Edgar Granados

April 26, 2019

CpE 142- Section 01

Dr. Faroughi

Project 5: ACC-ISA CPU Verilog Modeling

Overview: The main objective of this project was to model the data path from HW 4(Problem 8.2) in Verilog. We would need to convert the instructions listed in table 1 to machine instructions, so that the ACC design would be able to execute them. Finally, we would need to print out the output of the model to make sure that the program executed correctly.

3-bit op-code	Instruction	RTN
001	LD data	ACC ← data //sign extended
010	LD (adrs)	ACC ← M[adrs]
011	ST (adrs)	M[adrs] ← ACC
100	ADD (adrs)	ACC ← ACC + M[adrs]
101	XOR (adrs)	ACC ← ACC ⊕ M[adrs]
110	JMP address	PP ←address //5 bits

Table 1: The following table lists the set of instructions that our design should contain

Source Code: The following code was written to model an ACC ISA, with its main memory organized as 32 x 1 memory. The instructions loaded into Instruction Memory would be 8 bits long, consisting of 3 bit opcodes, and 5bit operations. In addition, the operand of the instruction was Sign Extended to 8 bits once loaded onto the ACC register.

main module

```
/*
Project 5 :Acc-ISA Verilog Model
Created By: Edgar Granados and Geff Friere
April 25, 2019
CpE 142- Section 01
*/
module accISA(clock, reset, instruction, acc, programCounter);
input clock, reset;
output[7:0] instruction,acc;
output[5:0] programCounter;
```

```
reg[5:0] programCounter;
reg[2:0] opcode;
reg[7:0] operand;
reg[7:0] acc, result;

wire[4:0] imAddress;
wire[5:0] dmAddress;
reg[7:0] instruction;

assign imAddress = programCounter[4:0];
assign dmAddress = operand[4:0];



reg[7:0] IM[0:31];
reg[7:0] DM[0:31];

parameter
NOP = 0,
LD_I = 1,
LD_D = 2,
ST_D = 3,
ADD_D = 4,
XOR_D = 5,
JMP = 6;

//Initialize the IM register with machine instructions.
initial begin
IM[0] = 8'h3E; //LD -2
IM[1] = 8'h7F; //ST [X]
IM[2] = 8'h26; //LD 6
IM[3] = 8'h7E; //ST [Y]
IM[4] = 8'h2B; //LD 11
IM[5] = 8'h7D; //ST [Z]
IM[6] = 8'h5F; //LD [X]
```

```
IM[7] = 8'h9E; //ADD [Y]

IM[8] = 8'h7C; //ST [TEMP]

IM[9] = 8'h3F; //LD -1

IM[10] = 8'hBD;//XOR [Z]

IM[11] = 8'h7D;//ST [Z]

IM[12] = 8'h21;//LD 1

IM[13] = 8'h9D;//ADD [Z]

IM[14] = 8'h9C;//ADD [TEMP]

IM[15] = 8'h7B;//ST[T]

IM[16] = 8'hC0;//JMP First Instruction

end


//-----------------Read Instruction Memory ------------------------//

always @ (*)

begin

if(reset != 1)

begin

	opcode <= IM[imAddress][7:5]; //Get the first 3 bits from the instruction
located in IM[imAddress]

	operand[4:0] <= IM[imAddress][4:0];//In memory Location IM[imAddress] get the
remain five bits i.e. from Bit4 to Bit0

	operand[7:5] <= {3{operand[4]}}; //Sign Extend the operand to become an 8-bit
operand

	instruction <= {opcode, operand[4:0]};

end

end


//---------------Decode and Execute -------------------------------//


always @(*)

begin

	case(opcode)

		NOP: result <= 8'hXX;

		LD_I: result <= operand;

		LD_D: result <= DM[dmAddress];
```

```
		ST_D: result <= acc;
		ADD_D: result <= acc + DM[dmAddress];
		XOR_D: result <= acc ^ DM[dmAddress];
		JMP: result <= 8'hXX;

		default: result <= 8'hXX;
	endcase
end

//--------------------------Fetch and WriteBack--------------------//
always @(posedge clock, posedge reset)
begin
	if(reset)
	begin
		programCounter <= 0;
		acc <= 0;
		instruction <= 8'h00;
	end
	else
	begin
		case (opcode)
			NOP: programCounter <= programCounter + 1;
			LD_I:
			begin
				acc <= result;
				programCounter <= programCounter + 1;
			end
			LD_D:
			begin
				acc <= result;
				programCounter <= programCounter + 1;
			end
			ST_D:
			begin
```

```
                        DM[dmAddress] <= result;
                        programCounter <= programCounter + 1;
                end
                ADD_D:
                begin
                        acc <= result;
                        programCounter <= programCounter + 1;
                end
                XOR_D:
                begin
                        acc <= result;
                        programCounter <= programCounter + 1;
                end
                JMP:
                        programCounter <= operand[2:0];
        endcase
    end
end
endmodule
```

