Opcode Decoder

By Sidhardh Burre

CS/ECE 2330 Digital Logic Design

February 13, 2021

Table of Contents

Contents

Table of Figures

1 Problem Statement

Design a digital decoding circuit to decode the 3-bit opcode for the instruction format shown in Figure 1. The 8 instructions and corresponding outputs are featured in Table 1. The corresponding command should be outputted only when execute is activated.

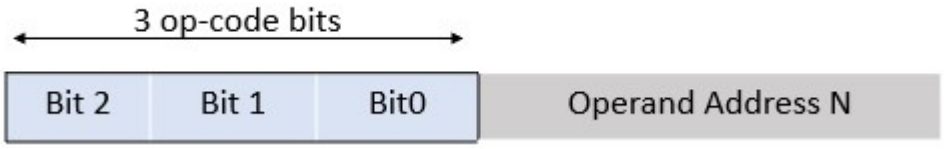

Figure 1: Simple CPU Instruction Format

Opcode	Mnemonic	Operation
000	Load N	Load D0 with the contents at memory address = N
001	Store N	Store contents of D0 at memory address = N
010	Add N	The contents at memory address = N are added to the contents of D0 and then stored in D0
011	Sub N	The contents at memory address = N are subtracted from the contents of D0 and then stored in D0
100	Inc N	The contents at memory address = N are incremented by 1
101	Dec N	The contents are memory address = N are incremented by 1
110	Bra N	The Program Counter (PC) is loaded with the memory address = N
111	Beq N	The Program Counter (PC) Is loaded with the memory address = N if the last arithmetic operation produced a result of zero (indicated by the zero bit of the ALU, Z = 1).

Table 1: Simple CPU Instructions

2 Analytical Design

The truth table for the opcode decoder is shown in Figure 2 along with the logical equations that allow the opcode input to be converted into the corresponding signal.

Table

File New Edit Create K-Map

Opc...	Opc...	Opc...	Exe...	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

All possible solutions

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

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

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

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

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

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

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

$$Beq = Execute \wedge Opcode_0 \wedge Opcode_1 \wedge Opcode_2$$

Figure 2: Truth Table and Logical Expressions for Opcode Decoder

The logical expressions were analyzed to form the circuit schematic shown in Figure 3.

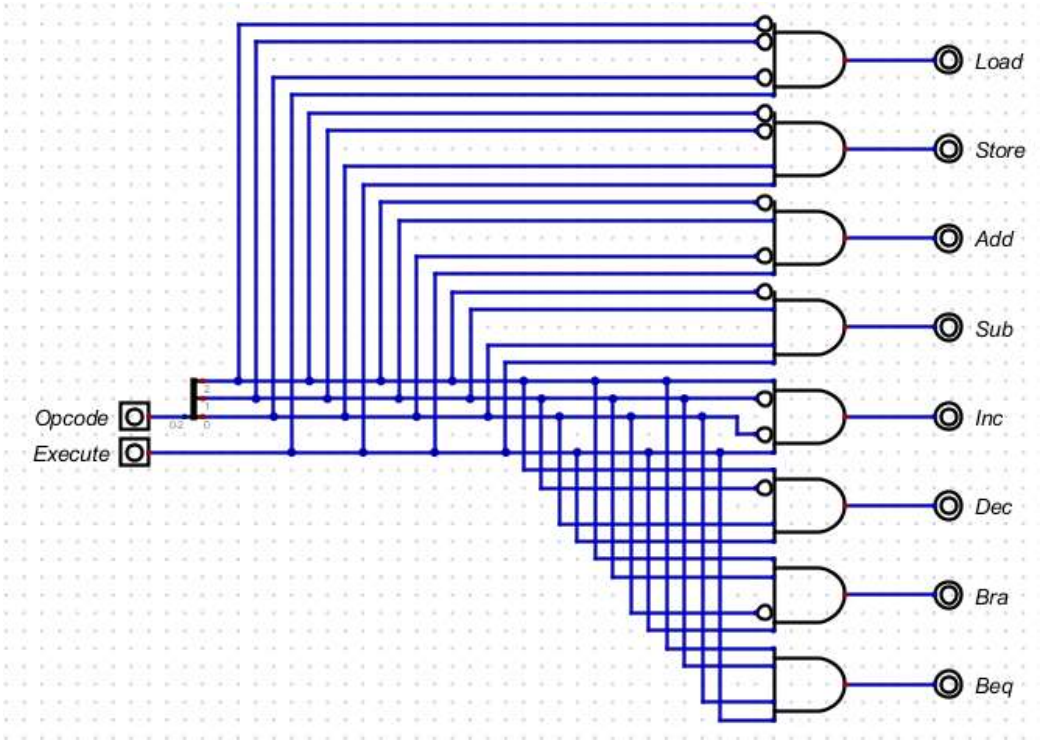

Figure 3: Circuit for Opcode Decoder

3 Numerical Verification

Shown in Figure 4, Figure 5, and Figure 6 are the numerical verification results, where an extensive set of input test vectors (Figure 4) were applied to the circuit.

