ECE 2330

Semester Project

Simple CPU

Winston Zhang

Table of Contents and Figures

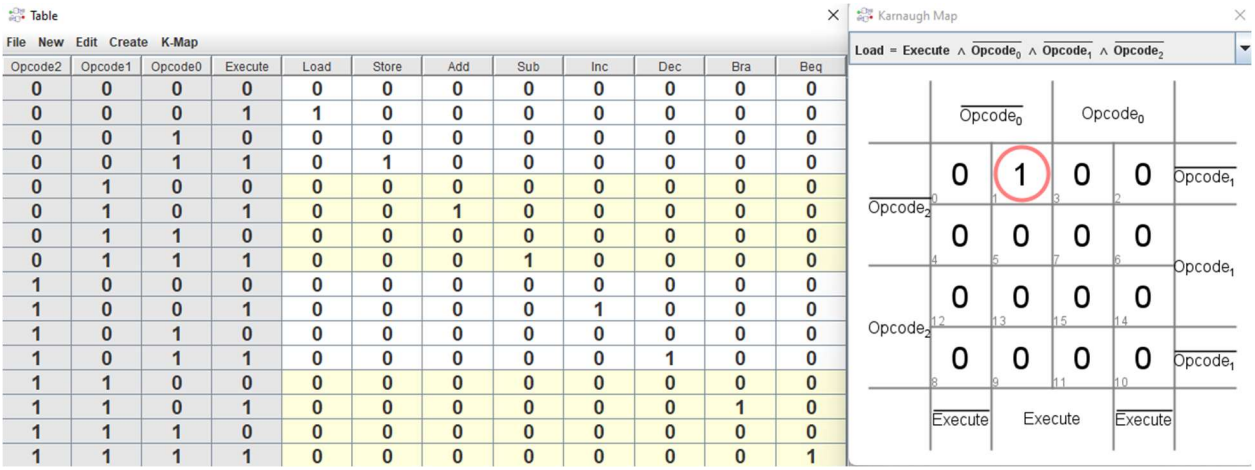
1. Opcode Decoder

1 Problem Statement

Design an opcode decoder - a digital circuit that will take in an opcode (instruction) signal of three bits, and an execution signal, producing a certain output signal corresponding to the instruction specified by the opcode signal.

2 Analytical Design

The truth table for the opcode decoder circuit, as well as its corresponding Karnaugh map is shown in Figure 1.



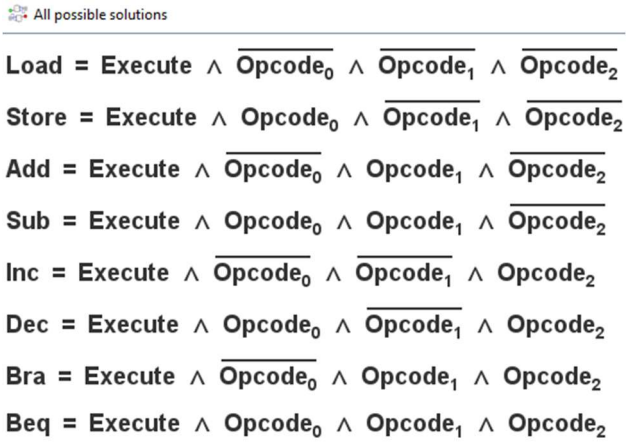
Table

File New Edit Create K-Map

Opcode2	Opcode1	Opcode0	Execute	Load	Store	Add	Sub	Inc	Dec	Bra	Beq
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	1

Figure 1: Truth Table and Karnaugh Map for Opcode Decoder Circuit

The logical expression for the output for each instruction is listed in Figure 2.



All possible solutions

Load = Execute ∧ $\overline{Opcode_0}$ ∧ $\overline{Opcode_1}$ ∧ $\overline{Opcode_2}$

Store = Execute ∧ $Opcode_0$ ∧ $\overline{Opcode_1}$ ∧ $\overline{Opcode_2}$

Add = Execute ∧ $\overline{Opcode_0}$ ∧ $Opcode_1$ ∧ $\overline{Opcode_2}$

Sub = Execute ∧ $Opcode_0$ ∧ $Opcode_1$ ∧ $\overline{Opcode_2}$

Inc = Execute ∧ $\overline{Opcode_0}$ ∧ $\overline{Opcode_1}$ ∧ $Opcode_2$

Dec = Execute ∧ $Opcode_0$ ∧ $\overline{Opcode_1}$ ∧ $Opcode_2$

Bra = Execute ∧ $\overline{Opcode_0}$ ∧ $Opcode_1$ ∧ $Opcode_2$

Beq = Execute ∧ $Opcode_0$ ∧ $Opcode_1$ ∧ $Opcode_2$

Figure 2: Logical Expressions for Each Instruction

The circuit implementation of the opcode decoder is shown in Figure 3.

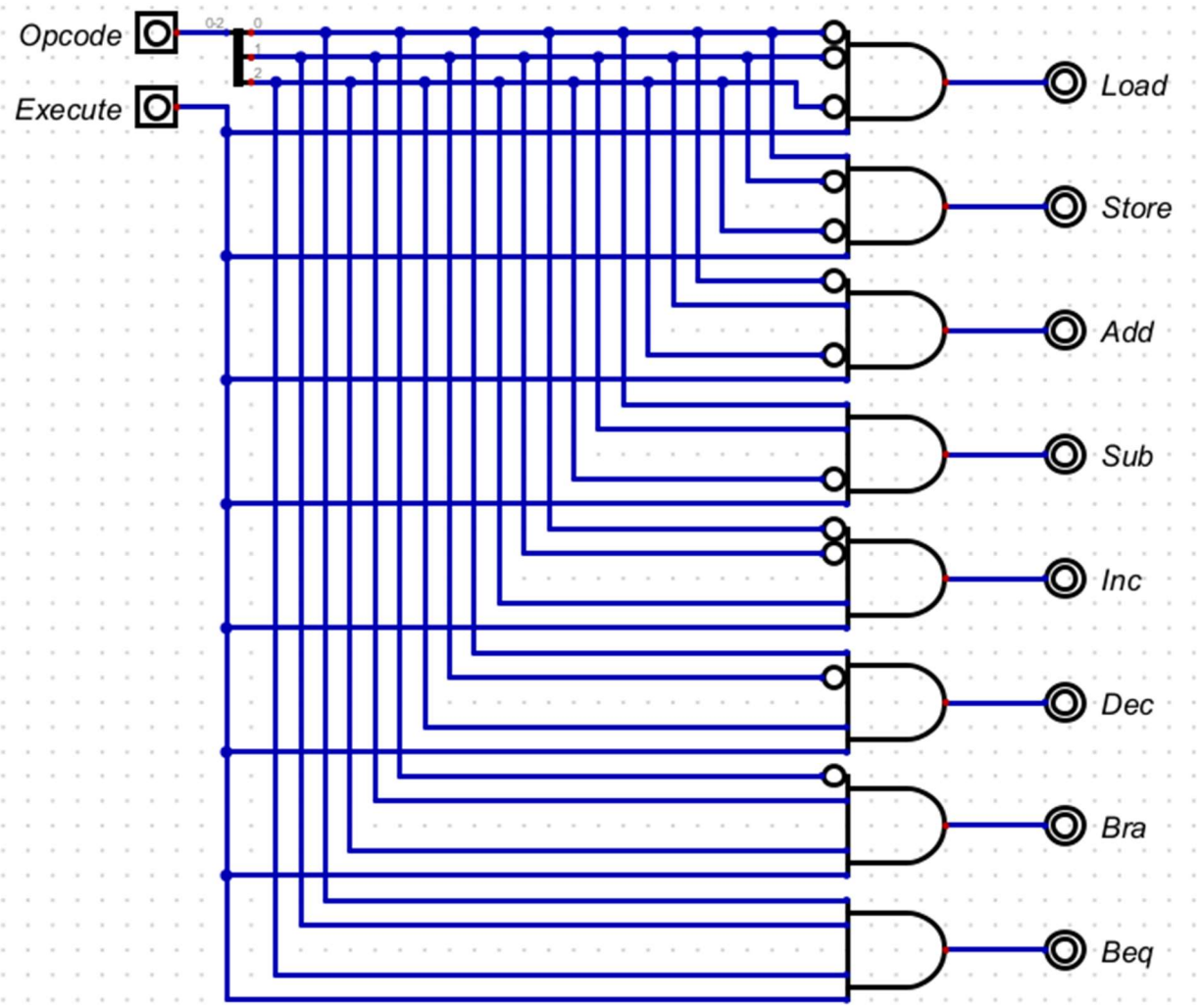


Figure 3: Circuit Implementation for Opcode Decoder

3 Numerical Verification

An exhaustive list of input vectors was loaded into a test case and applied to the circuit via a test bench. The numerical results are shown in Figure 4, Figure 5, and Figure 6.

Test data

```
# Test vectors for opcode decoder
# Inputs: Opcode (3 bits), Execute
# Outputs: Load, Store, Add, Sub, Inc, Dec, Bra, Beq
# When Execute = 0, all output equal 0
# When Execute = 1, decode Opcode
Opcode Execute Load Store Add Sub Inc Dec Bra Beq
# Load instruction
0      0       0    0     0   0   0   0   0
0      1       1    0     0   0   0   0   0
# Store instruction
1      0       0    0     0   0   0   0   0
1      1       0    1     0   0   0   0   0
# Add instruction
2      0       0    0     0   0   0   0   0
2      1       0    0     1   0   0   0   0
# Subtract instruction
3      0       0    0     0   0   0   0   0
3      1       0    0     0   1   0   0   0
# Increment instruction
4      0       0    0     0   0   0   0   0
4      1       0    0     0   0   1   0   0
# Decrement instruction
5      0       0    0     0   0   0   0   0
5      1       0    0     0   0   0   1   0
# Branch instruction
6      0       0    0     0   0   0   0   0
6      1       0    0     0   0   0   0   1
# Branch if equal to zero instruction
7      0       0    0     0   0   0   0   0
7      1       0    0     0   0   0   0   1
```

Figure 4: Input Test Vectors

passed

	Opcode	Execute	Load	Store	Add	Sub	Inc	Dec	Bra	Beq
L8	0	0	0	0	0	0	0	0	0	0
L9	0	1	1	0	0	0	0	0	0	0
L11	1	0	0	0	0	0	0	0	0	0
L12	1	1	0	1	0	0	0	0	0	0
L14	2	0	0	0	0	0	0	0	0	0
L15	2	1	0	0	1	0	0	0	0	0
L17	3	0	0	0	0	0	0	0	0	0
L18	3	1	0	0	0	1	0	0	0	0
L20	4	0	0	0	0	0	0	0	0	0
L21	4	1	0	0	0	0	1	0	0	0
L23	5	0	0	0	0	0	0	0	0	0
L24	5	1	0	0	0	0	0	1	0	0
L26	6	0	0	0	0	0	0	0	0	0
L27	6	1	0	0	0	0	0	0	1	0
L29	7	0	0	0	0	0	0	0	0	0
L30	7	1	0	0	0	0	0	0	0	1

