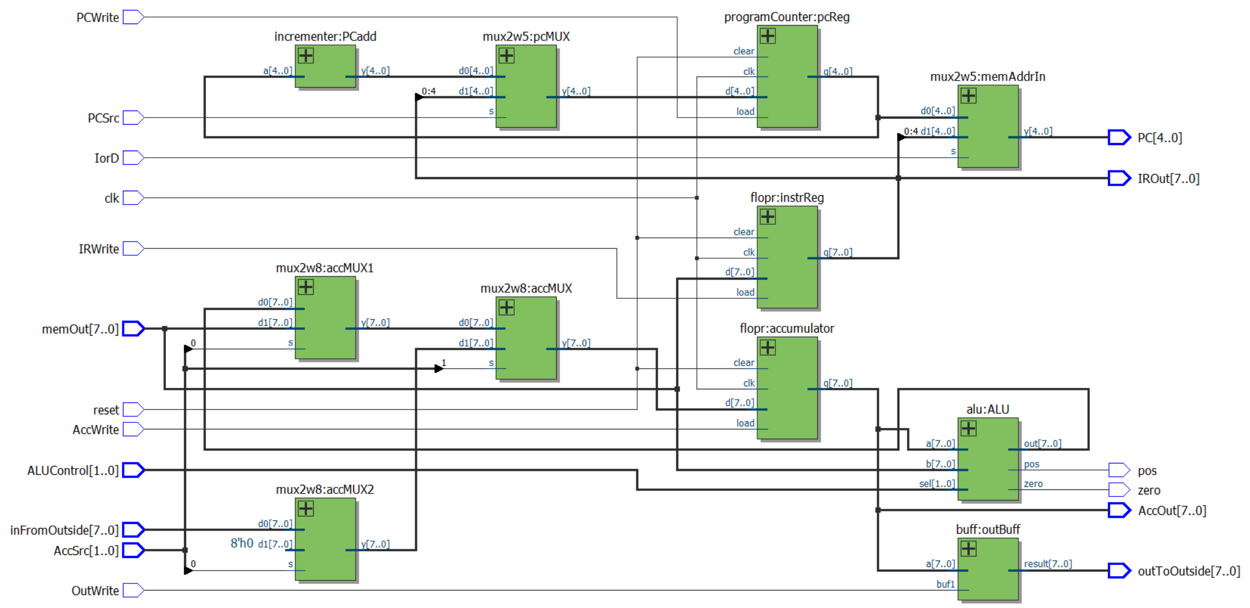
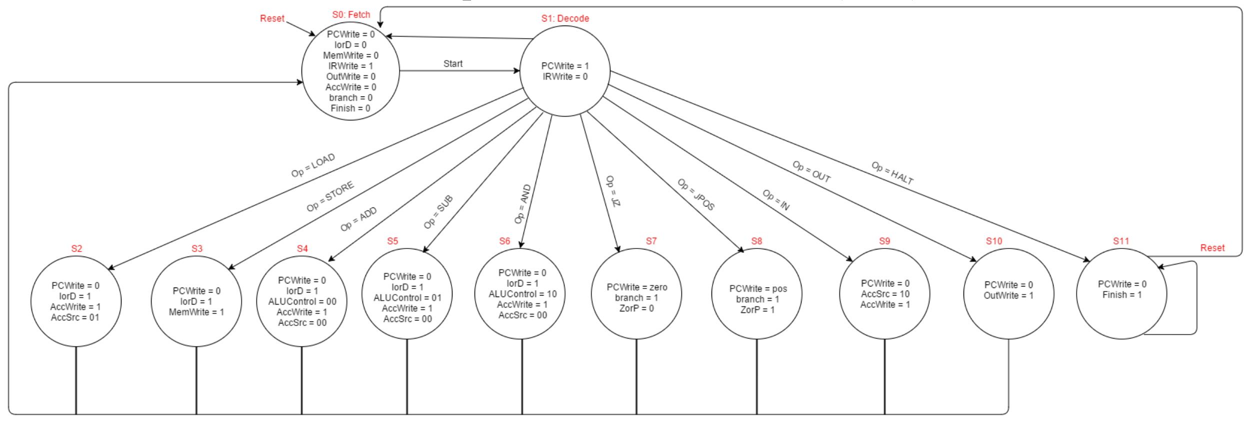
Final Report