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 outputs 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
0       1       1       0       0       0       0       0       0       0
#Store instruction
1       0       0       0       0       0       0       0       0       0
1       1       0       1       0       0       0       0       0       0
# Add instruction
2       0       0       0       0       0       0       0       0       0
2       1       0       0       1       0       0       0       0       0
# Subtract instruction
3       0       0       0       0       0       0       0       0       0
3       1       0       0       0       1       0       0       0       0
# Increment instruction
4       0       0       0       0       0       0       0       0       0
4       1       0       0       0       0       1       0       0       0
# Decrement instruction
5       0       0       0       0       0       0       0       0       0
5       1       0       0       0       0       0       1       0       0
# Branch instruction
6       0       0       0       0       0       0       0       0       0
6       1       0       0       0       0       0       0       1       0
# Branch if equal to zero instruction
7       0       0       0       0       0       0       0       0       0
7       1       0       0       0       0       0       0       0       1
```

Figure 4: Input Test Vectors

passed

	Opco...	Exec...	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 (Table)

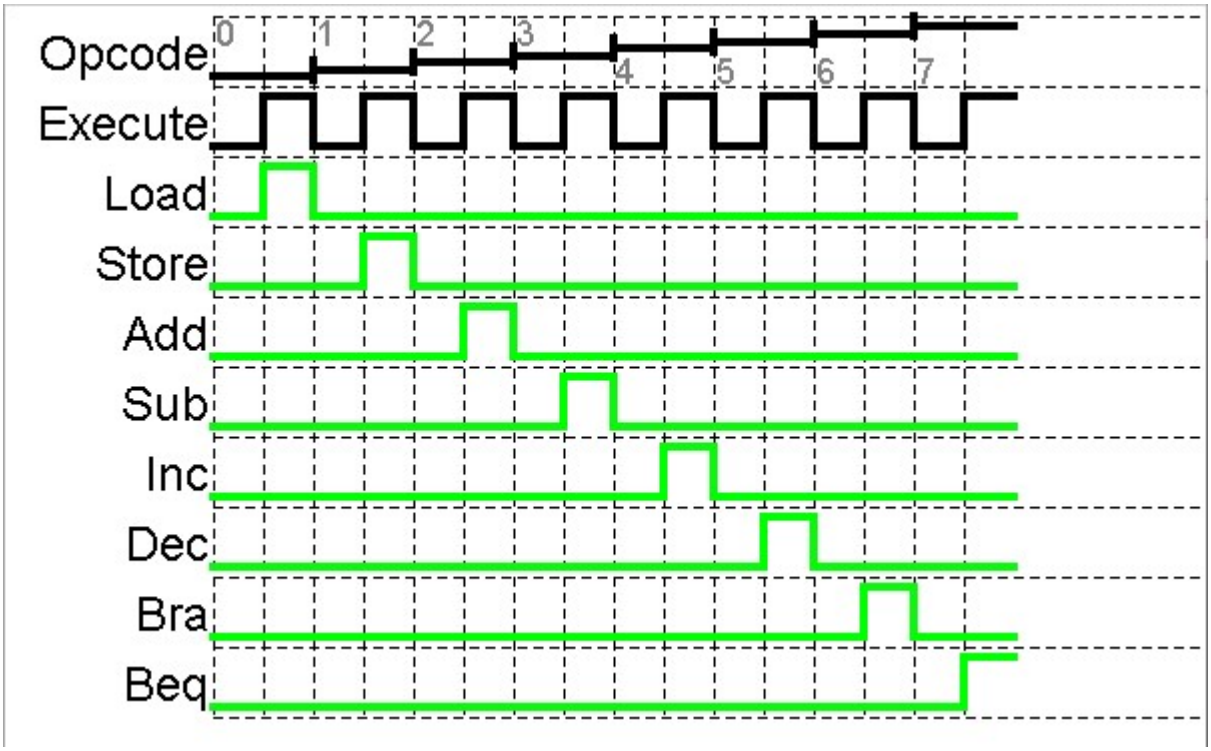

Figure 6: Test Results (Graph)

4 Summary

The problem requirement for this assignment was to design an opcode decoder circuit gated by an execute input that triggers a unique output for each unique value of the opcode input. When the execute produces a 0, nothing is output but when the execute is a 1, the corresponding opcode output is outputted.

The truth table and corresponding logical equations were essential to defining functionality and developing a logic expression, that was then used to specify a digital circuit solution. The digital circuit was tested with an extensive set of input vectors (16 test vectors for 9 inputs) and produced the expected output for each test vector. Therefore, the specified digital circuit design solution satisfies the problem requirements.