Test Bench

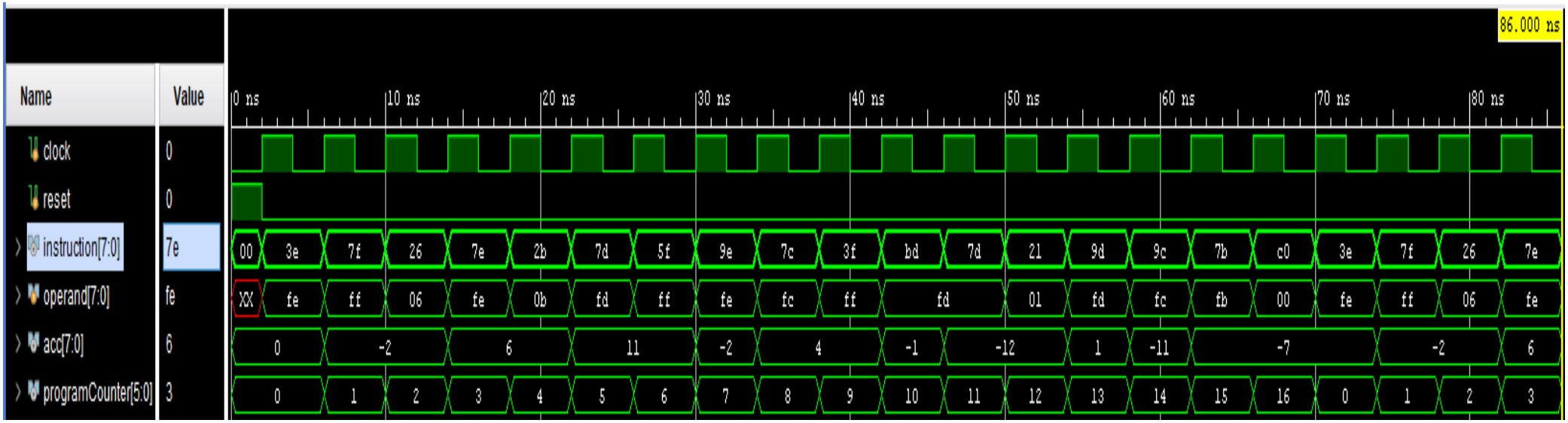
```
`timescale 1ns / 1ps
module accISA_tb();
reg clock, reset;
wire[7:0] instruction,acc;
wire[5:0] programCounter;
wire signed[7:0] decimal;
assign decimal = acc;
accISA uut(clock, reset, instruction, acc, programCounter);
initial clock = 0;
always #2 clock = ~clock;
initial begin
$display("Created By: Edgar Granados and Geff Friere\n");
reset = 1;
```

```
#2
reset = 0;
#80;
$stop;
$finish;
end
always@(posedge clock)
begin
$display("Time = %4d PP = %h ACC = Hex:%h  Signed Decimal:%d\n", $time,
programCounter, acc,decimal);
end
endmodule
```