1. Schematic of the DATAPATH (from RTL Viewer)

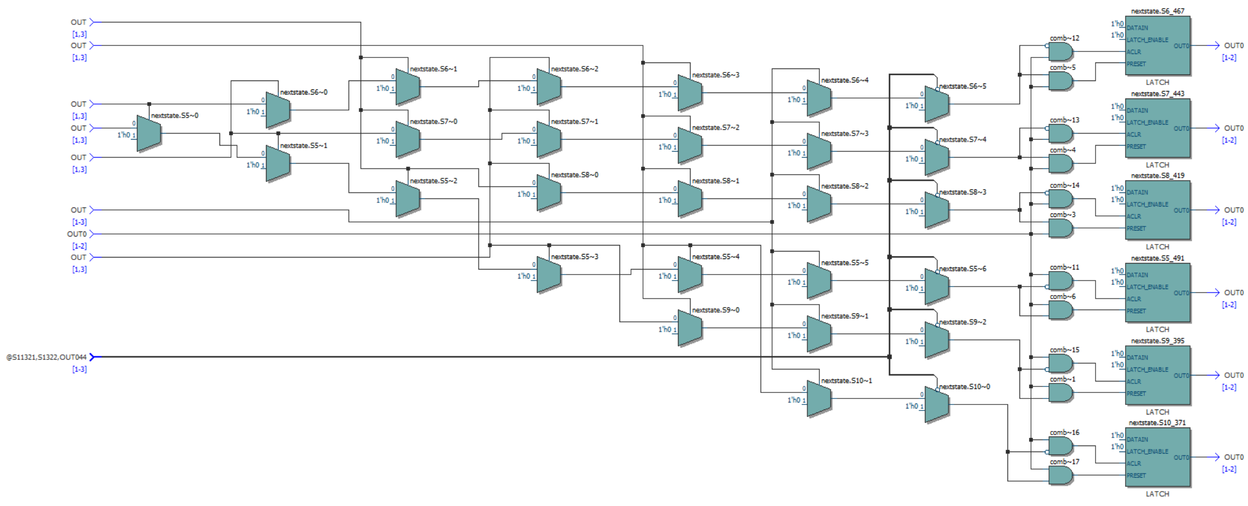


2. Control Unit

State Graph of Finite State Machine (FSM)



Schematic of Control Unit (from RTL Viewer)



Control Word Table (Per Cycle Per Instruction)

Instr	St	ZorP	branch	PCWrite	IorD	MemWrite	IRWrite	OutWrite	AccSrc	AccWrite	ALUControl
LOAD	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	0	1	0	0	0	01	1	XX
STORE	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	0	1	1	0	0	XX	0	XX
ADD	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	0	1	0	0	0	00	1	00
SUB	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	0	1	0	0	0	00	1	01
AND	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	1	0	0	0	0	00	1	10
JZ	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	0	1	zero	0	0	0	0	XX	0	XX
JPOS	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	1	1	pos	0	0	0	0	XX	0	XX
IN	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	0	0	0	0	0	10	1	XX
OUT	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	0	0	0	0	1	XX	0	XX
HALT	F	X	0	0	0	0	1	0	XX	0	XX
	D	X	0	1	0	0	0	0	XX	0	XX
	E	X	0	0	0	0	0	0	XX	0	XX

3. Verilog Modules

Components of the Datapath

alu.v (Arithmetic Logic Unit)

```verilog
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Arithmetic Logic Unit (ALU) Module
 */
module alu (a, b, sel, out, zero, pos);
      input [7:0] a, b;       // a - Accumulator, b - Memory
      input [1:0] sel;        // from Control Unit (ALUControl signal)
      output reg [7:0] out;   // fed back to Accumulator MUX
      output reg zero, pos;   // used for JZ, JPOS instructions

      initial begin
            out = 0;
            zero = 1'b0;
            pos = 1'b0;
      end

      always @ (*) begin
            case(sel)
                  2'b00: begin // ADD
                        out = a + b;
                        if(a == 0) zero = 1;
                        else zero = 0;
                        if(a > 0) pos = 1;
                        else pos = 0;
                  end
                  2'b01: begin // SUB
                        out = a - b;
                        if(a == 0) zero = 1;
                        else zero = 0;
                        if(a > 0) pos = 1;
                        else pos = 0;
                  end
                  2'b10: begin // AND
                        out = a & b;
                        if(a == 0) zero = 1;
                        else zero = 0;
                        if(a > 0) pos = 1;
                        else pos = 0;
                  end
            endcase
      end
endmodule
```

buff.v (Tri-State Buffer)

