Evan Smith

SoC Final Project

CSE-664

8-Bit Processor

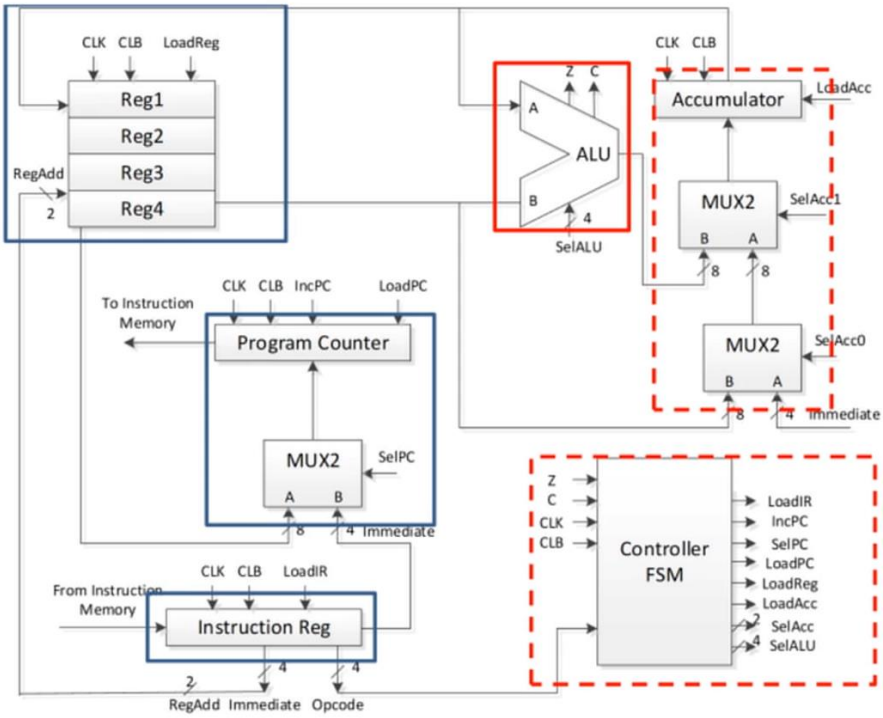
Design Overview

- 8 Bit words
- 255-word instruction memory size
- Instructions:

Code	Operation	Description
0000	NOP	Do no operations, increment PC
0001	ADD	Add ACC to IMM Register
0010	SUB	Subtract ACC from IMM Register
0011	NOR	Perform NOR on ACC and IMM
0100	LRA	Load IMM Register to ACC
0101	LAC	Load ACC to IMM Register
0110	ZBR	Branch to instruction in IMM Register if ACC is zero
0111	ZBI	Branch to instruction in IMM if ACC is zero
1000	CBR	Branch to instruction in IMM Register if ACC is negative
1001	------	Not used
1010	CBI	Branch to instruction in IMM if ACC is negative
1011	SHL	Shift ACC left one bit (respecting sign)
1100	SHR	Shift ACC right one bit (respecting sign)
1101	LIA	Load IMM to ACC
1110	------	Not used
1111	HALT	Do no operations and do not increment PC or load instruction. EOF

- 2 CPI
- Components: ALU, Accumulator, FSM Controller, Instruction Reg, Instruction Memory (disk-based), Register File.

Datapath



Datapath is from spec, with the addition of an Instruction Memory component to read from disk rather than hard-coding the program in the testbench.

Logic Overview

My goal for this design was to focus on a behavior-modelling approach, since my previous experience was exclusively with structural models. To that end, I have combined the MUX units into the logical elements they support to make the datapath more concise. Additionally, this let me work on some of the more complex features of Verilog like file read/write to streamline the user experience.

