Datapath and Control Unit Implementation

1 Overview

- Each unique component of the datapath and the control unit was designed using Verilog.
- The control unit and datapath components were thoroughly simulated.
- For each component, annotated simulation results are provided.
- All Verilog component code can be seen in Appendix A.
- All test-bench code can be seen in Appendix B.

2 Components

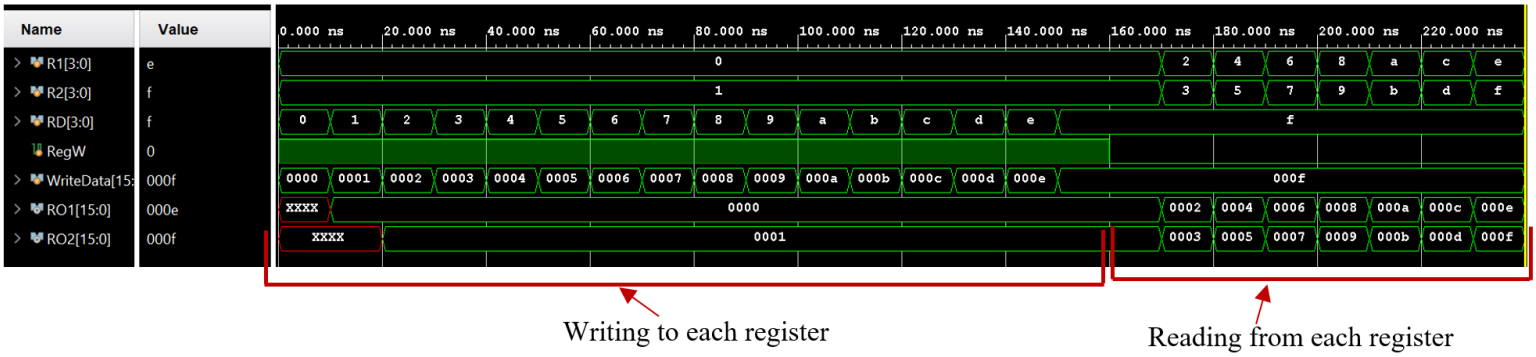
In the sections below, each component of the datapath is described and the its simulation results are discussed.

2.1 Register File

The register file is a multi-port memory array that contains the values of all the registers. The register file accepts three 4-bit inputs, which denote the two registers to be read (R1 and R2) and the register to be written to (RD). It also has a 1-bit write enable (RegW) that controls when a register is written to. The register file also has a 16-bit write data input (WriteData), which contains the value to be written to a register if needed. The register then outputs two 16-bit values (RO1 and RO2), which are the contents of the two registers selected to read.

Inputs: R1 (4-bits), R2 (4-bits), RD (4-bits), RegW (1-bit), WriteData (16-bits)
Outputs: RO1 (16-bits), RO2 (16-bits)

In the simulation, a value was written to each register, then each register was read. The simulation results did match the expected output. The simulation waveform and console output can be seen below.



```
R1:  0, R2:  1, RD:  0, RegW: 1, WriteData:    0, RO1:    x, RO2:    x
R1:  0, R2:  1, RD:  1, RegW: 1, WriteData:    1, RO1:    0, RO2:    x
R1:  0, R2:  1, RD:  2, RegW: 1, WriteData:    2, RO1:    0, RO2:    1
R1:  0, R2:  1, RD:  3, RegW: 1, WriteData:    3, RO1:    0, RO2:    1
R1:  0, R2:  1, RD:  4, RegW: 1, WriteData:    4, RO1:    0, RO2:    1
R1:  0, R2:  1, RD:  5, RegW: 1, WriteData:    5, RO1:    0, RO2:    1
R1:  0, R2:  1, RD:  6, RegW: 1, WriteData:    6, RO1:    0, RO2:    1
R1:  0, R2:  1, RD:  7, RegW: 1, WriteData:    7, RO1:    0, RO2:    1
R1:  0, R2:  1, RD:  8, RegW: 1, WriteData:    8, RO1:    0, RO2:    1
R1:  0, R2:  1, RD:  9, RegW: 1, WriteData:    9, RO1:    0, RO2:    1
R1:  0, R2:  1, RD: 10, RegW: 1, WriteData:   10, RO1:    0, RO2:    1
R1:  0, R2:  1, RD: 11, RegW: 1, WriteData:   11, RO1:    0, RO2:    1
R1:  0, R2:  1, RD: 12, RegW: 1, WriteData:   12, RO1:    0, RO2:    1
R1:  0, R2:  1, RD: 13, RegW: 1, WriteData:   13, RO1:    0, RO2:    1
R1:  0, R2:  1, RD: 14, RegW: 1, WriteData:   14, RO1:    0, RO2:    1
R1:  0, R2:  1, RD: 15, RegW: 1, WriteData:   15, RO1:    0, RO2:    1
R1:  0, R2:  1, RD: 15, RegW: 0, WriteData:   15, RO1:    0, RO2:    1
R1:  2, R2:  3, RD: 15, RegW: 0, WriteData:   15, RO1:    2, RO2:    3
R1:  4, R2:  5, RD: 15, RegW: 0, WriteData:   15, RO1:    4, RO2:    5
R1:  6, R2:  7, RD: 15, RegW: 0, WriteData:   15, RO1:    6, RO2:    7
R1:  8, R2:  9, RD: 15, RegW: 0, WriteData:   15, RO1:    8, RO2:    9
R1: 10, R2: 11, RD: 15, RegW: 0, WriteData:   15, RO1:   10, RO2:   11
R1: 12, R2: 13, RD: 15, RegW: 0, WriteData:   15, RO1:   12, RO2:   13
R1: 14, R2: 15, RD: 15, RegW: 0, WriteData:   15, RO1:   14, RO2:   15
```

2.2 ALU

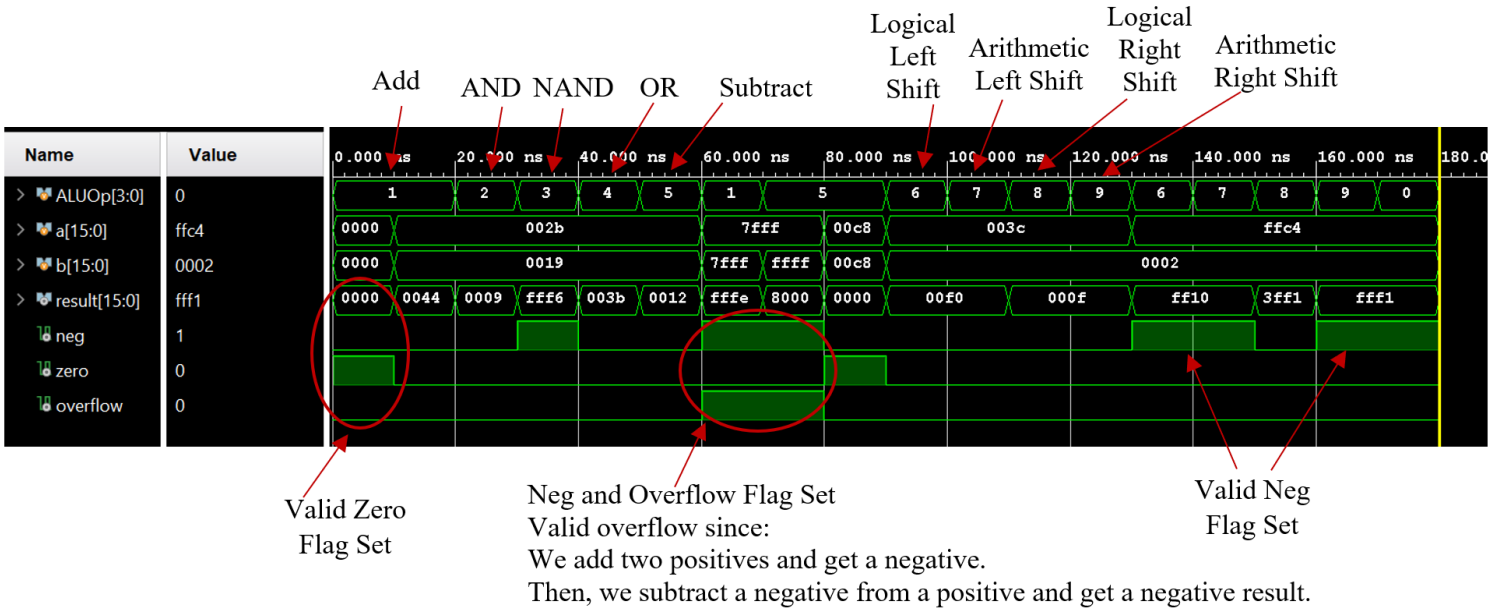
The arithmetic logic unit (ALU) can perform a variety of arithmetic operations. The ALU accepts two 16-bit inputs (A and B), which are the values used in the arithmetic operation. A 4-bit input control signal (ALUOp) indicates what operation is to be performed. The ALU then outputs the 16-bit result (R). It also has three 1-bit output flags: Negative (N), Overflow (O), and Zero (Z).

