Tyler Kim

Digital Logic Design

Simple CPU

Description

The purpose of this learning activity is to test the final CPU after combining all the other components made before. More specifically, I was tasked to create a program to test all the functions of the CPU as a whole.

Schematic

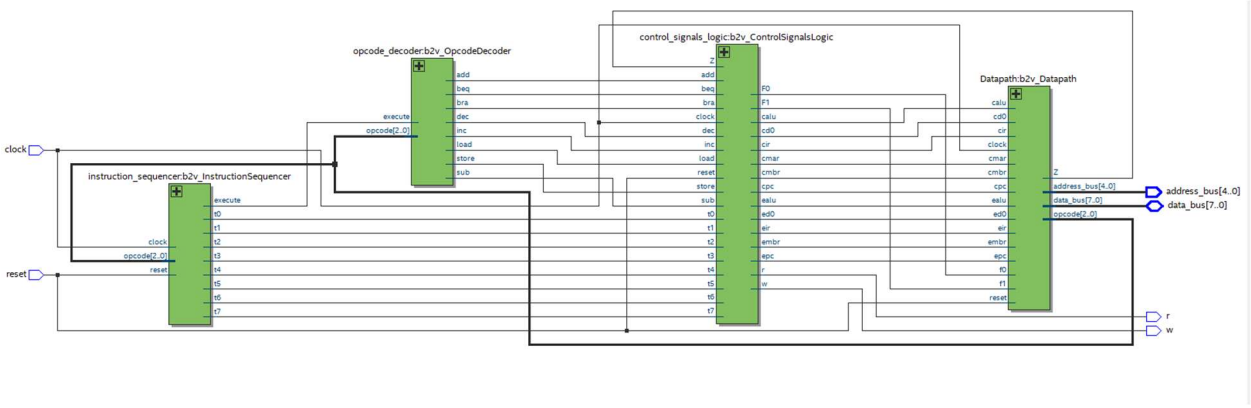

Figure 1: Simple CPU Schematic

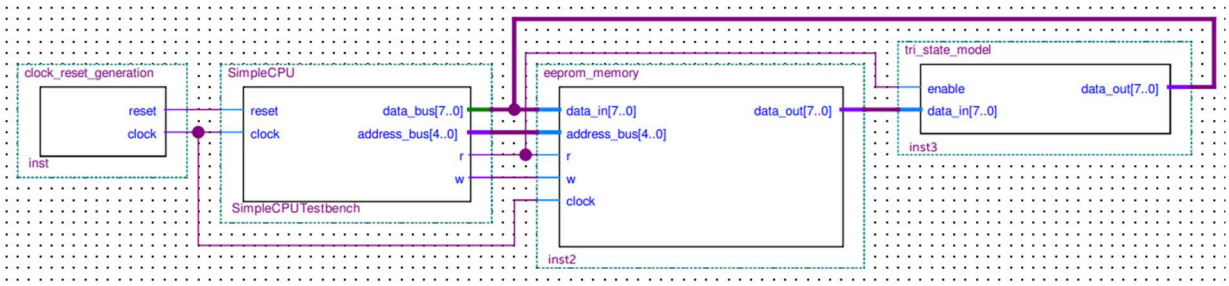

Figure 2: Simple Computer System

The schematic in Figure 1 illustrates the schematic for the simple CPU only using the components created from previous learning activities. The schematic in Figure 2 shows the entire computer system with the clock reset generation, CPU, memory, and tri-state model.

Program Explanation

The program used to verify the functionality of the simple CPU follows a simple logic.