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Tri-State Buffer Module
 */
module buff(
      output reg [7:0] result,      // Data Output
      input[7:0] a,                 // Tri-State Bus Control
      input buf1                    // Data Input
);
      always @(*)
            if(buf1 == 1)
                  result = a;
            else
                  result = 8'bzzzz_zzzz;
endmodule
```

flopr.v (Resettable Register)

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Resettable Flip-Flop Module
 * (Accumulator & Instruction Register)
 */
module flopr(
      input clk, clear, load,       // clock, clear, & load
      input [7:0] d,                // data in
      output reg [7:0] q            // data out
);
      // used for Accumulator & Instruction Register
      always @(posedge clk, posedge clear)
            if(clear) q <= 0;
            else if(load) q <= d;
endmodule
```

incrementer.v (Incrementer)

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Incrementer Module
 */
module incrementer (input [4:0] a, output [4:0] y);
      assign y = a + 1; // takes 5-bit input and adds 1, used for PC
endmodule
```

mux2w5.v (2-to-1 MUX with 5-bit Data Width)

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * 2-to-1 MUX with 5-bit Data Width Module
 */
module mux2w5(
      input [4:0] d0, d1,      // Two Data Inputs
      input s,                 // Selector Input (from Control Unit)
      output [4:0] y           // MUX Output
);
      assign y = s ? d1 : d0;
endmodule
```

mux2w8.v (2-to-1 MUX with 8-bit Data Width)

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * 2-to-1 MUX (8-bit data width) Module
 */
module mux2w8(
      input [7:0] d0, d1,      // Two Data Inputs
      input s,                 // Selector Input (from Control Unit)
      output [7:0] y           // MUX Output
);
      assign y = s ? d1 : d0;
endmodule
```

programCounter.v (Program Counter Register)

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Program Counter Register Module
 */
module programCounter(
      input clk, clear, load,        // Clock, Clear, & Load Signal
      input [4:0] d,                 // Data Input
      output reg [4:0] q             // Data Output
);
      always @(posedge clk, posedge clear)
            if(clear) q <= 0;
            else if(load) q <= d;
endmodule
```

Datapath – datapath.v

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Datapath Module
 */
module datapath(
      input clk, reset,                    // Clock & Reset Input
      input PCWrite, IorD, PCSrc,          // *
      input IRWrite, AccWrite, OutWrite,   // * Signals from Control Unit
      input [1:0] AccSrc, ALUControl,      // *
      output zero, pos,                    // Branch Condition Signals: used
for JZ, JPOS instructions
      output [4:0] PC,                     // Program Counter Output
      input [7:0] memOut, inFromOutside,   // Output From Memory & Outside:
comes from Top Module
      output [7:0] IROut,                  // Instruction Register Output
      output [7:0] AccOut, outToOutside    // Accumulator Output & Output to
Outside: used for Top Module
);
      wire [4:0] pcRegIn, pcRegOut, PCplus1;
      wire [7:0] accMUX1out, accMUX2out, accIn;
      wire [7:0] ALUOut;

      // next PC logic
      mux2w5 pcMUX(PCplus1, IROut[4:0], PCSrc, pcRegIn);
      programCounter pcReg(clk, reset, PCWrite, pcRegIn, pcRegOut);
      incrementer PCadd(pcRegOut, PCplus1);
      mux2w5 memAddrIn(pcRegOut, IROut[4:0], IorD, PC);

      // instruction register
      flopr instrReg(clk, reset, IRWrite, memOut, IROut);

      // accumulator register
      mux2w8 accMUX1(ALUOut, memOut, AccSrc[0], accMUX1out);
      mux2w8 accMUX2(inFromOutside, 8'b0, AccSrc[0], accMUX2out);
      mux2w8 accMUX(accMUX1out, accMUX2out, AccSrc[1], accIn);
      flopr accumulator(clk, reset, AccWrite, accIn, AccOut);