Inputs: A (16-bits), B (16-bits), ALUOp (4-bits)
Outputs: R (16-bits), O (1-bit), N (1-bit), Z (1-bit)

The ALU operations can be seen in the table below.

ALUOp	Operation
0000	Nothing
0001	Add
0010	AND
0011	NAND
0100	OR
0101	Subtract
0110	Logical Left Shift
0111	Arithmetic Left Shift
1000	Logical Right Shift
1001	Arithmetic Right Shift

In the simulation, each ALU operation was tested. Each operation result was as expected. The flag functionality was also tested. The negative, zero, and overflow flags did function correctly. The simulation waveform and console output can be seen below.



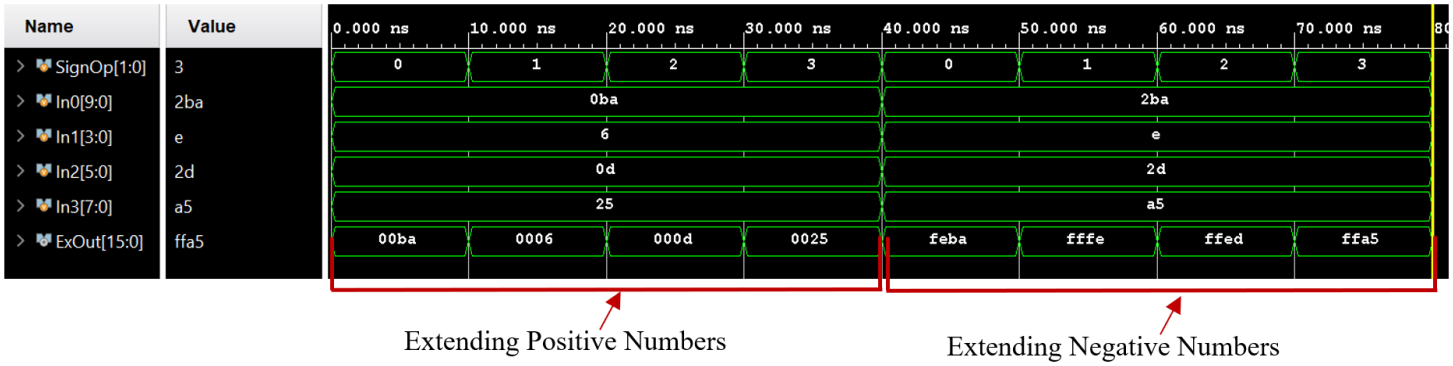
```
ALUOp:  1, Input A:     0, Input B:     0, Result:     0, Negative: 0, Zero: 1, Overflow: 0
ALUOp:  1, Input A:    43, Input B:    25, Result:    68, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  2, Input A:    43, Input B:    25, Result:     9, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  3, Input A:    43, Input B:    25, Result: 65526, Negative: 1, Zero: 0, Overflow: 0
ALUOp:  4, Input A:    43, Input B:    25, Result:    59, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  5, Input A:    43, Input B:    25, Result:    18, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  1, Input A: 32767, Input B: 32767, Result: 65534, Negative: 1, Zero: 0, Overflow: 1
ALUOp:  5, Input A: 32767, Input B: 65535, Result: 32768, Negative: 1, Zero: 0, Overflow: 1
ALUOp:  5, Input A:   200, Input B:   200, Result:     0, Negative: 0, Zero: 1, Overflow: 0
ALUOp:  6, Input A:    60, Input B:     2, Result:   240, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  7, Input A:    60, Input B:     2, Result:   240, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  8, Input A:    60, Input B:     2, Result:    15, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  9, Input A:    60, Input B:     2, Result:    15, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  6, Input A: 65476, Input B:     2, Result: 65296, Negative: 1, Zero: 0, Overflow: 0
ALUOp:  7, Input A: 65476, Input B:     2, Result: 65296, Negative: 1, Zero: 0, Overflow: 0
ALUOp:  8, Input A: 65476, Input B:     2, Result: 16369, Negative: 0, Zero: 0, Overflow: 0
ALUOp:  9, Input A: 65476, Input B:     2, Result: 65521, Negative: 1, Zero: 0, Overflow: 0
ALUOp:  0, Input A: 65476, Input B:     2, Result: 65521, Negative: 1, Zero: 0, Overflow: 0
```

2.3 Sign Extender

The sign extender unit extends the sign bit of the input value. The sign extender can accept inputs of varying lengths: In0 (10-bits), In1 (4-bits), In2 (6-bits), and In3 (8-bits). It also has a 2-bit input select signal (SignOp), which determine which input should be extended. The extended value is then outputted as a 16-bit value (ExOut).

Inputs: In0 (10-bits), In1 (4-bits), In2 (6-bits), In3 (8-bits), SignOp (2-bits)
Outputs: ExOut (16-bits)

In the simulation, various positive and negative numbers were tested. All of the possible input sizes were tested using both positive and negative numbers. The sign extender was observed to function as expected. The simulation waveform and console output can be seen below.



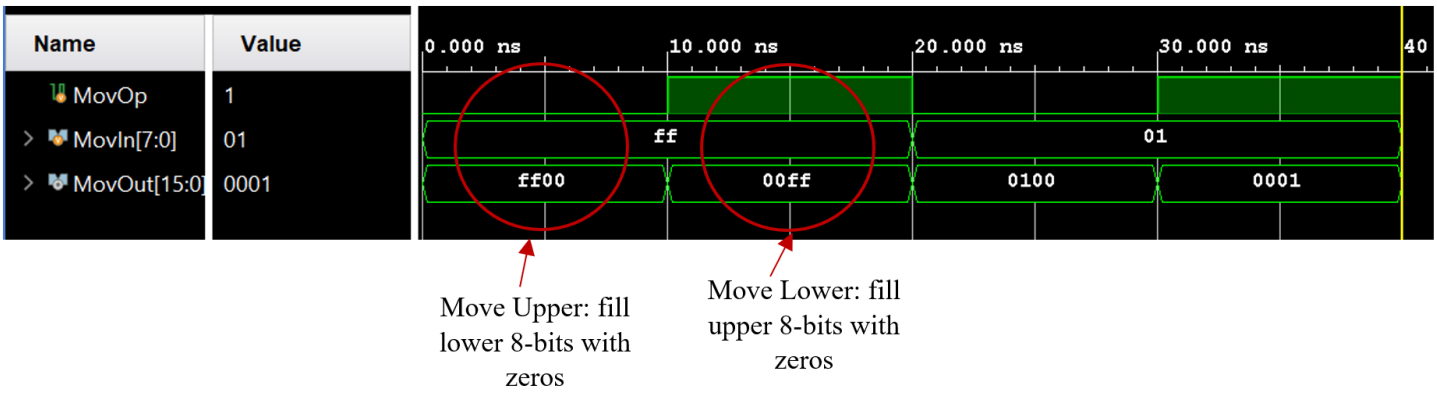

```
SignOp: 0, In0:  186, In1:  6, In2: 13, In3:  37, ExOut:   186
SignOp: 1, In0:  186, In1:  6, In2: 13, In3:  37, ExOut:     6
SignOp: 2, In0:  186, In1:  6, In2: 13, In3:  37, ExOut:    13
SignOp: 3, In0:  186, In1:  6, In2: 13, In3:  37, ExOut:    37
SignOp: 0, In0:  698, In1: 14, In2: 45, In3: 165, ExOut: 65210
SignOp: 1, In0:  698, In1: 14, In2: 45, In3: 165, ExOut: 65534
SignOp: 2, In0:  698, In1: 14, In2: 45, In3: 165, ExOut: 65517
SignOp: 3, In0:  698, In1: 14, In2: 45, In3: 165, ExOut: 65445
```

2.4 Move Extender

The move extender is used for the move lower (movl) and mov upper (movu) instructions. It accepts an 8-bit input (MovIn) and a 1-bit control signal (MovOp). The move extender will place 8 0's to either the bottom or top of the 8-bit input depending on MovOp. The 16-bit value will then be outputted (MovOut).

