

# EE 337 / EE 309 - Microprocessor Design Report

OJAS APOORVA KANHERE  
12D070002

PRAVEEN AGRAWAL  
12D020030

KOTWAL ALANKAR SHASHIKANT  
12D070010

October 31, 2014

# 1 Introduction

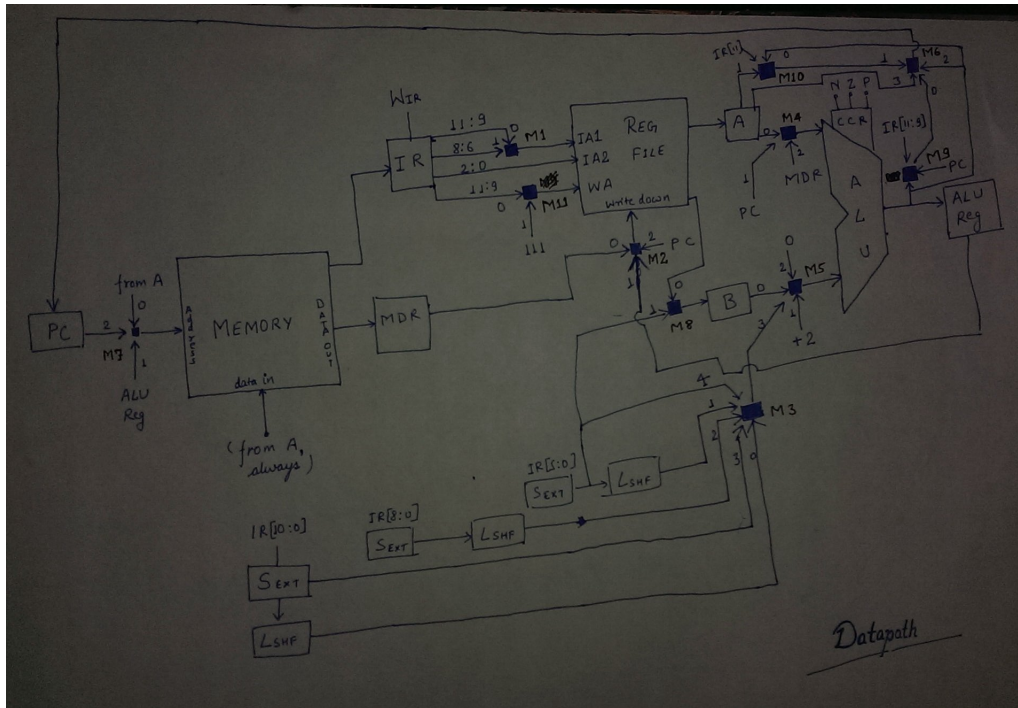
The aim of this exercise was to design and implement an 8-bit RISC processor on an FPGA. The ISA used was the LC-3b instruction set. You may find the instruction set [here](#).

In RISC design, we have to choose between a multi-cycle implementation and a single-cycle implementation. We chose to implement a multi-cycle data-path because all instructions do not take an equal amount of time, and the multi-cycle implementation makes sure that each instruction gets over in the minimum amount of time it requires. Also the single-cycle data-path is more complicated as compared to the multi-cycle data-path, and we save significant resources on the FPGA by doing a multi-cycle implementation.

You may find all the code for our project [here](#)

# 2 Data-path

The data-path we designed is summarised in the diagram below.