Figure 5: Test Results in Tabular Form

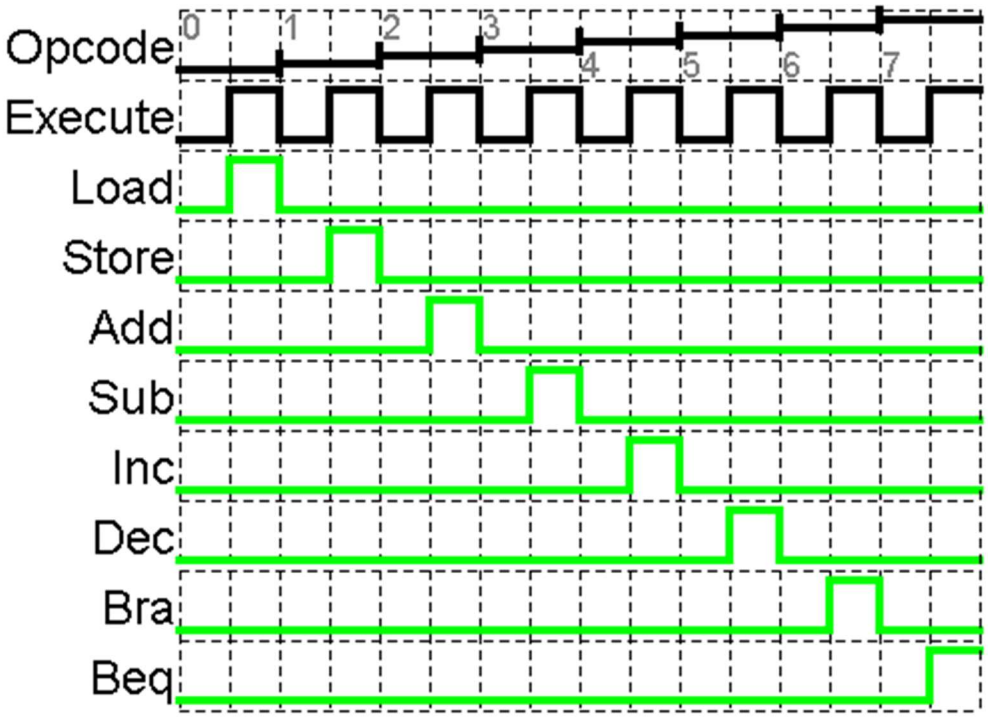


Figure 6: Test Results in Graphical Form

4 Summary

The goal of this project assignment was to create a circuit that could interpret an opcode input of three bits as well as an execution input of one bit, and produce the correct output signal corresponding to the instruction specified by the opcode input. In other words, if the opcode was 101 (3 in binary), the circuit would produce a signal on the third output pin, corresponding to the third instruction, which in this case, happens to be Addition. All of these output signals would only be produced if the Execute input signal was true.

A truth table and Karnaugh map were used to define the expected outputs in relation to inputs, and a digital circuit solution was developed. An exhaustive list of input test vectors was tested against this circuit, and all inputs produced the expected output for each test vector. Therefore, the specified digital circuit design satisfies the problem set out to be solved in the original problem statement.

2. Control Signal Logic

1 Problem Statement

Design circuits to generate the control signals for the Central Processing Unit. This circuit takes inputs from the Opcode Decoder, as well as various Control Signals, and outputs the corresponding signal for a set of Register Transfer Level operations, or microinstructions. This architecture also includes the circuit design for the MemoryAddress Register, Memory Buffer Register, Instruction Register, Program Counter, Data register, Arithmetic & Logic Unit, and ALU register.

2 Analytical Design

The truth table for the control signal logic circuit is shown in Figure 1, courtesy of Professor Johnson.

		Instruction Sequence Timing Signals								Control Actions														
Instruction	Operations (RTL)	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	R E_{MSR}	W	C_{MAR}	C_{MBR}	C_{PC}	C_{IR}	C_{D0}	C_{ALU}	E_{MBR}	E_{PC}	E_{IR}	E_{D0}	E_{ALU}	F_1	F_0
Fetch	MAR ← [PC]	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
	IR ← [MAR]	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	ALU (Q) ← [PC]	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0
	PC ← [ALU]	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
Load	MAR ← [IR]	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
	D0 ← [MAR]	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Store	MAR ← [IR]	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
	[MAR] ← [D0]	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0
Add	MAR ← [IR]	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
	MBR ← [MAR]	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	ALU (P) ← [MBR]	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
	D0 ← [ALU]	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
Sub	MAR ← [IR]	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
	MBR ← [MAR]	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	ALU (P) ← [MBR]	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1
	D0 ← [ALU]	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
Inc	MAR ← [IR]	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
	MBR ← [MAR]	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	ALU (P) ← [MBR]	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0
	[MAR] ← [ALU]	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0
Dec	MAR ← [IR]	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
	MBR ← [MAR]	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	ALU (P) ← [MBR]	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1
	[MAR] ← [ALU]	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0
Bra	PC ← [IR]	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
Beq	If Z = 1 then PC ← [IR]	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0

Figure 1: Truth Table for Control Signal Logic Circuit

The logical expression for each instruction type was derived from the truth table and further reduced using boolean algebra, mostly the distributive property. The equations for each instruction type are show in Table 1:

Read	T1 ∨ (T5 ∧ (Load ∨ Add ∨ Sub ∨ Inc ∨ Dec))
Write	(T5 ∧ Store) ∨ (T7 ∧ (Inc ∨ Dec))
Cmar	T0 ∨ (T4 ∧ (Load ∨ Store ∨ Add ∨ Sub ∨ Inc ∨ Dec))
Cmbr	T5 ∧ (Add ∨ Sub ∨ Inc ∨ Dec)
Embr	T6 ∧ (Add ∨ Sub ∨ Inc ∨ Dec)
Cir	T1
Eir	T4 ∧ (Bra ∨ Load ∨ Store ∨ Add ∨ Sub ∨ Inc ∨ Dec ∨ (Z ∧ Beq))
Cpc	T3 ∨ (T4 ∧ (Bra ∨ (Z ∧ Beq))
Epc	T0 ∨ T2
Cd0	(T5 ∧ Load) ∨ (T7 ∧ (Add ∨ Sub))
Ed0	T5 ∨ Store
Calu	T2 ∨ (T6 ∧ (Add ∨ Sub ∨ Inc ∨ Dec))
Ealu	T3 ∨ (T7 ∧ (Add ∨ Sub ∨ Inc ∨ Dec))
F0	T6 ∧ (Inc ∨ Dec)
F1	T2 ∨ (T6 ∧ (Inc ∨ Dec))

Table 1: Logical Expressions for Each Microinstruction

The circuit implementation of the control signal logic circuit is shown in Figure 2.

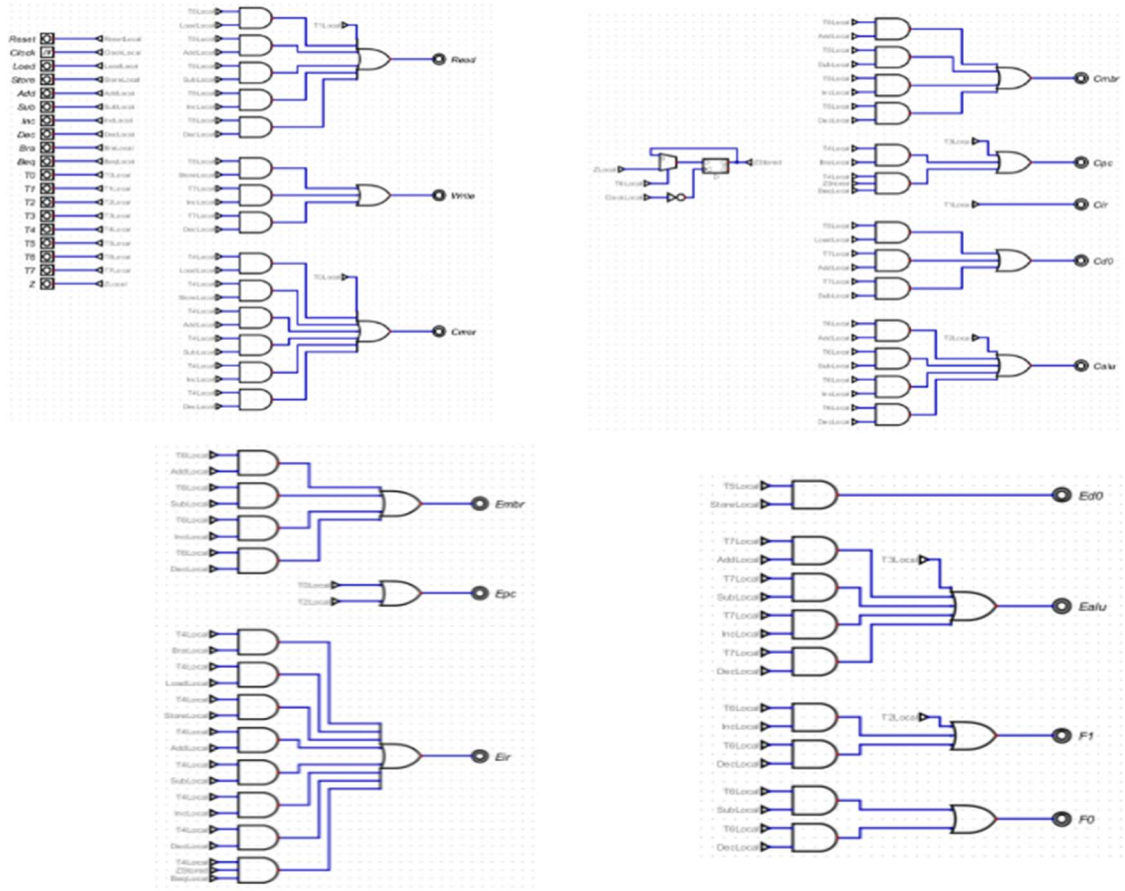


Figure 2: Circuit Implementation for Control Signal Logic Circuit

3 Numerical Verification

An exhaustive list of input vectors was loaded into a test case and applied to the circuit via a test bench. The numerical results are shown in Figure 3, Figure 4, and Figure 5.