Inputs: MovIn (8-bits), MovOp (1-bit)
Outputs: MovOut (16-bits)

In the simulation, various values were tested using both MovOp signals. The move extender was observed to function as expected. The simulation waveform and console output can be seen below.



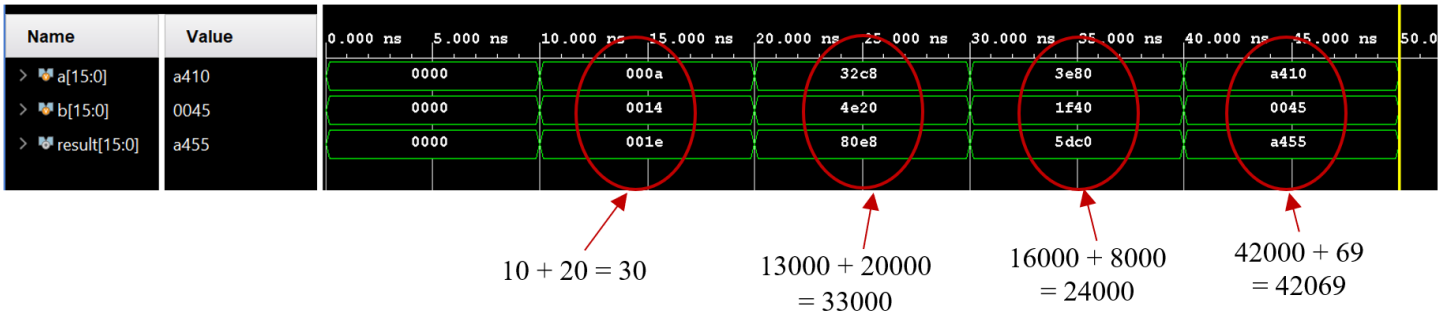

```
MovOp: 0, MovIn: 255, MovOut: 65280
MovOp: 1, MovIn: 255, MovOut:   255
MovOp: 0, MovIn:   1, MovOut:   256
MovOp: 1, MovIn:   1, MovOut:     1
```

2.5 Adder

There are two adders used in addition to the ALU. Both adders accept two 16-bits inputs (A and B) and output a 16-bit value (Result).

Inputs: A (16-bits), B (16-bits)
Outputs: Result (16-bits)

In the simulation, various input values were used. The adder did produce the correct result for each of the different input values. The simulation waveform and console output can be seen below.




```
Input A:     0, Input B:     0, Result:     0
Input A:    10, Input B:    20, Result:    30
Input A: 13000, Input B: 20000, Result: 33000
Input A: 16000, Input B:  8000, Result: 24000
Input A: 42000, Input B:    69, Result: 42069
```

2.6 Control Unit

The control unit receives the opcode, LS, R, and AD bits from the instruction memory and sets control signals for the other datapath components in order to execute the proper instruction. The opcode identifies the instruction, LS is used to differentiate load and store, R differentiates jump and return, and AD indicates the direction and arithmetic of shifts. The control unit then outputs: register write enable (RegW), memory read enable (MemR), memory write enable (MemW), branch logic control (BrOp), ALU logic control (ALUOp), move extender control (MovOp), sign extender control (SignOp), and multiplexer control signals (MuxC0 - MuxC4).

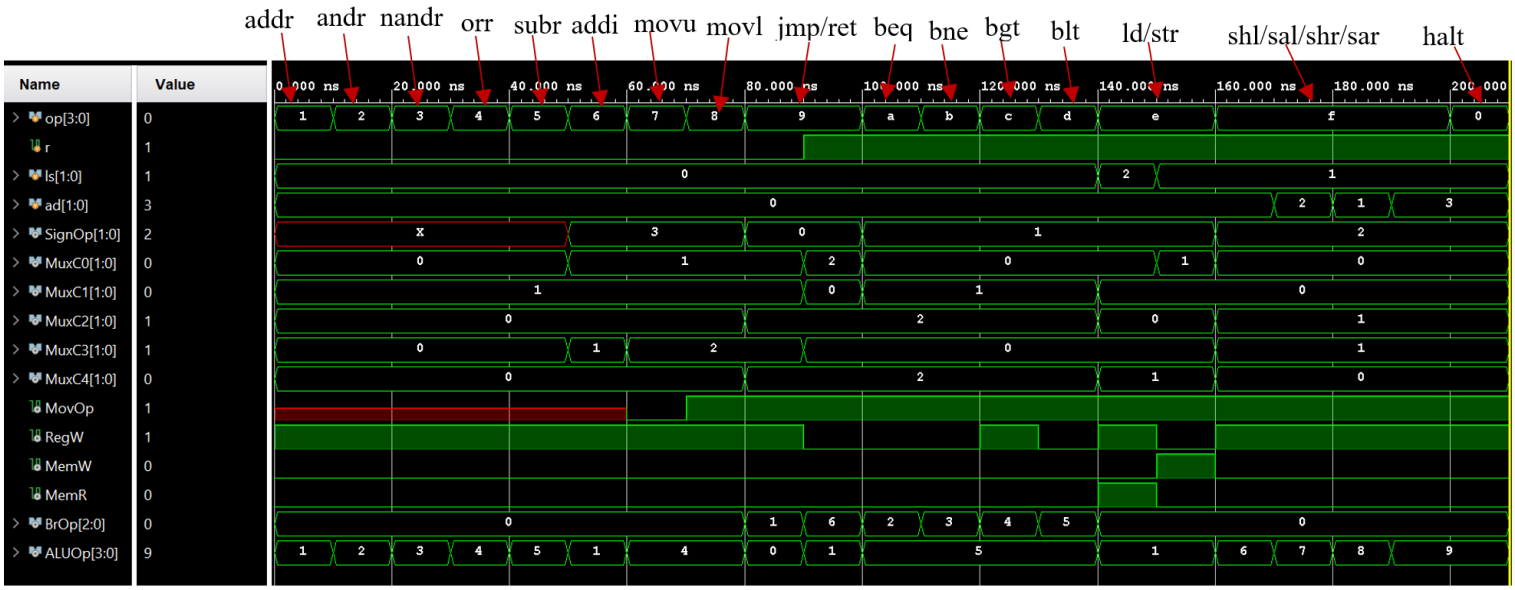
Inputs: opcode (4-bits), LS (2-bits), R (1-bit), AD (2-bits)
Outputs: RegW (1-bit), MemR (1-bit), MemW (1-bit), BrOp (3-bits), ALUOp (4-bits), MovOp (1-bit), SignOp (2-bits), MuxC0 (2-bits), MuxC0 (2-bits), MuxC1 (2-bits), MuxC2 (2-bits), MuxC3 (2-bits), MuxC4 (2-bits)

The truth tables for the control unit can be seen below.