Results:

```
Created By: Edgar Granados and Geff Friere

Time =    2 PP = 00 ACC = Hex:00  Signed Decimal:   0

Time =    6 PP = 00 ACC = Hex:00  Signed Decimal:   0

Time =   10 PP = 01 ACC = Hex:fe  Signed Decimal:  -2

Time =   14 PP = 02 ACC = Hex:fe  Signed Decimal:  -2

Time =   18 PP = 03 ACC = Hex:06  Signed Decimal:   6

Time =   22 PP = 04 ACC = Hex:06  Signed Decimal:   6

Time =   26 PP = 05 ACC = Hex:0b  Signed Decimal:  11

Time =   30 PP = 06 ACC = Hex:0b  Signed Decimal:  11

Time =   34 PP = 07 ACC = Hex:fe  Signed Decimal:  -2

Time =   38 PP = 08 ACC = Hex:04  Signed Decimal:   4

Time =   42 PP = 09 ACC = Hex:04  Signed Decimal:   4

Time =   46 PP = 0a ACC = Hex:ff  Signed Decimal:  -1

Time =   50 PP = 0b ACC = Hex:f4  Signed Decimal: -12

Time =   54 PP = 0c ACC = Hex:f4  Signed Decimal: -12

Time =   58 PP = 0d ACC = Hex:01  Signed Decimal:   1

Time =   62 PP = 0e ACC = Hex:f5  Signed Decimal: -11

Time =   66 PP = 0f ACC = Hex:f9  Signed Decimal:  -7

Time =   70 PP = 10 ACC = Hex:f9  Signed Decimal:  -7

Time =   74 PP = 00 ACC = Hex:f9  Signed Decimal:  -7

Time =   78 PP = 01 ACC = Hex:fe  Signed Decimal:  -2

Time =   82 PP = 02 ACC = Hex:fe  Signed Decimal:  -2

) relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:06 . Memory (MB): peak = 945.809 ; gain = 0.000
```

WaveForm:


86.000 ns
Name
Value
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
clock
0
reset
0
instruction[7:0]
7e
00 3e 7f 26 7e 2b 7d 5f 9e 7c 3f bd 7d 21 9d 9c 7b c0 3e 7f 26 7e
operand[7:0]
fe
XX fe ff 06 fe 0b fd ff fe fc ff fd 01 fd fc fb 00 fe ff 06 fe
acc[7:0]
6
0 -2 6 11 -2 4 -1 -12 1 -11 -7 -2 6
programCounter[5:0]
3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 0 1 2 3

Analysis:

At Time 0ns, reset = 1:

Thus, the Accumulator register and ProgramCounter(pp) are set to 0. In addition, I also included the signal: instruction to be set to a value of 0, since the model is being reset.

At Time 2ns, reset = 0 and IM [0] is Fetched (LD -2 or 3E in HEX):

At this time the first instruction in IM(**3E**) is fetched and executed. However, since this is the instant at which the instruction (LD -2) will be executed, the value of stored in ACC (-2 or FE in HEX) will not appear until the next rising edge cycle. The waveform and output results demonstrate this as expected.

At Time 6ns, IM [1] is Fetched (ST [X] or 7F in HEX):

At this time, the value in ACC has updated as expected to -2(FE); and the instruction stored in IM[1] is executed. Since this instruction is a STORE, no modification to the Accumulator Register is required. The waveform and output results demonstrate this as expected.

At Time 10ns, IM [2] is Fetched (LD 6 or 26 in HEX):

At this time, the instruction in IM[2](26) is fetched and executed. Once change in ACC will not show until the next rising edge cycle, because at this very instant the instruction is executed. The waveform and output results demonstrate this as expected.

At Time 14ns, IM [3] is Fetched (ST [Y] or 7E in HEX):

At this time, the fourth instruction ST [Y] is fetched and executed. The value of the ACC is stored in Memory Address [Y]. The waveform and output results demonstrate this as expected.

At Time 18ns, IM [4] is Fetched (LD 11 or 2B in HEX):

At this time, the model fetches and executes the 5th instruction LD 11. Since, this instruction is being executed at this very instant, the change in ACC won't be visible until the next rising edge cycle. The waveform and output results demonstrate this as expected.

At Time 22ns, IM [5] is Fetched (ST[Z] or 7D in HEX):

At this time the value in the ACC register is stored onto Memory Address Z. The waveform and output results demonstrate this as expected.

At Time 26ns, IM [6] is Fetched (LD [X] or 5F in HEX):

At this time, the value stored in Memory X (-2) is loaded onto the ACC register. Again, the change will be visible on the next rising edge clock. The waveform and output results demonstrate this as expected.

At Time 30ns, IM [7] is Fetched (ADD[Y] or 9E in HEX):

At this time, ACC holds the value of -2 and we execute instruction 9E, which is going to ADD the value stored in Memory Address [Y] to the ACC Register. Therefore, the next clock cycle should show ACC = 4. The waveform and output results demonstrate this as expected.