The overall logical steps for this project are:

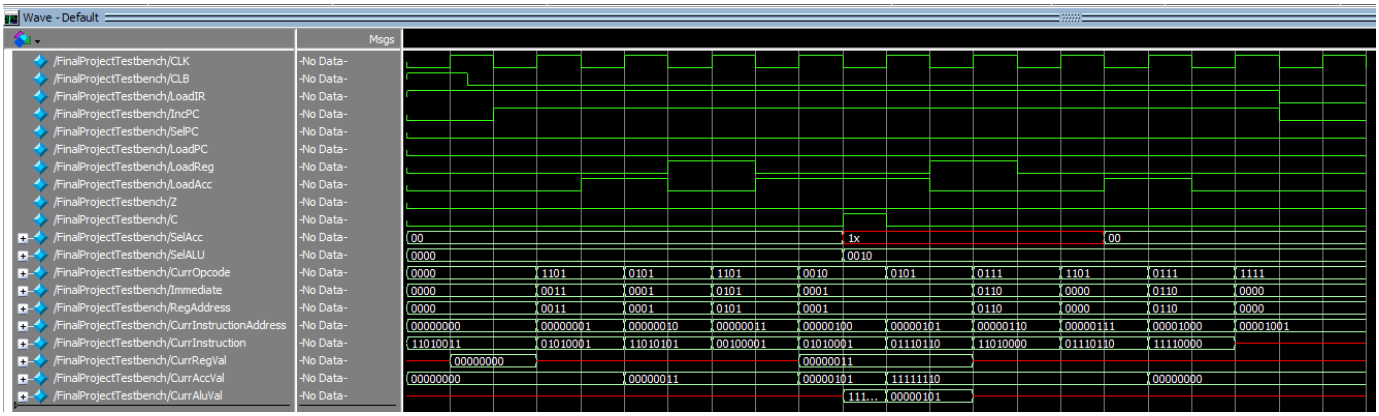
1. Reset
2. Load/Parse first instruction from memory
3. Determine command signals based on that instruction
4. Execute the command while parsing the next instruction
5. Repeat 3-4 until HALT operation is read.

This flow hinges on the fact that both the RegisterFile and InstructionMemory use constant assignment, so their output values update in phase with changes to their address inputs. This clock-independence allows the processor to be 2 CPI.

Sample Code

```
//Sample Program - a For-Loop!
11010011 //0-   Load 8
01010001 //1-   Store in Reg1
11010101 //2-   Load 4
00100001 //3-   Subtract Acc from Reg1
01010001 //4-   Store in Reg1
01110110 //5-   Z Jump to Instruction 8 (end)
11010000 //6-   Load 0
01110110 //7-   Z Jump to Instruction 2
11110000 //8-   HALT
```

Output



Waveform is hard to read, so I rigged up a write to file that helps show operation:

```
time =                    10
OpCode = 0000
Immediate = 0000
Acc = 00000000
CurrRegValue = 00000000

time =                    20
OpCode = 1101
Immediate = 0011
Acc = 00000000
CurrRegValue = xxxxxxxx

time =                    30
OpCode = 0101
Immediate = 0001
Acc = 00000011
CurrRegValue = xxxxxxxx

time =                    40
OpCode = 1101
Immediate = 0101
Acc = 00000011
CurrRegValue = xxxxxxxx

time =                    50
OpCode = 0010
Immediate = 0001
Acc = 00000101
CurrRegValue = 00000011

time =                    60
OpCode = 0101
Immediate = 0001
Acc = 11111110
CurrRegValue = 00000011

time =                    70
OpCode = 0111
Immediate = 0110
Acc = 11111110
CurrRegValue = xxxxxxxx

time =                    80
OpCode = 1101
Immediate = 0000
Acc = 11111110
CurrRegValue = xxxxxxxx

time =                    90
OpCode = 0111
Immediate = 0110
Acc = 00000000
CurrRegValue = xxxxxxxx

time =                   100
OpCode = 1111
Immediate = 0000
Acc = 00000000
CurrRegValue = xxxxxxxx

time =                   110
OpCode = 1111
Immediate = 0000
Acc = 00000000
CurrRegValue = xxxxxxxx
```