Inputs				Outputs						
opcode	**R**	**LS**	**AD**	**RegW**	**MemR**	**MemW**	**BrOp**	**ALUOp**	**MovOp**	**SignOp**
0001	X	X	X	1	0	0	000	0001	X	X
0010	X	X	X	1	0	0	000	0010	X	X
0011	X	X	X	1	0	0	000	0011	X	X
0100	X	X	X	1	0	0	000	0100	X	X
0101	X	X	X	1	0	0	000	0101	X	X
0110	X	X	X	1	0	0	000	0001	X	11
0111	X	X	X	1	0	0	000	0100	0	X
1000	X	X	X	1	0	0	000	0100	1	X
1001	0	X	X	1	0	0	001	0000	X	00
1001	1	X	X	0	0	0	110	0001	X	X
1010	X	X	X	0	0	0	010	0101	X	01
1011	X	X	X	0	0	0	011	0101	X	01
1100	X	X	X	0	0	0	100	0101	X	01
1101	X	X	X	0	0	0	101	0101	X	01
1110	X	10	X	1	1	0	000	0001	X	X
1110	X	01	X	0	0	1	000	0001	X	X
1111	X	X	00	1	0	0	000	0110	X	10
1111	X	X	10	1	0	0	000	0111	X	10
1111	X	X	01	1	0	0	000	1000	X	10
1111	X	X	11	1	0	0	000	1001	X	10
0000	X	X	X	X	X	X	X	X	X	X

Inputs				Outputs				
opcode	**R**	**LS**	**AD**	**MuxC0**	**MuxC1**	**MuxC2**	**MuxC3**	**MuxC4**
0001	X	X	X	00	01	00	00	00
0010	X	X	X	00	01	00	00	00
0011	X	X	X	00	01	00	00	00
0100	X	X	X	00	01	00	00	00
0101	X	X	X	00	01	00	00	00
0110	X	X	X	01	X	00	01	00
0111	X	X	X	01	X	00	10	00
1000	X	X	X	01	X	00	10	00
1001	0	X	X	X	X	10	X	10
1001	1	X	X	10	00	X	00	X
1010	X	X	X	00	01	X	00	X
1011	X	X	X	00	01	X	00	X
1100	X	X	X	00	01	X	00	X
1101	X	X	X	00	01	X	00	X
1110	X	10	X	00	00	00	00	01
1110	X	01	X	01	00	X	00	X
1111	X	X	00	00	X	01	01	00
1111	X	X	10	00	X	01	01	00
1111	X	X	01	00	X	01	01	00
1111	X	X	11	00	X	01	01	00
0000	X	X	X	X	X	X	X	X

In the simulation, each opcode was tested along with the R, LS, and AD fields. The control unit did produce the output described in the above truth tables. The simulation waveform and console output can be seen below.



Inputs

Outputs

```
Opcode:  1, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  1, MovOp: x, SignOp: x, MuxC0: 0, MuxC1: 1, MuxC2: 0, MuxC3: 0, MuxC4: 0
Opcode:  2, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  2, MovOp: x, SignOp: x, MuxC0: 0, MuxC1: 1, MuxC2: 0, MuxC3: 0, MuxC4: 0
Opcode:  3, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  3, MovOp: x, SignOp: x, MuxC0: 0, MuxC1: 1, MuxC2: 0, MuxC3: 0, MuxC4: 0
Opcode:  4, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  4, MovOp: x, SignOp: x, MuxC0: 0, MuxC1: 1, MuxC2: 0, MuxC3: 0, MuxC4: 0
Opcode:  5, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  5, MovOp: x, SignOp: x, MuxC0: 0, MuxC1: 1, MuxC2: 0, MuxC3: 0, MuxC4: 0
Opcode:  6, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  1, MovOp: x, SignOp: 3, MuxC0: 1, MuxC1: 1, MuxC2: 0, MuxC3: 1, MuxC4: 0
Opcode:  7, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  4, MovOp: 0, SignOp: 3, MuxC0: 1, MuxC1: 1, MuxC2: 0, MuxC3: 2, MuxC4: 0
Opcode:  8, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  4, MovOp: 1, SignOp: 3, MuxC0: 1, MuxC1: 1, MuxC2: 0, MuxC3: 2, MuxC4: 0
Opcode:  9, R Field: 0, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 1, ALUOp:  0, MovOp: 1, SignOp: 0, MuxC0: 1, MuxC1: 1, MuxC2: 2, MuxC3: 2, MuxC4: 2
Opcode:  9, R Field: 1, LS Field: 0, AD Field: 0, RegW: 0, MemR: 0, MemW: 0, BrOp: 6, ALUOp:  1, MovOp: 1, SignOp: 0, MuxC0: 2, MuxC1: 0, MuxC2: 2, MuxC3: 0, MuxC4: 2
Opcode: 10, R Field: 1, LS Field: 0, AD Field: 0, RegW: 0, MemR: 0, MemW: 0, BrOp: 2, ALUOp:  5, MovOp: 1, SignOp: 1, MuxC0: 0, MuxC1: 1, MuxC2: 2, MuxC3: 0, MuxC4: 2
Opcode: 11, R Field: 1, LS Field: 0, AD Field: 0, RegW: 0, MemR: 0, MemW: 0, BrOp: 3, ALUOp:  5, MovOp: 1, SignOp: 1, MuxC0: 0, MuxC1: 1, MuxC2: 2, MuxC3: 0, MuxC4: 2
Opcode: 12, R Field: 1, LS Field: 0, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 4, ALUOp:  5, MovOp: 1, SignOp: 1, MuxC0: 0, MuxC1: 1, MuxC2: 2, MuxC3: 0, MuxC4: 2
Opcode: 13, R Field: 1, LS Field: 0, AD Field: 0, RegW: 0, MemR: 0, MemW: 0, BrOp: 5, ALUOp:  5, MovOp: 1, SignOp: 1, MuxC0: 0, MuxC1: 1, MuxC2: 2, MuxC3: 0, MuxC4: 2
Opcode: 14, R Field: 1, LS Field: 2, AD Field: 0, RegW: 1, MemR: 1, MemW: 0, BrOp: 0, ALUOp:  1, MovOp: 1, SignOp: 1, MuxC0: 0, MuxC1: 0, MuxC2: 0, MuxC3: 0, MuxC4: 1
Opcode: 14, R Field: 1, LS Field: 1, AD Field: 0, RegW: 0, MemR: 0, MemW: 1, BrOp: 0, ALUOp:  1, MovOp: 1, SignOp: 1, MuxC0: 1, MuxC1: 0, MuxC2: 0, MuxC3: 0, MuxC4: 1
Opcode: 15, R Field: 1, LS Field: 1, AD Field: 0, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  6, MovOp: 1, SignOp: 2, MuxC0: 0, MuxC1: 0, MuxC2: 1, MuxC3: 1, MuxC4: 0
Opcode: 15, R Field: 1, LS Field: 1, AD Field: 2, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  7, MovOp: 1, SignOp: 2, MuxC0: 0, MuxC1: 0, MuxC2: 1, MuxC3: 1, MuxC4: 0
Opcode: 15, R Field: 1, LS Field: 1, AD Field: 1, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  8, MovOp: 1, SignOp: 2, MuxC0: 0, MuxC1: 0, MuxC2: 1, MuxC3: 1, MuxC4: 0
Opcode: 15, R Field: 1, LS Field: 1, AD Field: 3, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  9, MovOp: 1, SignOp: 2, MuxC0: 0, MuxC1: 0, MuxC2: 1, MuxC3: 1, MuxC4: 0
Opcode:  0, R Field: 1, LS Field: 1, AD Field: 3, RegW: 1, MemR: 0, MemW: 0, BrOp: 0, ALUOp:  9, MovOp: 1, SignOp: 2, MuxC0: 0, MuxC1: 0, MuxC2: 1, MuxC3: 1, MuxC4: 0
```