At Time 34ns, IM [8] is Fetched (ST [TEMP] or 7C in HEX):

At this time the contents stored in ACC register (4) is saved onto Memory Address TEMP. Because this is a store instruction, the contents of ACC should not change. The waveform and output results demonstrate this as expected.

At Time 38ns, IM [9] is Fetched (LD -1 or 3F in HEX):

At this time, we execute the instruction LD -1, which will change the contents of the ACC register. Again, this change will not appear until the very next clock cycle because this instruction is being executed at this very instant. The waveform and output results demonstrate this as expected.

At Time 42ns, IM [10] is Fetched (XOR [Z] or BD in HEX):

At this time we are going to XOR the ACC [-1] with the value stored in memory [Z]. This will give us the 1's complement of that value. The waveform and output results demonstrate this as expected.

At Time 46ns, IM [11] is Fetched (ST [Z] or 7D in HEX):

At this time, we are going to store the value that ACC contains onto Memory Address [Z]

At Time 50ns, IM [12] is Fetched (LD 1 or 21 in HEX):

In this instruction, we load the ACC register with the immediate value 1. Again, the change in the ACC register will be present on the next rising edge clock cycle. The waveform and output results demonstrate this as expected.

At Time 54ns, IM [13] is Fetched (ADD [Z] or 9D in HEX):

At this time, the ACC register contains the value 1. We are going to add the value located in Memory Address Z to the ACC register. The change in ACC, will appear on the next rising edge clock cycle. The waveform and output results demonstrate this as expected.

At Time 58ns, IM [14] is Fetched (ADD [TEMP] or 9C in HEX):

At this time, the ACC register contains the value -11. We are going to add the value located in Memory Address TEMP to the ACC register. The change in ACC, will appear on the next rising edge clock cycle. The waveform and output results demonstrate this as expected.

At Time 62ns, IM [15] is Fetched (ST[T] or 7B in HEX):

At this time, the value that is stored in the ACC register (-7) will be stored in Memory Address T. This value is the result of the expression T = X + Y - Z. The waveform and output results demonstrate this as expected.

At Time 66ns, IM [16] is Fetched (JMP 0 or C0 in HEX):

At this time, we execute a JUMP instruction that will bring us to the beginning of the program. PP will change on the next rising edge cycle, but ACC will remain the same. The waveform and output results demonstrate this as expected.

All subsequent instructions (after JUMP instruction) are the same instructions that were executed before.

Analysis Result:

We were asked to write a program to execute the following expression T = X + Y – Z. With

X = -2

Y= 6

Z = 11

The result should have been -7 and from our Verilog model our final result that was stored in Memory Address T was also -7.

Analysis (Part II: Modified Program):

For the modified program, all we needed to do was modify instructions IM[0], IM[2], and IM[4]