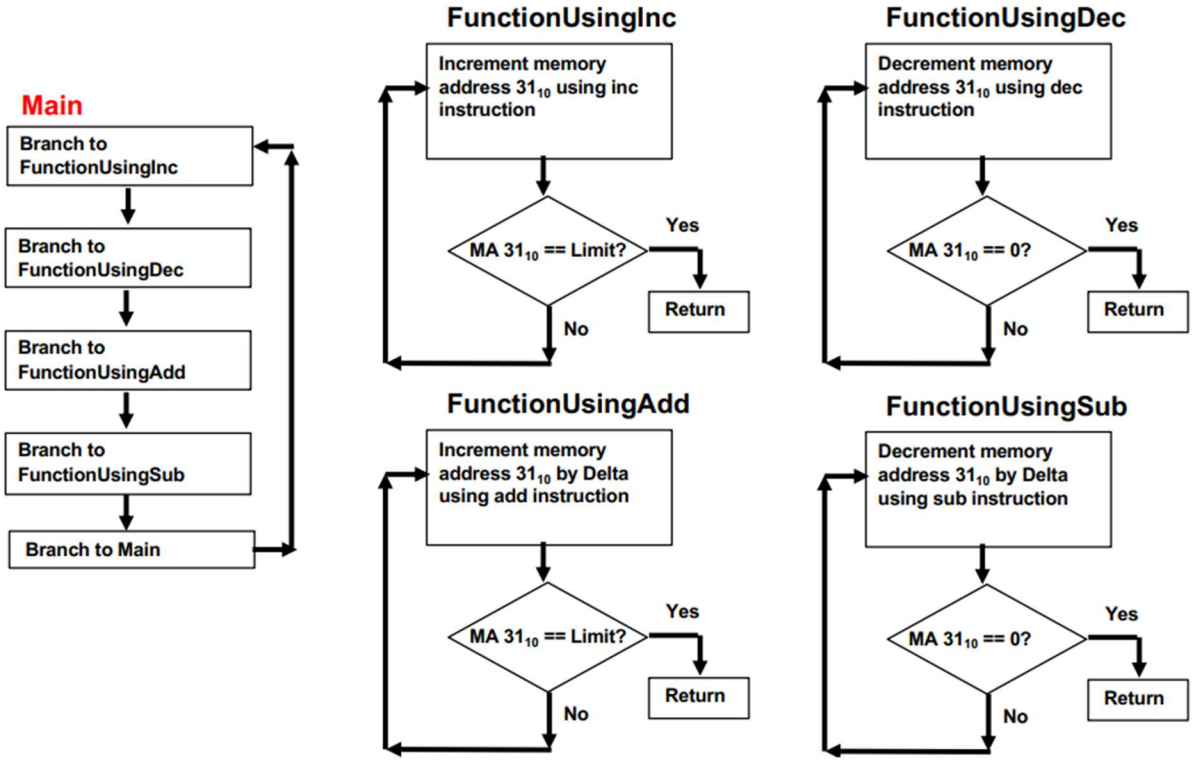

Figure 3: Flowchart of Program

Memory Address (hex)	Memory Address (decimal)	Data (Hex)	Memory File Contents	Instruction	Address (Decimal)	Description
00	0	C5	00/C5;	**bra**	5	// (Main) unconditionally branches to FunctionUsingInc in Memory Address = 5
01	1	CA	01/CA;	**bra**	10	//unconditionally branches to FunctionUsingDec in Memory Address = 10
02	2	CD	02/CD;	**bra**	13	//unconditionally branches to FunctionUsingAdd in Memory Address = 13
03	3	D4	03/D4;	**bra**	20	//unconditionally branches to FunctionUsingSub; MA = 19
04	4	C0	04/C0;	**bra**	0	//unconditionally branches to Main; MA = 0
05	5	9F	05/9F;	**inc**	31	//(FunctionUsingInc) Beginning of FunctionUsingInc; increment MA = 31
06	6	1F	06/1F;	**load**	31	//load value in MA = 31 into D0
07	7	7C	07/7C;	**sub**	28	//Subtract value in D0 by limit
08	8	E1	08/E1;	**beq**	1	//invoke function again; runs only when MA = 31 is not equal to 0
09	9	C5	09/C5;	**bra**	5	//unconditionally branch to the beginning of the function
0A	10	BF	0A/BF;	**dec**	31	//decrement using MA = 31; (FunctionUsingDec)
0B	11	E2	0B/E2;	**beq**	2	//If MA = 31 is equal to the limit; yes => go to invocation
0C	12	CA	0C/CA;	**bra**	10	//invokes function again if MA = 31 is not equal to 0
0D	13	1F	0D/1F;	**load**	31	//(FunctionUsingAdd) loads the value in MA = 31 into D0
0E	14	5D	0E/5D;	**add**	29	//adds delta to D0
0F	15	3F	0F/3F;	**store**	31	//stores the value into MA = 31
10	16	1F	10/1F;	**load**	31	//load value in MA = 31 into D0
11	17	7C	11/7C;	**sub**	28	//Subtract value in D0 by limit
12	18	E3	12/E3;	**beq**	3	//If MA = 31 is equal to the limit; yes => go to invocation
13	19	CD	13/CD;	**bra**	13	//unconditionally branch to the beginning of the function
14	20	1F	14/1F;	**load**	31	//(FunctionUsingSub) load MA=31 into D0 register
15	21	7D	15/7D;	**sub**	29	//subtract the value by delta
16	22	3F	16/3F;	**store**	31	//store the value into MA = 31
17	23	1F	17/1F;	**load**	31	//load MA=31 back into D0
18	24	7E	18/7E;	**sub**	30	//subtract from 0
19	25	E4	19/E4;	**beq**	4	//conditional statement, if value in MA=31 is 0 then go to invocation
1A	26	D4	1A/D4;	**bra**	20	//unconditionally branch to the beginning of the function
1B	27	00	1B/00;	**load**	0	
1C	28	0A	1C/0A;	data	10	**limit**
1D	29	02	1D/02;	data	2	**delta**
1E	30	00	1E/00;	data	0	
1F	31	00	1F/00;	data	0	