2.7 Branch Control Unit

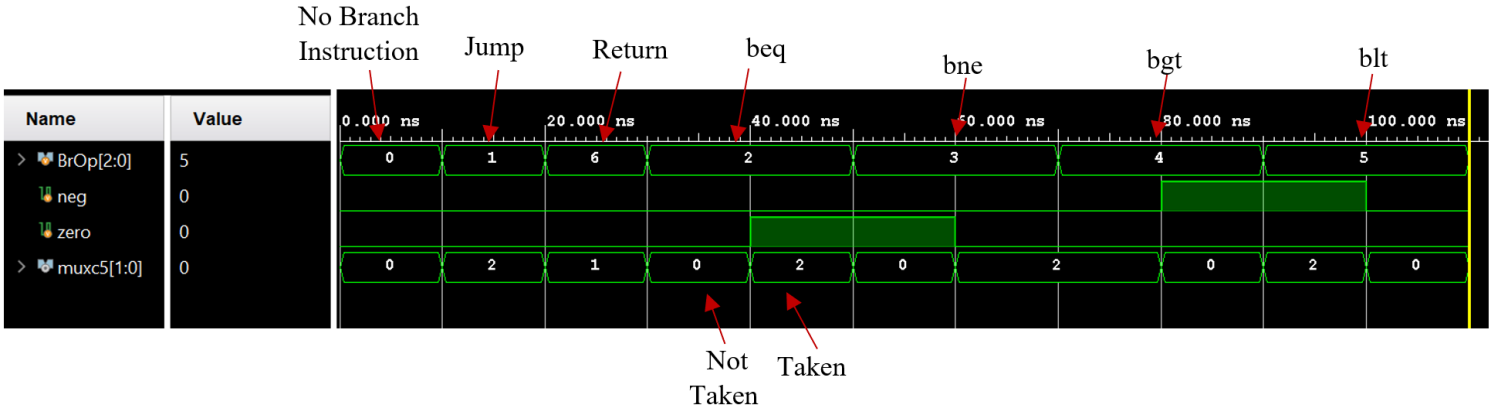
The branch control unit controls whether a branch or jump is taken. The branch control unit receives a branch logic control (BrOp) signal from the main control unit, which identifies what type of branch or jump is being evaluated. It also accepts the negative (N) and Zero (Z) flags from the ALU to determine whether or not a branch should be taken. The branch control unit then outputs the mux 5 control signal (MuxC5), which determines what value is loaded into the PC.

Inputs: BrOp (3-bits), N (1-bit), Z (1-bit)
Outputs: MuxC5 (2-bits)

The truth table for the branch control unit can be seen below.

Inputs			**Outputs**
BranchOp	**N**	**Z**	**MuxC5**
000	X	X	00
001	X	X	10
110	X	X	01
010	X	1	10
010	X	0	00
011	X	0	10
011	X	1	00
100	0	0	10
100	1	X	00
101	1	0	10
101	0	X	00

In the simulation, each BrOp signal was tested along with various N and Z flags. The branch control unit did produce the output described in the above truth table. The simulation waveform and console output can be seen below.



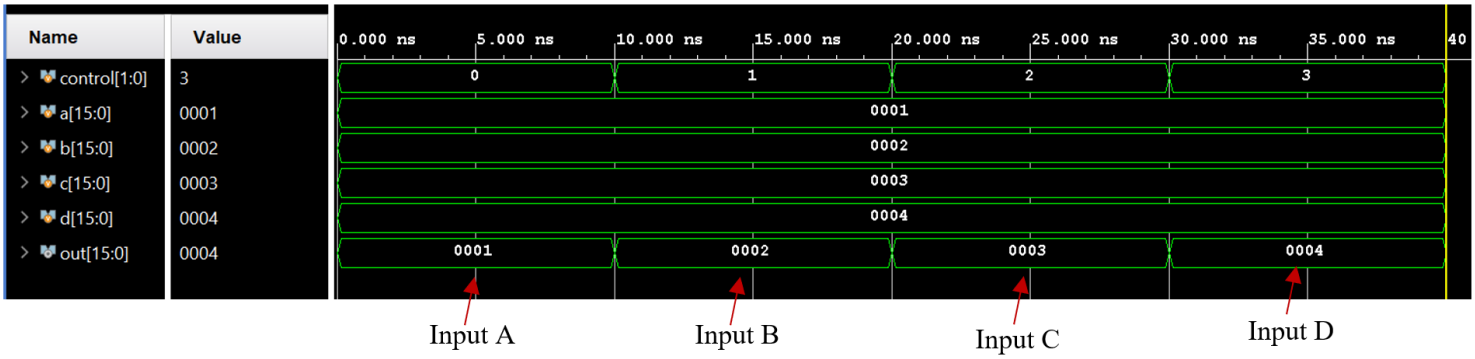

```
BrOp: 0, Negative: 0, Zero: 0, MuxC5(output): 0
BrOp: 1, Negative: 0, Zero: 0, MuxC5(output): 2
BrOp: 6, Negative: 0, Zero: 0, MuxC5(output): 1
BrOp: 2, Negative: 0, Zero: 0, MuxC5(output): 0
BrOp: 2, Negative: 0, Zero: 1, MuxC5(output): 2
BrOp: 3, Negative: 0, Zero: 1, MuxC5(output): 0
BrOp: 3, Negative: 0, Zero: 0, MuxC5(output): 2
BrOp: 4, Negative: 0, Zero: 0, MuxC5(output): 2
BrOp: 4, Negative: 1, Zero: 0, MuxC5(output): 0
BrOp: 5, Negative: 1, Zero: 0, MuxC5(output): 2
BrOp: 5, Negative: 0, Zero: 0, MuxC5(output): 0
```

2.8 4-to-1 Multiplexer

Several 4-to-1 multiplexers are used throughout the datapath to select one output from multiple inputs. The multiplexers used throughout the datapath are listed below. Each multiplexer has a control signal (MuxC#) from the control unit, which selects what input to pass.

In the simulation, various control signals were used to pass each of the four inputs. The multiplexer was observed to work correctly. The simulation waveform and console

output can be seen below.

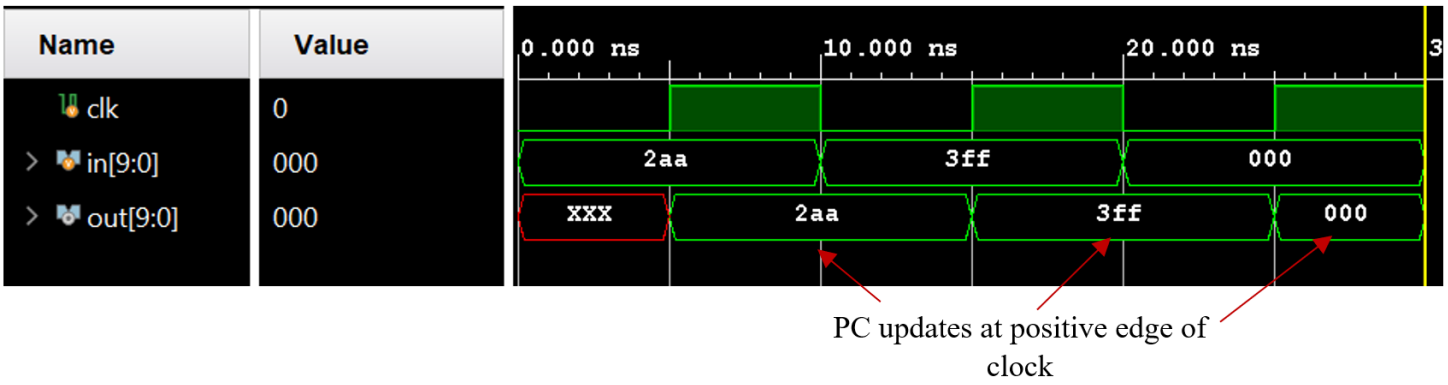



```
Control: 0, Input A:     1, Input B:     2, Input C:     3, Input D:     4, Output:     1
Control: 1, Input A:     1, Input B:     2, Input C:     3, Input D:     4, Output:     2
Control: 2, Input A:     1, Input B:     2, Input C:     3, Input D:     4, Output:     3
Control: 3, Input A:     1, Input B:     2, Input C:     3, Input D:     4, Output:     4
```

2.9 Program Counter

The program counter is a 10-bit register that holds the address of the next instruction. The PC will be updated when the positive edge of the clock is detected. This change will then trigger changes in all the other datapath components in order to execute the desired instruction.