      // ALU logic
      alu ALU(AccOut, memOut, ALUControl, ALUOut, zero, pos);

      // out tri-state buffer
      buff outBuff(outToOutside, AccOut, OutWrite);
endmodule
```

Controller/Control Unit – controller.v

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Controller Module
 */
module controller(
      input clk, reset, start,      // Clock, Reset, & Start Signal
      input [7:0] instr,            // Instruction Input
      output PCSrc,
      output reg PCWrite, IorD, MemWrite,
      output reg IRWrite, OutWrite,
      output reg [1:0] AccSrc, ALUControl,
      output reg AccWrite,
      input zero, pos               // Branch Condition Input (from ALU)
);
      reg branch, ZorP;
      reg [4:0] state;
      reg [4:0] nextstate;
      parameter S0 = 0; // reset
      parameter S1 = 1;
      parameter S2 = 2;
      parameter S3 = 3;
      parameter S4 = 4;
      parameter S5 = 5;
      parameter S6 = 6;
      parameter S7 = 7;
      parameter S8 = 8;
      parameter S9 = 9;
      parameter S10 = 10;
      parameter S11 = 11;

      always @(posedge clk, posedge reset)
            if(reset) state <= S0;
            else state <= nextstate;

      always @(state, instr, start) begin
      case(state)
            S0: begin
                  if(start) begin
                        PCWrite <= 1'b0;
                        IorD <= 1'b0;
                        MemWrite <= 1'b0;
                        IRWrite <= 1'b1;
                        OutWrite <= 1'b0;
                        AccSrc <= 2'bxx;
                        AccWrite <= 1'b0;
                        ALUControl <= 2'bxx;
                        branch <= 1'b0;
                        ZorP <= 1'bx;
                        nextstate <= S1;
                  end
                  else nextstate <= S0;
            end

            S1: begin
```

```verilog
            PCWrite <= 1'b1;
            IRWrite <= 1'b0;
            if(instr[7:5] == 3'b000) nextstate <= S2; //opcode = LOAD
            if(instr[7:5] == 3'b001) nextstate <= S3; //opcode = STORE
            if(instr[7:5] == 3'b010) nextstate <= S4; //opcode = ADD
            if(instr[7:5] == 3'b011) nextstate <= S5; //opcode = SUB
            if(instr[7:5] == 3'b100) nextstate <= S6; //opcode = AND
            if(instr[7:5] == 3'b101) nextstate <= S7; //opcode = JZ
            if(instr[7:5] == 3'b110) nextstate <= S8; //opcode = JPOS
            if(instr[7:3] == 5'b11100) nextstate <= S9; //opcode = IN
            if(instr[7:3] == 5'b11101) nextstate <= S10; //opcode = OUT
            if(instr[7:4] == 4'b1111) nextstate <= S11; //opcode = HALT
      end

      S2: begin // LOAD
            PCWrite <= 1'b0;
            IorD <= 1'b1;
            AccWrite <= 1'b1;
            AccSrc <= 2'b01;
            nextstate <= S0;
      end

      S3: begin // STORE
            PCWrite <= 1'b0;
            IorD <= 1'b1;
            MemWrite <= 1'b1;
            nextstate <= S0;
      end

      S4: begin // ADD
            PCWrite <= 1'b0;
            IorD <= 1'b1;
            ALUControl <= 2'b00;
            AccSrc <= 2'b00;
            AccWrite <= 1'b1;
            nextstate <= S0;
      end

      S5: begin // SUB
            PCWrite <= 1'b0;
            IorD <= 1'b1;
            ALUControl <= 2'b01;
            AccSrc <= 2'b00;
            AccWrite <= 1'b1;
            nextstate <= S0;
      end

      S6: begin // AND
            PCWrite <= 1'b0;
            IorD <= 1'b1;
            ALUControl <= 2'b10;
            AccSrc <= 2'b00;
            AccWrite <= 1'b1;
            nextstate <= S0;
      end
```

```
            S7: begin // JZ
                  PCWrite <= zero;
                  branch <= 1'b1;
                  ZorP <= 1'b0;
                  nextstate <= S0;
            end

            S8: begin // JPOS
                  PCWrite <= pos;
                  branch <= 1'b1;
                  ZorP <= 1'b1;
                  nextstate <= S0;
            end

            S9: begin // IN
                  PCWrite <= 1'b0;
                  AccSrc <= 2'b10;
                  AccWrite <= 1'b1;
                  nextstate <= S0;
            end

            S10: begin // OUT
                  PCWrite <= 1'b0;
                  OutWrite <= 1'b1;
                  nextstate <= S0;
            end

            S11: begin // HALT
                  PCWrite <= 1'b0;
                  nextstate <= S11;
            end
            default: nextstate <= S0;
      endcase
      end
      assign PCSrc = ZorP ? (branch & pos) : (branch & zero);
endmodule
```