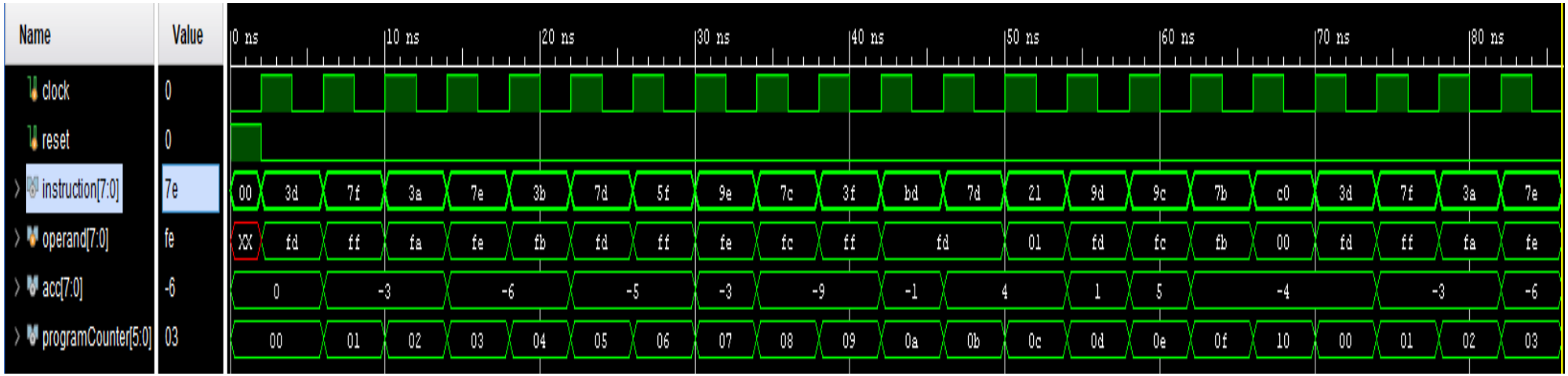
to the new respective values of: X = -3, Y = -6, and Z =-5.

```
//Initialize the IM register with machine instructions.
initial begin
IM[0] = 8'h3D; //LD -3
IM[1] = 8'h7F; //ST [X]
IM[2] = 8'h3A; //LD -6
IM[3] = 8'h7E; //ST [Y]
IM[4] = 8'h3B; //LD -5
IM[5] = 8'h7D; //ST [Z]
IM[6] = 8'h5F; //LD [X]
IM[7] = 8'h9E; //ADD [Y]
IM[8] = 8'h7C; //ST [TEMP]
IM[9] = 8'h3F; //LD -1
IM[10] = 8'hBD;//XOR [Z]
IM[11] = 8'h7D;//ST [Z]
IM[12] = 8'h21;//LD 1
IM[13] = 8'h9D;//ADD [Z]
IM[14] = 8'h9C;//ADD [TEMP]
IM[15] = 8'h7B;//ST[T]
IM[16] = 8'hC0;//JMP First Instruction
end
```

Results (Part II):

```
Created By: Edgar Granados and Geff Friere

Time =    2 PP = 00 ACC = Hex:00  Signed Decimal:   0

Time =    6 PP = 00 ACC = Hex:00  Signed Decimal:   0

Time =   10 PP = 01 ACC = Hex:fd  Signed Decimal:  -3

Time =   14 PP = 02 ACC = Hex:fd  Signed Decimal:  -3

Time =   18 PP = 03 ACC = Hex:fa  Signed Decimal:  -6

Time =   22 PP = 04 ACC = Hex:fa  Signed Decimal:  -6

Time =   26 PP = 05 ACC = Hex:fb  Signed Decimal:  -5

Time =   30 PP = 06 ACC = Hex:fb  Signed Decimal:  -5

Time =   34 PP = 07 ACC = Hex:fd  Signed Decimal:  -3

Time =   38 PP = 08 ACC = Hex:f7  Signed Decimal:  -9

Time =   42 PP = 09 ACC = Hex:f7  Signed Decimal:  -9

Time =   46 PP = 0a ACC = Hex:ff  Signed Decimal:  -1

Time =   50 PP = 0b ACC = Hex:04  Signed Decimal:   4

Time =   54 PP = 0c ACC = Hex:04  Signed Decimal:   4

Time =   58 PP = 0d ACC = Hex:01  Signed Decimal:   1

Time =   62 PP = 0e ACC = Hex:05  Signed Decimal:   5

Time =   66 PP = 0f ACC = Hex:fc  Signed Decimal:  -4

Time =   70 PP = 10 ACC = Hex:fc  Signed Decimal:  -4

Time =   74 PP = 00 ACC = Hex:fc  Signed Decimal:  -4

Time =   78 PP = 01 ACC = Hex:fd  Signed Decimal:  -3

Time =   82 PP = 02 ACC = Hex:fd  Signed Decimal:  -3

relaunch_sim: Time (s): cpu = 00:00:01 ; elapsed = 00:00:07 . Memory (MB): peak = 1043.242 ; gain = 0.000
```

Waveform:



Analysis Result:

Since, the data path of the Verilog model is exactly the same as the previous one; the step by step analysis will be the same. Therefore, we can just analyze the result to see if it matches with our expected value. We were asked to write a program to execute the following expression T = X + Y – Z. With

X = -3

Y= -6

Z = -5

The result should have been -4 and from our Verilog model our final result that was stored in Memory Address T was also -4. At Time = 62 ns, the instruction ST [T] is executed to store the value in ACC (-4) onto the Memory Address T.

Contribution:

Edgar- 50 % writing the Verilog Code for the data path and testbench.

Geff- 50% Analyzing the Waveform and writing down results. Also, taking snippets of code/waveforms for final Report.