Test data

```
# Test vectors for control signals logic
Reset Clock Load Store Add Sub Inc Dec Bra Beq T0 T1 T2 T3 T4 T5 T6 T7 Z Read Write Cmar Cmbr Cpc Cir Cd0 Calu Embr Epc Eir Ed0 Ealu F1 F0
# Fetch: MAR ← [PC]
0     1     0     0     0   0   0   0   0   0   1  0  0  0  0  0  0  0  0 0    0     1     0     0   0   0   0    0   1   0   0   0    0  0
0     0     0     0     0   0   0   0   0   0   1  0  0  0  0  0  0  0  0 0    0     1     0     0   0   0   0    0   1   0   0   0    0  0
# IR ← [MAR]
0     1     0     0     0   0   0   0   0   0   0  1  0  0  0  0  0  0  0 1    0     0     0     0   1   0   0    0   0   0   0   0    0  0
0     0     0     0     0   0   0   0   0   0   0  1  0  0  0  0  0  0  0 1    0     0     0     0   1   0   0    0   0   0   0   0    0  0
# ALU (Q) ← [PC]
0     1     0     0     0   0   0   0   0   0   0  0  1  0  0  0  0  0  0 0    0     0     0     0   0   0   1    0   1   0   0   0    1  0
0     0     0     0     0   0   0   0   0   0   0  0  1  0  0  0  0  0  0 0    0     0     0     0   0   0   1    0   1   0   0   0    1  0
# PC ← [ALU]
0     1     0     0     0   0   0   0   0   0   0  0  0  1  0  0  0  0  0 0    0     0     0     1   0   0   0    0   0   0   0   1    0  0
0     0     0     0     0   0   0   0   0   0   0  0  0  1  0  0  0  0  0 0    0     0     0     1   0   0   0    0   0   0   0   1    0  0
```

Figure 3: Input Test Vectors

passed

	Re...	Clo...	Load	Store	Add	Sub	Inc	Dec	Bra	Beq	T0	T1	T2	T3	T4	T5	T6	T7	Z	Re...	Write	Cm...	Cm...	Cpc	Cir	Cd0	Calu	Em...	Epc	Eir	Ed0	Ealu	F1	F0
L4	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
L5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
L7	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
L8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
L10	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0
L11	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0
L13	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
L14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 4: Test Results in Tabular Form

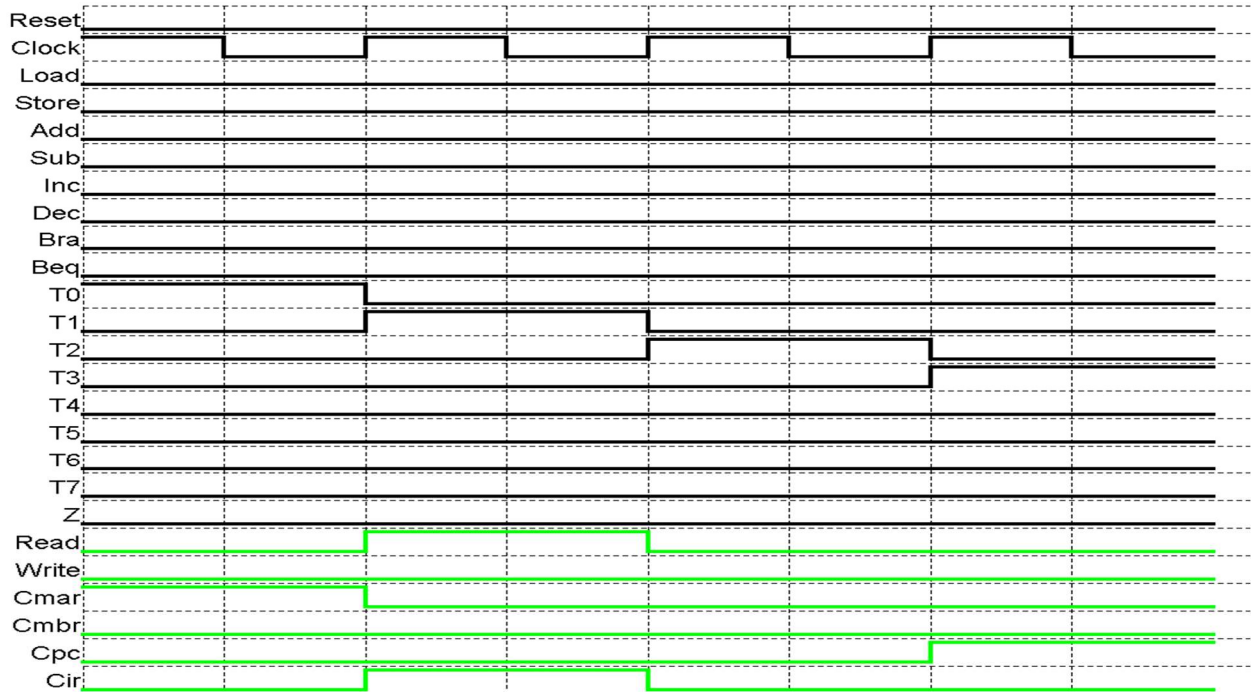


Figure 5: Test Results in Partial Graphical Form

Figure 5 only shows part of the test vectors, as there were too many rows to display in the test case results screen of the Digital logic simulator, and there was no way to scroll down, only zoom out along the horizontal axis.

4 Summary

The goal of this project assignment was to create a circuit that could interpret a series of input instructions, as well as control signals, and produce the correct microinstruction for the CPU.

A truth table, provided by the assignment instructions PDF, was used to define the circuit's expected outputs in response to a set of several inputs. Logical expressions were derived from the table and reduced using boolean algebra, which were then converted into circuit form in the Digital logic simulator. A set of input test vectors was tested against this circuit, and all inputs produced the expected output for each test vector. Therefore, the specified digital circuit design satisfies the problem set out to be solved in the original problem statement.

3. Instruction Sequencer FSM

1 Problem Statement

2 Analytical Design

The project assignment offers a diagram illustrating the FSM, which has been recreated in the ZyBook FSM simulator for this report in Figure 1:

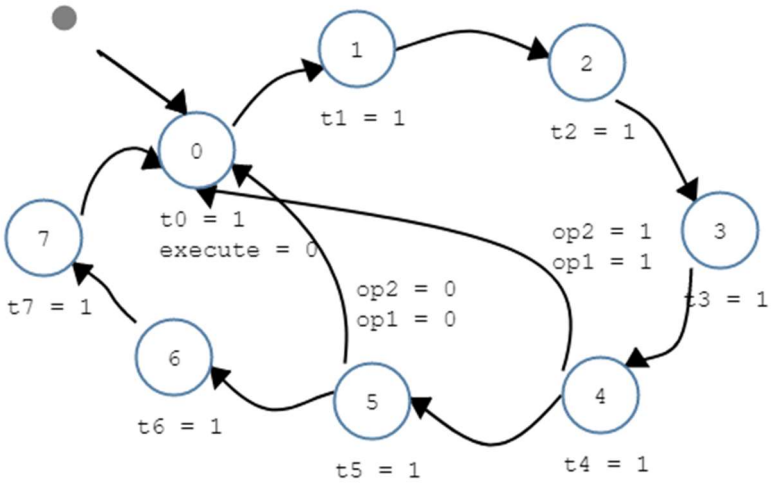


Figure 1: Finite State Machine Representation of Instruction Sequencer

The project assignment offers three distinct tables, which represent the behavior of the FSM. The first table described the sequence of state transitions for each opcode following consecutive clock signals. For each opcode, the stored state starts as 001 and ends with 000 - every cell left blank indicates a “don’t care”, as under normal operation that situation should not occur. These transitions are represented in Table 1 (found on next page):

	Next State	Load (000)	Store (001)	Add (010)	Sub (011)	Inc (100)	Dec (101)	Bra (110)	Beq (111)
Current State	**000**	001	001	001	001	001	001	001	001
	001	010	010	010	010	010	010	010	010
	010	011	011	011	011	011	011	011	011
	011	100	100	100	100	100	100	100	100
	100	101	101	101	101	101	101	000	000
	101	000	000	110	110	110	110		
	110			111	111	111	111		
	111			000	000	000	000		

Table 1: State Transition Table for Each Opcode Instruction and State

The second table details the expected outputs for timing pulses for each opcode instruction in accordance with each successive clock signal. This table has been recreated in this report and is reflected in Table 2 (found on next page):

	Output	Load (000)	Store (001)	Add (010)	Sub (011)	Inc (100)	Dec (101)	Bra (110)	Beq (111)
Current State	**000**	T0	T0	T0	T0	T0	T0	T0	T0
	001	T1	T1	T1	T1	T1	T1	T1	T1
	010	T2	T2	T2	T2	T2	T2	T2	T2
	011	T3	T3	T3	T3	T3	T3	T3	T3
	100	T4	T4	T4	T4	T4	T4	T4	T4

	Output	Load (000)	Store (001)	Add (010)	Sub (011)	Inc (100)	Dec (101)	Bra (110)	Beq (111)
Current State	000	T0	T0	T0	T0	T0	T0	T0	T0
	001	T1	T1	T1	T1	T1	T1	T1	T1
	101	T5	T5	T5	T5	T5	T5		
	110			T6	T6	T6	T6		
	111			T7	T7	T7	T7		

Table 2: Output for Each Opcode Instruction and State

The third table is very similar to table two, describing the expected behavior for the Execute output:

	Execute	Load (000)	Store (001)	Add (010)	Sub (011)	Inc (100)	Dec (101)	Bra (110)	Beq (111)
Current State	000	0	0	0	0	0	0	0	0
	001	0	0	0	0	0	0	0	0
	010	0	0	0	0	0	0	0	0
	011	0	0	0	0	0	0	0	0
	100	1	1	1	1	1	1	1	1
	101	1	1	1	1	1	1		
	110			1	1	1	1		
	111			1	1	1	1		

Table 3: Execute Output for Each Opcode Instruction and State

The suggested structure was to break the circuit down into three separate embedded circuits. The first embedded circuit was to be the ClearLogic; it's purpose is to account for the special cases for the Load, Store, Bra, and Beq opcodes, which transition back to state after reaching state 4 (Load, Store) or 5 (Bra, Beq). A truth table was handwritten describing the expected output, and boolean algebra was used to reduce the expression to sum-of-minterms form:

Clear = Op1'Op2'T5 + Op1'Op2'T4

This ensures that the FSM will correctly transition to state 000 for the correct opcode and state. This expression was the converted into circuit form, and is shown in Figure 2:

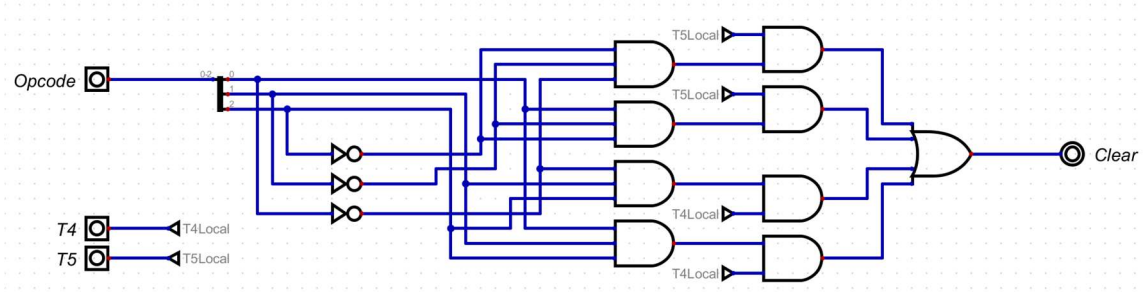


Figure 2: ClearLogic Circuit Implementation

The second embedded circuit was to be the 3-Bit Binary Counter. This circuit is to handle the state transitions based on the signal from the ClearLogic Circuit, as well as a clock input. Similarly, equations were created based on a hand-drawn truth table and are as follows:

Q0 = Clear'Q0'

Q1 = Clear'(Q1'Q0 + Q1Q0')

Q2 = Clear'(Q2'Q1Q0 + Q2Q1' + Q2Q0')

These were converted to circuit form, in conjunction with D flip-flops to ensure state transitions occur with rising clock edges. The circuit implementation is shown in Figure 3:

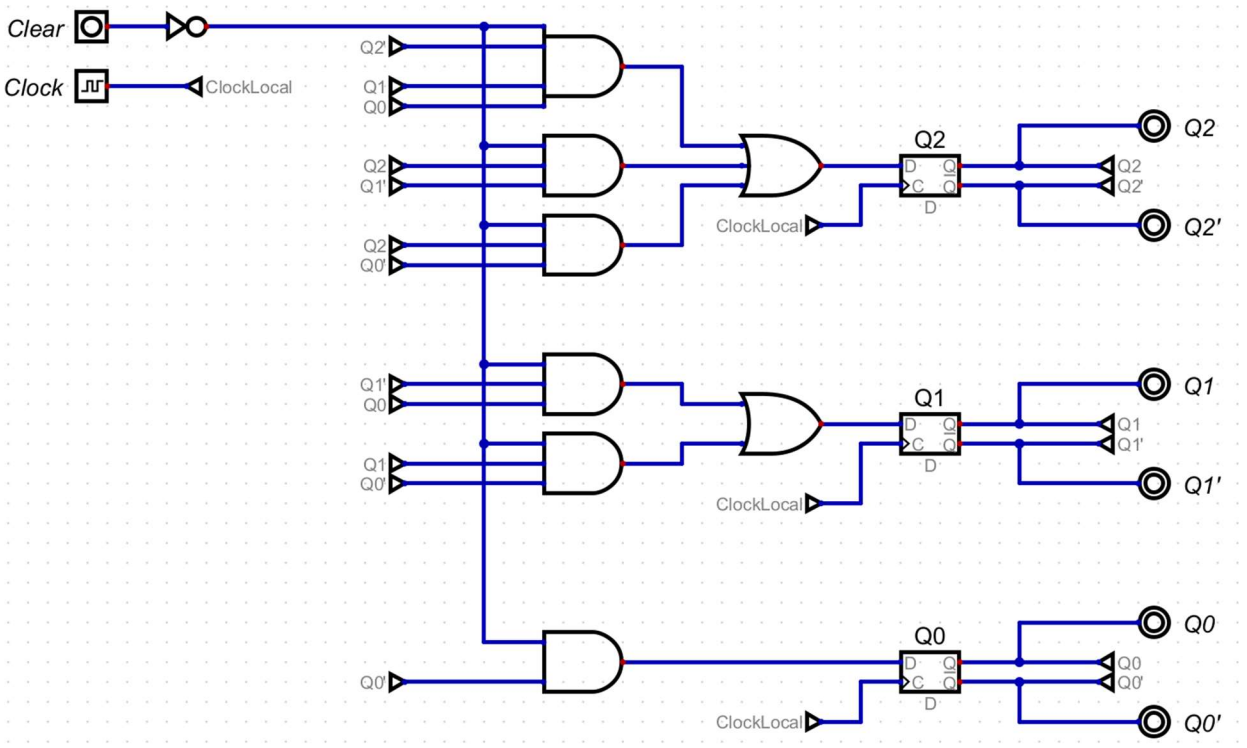


Figure 3: 3-Bit Counter Circuit Implementation

The third and final embedded circuit was the OutputLogic. As suggested by the name, this handles the logic for the expected output based on the current state, which was encoded in binary by the 3-Bit Binary Counter. Mirroring the prior two embedded circuits, logical expressions with derived from a truth table and simplified (found on next page):

T0 = Q2'Q1'Q0'

T1 = Q2'Q1'Q0

T2 = Q2'Q1Q0'

T3 = Q2'Q1Q0

T4 = Q2Q1'Q0'

T5 = Q2Q1'Q0

T6 = Q2Q1Q0'

T7 = Q2Q1Q0

Execute = T4 + T5 + T6 + T7

These equations were then accordingly converted to circuit form, as shown in Figure 4:

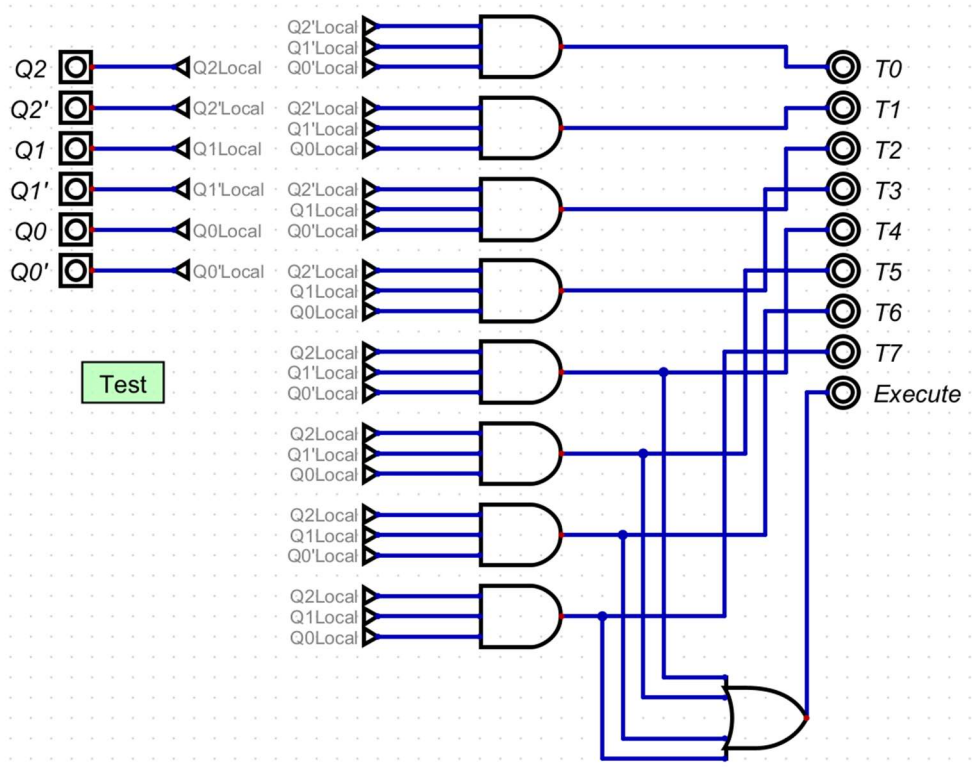


Figure 4: Output Logic Circuit Implementation

Finally, all three embedded circuits were wired up together in a larger Instruction Sequencer FSM circuit, which itself was embedded in a test bench. Both circuits are shown in Figures 5, 6, respectively:

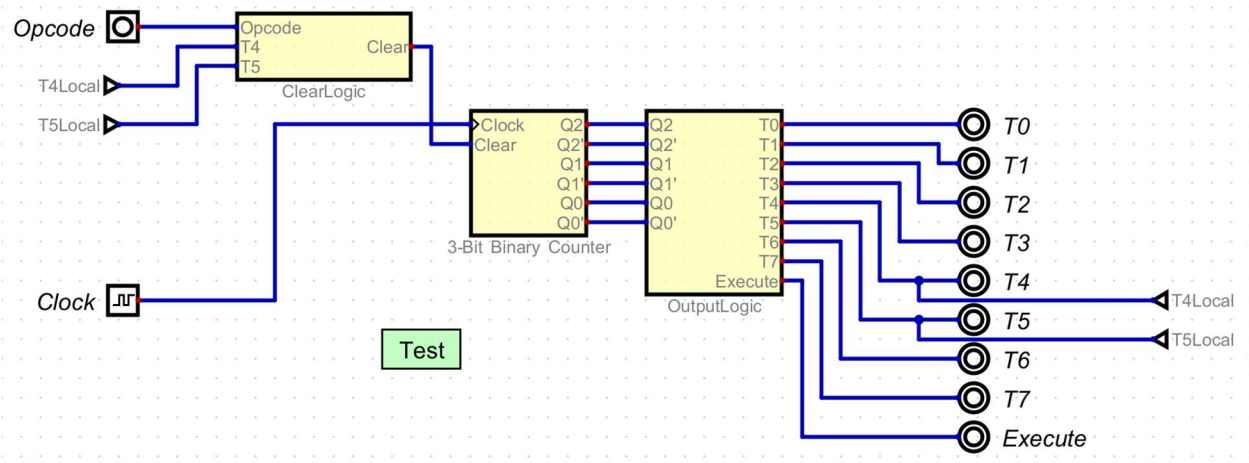


Figure 5: Instruction Sequencer FSM Circuit Implementation

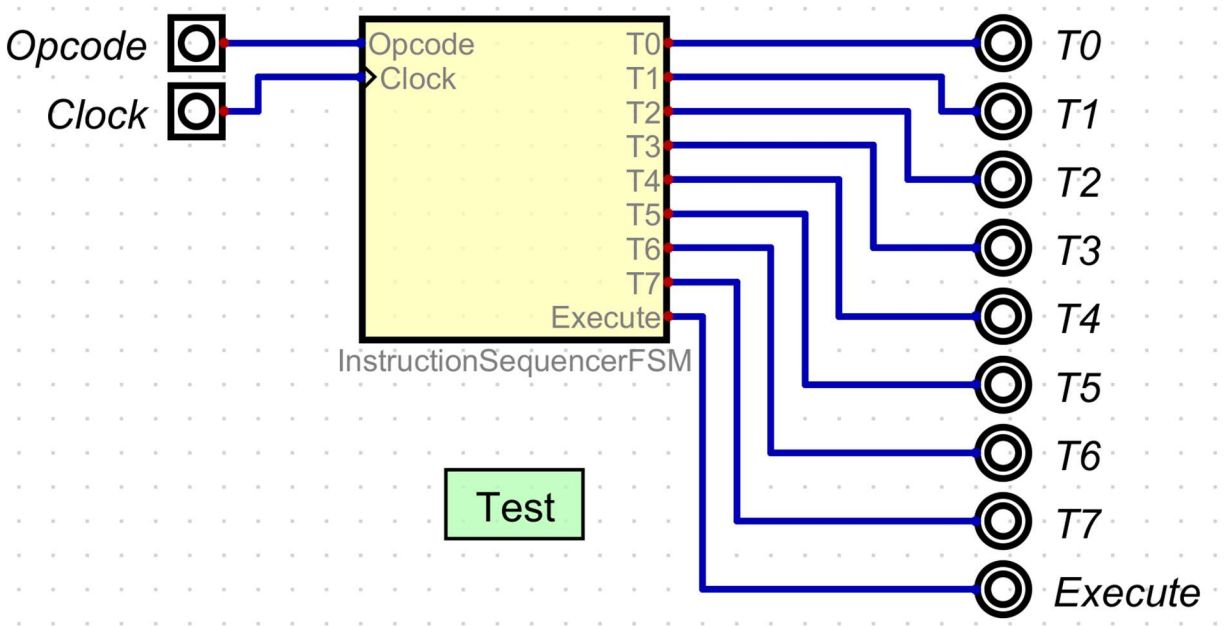


Figure 6: Instruction Sequencer FSM Test Bench

3 Numerical Verification

A very large list of test vectors were loaded into a test case and applied to the circuit via the test bench. These test vectors as well as proof that the circuit correctly passed the test case are shown in Figures 7, 8, 9:

	Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
2	0	0	1	0	0	0	0	0	0	0	0
3	1	0	0	1	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0
5	1	0	0	0	1	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0
7	1	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0	0	0
9	1	0	0	0	0	0	1	0	0	0	1
10	0	0	0	0	0	0	1	0	0	0	1
11	1	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	1	0	0	1
13	#Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
14	1	1	1	0	0	0	0	0	0	0	0
15	0	1	1	0	0	0	0	0	0	0	0
16	1	1	0	1	0	0	0	0	0	0	0
17	0	1	0	1	0	0	0	0	0	0	0
18	1	1	0	0	1	0	0	0	0	0	0
19	0	1	0	0	1	0	0	0	0	0	0
20	1	1	0	0	0	1	0	0	0	0	0
21	0	1	0	0	0	1	0	0	0	0	0
22	1	1	0	0	0	0	1	0	0	0	1
23	0	1	0	0	0	0	1	0	0	0	1
24	1	1	0	0	0	0	0	1	0	0	1
25	0	1	0	0	0	0	0	1	0	0	1
26	#Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
27	1	2	1	0	0	0	0	0	0	0	0
28	0	2	1	0	0	0	0	0	0	0	0
29	1	2	0	1	0	0	0	0	0	0	0
30	0	2	0	1	0	0	0	0	0	0	0
31	1	2	0	0	1	0	0	0	0	0	0
32	0	2	0	0	1	0	0	0	0	0	0
33	1	2	0	0	0	1	0	0	0	0	0
34	0	2	0	0	0	1	0	0	0	0	0
35	1	2	0	0	0	0	1	0	0	0	1
36	0	2	0	0	0	0	1	0	0	0	1
37	1	2	0	0	0	0	0	1	0	0	1
38	0	2	0	0	0	0	0	1	0	0	1
39	1	2	0	0	0	0	0	0	1	0	1
40	0	2	0	0	0	0	0	0	1	0	1
41	1	2	0	0	0	0	0	0	0	1	1
42	0	2	0	0	0	0	0	0	0	1	1

	#Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
44	1	3	1	0	0	0	0	0	0	0	0
45	0	3	1	0	0	0	0	0	0	0	0
46	1	3	0	1	0	0	0	0	0	0	0
47	0	3	0	1	0	0	0	0	0	0	0
48	1	3	0	0	1	0	0	0	0	0	0
49	0	3	0	0	1	0	0	0	0	0	0
50	1	3	0	0	0	1	0	0	0	0	0
51	0	3	0	0	0	1	0	0	0	0	0
52	1	3	0	0	0	0	1	0	0	0	1
53	0	3	0	0	0	0	1	0	0	0	1
54	1	3	0	0	0	0	0	1	0	0	1
55	0	3	0	0	0	0	0	1	0	0	1
56	1	3	0	0	0	0	0	0	1	0	1
57	0	3	0	0	0	0	0	0	1	0	1
58	1	3	0	0	0	0	0	0	0	1	1
59	0	3	0	0	0	0	0	0	0	1	1
60	#Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
61	1	4	1	0	0	0	0	0	0	0	0
62	0	4	1	0	0	0	0	0	0	0	0
63	1	4	0	1	0	0	0	0	0	0	0
64	0	4	0	1	0	0	0	0	0	0	0
65	1	4	0	0	1	0	0	0	0	0	0
66	0	4	0	0	1	0	0	0	0	0	0
67	1	4	0	0	0	1	0	0	0	0	0
68	0	4	0	0	0	1	0	0	0	0	0
69	1	4	0	0	0	0	1	0	0	0	1
70	0	4	0	0	0	0	1	0	0	0	1
71	1	4	0	0	0	0	0	1	0	0	1
72	0	4	0	0	0	0	0	1	0	0	1
73	1	4	0	0	0	0	0	0	1	0	1
74	0	4	0	0	0	0	0	0	1	0	1
75	1	4	0	0	0	0	0	0	0	1	1
76	0	4	0	0	0	0	0	0	0	1	1

	#Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
78	1	5	1	0	0	0	0	0	0	0	0
79	0	5	1	0	0	0	0	0	0	0	0
80	1	5	0	1	0	0	0	0	0	0	0
81	0	5	0	1	0	0	0	0	0	0	0
82	1	5	0	0	1	0	0	0	0	0	0
83	0	5	0	0	1	0	0	0	0	0	0
84	1	5	0	0	0	1	0	0	0	0	0
85	0	5	0	0	0	1	0	0	0	0	0
86	1	5	0	0	0	0	1	0	0	0	1
87	0	5	0	0	0	0	1	0	0	0	1
88	1	5	0	0	0	0	0	1	0	0	1
89	0	5	0	0	0	0	0	1	0	0	1
90	1	5	0	0	0	0	0	0	1	0	1
91	0	5	0	0	0	0	0	0	1	0	1
92	1	5	0	0	0	0	0	0	0	1	1
93	0	5	0	0	0	0	0	0	0	1	1
94	#Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
95	1	6	1	0	0	0	0	0	0	0	0
96	0	6	1	0	0	0	0	0	0	0	0
97	1	6	0	1	0	0	0	0	0	0	0
98	0	6	0	1	0	0	0	0	0	0	0
99	1	6	0	0	1	0	0	0	0	0	0
100	0	6	0	0	1	0	0	0	0	0	0
101	1	6	0	0	0	1	0	0	0	0	0
102	0	6	0	0	0	1	0	0	0	0	0
103	1	6	0	0	0	0	1	0	0	0	1
104	0	6	0	0	0	0	1	0	0	0	1
105	#Clock	Opcode	T0	T1	T2	T3	T4	T5	T6	T7	Execute
106	1	7	1	0	0	0	0	0	0	0	0
107	0	7	1	0	0	0	0	0	0	0	0
108	1	7	0	1	0	0	0	0	0	0	0
109	0	7	0	1	0	0	0	0	0	0	0
110	1	7	0	0	1	0	0	0	0	0	0
111	0	7	0	0	1	0	0	0	0	0	0
112	1	7	0	0	0	1	0	0	0	0	0
113	0	7	0	0	0	1	0	0	0	0	0
114	1	7	0	0	0	0	1	0	0	0	1
115	0	7	0	0	0	0	1	0	0	0	1

Figure 7: Circuit Test Vectors

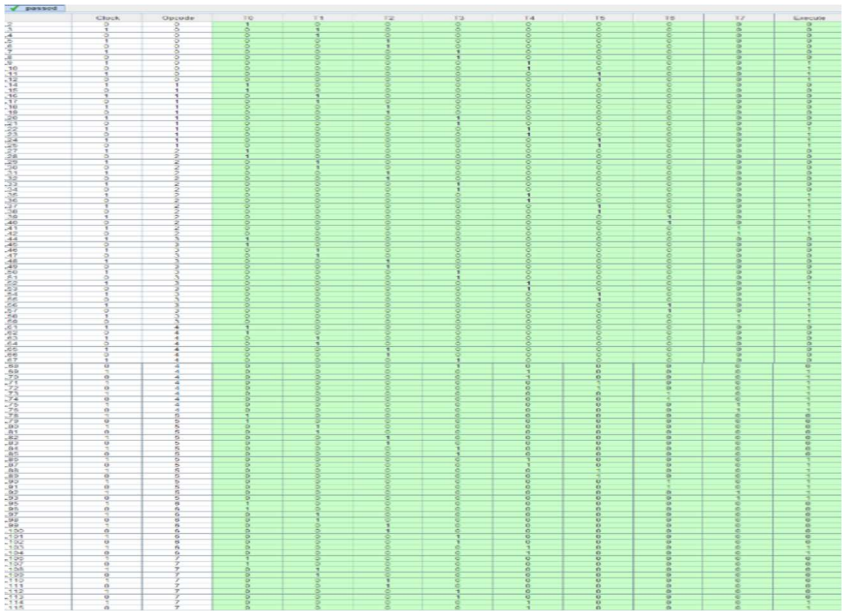


Figure 8: Test Case Passed

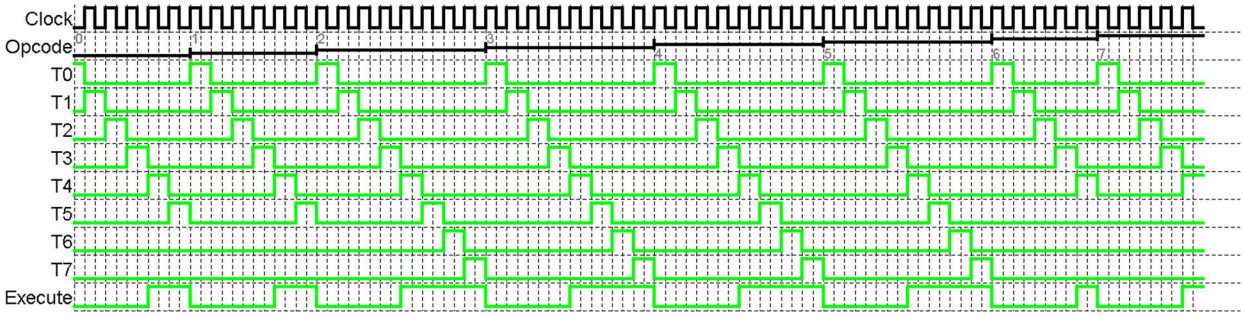


Figure 9: Test Case Passed in Graphical Form

4 Summary

The goal of this project assignment was to create a circuit that could interpret a 3-bit opcode input and correctly generate the expected outputs based on state transitions according to a clock signal.

Several truth tables were used to define the circuit's expected state transitions, as well as outputs in accordances to said states. These were obtained by interpretation of an FSM diagram, converting its states and state transitions into said truth tables. Logical expressions were derived from the tables and reduced using boolean algebra, which were then converted into circuit form in the Digital logic simulator. These circuit implementations made use of AND/OR/NOT logic gates, as well as D flip-flops. A set of input test vectors was tested against this circuit, and all inputs produced the expected output for each test vector. Therefore, the specified digital circuit design satisfies the problem set out to be solved in the original problem statement.