Microprocessor – microProc.v

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Microprocessor Module
 */
module microProc(
      input clk, reset, start,            // Clock, Reset, & Start Signal
      output [4:0] PC,                    // Program Counter Output
      input [7:0] inFromOutside, memOut,  // Input From Outside, Memory
Output
      output OutWrite, MemWrite,
      output [7:0] AccOut, outToOutside
);
      wire PCWrite, IorD, PCSrc, IRWrite, AccWrite;
      wire [1:0] ALUControl, AccSrc;
      wire zero, pos;
      wire [7:0] IROut;
	controller CU(
		clk, reset, start, IROut, PCSrc, PCWrite,
		IorD, MemWrite, IRWrite, OutWrite, AccSrc,
		ALUControl, AccWrite, zero, pos
	);
	datapath DP(
		clk, reset, PCWrite, IorD, PCSrc,
		IRWrite, AccWrite, OutWrite,
		AccSrc, ALUControl, zero, pos, PC,
		memOut, inFromOutside, IROut,
		AccOut, outToOutside
	);
endmodule
```

Combined Instruction/Data Memory – mem.v

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Combined Instruction/Data Memory Module
 */
module mem(
	input CLK,		// Clock Input
	input [4:0] addr,	// Address Input (from MUX)
	input [7:0] WD,		// Write Data Input
	input WE,		// Write Enable Input
	output [7:0] RD		// Read Data Output
);
	reg [7:0] RAM[31:0]; // RAM limited to 32-bit
	initial $readmemb ("memfile.dat", RAM);
	assign RD = RAM[addr];
	always @(posedge CLK)
		if(WE) RAM[addr] <= WD;
endmodule
```

Top Module – top.v

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Top Module
 */
module top(
	input clk, reset, start,			// Clock, Reset, &
Start Input
	input [7:0] inFromOutside,			// Input from Outside
	output [7:0] AccOut, outToOutside, memOut,	// Accumulator Output,
Output from Outside, & Memory Output
	output [4:0] PC,				// Program
Counter/Address to the Memory
	output MemWrite, OutWrite			// Memory Write &
OutWrite Signal (from Control Unit)
);
	microProc mp(
		clk, reset, start, PC, inFromOutside, memOut,
		OutWrite, MemWrite, AccOut, outToOutside
	);
	mem memory(clk, PC, AccOut, MemWrite, memOut);
endmodule
```

Testbench Module (For N = 5) – testb.v

```
/* Reiner Dizon
 * CPE 300 Take Home Final
 * Simple 8-bit microprocessor (MP-8)
 * Testbench Module
 */
module testb;
      reg clk, reset, start;
      reg [7:0] inFromOutside;
      wire [7:0] AccOut, outToOutside, memOut;
      wire [4:0] addr;
      wire MemWrite, OutWrite;

      // device under test
      top dut(clk, reset, start, inFromOutside, AccOut, outToOutside, memOut,
addr, MemWrite, OutWrite);

      initial begin
            start <= 0;
            reset <= 1;
            #22;
            start <= 1;
            #22;
            reset <= 0;
            inFromOutside <= 5;
      end

      always begin
            clk <= 0;
            #5;
            clk <= 1;
            #5;
      end