In the simulation, various inputs were used along with a clock signal. The PC was observed to only update when the positive edge of the clock was detected. Thus, the PC performed correctly. The simulation waveform and console output can be seen below.




```
Clock: 0, In:   682, Out:     x
Clock: 1, In:   682, Out:   682
Clock: 0, In:  1023, Out:   682
Clock: 1, In:  1023, Out:  1023
Clock: 0, In:     0, Out:  1023
Clock: 1, In:     0, Out:     0
Clock: 0, In:     0, Out:     0
```

Appendix A Component Code

A.1 Register File

```
module reg_file(R1, R2, RD, RegW, WriteData, RO1, RO2);
    input [3:0] R1;
    input [3:0] R2;
    input [3:0] RD;
    input RegW;
    input [15:0] WriteData;
    output reg [15:0] RO1;
    output reg [15:0] RO2;
    reg [15:0] regs [15:0];

    always @ (R1 or R2 or RD or RegW or WriteData)
    begin
        if (RegW == 1)
            regs[RD] <= WriteData;
        RO1 <= regs[R1];
        RO2 <= regs[R2];
    end
endmodule
```

A.2 ALU

```
module alu(ALUOp, a, b, r, n, z, o);
    input [3:0] ALUOp;
    input signed [15:0] a;
    input signed [15:0] b;
    output reg [15:0] r;
    output reg n, z, o;

    always @*
    begin
        n = 0;
        z = 0;
        o = 0;
        case (ALUOp)
            4'b0000: r = r;
            4'b0001: begin
                        r = a + b; o = (a[15] ^ b[15]) ? 0:(r[15] ^ a[15]);
```

```
                    end
            4'b0010: r = a & b;
            4'b0011: r = ~(a & b);
            4'b0100: r = a | b;
            4'b0101: begin
                        r = a - b; o = (a[15] ^ b[15]) ? (r[15] ~^ b[15]):0;
                    end
            4'b0110: r = a << b;
            4'b0111: r = a <<< b;
            4'b1000: r = a >> b;
            4'b1001: r = a >>> b;
            default: r = 0;
        endcase
        n = (r[15] == 1'b1) ? 1:0;
        z = (r == 0) ? 1:0;
    end
endmodule
```

A.3 Sign Extender

```
module sign_ext(SignOp, In0, In1, In2, In3, ExOut);
    input [1:0] SignOp;
    input [9:0] In0;
    input [3:0] In1;
    input [5:0] In2;
    input [7:0] In3;
    output reg [15:0] ExOut;

    always @ (SignOp or In0 or In1 or In2 or In3)
    begin
        case (SignOp)
            2'b00: ExOut = { {6{In0[9]}}, In0 };
            2'b01: ExOut = { {12{In1[3]}}, In1 };
            2'b10: ExOut = { {10{In2[5]}}, In2 };
            2'b11: ExOut = { {8{In3[7]}}, In3 };
        endcase
    end
endmodule
```

A.4 Move Extender

```
module mov_ext(MovOp, MovIn, MovOut);
    input MovOp;
    input [7:0] MovIn;
    output reg [15:0] MovOut;

    always @ (MovOp or MovIn)
    begin
        if (MovOp == 0)
            MovOut = { MovIn, {8{1'b0}} };
        else if (MovOp == 1)
            MovOut = { {8{1'b0}}, MovIn };
    end
endmodule
```

A.5 Adder

```
module adder(a, b, y);
    input [15:0] a;
    input [15:0] b;
    output reg [15:0] y;

    always @ (a or b)
    begin
        y = a + b;
    end
endmodule
```

A.6 Control Unit

```
module control(op, r, ls, ad, RegW, MemR, MemW, BrOp, ALUOp, MovOp,
                SignOp, MuxC0, MuxC1, MuxC2, MuxC3, MuxC4);
    input [3:0] op;
    input r;
    input [1:0] ls, ad;
    output reg [1:0] SignOp, MuxC0, MuxC1, MuxC2, MuxC3, MuxC4;
    output reg MovOp, RegW, MemW, MemR;
    output reg [2:0] BrOp;
    output reg [3:0] ALUOp;
```

```
10
11      always @ (op or r or ls or ad)
12      begin
13          case(op)
14              4'b0001: begin
15                      RegW <= 1'b1;
16                      MemR <= 1'b0;
17                      MemW <= 1'b0;
18                      BrOp <= 3'b000;
19                      ALUOp <= 4'b0001;
20                      MuxC0 <= 2'b00;
21                      MuxC1 <= 2'b01;
22                      MuxC2 <= 2'b00;
23                      MuxC3 <= 2'b00;
24                      MuxC4 <= 2'b00;
25                  end
26              4'b0010: begin
27                      RegW <= 1'b1;
28                      MemR <= 1'b0;
29                      MemW <= 1'b0;
30                      BrOp <= 3'b000;
31                      ALUOp <= 4'b0010;
32                      MuxC0 <= 2'b00;
33                      MuxC1 <= 2'b01;
34                      MuxC2 <= 2'b00;
35                      MuxC3 <= 2'b00;
36                      MuxC4 <= 2'b00;
37                  end
38              4'b0011: begin
39                      RegW <= 1'b1;
40                      MemR <= 1'b0;
41                      MemW <= 1'b0;
42                      BrOp <= 3'b000;
43                      ALUOp <= 4'b0011;
44                      MuxC0 <= 2'b00;
45                      MuxC1 <= 2'b01;
46                      MuxC2 <= 2'b00;
47                      MuxC3 <= 2'b00;
48                      MuxC4 <= 2'b00;
49                  end
50              4'b0100: begin
51                      RegW <= 1'b1;
```

```
52                      MemR <= 1'b0;
53                      MemW <= 1'b0;
54                      BrOp <= 3'b000;
55                      ALUOp <= 4'b0100;
56                      MuxC0 <= 2'b00;
57                      MuxC1 <= 2'b01;
58                      MuxC2 <= 2'b00;
59                      MuxC3 <= 2'b00;
60                      MuxC4 <= 2'b00;
61                  end
62              4'b0101: begin
63                      RegW <= 1'b1;
64                      MemR <= 1'b0;
65                      MemW <= 1'b0;
66                      BrOp <= 3'b000;
67                      ALUOp <= 4'b0101;
68                      MuxC0 <= 2'b00;
69                      MuxC1 <= 2'b01;
70                      MuxC2 <= 2'b00;
71                      MuxC3 <= 2'b00;
72                      MuxC4 <= 2'b00;
73                  end
74              4'b0110: begin
75                      RegW <= 1'b1;
76                      MemR <= 1'b0;
77                      MemW <= 1'b0;
78                      BrOp <= 3'b000;
79                      ALUOp <= 4'b0001;
80                      SignOp <= 2'b11;
81                      MuxC0 <= 2'b01;
82                      MuxC2 <= 2'b00;
83                      MuxC3 <= 2'b01;
84                      MuxC4 <= 2'b00;
85                  end
86              4'b0111: begin
87                      RegW <= 1'b1;
88                      MemR <= 1'b0;
89                      MemW <= 1'b0;
90                      BrOp <= 3'b000;
91                      ALUOp <= 4'b0100;
92                      MovOp <= 1'b0;
93                      MuxC0 <= 2'b01;
```

```
94                      MuxC2 <= 2'b00;
95                      MuxC3 <= 2'b10;
96                      MuxC4 <= 2'b00;
97                  end
98              4'b1000: begin
99                      RegW <= 1'b1;
100                     MemR <= 1'b0;
101                     MemW <= 1'b0;
102                     BrOp <= 3'b000;
103                     ALUOp <= 4'b0100;
104                     MovOp <= 1'b1;
105                     MuxC0 <= 2'b01;
106                     MuxC2 <= 2'b00;
107                     MuxC3 <= 2'b10;
108                     MuxC4 <= 2'b00;
109                 end
110             4'b1001: begin
111                     if (r == 1'b0)
112                     begin
113                         RegW <= 1'b1;
114                         MemR <= 1'b0;
115                         MemW <= 1'b0;
116                         BrOp <= 3'b001;
117                         ALUOp <= 4'b0000;
118                         SignOp <= 2'b00;
119                         MuxC2 <= 2'b10;
120                         MuxC4 <= 2'b10;
121                     end
122                     else if (r == 1'b1)
123                     begin
124                         RegW <= 1'b0;
125                         MemR <= 1'b0;
126                         MemW <= 1'b0;
127                         BrOp <= 3'b110;
128                         ALUOp <= 4'b0001;
129                         MuxC0 <= 2'b10;
130                         MuxC1 <= 2'b00;
131                         MuxC3 <= 2'b00;
132                     end
133                 end
134             4'b1010: begin
135                     RegW <= 1'b0;
```

```
136                     MemR <= 1'b0;
137                     MemW <= 1'b0;
138                     BrOp <= 3'b010;
139                     ALUOp <= 4'b0101;
140                     SignOp <= 2'b01;
141                     MuxC0 <= 2'b00;
142                     MuxC1 <= 2'b01;
143                     MuxC3 <= 2'b00;
144                 end
145             4'b1011: begin
146                     RegW <= 1'b0;
147                     MemR <= 1'b0;
148                     MemW <= 1'b0;
149                     BrOp <= 3'b011;
150                     ALUOp <= 4'b0101;
151                     SignOp <= 2'b01;
152                     MuxC0 <= 2'b00;
153                     MuxC1 <= 2'b01;
154                     MuxC3 <= 2'b00;
155                 end
156             4'b1100: begin
157                     RegW <= 1'b1;
158                     MemR <= 1'b0;
159                     MemW <= 1'b0;
160                     BrOp <= 3'b100;
161                     ALUOp <= 4'b0101;
162                     SignOp <= 2'b01;
163                     MuxC0 <= 2'b00;
164                     MuxC1 <= 2'b01;
165                     MuxC3 <= 2'b00;
166                 end
167             4'b1101: begin
168                     RegW <= 1'b0;
169                     MemR <= 1'b0;
170                     MemW <= 1'b0;
171                     BrOp <= 3'b101;
172                     ALUOp <= 4'b0101;
173                     SignOp <= 2'b01;
174                     MuxC0 <= 2'b00;
175                     MuxC1 <= 2'b01;
176                     MuxC3 <= 2'b00;
177                 end
```

```
178             4'b1110: begin
179                     if (ls == 2'b10)
180                     begin
181                         RegW <= 1'b1;
182                         MemR <= 1'b1;
183                         MemW <= 1'b0;
184                         BrOp <= 3'b000;
185                         ALUOp <= 4'b0001;
186                         MuxC0 <= 2'b00;
187                         MuxC1 <= 2'b00;
188                         MuxC2 <= 2'b00;
189                         MuxC3 <= 2'b00;
190                         MuxC4 <= 2'b01;
191                     end
192                     else if (ls == 2'b01)
193                     begin
194                         RegW <= 1'b0;
195                         MemR <= 1'b0;
196                         MemW <= 1'b1;
197                         BrOp <= 3'b000;
198                         ALUOp <= 4'b0001;
199                         MuxC0 <= 2'b01;
200                         MuxC1 <= 2'b00;
201                         MuxC3 <= 2'b00;
202                     end
203                   end
204             4'b1111: begin
205                     if (ad == 2'b00)
206                     begin
207                         RegW <= 1'b1;
208                         MemR <= 1'b0;
209                         MemW <= 1'b0;
210                         BrOp <= 3'b000;
211                         ALUOp <= 4'b0110;
212                         SignOp <= 2'b10;
213                         MuxC0 <= 2'b00;
214                         MuxC2 <= 2'b01;
215                         MuxC3 <= 2'b01;
216                         MuxC4 <= 2'b00;
217                     end
218                     else if (ad == 2'b10)
219                     begin
```

```
220                             RegW <= 1'b1;
221                             MemR <= 1'b0;
222                             MemW <= 1'b0;
223                             BrOp <= 3'b000;
224                             ALUOp <= 4'b0111;
225                             SignOp <= 2'b10;
226                             MuxC0 <= 2'b00;
227                             MuxC2 <= 2'b01;
228                             MuxC3 <= 2'b01;
229                             MuxC4 <= 2'b00;
230                         end
231                         else if (ad == 2'b01)
232                         begin
233                             RegW <= 1'b1;
234                             MemR <= 1'b0;
235                             MemW <= 1'b0;
236                             BrOp <= 3'b000;
237                             ALUOp <= 4'b1000;
238                             SignOp <= 2'b10;
239                             MuxC0 <= 2'b00;
240                             MuxC2 <= 2'b01;
241                             MuxC3 <= 2'b01;
242                             MuxC4 <= 2'b00;
243                         end
244                         else if (ad == 2'b11)
245                         begin
246                             RegW <= 1'b1;
247                             MemR <= 1'b0;
248                             MemW <= 1'b0;
249                             BrOp <= 3'b000;
250                             ALUOp <= 4'b1001;
251                             SignOp <= 2'b10;
252                             MuxC0 <= 2'b00;
253                             MuxC2 <= 2'b01;
254                             MuxC3 <= 2'b01;
255                             MuxC4 <= 2'b00;
256                         end
257                     end
258             endcase
259         end
260 endmodule
```

