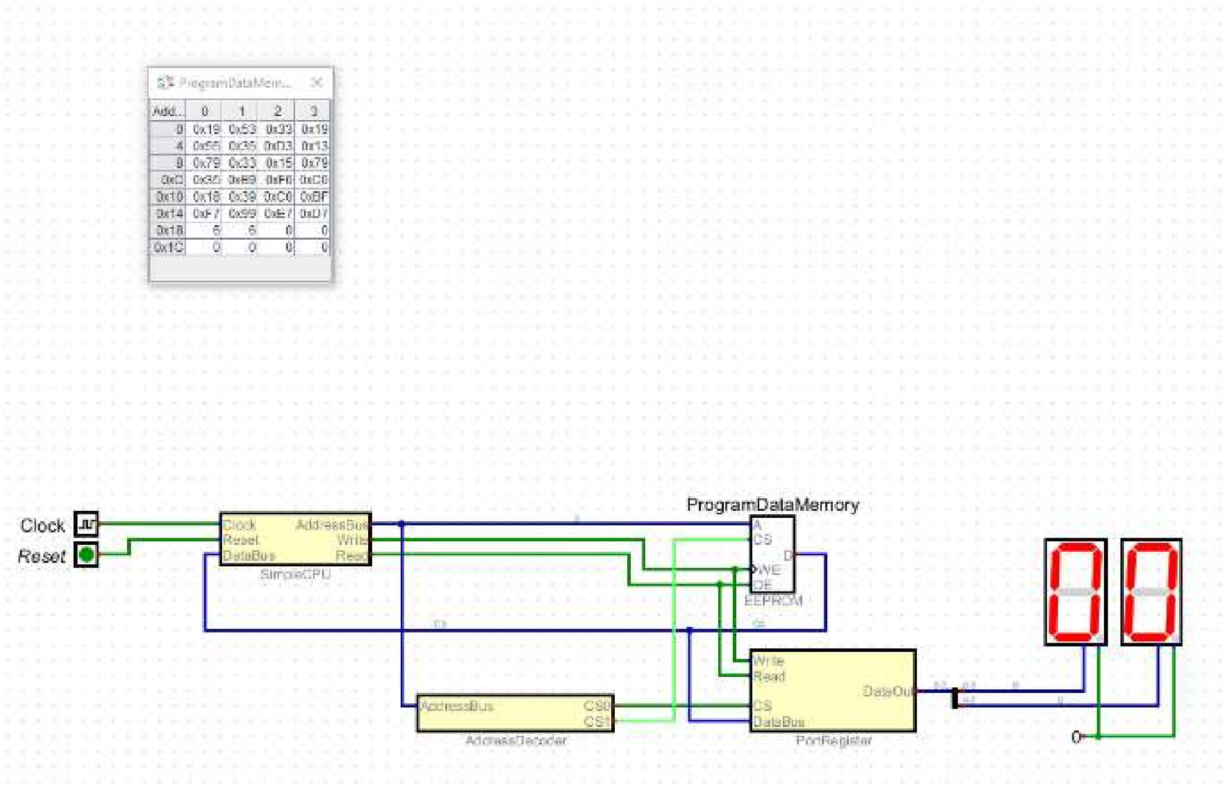
Add...	0	1	2	3
0	0x19	0x53	0x33	0x19
4	0x56	0x35	0xD3	0x13
8	0x79	0x33	0x15	0x79
0xC	0x35	0x89	0xF0	0xC0
0x10	0x18	0x39	0xC0	0xDC
0x14	0xF7	0x9C	0xE7	0xD7
0x18	6	3	0	0
0x1C	0	0	0	0

Figure 12: Third Address (0x1C) Returning to 0x00


ProgramDataMem...

Add...	0	1	2	3
0	0x19	0x53	0x33	0x19
4	0x56	0x35	0xD3	0x13
8	0x79	0x33	0x15	0x79
0xC	0x35	0x89	0xF0	0xC0
0x10	0x18	0x39	0xC0	0xD9
0x14	0xF7	0x99	0xE7	0xD7
0x18	6	0	0	0
0x1C	0	0	0	0

Figure 13: Data Memory Counter (0x19) at 0x00



Add...	0	1	2	3
0	0x19	0x53	0x33	0x19
4	0x55	0x35	0xD3	0x13
8	0x79	0x33	0x15	0x79
0xC	0x35	0xB9	0xF0	0xC0
0x10	0x18	0x39	0xC0	0xBF
0x14	0xF7	0x99	0xE7	0xD7
0x18	6	6	0	0
0x1C	0	0	0	0

Figure 14: Data Memory Counter (0x19) Set to 0x06

Because the program can accomplish all its actions successfully, as evidenced by its ability to complete each step as well as its ability to loop, we can conclude that the design satisfies the problem requirements.

4 Summary

The problem requirement for this assignment was to design a hierarchical schematic of a simple computer system as detailed in Figure 1 followed by designing a memory mapped I/O device as shown in Figure 3. This was to be built on top of the pre-existing CPU structure with the unique addition of the Port Register and the Address Decoder.

By employing the diagram displayed in Figures 1 and 3, the circuits detailed in Figures 4 to 7 were created. These circuits were tested with a self-modifying program that modifies its own instructions depending on the value of its Data Memory Counter(0x19). As it was shown that the Address Decoder and Port Register were used and the program completed a loop and was to begin the next loop, we can conclude that the specified digital circuit design solution satisfies the problem requirements.