      always @(posedge clk) begin
            if(addr === 30 && memOut === 15) begin
                        $display ("Write Data:      Expected: 15
Actual: %d", memOut);
                        $display ("Simulation succeeded");
            end
            else if(addr === 16) begin
                  if(AccOut !== 15) begin
                        $display ("Write Data:      Expected: 15
Actual: %d", AccOut);
                        $display ("Simulation failed");
                  end
                  $display ("Simulation done");
                  $stop;
            end
      end
endmodule
```

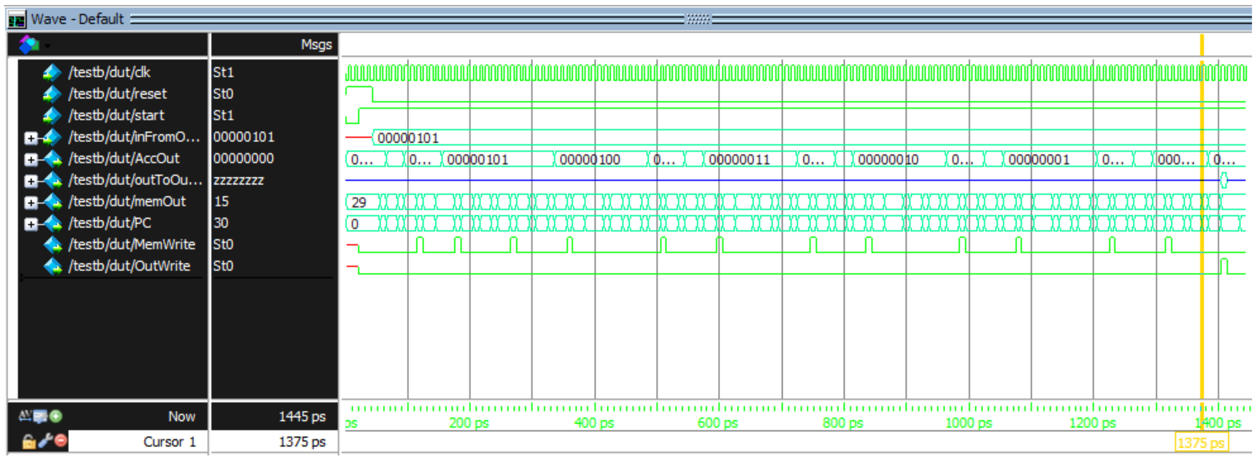
Memory Initialization File – memfile.dat

```
00011101
01111101
00111110
11100111
00111111
00011111
01011110
00111110
00011111
01111101
00111111
10101101
11000101
00011110
11101111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
11111111
00000001
00000000
00000000
```

4. Outcomes of the Testbench

For N = 5:

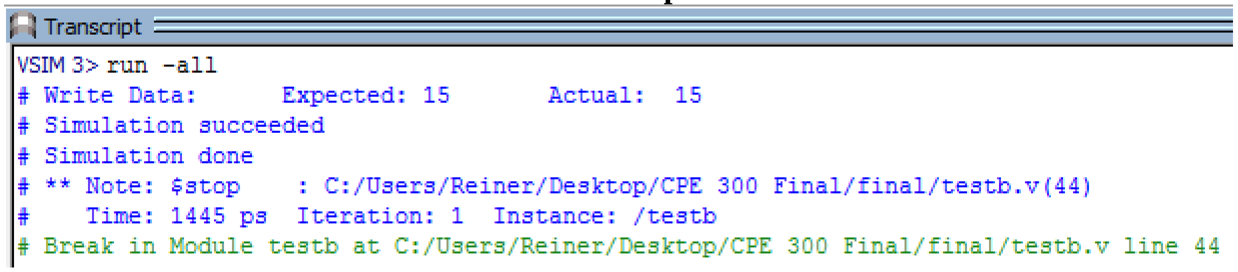
Waveform



```
Wave - Default
                          Msgs
/testb/dut/clk            St1
/testb/dut/reset          St0
/testb/dut/start          St1
/testb/dut/inFromO...     00000101
/testb/dut/AccOut         00000000
/testb/dut/outToOu...     zzzzzzzz
/testb/dut/memOut         15
/testb/dut/PC             30
/testb/dut/MemWrite       St0
/testb/dut/OutWrite       St0