A.7 Branch Control Unit

```
module branch_control(BrOp, neg, zero, muxc5);
    input [2:0] BrOp;
    input neg;
    input zero;
    output reg [1:0] muxc5;

    always @ (BrOp, neg, zero)
    begin
        case(BrOp)
            3'b000: muxc5 = 2'b00;
            3'b001: muxc5 = 2'b10;
            3'b110: muxc5 = 2'b01;
            3'b010: muxc5 = (zero == 1'b1) ? 2'b10:2'b00;
            3'b011: muxc5 = (zero == 1'b1) ? 2'b00:2'b10;
            3'b100: muxc5 = (neg == 1'b1) ? 2'b00:2'b10;
            3'b101: muxc5 = (neg == 1'b1) ? 2'b10:2'b00;
        endcase
    end
endmodule
```

A.8 4-to-1 Multiplexer

```
module mux_4to1(control, a, b, c, d, out);
    input [1:0] control;
    input [15:0] a;
    input [15:0] b;
    input [15:0] c;
    input [15:0] d;
    output reg [15:0] out;

    always @ (control or a or b or c or d)
    begin
        case (control)
            2'b00: out = a;
            2'b01: out = b;
            2'b10: out = c;
            2'b11: out = d;
        endcase
    end
```

```
18 endmodule
```

A.9 Program Counter

```
1  module pc(clk, in, out);
2      input clk;
3      input [9:0] in;
4      output reg [9:0] out;
5
6      always @ (posedge clk)
7      begin
8          out = in;
9      end
10 endmodule
```

Appendix B Test-bench Code

B.1 Register File

```
1  module reg_file_tb();
2      reg [3:0] R1;
3      reg [3:0] R2;
4      reg [3:0] RD;
5      reg RegW;
6      reg [15:0] WriteData;
7      wire [15:0] RO1;
8      wire [15:0] RO2;
9
10     reg_file REG (R1, R2, RD, RegW, WriteData, RO1, RO2);
11
12     initial
13     begin
14         $monitor("R1: %d, R2: %d, RD: %d, RegW: %d, WriteData: %d,
15                   RO1: %d, RO2: %d", R1, R2, RD, RegW, WriteData, RO1, RO2);
16         RegW = 1; RD = 0; WriteData = 0; R1 = 0; R2 = 1;
17         #10; RD = 1; WriteData = 1;
18         #10; RD = 2; WriteData = 2;
19         #10; RD = 3; WriteData = 3;
20         #10; RD = 4; WriteData = 4;
```

```
21          #10; RD = 5; WriteData = 5;
22          #10; RD = 6; WriteData = 6;
23          #10; RD = 7; WriteData = 7;
24          #10; RD = 8; WriteData = 8;
25          #10; RD = 9; WriteData = 9;
26          #10; RD = 10; WriteData = 10;
27          #10; RD = 11; WriteData = 11;
28          #10; RD = 12; WriteData = 12;
29          #10; RD = 13; WriteData = 13;
30          #10; RD = 14; WriteData = 14;
31          #10; RD = 15; WriteData = 15;
32          #10; RegW = 0; R1 = 0; R2 = 1;
33          #10; R1 = 2; R2 = 3;
34          #10; R1 = 4; R2 = 5;
35          #10; R1 = 6; R2 = 7;
36          #10; R1 = 8; R2 = 9;
37          #10; R1 = 10; R2 = 11;
38          #10; R1 = 12; R2 = 13;
39          #10; R1 = 14; R2 = 15;
40      end
41  endmodule
```