Figure 4: Implementation in Excel

Figure 3 shows a simple flowchart of the logic of the program used to verify the functionality of the CPU and Figure 4 illustrates the actual implementation. In short, the program has four functions and two constants: delta and limit. In the case of my particular implementation, delta and limit were set to 2 and 10 respectively. The program first tests the *FunctionUsingInc* function where it increments the value in memory address 31 until it reaches the limit. When it reaches the limit, it decrements the value until it reaches 0 through *FunctionUsingDec*. Then the program increments the value in memory address 31 by delta using *FunctionUsingAdd* until reaches the limit and then decrements by delta using *FunctionUsingSub* until it reaches the value of 0.

Verification

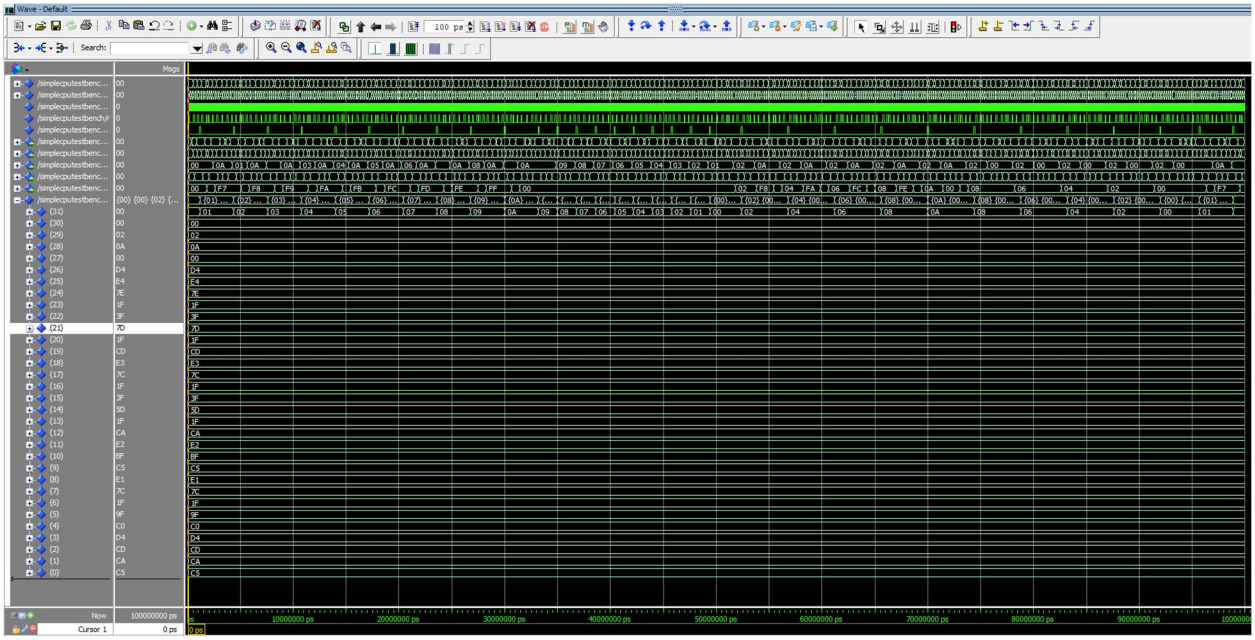

Figure 5: Overall Verification Test

Figure 5 shows the overall verification tests of the functionality of the program and the functionality of the CPU where memory address 31 is the variable that was transformed by the operations the most. To further illustrate the success of the functionality of the program and CPU, the verification section in this report is divided into four additional parts for each function.