4. Datapath

1 Problem Statement

Design a circuit to act as the datapath in the simple computer system. The datapath's purpose is to store data that defines the state of the CPU at any given time. This information is stored in several registers: the Memory Address Register (MAR), Memory Buffer Register (MBR), Instruction Register (IR), Program Counter (PC), and Data Register (D0). The datapath also contains an Arithmetic Logic Unit (ALU) which performs addition, subtraction, incrementation, and decrementation as arithmetic operations.

2 Analytical Design

A circuit was devised that correlated inputs to their respective registers, as well as the ALU inputs to the ALU embedded circuit. This circuit design closely mimics the figure provided in the assignment description depicting a simple CPU datapath. The Memory Address Register was connected to the AddressBus output and the Instruction Register connected to the Opcode output. Splitters were used to separate the 8-bit data bus input into 5-bit and 3-bit outputs for the AddressBus and Opcode outputs, respectively. The circuit design of the datapath is shown in Figure 1:

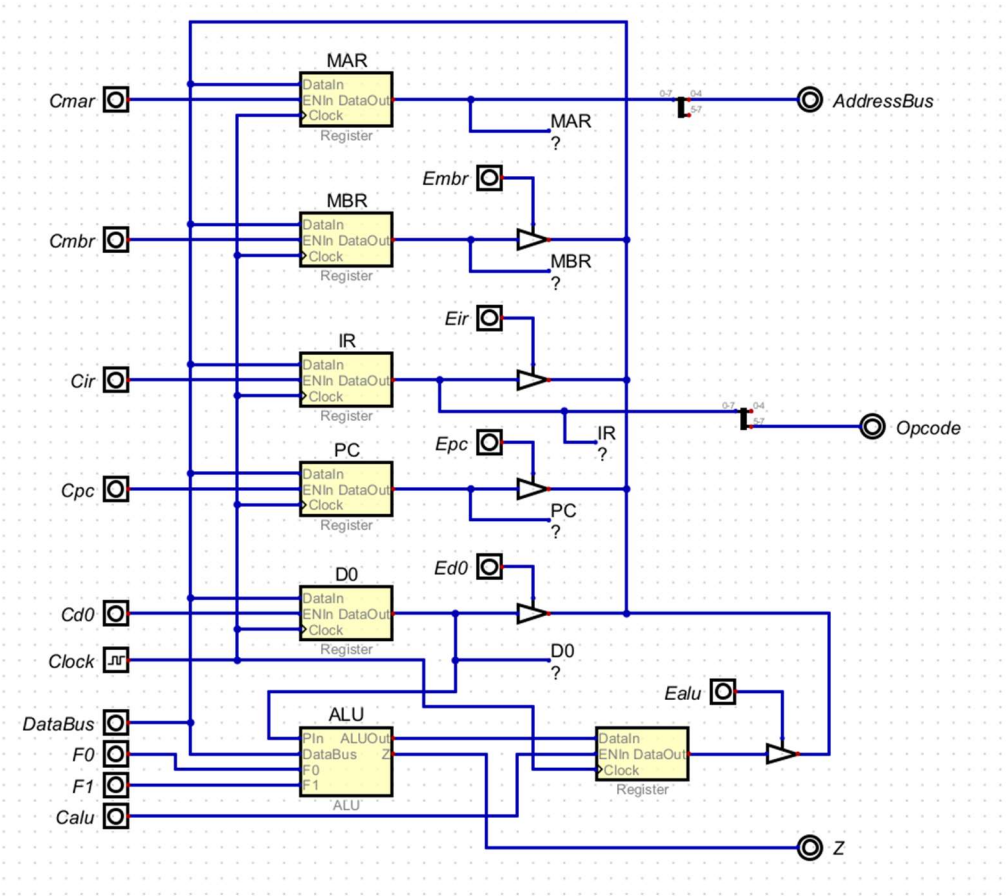


Figure 1: Circuit Diagram of Datapath

The registers were designed through use of multiplexers and D-flip-flops, with the ENIn input acting as the selector signal for the multiplexers. This multiplexer/flip-flop combination was used for each of the 8 bits that the DataIn input was split into. The flip-flops then output their loaded values upon a rising edge of clock input, thus coming together to form a simple register. The circuit design of the register is shown in Figure 2:

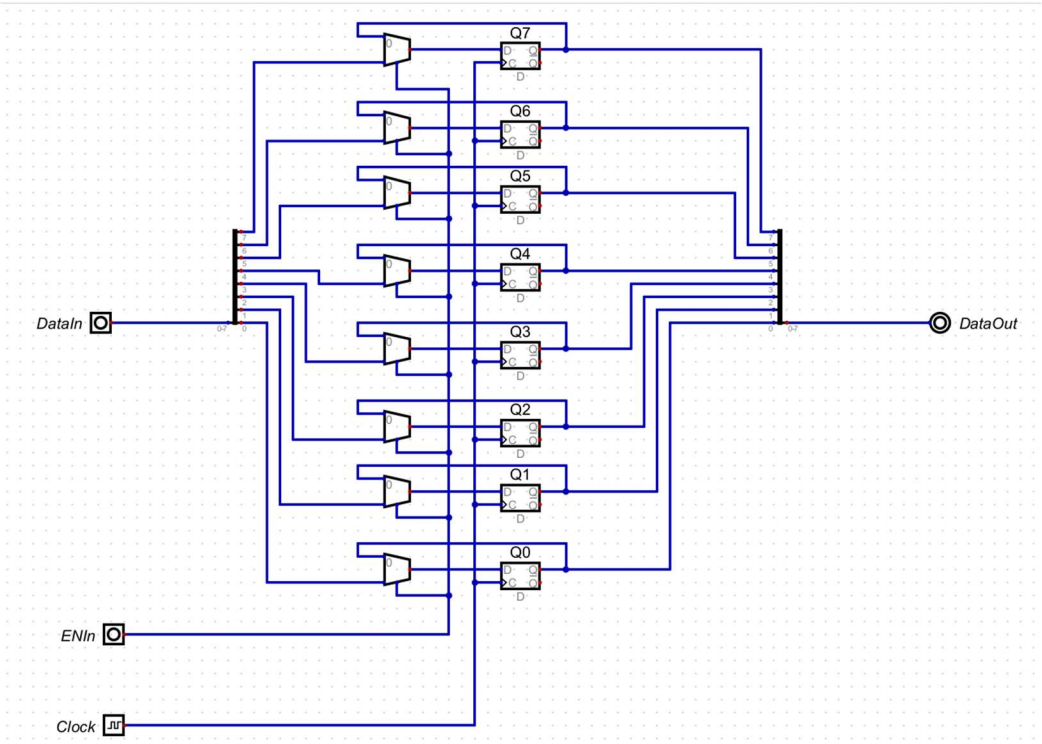


Figure 2: Circuit Diagram of Register

The ALU was designed as a ripple carry adder, with the operation being executed depending on the F0 and F1 inputs. The F0 and F1 inputs were put together with a splitter and acted as selector bits for a 2-bit Decoder. Within the overall structure ripple carry adder structure of the ALU, a full adder was used for each bit. The full adder was designed using a combination of logic gates, derived from the expressions:

$$S = (A \text{ XOR } B) \text{ XOR } Cin = (AB' + A'B)Cin' + (AB' + A'B)'Cin$$

$$Cout = AB + BCin + ACin$$

The circuit implementation of the full adder is shown in Figure 3:

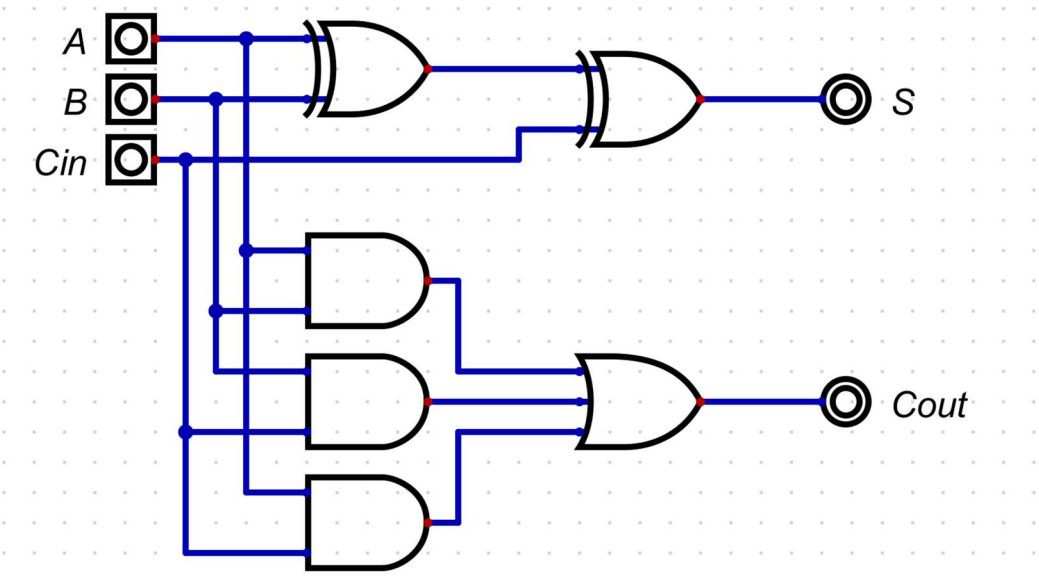


Figure 3: Circuit Diagram of Full Adder

The ALU circuit implementation with the full adders and input decoder are shown in Figure 4:

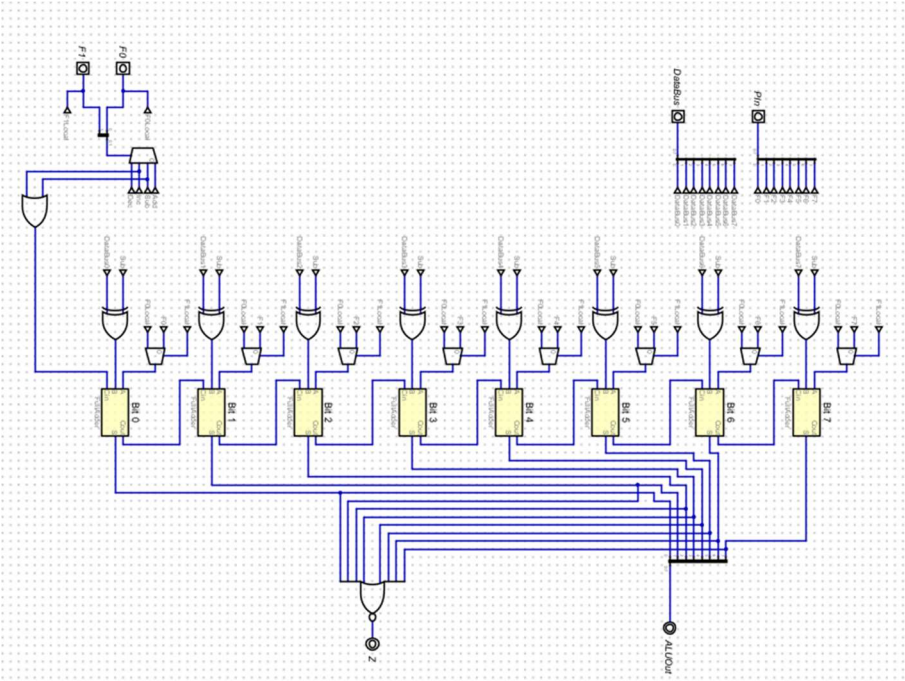


Figure 4: Circuit Diagram of Arithmetic and Logic Unit

The datapath was then implemented as an imbedded circuit inside of a simple computer system, shown in Figure 5:

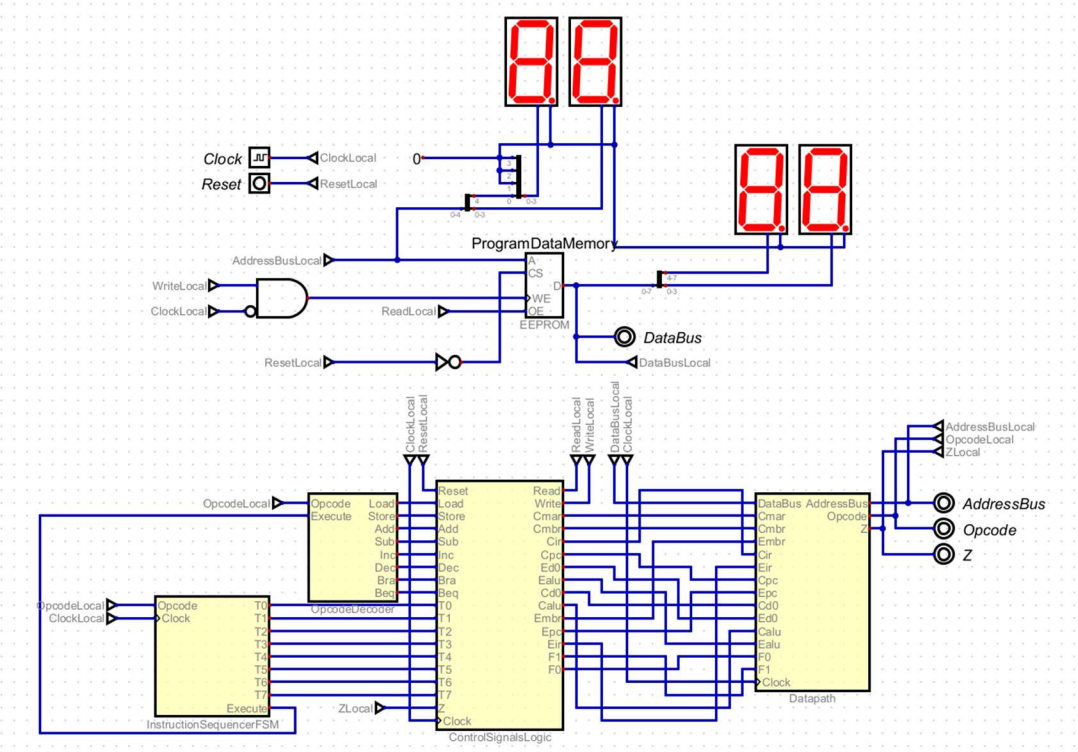


Figure 5: Simple Computer System

The simple computer system was then used as a means for verifying the functionality of the datapath. Test vectors were then loaded and tested against this computer system.

3 Numerical Verification

Test vectors were loaded into a test case in the digital logic simulator as described in the assignment description. All test vectors passed and are shown in multiple forms in Figures 6, 7, 8:

Test data

Reset	Clock	AddressBus	DataBus	MAR	MBR	IR	PC	D0	Opcode	Z
0	0	0	0	0	0	0	0	0	0	1
0	1	0	25	0	0	0	0	0	0	0
0	0	0	25	0	0	0	0	0	0	0
0	1	0	0	0	0	25	0	0	0	0
0	0	0	0	0	0	25	0	0	0	0
0	1	0	1	0	0	25	0	0	0	0
0	0	0	1	0	0	25	0	0	0	0
0	1	0	25	0	0	25	1	0	0	0
0	0	0	25	0	0	25	1	0	0	0
0	1	25	6	25	0	25	1	0	0	0
0	0	25	6	25	0	25	1	0	0	0
0	1	25	1	25	0	25	1	6	0	0
0	0	25	1	25	0	25	1	6	0	0
0	1	1	83	1	0	25	1	6	0	0
0	0	1	83	1	0	25	1	6	0	0
0	1	1	1	1	0	83	1	6	2	0
0	0	1	1	1	0	83	1	6	2	0
0	1	1	2	1	0	83	1	6	2	0
0	0	1	2	1	0	83	1	6	2	0
0	1	1	83	1	0	83	2	6	2	0
0	0	1	83	1	0	83	2	6	2	0
0	1	19	185	83	0	83	2	6	2	0
0	0	19	185	83	0	83	2	6	2	0
0	1	19	185	83	185	83	2	6	2	0
0	0	19	185	83	185	83	2	6	2	0
0	1	19	191	83	185	83	2	6	2	0
0	0	19	191	83	185	83	2	6	2	0
0	1	19	2	83	185	83	2	191	2	0
0	0	19	2	83	185	83	2	191	2	0
0	1	2	51	2	185	83	2	191	2	0
0	0	2	51	2	185	83	2	191	2	0
0	1	2	2	2	185	51	2	191	1	0
0	0	2	2	2	185	51	2	191	1	0
0	1	2	3	2	185	51	2	191	1	0
0	0	2	3	2	185	51	2	191	1	0
0	1	2	51	2	185	51	3	191	1	0
0	0	2	51	2	185	51	3	191	1	0
0	1	19	191	51	185	51	3	191	1	0
0	0	19	191	51	185	51	3	191	1	0
0	1	19	3	51	185	51	3	191	1	0
0	0	19	3	51	185	51	3	191	1	0

Figure 6: Test Vector Data

passed

	Reset	Clock	Addres...	DataBus	MAR	MBR	IR	PC	D0	Opcode	Z
L2	0	0	0	0	0	0	0	0	0	0	1
L3	0	1	0	0x19	0	0	0	0	0	0	0
L4	0	0	0	0x19	0	0	0	0	0	0	0
L5	0	1	0	0	0	0	0x19	0	0	0	0
L6	0	0	0	0	0	0	0x19	0	0	0	0
L7	0	1	0	1	0	0	0x19	0	0	0	0
L8	0	0	0	1	0	0	0x19	0	0	0	0
L9	0	1	0	0x19	0	0	0x19	1	0	0	0
L10	0	0	0	0x19	0	0	0x19	1	0	0	0
L11	0	1	0x19	6	0x19	0	0x19	1	0	0	0
L12	0	0	0x19	6	0x19	0	0x19	1	0	0	0
L13	0	1	0x19	1	0x19	0	0x19	1	6	0	0
L14	0	0	0x19	1	0x19	0	0x19	1	6	0	0
L15	0	1	1	0x53	1	0	0x19	1	6	0	0
L16	0	0	1	0x53	1	0	0x19	1	6	0	0
L17	0	1	1	1	1	0	0x53	1	6	2	0
L18	0	0	1	1	1	0	0x53	1	6	2	0
L19	0	1	1	2	1	0	0x53	1	6	2	0
L20	0	0	1	2	1	0	0x53	1	6	2	0
L21	0	1	1	0x53	1	0	0x53	2	6	2	0
L22	0	0	1	0x53	1	0	0x53	2	6	2	0
L23	0	1	0x13	B9	0x53	0	0x53	2	6	2	0
L24	0	0	0x13	B9	0x53	0	0x53	2	6	2	0
L25	0	1	0x13	B9	0x53	B9	0x53	2	6	2	0
L26	0	0	0x13	B9	0x53	B9	0x53	2	6	2	0
L27	0	1	0x13	BF	0x53	B9	0x53	2	6	2	0
L28	0	0	0x13	BF	0x53	B9	0x53	2	6	2	0
L29	0	1	0x13	2	0x53	B9	0x53	2	BF	2	0
L30	0	0	0x13	2	0x53	B9	0x53	2	BF	2	0
L31	0	1	2	0x33	2	B9	0x53	2	BF	2	0
L32	0	0	2	0x33	2	B9	0x53	2	BF	2	0
L33	0	1	2	2	2	B9	0x33	2	BF	1	0
L34	0	0	2	2	2	B9	0x33	2	BF	1	0
L35	0	1	2	3	2	B9	0x33	2	BF	1	0
L36	0	0	2	3	2	B9	0x33	2	BF	1	0
L37	0	1	2	0x33	2	B9	0x33	3	BF	1	0
L38	0	0	2	0x33	2	B9	0x33	3	BF	1	0
L39	0	1	0x13	BF	0x33	B9	0x33	3	BF	1	0
L40	0	0	0x13	BF	0x33	B9	0x33	3	BF	1	0
L41	0	1	0x13	3	0x33	B9	0x33	3	BF	1	0
L42	0	0	0x13	3	0x33	B9	0x33	3	BF	1	0

Figure 7: Test Case Passed

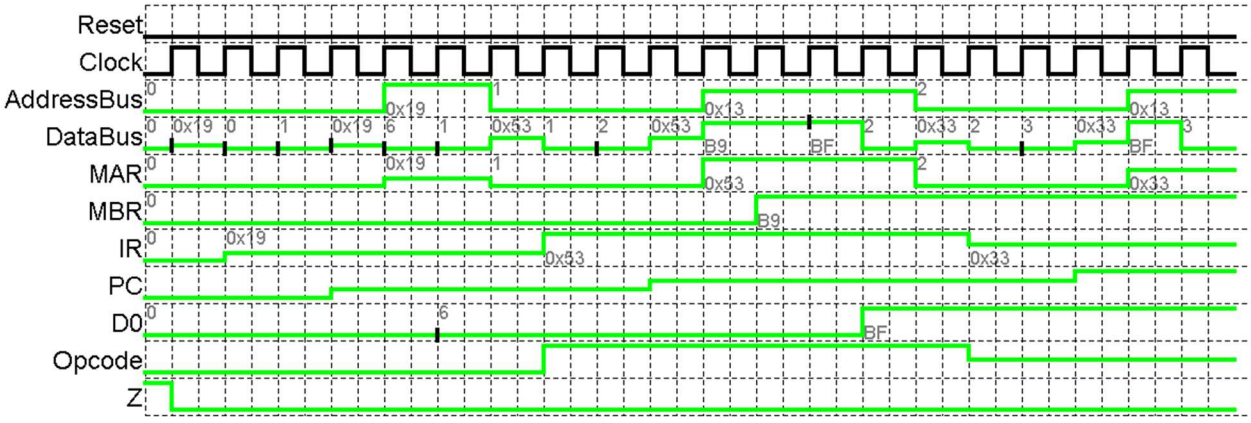


Figure 8: Test Vectors in Graphical Form

4 Summary

The goal of this project assignment was to create a circuit that could act as the datapath component to the simple computer system.