Now        1445 ps
Cursor 1   1375 ps
```

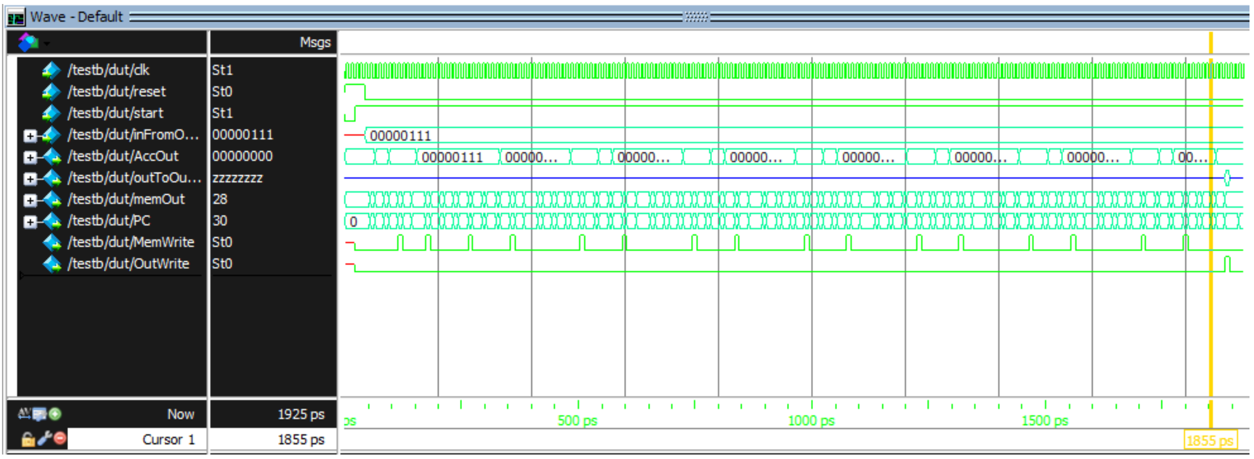
Transcript



```
Transcript
VSIM 3> run -all
# Write Data:      Expected: 15       Actual:  15
# Simulation succeeded
# Simulation done
# ** Note: $stop    : C:/Users/Reiner/Desktop/CPE 300 Final/final/testb.v(44)
#    Time: 1445 ps  Iteration: 1  Instance: /testb
# Break in Module testb at C:/Users/Reiner/Desktop/CPE 300 Final/final/testb.v line 44
```

For N = 7:

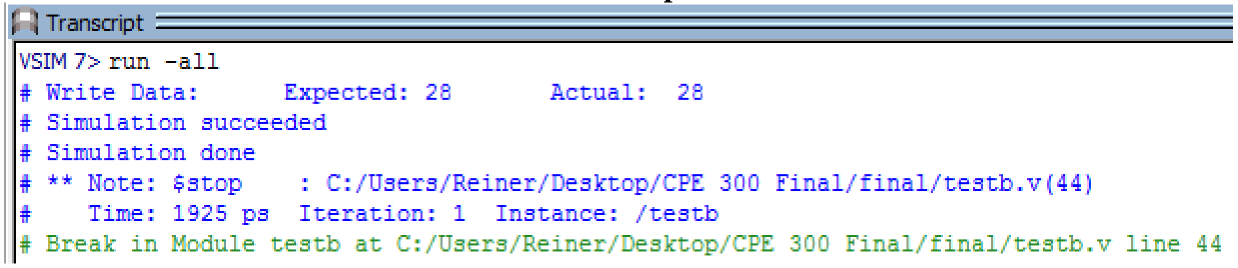
Waveform



```
Wave - Default
                          Msgs
/testb/dut/clk            St1
/testb/dut/reset          St0
/testb/dut/start          St1
/testb/dut/inFromO...     00000111
/testb/dut/AccOut         00000000
/testb/dut/outToOu...     zzzzzzzz
/testb/dut/memOut         28
/testb/dut/PC             30
/testb/dut/MemWrite       St0
/testb/dut/OutWrite       St0

Now                       1925 ps
Cursor 1                  1855 ps
```

Transcript



```
Transcript
VSIM 7> run -all
# Write Data:      Expected: 28       Actual:  28
# Simulation succeeded
# Simulation done
# ** Note: $stop    : C:/Users/Reiner/Desktop/CPE 300 Final/final/testb.v(44)
#    Time: 1925 ps  Iteration: 1  Instance: /testb
# Break in Module testb at C:/Users/Reiner/Desktop/CPE 300 Final/final/testb.v line 44
```