FunctionUsingInc

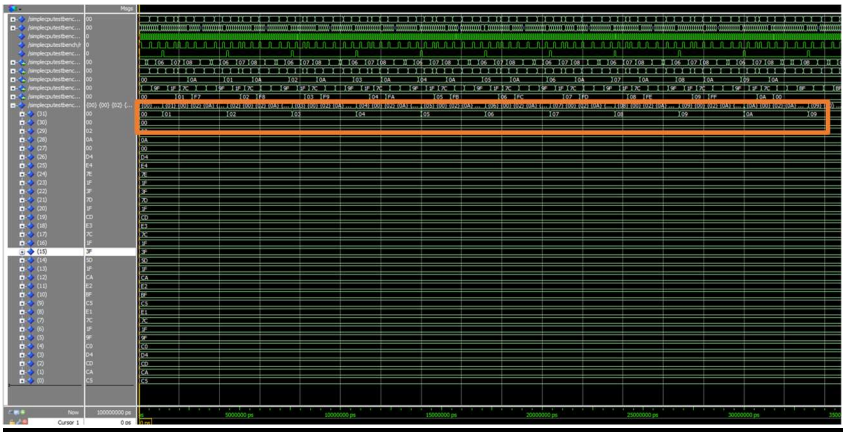

Figure 6: FunctionUsingInc Verification

Figure 6 shows the *FunctionUsingInc* works correctly because the value in memory address 31 increases by one every time. When the value in memory address 31 equals 10, it proceeds to begin decreasing showing the start of the *FunctionUsingDec* function.

FunctionUsingDec

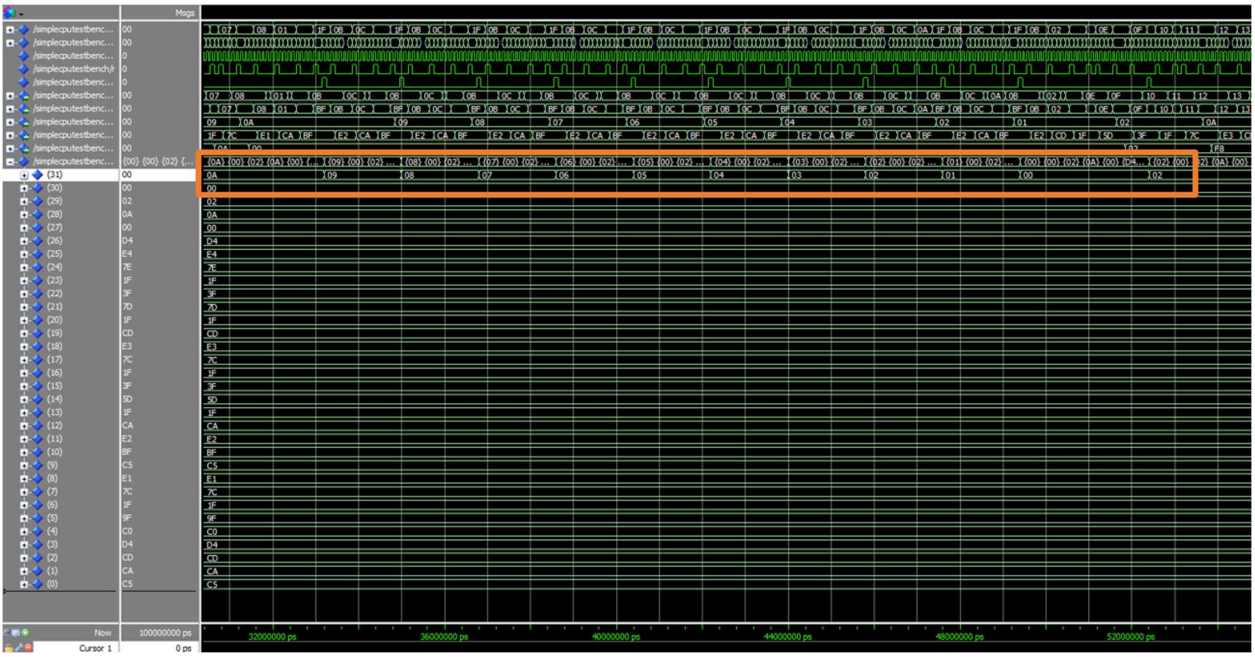

Figure 7: FunctionUsingDec Verification

Figure 7 shows that the *FunctionUsingDec* works correctly because when the value in memory address 31 is 10, it begins decreasing until it reaches 0. Then, the value starts transforming by the *FunctionUsingAdd* function.