B.2 ALU

```
1   module alu_tb();
2       reg [3:0] ALUOp;
3       reg [15:0] a;
4       reg [15:0] b;
5       wire [15:0] result;
6       wire neg, zero, overflow;
7
8       alu ALU1 (ALUOp, a, b, result, neg, zero, overflow);
9
10      initial
11      begin
12          $monitor("ALUOp: %d, Input A: %d, Input B: %d, Result: %d,
13                    Negative: %d, Zero: %d, Overflow: %d", ALUOp, a,
14                    b, result, neg, zero, overflow);
15          ALUOp = 4'b0001; a = 0; b = 0;
16          #10; a = 43; b = 25;
17          #10; ALUOp = 4'b0010;
```

```
        #10; ALUOp = 4'b0011;
        #10; ALUOp = 4'b0100;
        #10; ALUOp = 4'b0101;
        #10; a = 16'b0111111111111111; b = 16'b0111111111111111;
        ALUOp = 4'b0001;
        #10; a = 16'b0111111111111111; b = 16'b1111111111111111;
        ALUOp = 4'b0101;
        #10; a = 200; b = 200; ALUOp = 4'b0101;
        #10; a = 60; b = 2; ALUOp = 4'b0110;
        #10; ALUOp = 4'b0111;
        #10; ALUOp = 4'b1000;
        #10; ALUOp = 4'b1001;
        #10; a = -60; b = 2; ALUOp = 4'b0110;
        #10; ALUOp = 4'b0111;
        #10; ALUOp = 4'b1000;
        #10; ALUOp = 4'b1001;
        #10; ALUOp = 4'b0000;
    end
endmodule
```

B.3 Sign Extender

```
module sign_ext_tb();
    reg [1:0] SignOp;
    reg [9:0] In0;
    reg [3:0] In1;
    reg [5:0] In2;
    reg [7:0] In3;
    wire [15:0] ExOut;

    sign_ext SE (SignOp, In0, In1, In2, In3, ExOut);

    initial
    begin
        $monitor("SignOp: %d, In0: %d, In1: %d, In2: %d, In3: %d,
                  ExOut: %d", SignOp, In0, In1, In2, In3, ExOut);
        In0 = 10'b0010111010; In1 = 4'b0110; In2 = 6'b001101;
        In3 = 8'b00100101; SignOp = 0;
        #10; SignOp = 1;
        #10; SignOp = 2;
        #10; SignOp = 3;
```

```
        #10; In0 = 10'b1010111010; In1 = 4'b1110; In2 = 6'b101101;
        In3 = 8'b10100101; SignOp = 0;
        #10; SignOp = 1;
        #10; SignOp = 2;
        #10; SignOp = 3;
    end
endmodule
```

B.4 Move Extender

```
module mov_ext_tb();
    reg MovOp;
    reg [7:0] MovIn;
    wire [15:0] MovOut;

    mov_ext ME (MovOp, MovIn, MovOut);

    initial
    begin
        $monitor("MovOp: %d, MovIn: %d, MovOut: %d", MovOp, MovIn, MovOut);
        MovIn = 8'b11111111; MovOp = 0;
        #10; MovOp = 1;
        #10; MovIn = 1; MovOp = 0;
        #10; MovOp = 1;
    end
endmodule
```

B.5 Adder

```
module adder_tb();
    reg [15:0] a;
    reg [15:0] b;
    wire [15:0] result;

    adder ADD (a, b, result);

    initial
    begin
        $monitor("Input A: %d, Input B: %d, Result: %d", a, b, result);
        a = 0; b =0;
```

```
        #10; a = 10; b = 20;
        #10; a = 13000; b = 20000;
        #10; a = 16000; b = 8000;
        #10; a = 42000; b = 69;
    end
endmodule
```

B.6 Control Unit

```
module control_tb();
    reg [3:0] op;
    reg r;
    reg [1:0] ls, ad;
    wire [1:0] SignOp, MuxC0, MuxC1, MuxC2, MuxC3, MuxC4;
    wire MovOp, RegW, MemW, MemR;
    wire [2:0] BrOp;
    wire [3:0] ALUOp;

    control CON (op, r, ls, ad, RegW, MemR, MemW, BrOp, ALUOp, MovOp,
                 SignOp, MuxC0, MuxC1, MuxC2, MuxC3, MuxC4);

    initial
    begin
        $monitor("Opcode: %d, R Field: %d, LS Field: %d, AD Field: %d,
                  RegW: %d, MemR: %d, MemW: %d, BrOp: %d, ALUOp: %d,
                  MovOp: %d, SignOp: %d, MuxC0: %d, MuxC1: %d, MuxC2: %d,
                  MuxC3: %d, MuxC4: %d", op, r, ls, ad, RegW, MemR, MemW,
                  BrOp, ALUOp, MovOp, SignOp, MuxC0, MuxC1, MuxC2, MuxC3,
                  MuxC4);
        op = 4'b0001; r = 0; ls = 0; ad = 0;
        #10; op = 4'b0010;
        #10; op = 4'b0011;
        #10; op = 4'b0100;
        #10; op = 4'b0101;
        #10; op = 4'b0110;
        #10; op = 4'b0111;
        #10; op = 4'b1000;
        #10; op = 4'b1001;
        #10; r = 1; op = 4'b1001;
        #10; op = 4'b1010;
        #10; op = 4'b1011;
```

```
33          #10; op = 4'b1100;
34          #10; op = 4'b1101;
35          #10; ls = 2'b10; op = 4'b1110;
36          #10; ls = 2'b01; op = 4'b1110;
37          #10; op = 4'b1111;
38          #10; ad = 2'b10; op = 4'b1111;
39          #10; ad = 2'b01; op = 4'b1111;
40          #10; ad = 2'b11; op = 4'b1111;
41          #10; op = 4'b0000;
42      end
43  endmodule
```

B.7 Branch Control Unit

```
1   module branch_control_tb();
2       reg [2:0] BrOp;
3       reg neg;
4       reg zero;
5       wire [1:0] muxc5;
6
7       branch_control BRC (BrOp, neg, zero, muxc5);
8
9       initial
10      begin
11      $monitor("BrOp: %d, Negative: %d, Zero: %d, MuxC5(output): %d",
12                BrOp, neg, zero, muxc5);
13      neg = 0; zero = 0; BrOp = 3'b000;
14      #10; BrOp = 3'b001;
15      #10; BrOp = 3'b110;
16      #10; BrOp = 3'b010;
17      #10; zero = 1; BrOp = 3'b010;
18      #10; BrOp = 3'b011;
19      #10; zero = 0; BrOp = 3'b011;
20      #10; BrOp = 3'b100;
21      #10; neg = 1; BrOp = 3'b100;
22      #10; BrOp = 3'b101;
23      #10; neg = 0; BrOp = 3'b101;
24      end
25  endmodule
```

B.8 4-to-1 Multiplexer

```
module mux_4to1_tb();
    reg [1:0] control;
    reg [15:0] a;
    reg [15:0] b;
    reg [15:0] c;
    reg [15:0] d;
    wire [15:0] out;

    mux_4to1 M1 (control, a, b, c, d, out);

    initial
    begin
        $monitor("Control: %d, Input A: %d, Input B: %d, Input C: %d,
                  Input D: %d, Output: %d", control, a, b, c, d, out);
        a = 1; b = 2; c = 3; d = 4; control = 0;
        #10; control = 1;
        #10; control = 2;
        #10; control = 3;
    end
endmodule
```

B.9 Program Counter

```
module pc_tb();
    reg clk;
    reg [9:0] in;
    wire [9:0] out;

    pc PC1 (clk, in, out);

    initial
    begin
        $monitor("Clock: %d, In: %d, Out: %d", clk, in, out);
        in = 10'b1010101010; clk = 0;
        #10; in = 10'b1111111111;
        #10; in = 0;
    end

    always
```

```
17      begin
18          #5; clk = ~clk;
19      end
20  endmodule
```