This was achieved by mirroring the design specifications given by the figure provided in the assignment description and related kick-off slides. Registers were designed through implementation of flip-flops, and the ALU was designed as a carry ripple adder, mirroring an implementation using full adders shown in lecture slides, as well as textbook content. The provided set of text vectors were loaded into the simulator and all values passed, verifying the design's functionality. Therefore, the specified digital circuit design satisfies the problem set out to be solved in the original problem statement.

5. Simple Computer System

1 Problem Statement

Design a simple computer system using all of the components designed up to this point in ECE 2330. These culminating components act as embedded circuits to create a hierarchical design of a computer system. This design will then be verified by implementing memory-mapped input/output and running two programs loaded into memory.

2 Analytical Design

Mirroring the first figure of the assignment description, a circuit was devised that connects a simple CPU to memory (EEPROM), which is where a program will be loaded into in order to verify the functionality of the computer system.

The first program, provided by the assignment description, is a self modifying program that employs use of all eight instructions, all components of the central processing unit, and all locations in memory.

After the first program was run in the computer system and confirmed to function correctly, a memory-mapped I/O device was appended to the system, which includes an address decoder, port register, and two seven-segment hex displays. The address decoder is an embedded circuit that returns true when the address being accessed is *MA = 0x1F*. This then enables the port register, in turn enabling the two seven-segment displays, which will display the increment/decrement operations. When the port register is not enabled, the program proceeds as detailed previously.

A second program was then written, described by Professor Johnson as, “more visually pleasing”. Upon starting the program, the displays will initialize to *00*, and sequentially increment up to *0A*, after which the displays will decrement back down to *00*. This cycle will continue until the program is terminated.

The address decoder was a relatively straightforward component, as it only returns true for one particular set of bits corresponding to the value *0x1F* (all bits are 1). The boolean algebra expression for the chip select output (CS) is defined:

$$\mathbf{CS0 = (AddressBus_0)(AddressBus_1)(AddressBus_2)(AddressBus_3)(AddressBus_4)}$$

The output *CS0* was then connected to the port register so that it would be enabled when the desired memory address was accessed. The address decoder’s second output, *CS1*, was defined as the complement to *CS0*. *CS1* was connected to the EEPROM so that when an address other than *0x1F* was accessed, the program would proceed as described. The circuit design of the address decoder is shown in Figure 1:

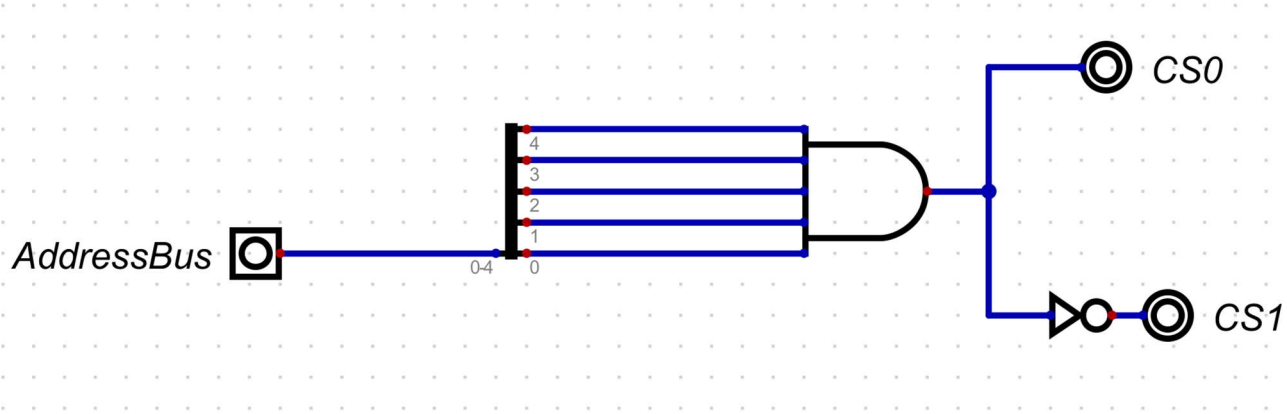


Figure 1: Circuit Design of Address Decoder

A port register was then designed that would be enabled by the address decoder, and output data to the displays. The write signal of the CPU is used as the clock input to the register, and the CS output of the address decoder was used as an enable signal. A tri-state device was then used so that the content of the port register does not change unless a write command occurs. This component has the ability to either read what is stored in the register and push it onto the databus, or to write what is on the databus to the register, depending on the boolean value of chip select input. The circuit design of the port register is shown in Figure 2:

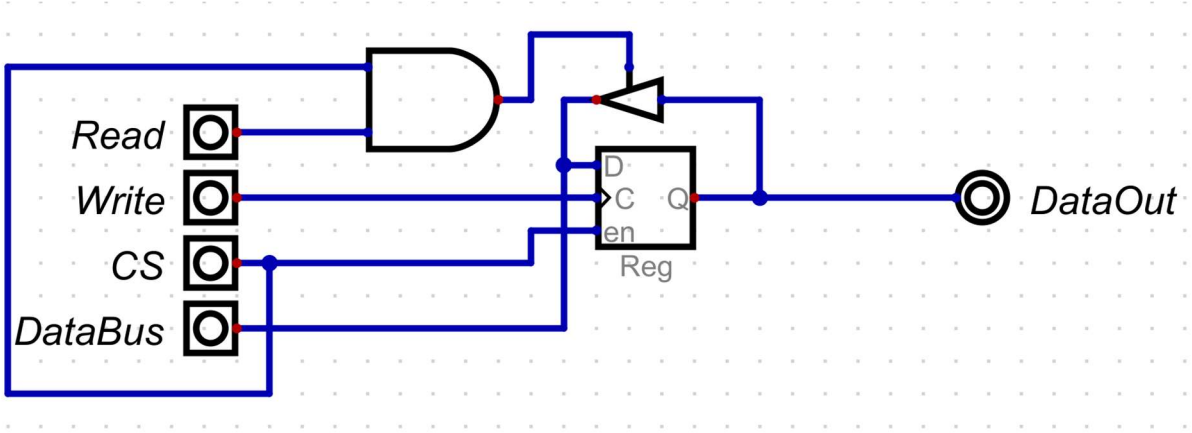


Figure 2: Circuit Design of Port Register

All of the above components were then placed into the Digital logical simulator, and their inputs and outputs were connected to form the simple computer system, now with a memory-mapped I/O device implemented. The circuit design of this simple computer is shown in Figure 3:

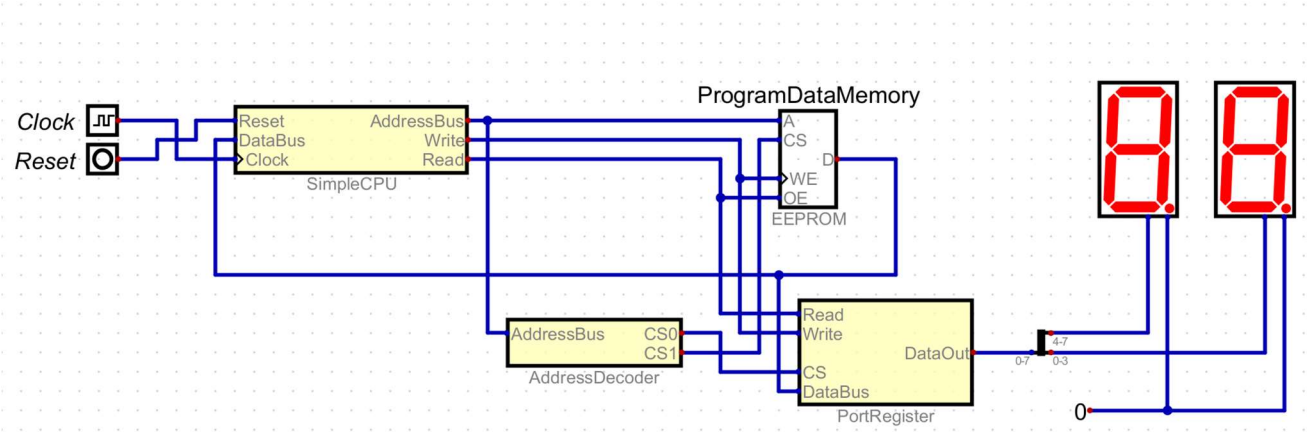


Figure 3: Circuit Design of Simple Computer System with Memory-Mapped I/O

3 Verification

Unlike previous project assignments, test vectors were not used to verify the functionality of the circuit. Because there are no outputs to measure, the functionality of the circuit was verified by simply observing the behavior of the system, both in memory and on the seven-segment displays. The two programs specified by the assignment description were loaded into the EEPROM on two separate files, and the circuit was run with the clock input set up as a real-time clock with a frequency of 10Hz. Both programs behaved as expected. The values loaded into the EEPROM for the first program are shown in Figure 4; the second program, Figure 5:

Add...	0x00	0x01	0x02	0x03
0x00	0x19	0x53	0x33	0x19
0x04	0x55	0x35	D3	0x13
0x08	0x79	0x33	0x15	0x79
0x0C	0x35	B9	F0	C0
0x10	0x18	0x39	C0	B9
0x14	F7	0x99	E7	D7
0x18	6	6	0	0
0x1C	0	0	0	0

Figure 4: EEPROM Values for Self-Modifying Program

Add...	0x00	0x01	0x02	0x03
0x00	0x18	3F	9F	0x19
0x04	7F	E7	C2	BF
0x08	1F	5F	E0	C7
0x0C	0	0	0	0
0x10	0	0	0	0
0x14	0	0	0	0
0x18	0	A	0	0
0x1C	0	0	0	0

Figure 3: EEPROM Values for Incrementing/Decrementing Display Program

4 Summary

The goal of this project assignment was to be a summative project that uses all of the components designed in this course over the past fourteen weeks.

This was achieved by mirroring the design specifications given by the figure provided in the assignment description and related kick-off slides. Components were implemented to create a central processing unit, which was then used as an embedded circuit in a larger circuit, which used memory, decoder, and register components to run two simple, self-modifying programs, which made use of every component designed, as well as every memory address. The programs both ran successfully and behaved as intended, so this project has succeeded in achieving the objectives set out to be accomplished by the assignment statement.

I'd like to thank Professor Johnson for teaching this course. As a computer science student, I was never aware or interested in low-level programming and data representation. ECE 2330 has shed light on how difficult it is to design computer systems, and I have a newfound appreciation for electrical engineering as a field. This course truly has given new meaning to the phrase "1s and 0s".

5 Honor Pledge

On my honor as a student, I have neither given nor received unauthorized assistance on this assignment.

Winston Zhang, wyz5rge