FunctionUsingAdd

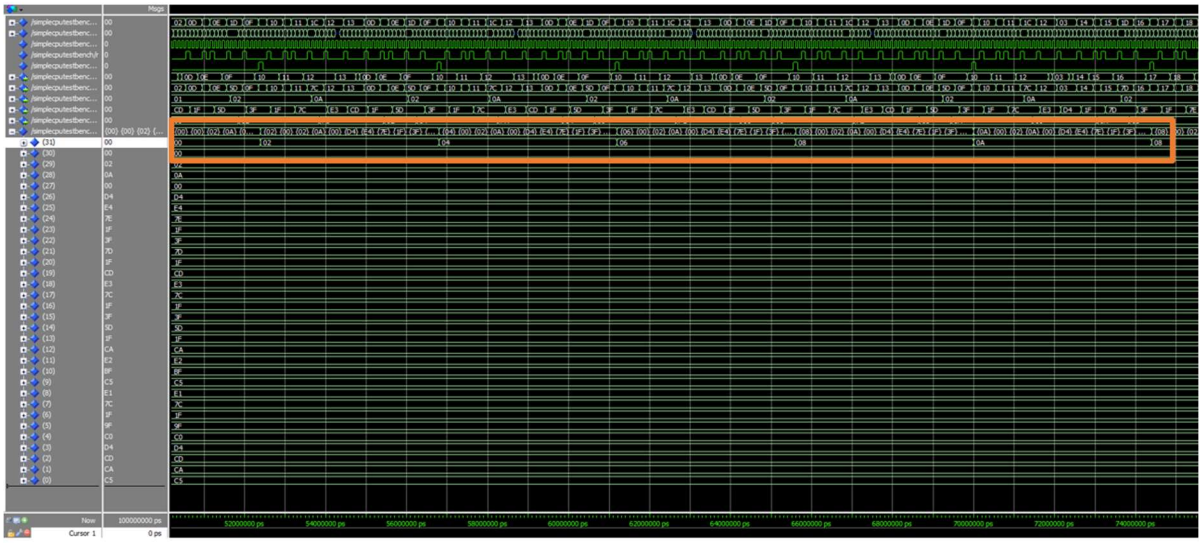

Figure 8: FunctionUsingAdd Verification

Figure 8 shows that the *FunctionUsingAdd* works correctly because when the value in memory address 31 is 0, it begins incrementing by delta, 2. Then, the value starts transforming by the *FunctionUsingSub* function when the value reaches 10.

FunctionUsingSub

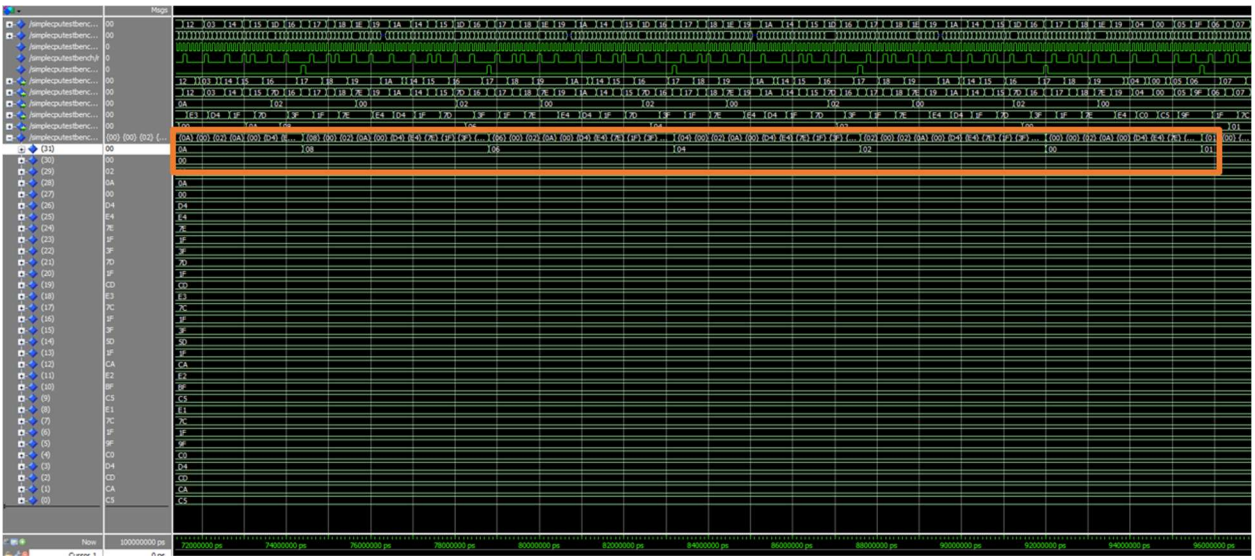

Figure 9: FunctionUsingSub Verification

Figure 8 shows that the *FunctionUsingSub* works correctly because when the value in memory address 31 is 10, it begins decrementing by delta, 2. Then, the value starts transforming by the *FunctionUsingInc* function when the value reaches 0 since the program will run as a continuous loop.