### 3 Controller

The controller implemented has two parts: logic driving current control signals and next state logic. The code for the current control signal values is as follows:

```
1 always@(*) begin
2     case(StateID)
3         1: begin
4             Mux1 = 1'b1;
5             Mux2 = 2'b11;
6             Mux3 = 3'b111;
7             Mux4 = 2'b01;
8             Mux5 = 2'b01;
9             Mux6 = 2'b10;
10            Mux7 = 2'b10;
11            Mux11 = 1'b1;
12
13            wrf = 1'b1;
14            wpc = ~|IR;
15            wir = 1'b0;
16            lccr = 1'b1;
17            aluop = 2'b00;
18            alushop = 2'b11;
19            wmem = 1'b1;
20            wa = 1'b1;
21            wb = 1'b1;
22            lalu = 1'b1;
23        end
24
25        2: begin
26            Mux1 = 1'b1;
27            Mux2 = 2'b11;
28            Mux3 = 3'b111;
29            Mux4 = 2'b11;
30            Mux5 = 2'b11;
31            Mux6 = 2'b11;
32            Mux7 = 2'b11;
33            Mux11 = 1'b1;
34
35            wrf = 1'b1;
36            wpc = 1'b1;
37            wir = 1'b1;
38            lccr = 1'b1;
39            aluop = 2'b11;
40            alushop = 2'b11;
41            wmem = 1'b1;
42            wa = 1'b0;
43            wb = 1'b0;
```

```

    lalu = 1'b1;
45 end

3: begin
    Mux1 = 1'b1;
47 Mux2 = 2'b11;
49 Mux3 = 3'b111;
    Mux4 = 2'b00;
51 Mux5 = 2'b00;
    Mux6 = 2'b11;
53 Mux7 = 2'b11;
    Mux11 = 1'b1;
55

    wrf = 1'b1;
    wpc = 1'b1;
57 wir = 1'b1;
    lccr = 1'b0;
59 aluop = {IR[15], IR[14]};
61 alushop = {IR[5], IR[4]};
    wmem = 1'b1;
63 wa = 1'b1;
    wb = 1'b1;
65 lalu = 1'b0;
67 end

4: begin
    Mux1 = 1'b1;
69 Mux2 = 2'b01;
71 Mux3 = 3'b111;
    Mux4 = 2'b11;
73 Mux5 = 2'b11;
    Mux6 = 2'b11;
75 Mux7 = 2'b11;
    Mux11 = 1'b0;
77

    wrf = 1'b0;
    wpc = 1'b1;
79 wir = 1'b1;
    lccr = 1'b1;
81 aluop = 2'b11;
    alushop = 2'b11;
83 wmem = 1'b1;
    wa = 1'b1;
85 wb = 1'b1;
    lalu = 1'b1;
87 end
89

5: begin
91 Mux1 = 1'b1;

```

```

93      Mux2 = 2'b11;
      Mux3 = 3'b010;
95      Mux4 = 2'b01;
      Mux5 = 2'b11;
97      Mux6 = 2'b00;
      Mux7 = 2'b11;
99      Mux11 = 1'b1;

101     wrf = 1'b1;
      wpc = 1'b0;
103     wir = 1'b1;
      lccr = 1'b1;
105     aluop = 2'b00;
      alushop = 2'b11;
107     wmem = 1'b1;
      wa = 1'b1;
109     wb = 1'b1;
      lalu = 1'b1;
111 end

113 6: begin
      Mux1 = 1'b1;
115     Mux2 = 2'b11;
      Mux3 = 3'b111;
117     Mux4 = 2'b11;
      Mux5 = 2'b11;
119     Mux6 = 2'b11;
      Mux7 = 2'b11;
121     Mux11 = 1'b1;

123     wrf = 1'b1;
      wpc = 1'b0;
125     wir = 1'b1;
      lccr = 1'b1;
127     aluop = 2'b11;
      alushop = 2'b11;
129     wmem = 1'b1;
      wa = 1'b1;
131     wb = 1'b1;
      lalu = 1'b1;
133 end

135 7: begin
      Mux1 = 1'b1;
137     Mux2 = 2'b10;
      Mux3 = 3'b111;
139     Mux4 = 2'b11;
      Mux5 = 2'b11;
141     Mux6 = 2'b11;

```

```

143     Mux7 = 2'b11;
      Mux11 = 1'b1;

145     wrf = 1'b0;
      wpc = 1'b1;
147     wir = 1'b1;
      lccr = 1'b1;
149     aluop = 2'b11;
      alushop = 2'b11;
151     wmem = 1'b1;
      wa = 1'b1;
153     wb = 1'b1;
      lalu = 1'b1;
155 end

157 8: begin
      Mux1 = 1'b1;
159     Mux2 = 2'b11;
      Mux3 = 3'b000;
161     Mux4 = 2'b01;
      Mux5 = 2'b11;
163     Mux6 = 2'b01;
      Mux7 = 2'b11;
165     Mux11 = 1'b1;

167     wrf = 1'b1;
      wpc = 1'b0;
169     wir = 1'b1;
      lccr = 1'b1;
171     aluop = 2'b00;
      alushop = 2'b11;
173     wmem = 1'b1;
      wa = 1'b1;
175     wb = 1'b1;
      lalu = 1'b1;
177 end

179 9: begin
      Mux1 = 1'b1;
181     Mux2 = 2'b11;
      Mux3 = 3'b100;
183     Mux4 = 2'b00;
      Mux5 = 2'b11;
185     Mux6 = 2'b11;
      Mux7 = 2'b11;
187     Mux11 = 1'b1;

189     wrf = 1'b1;
      wpc = 1'b1;

```

```

191     wir = 1'b1;
192     lccr = 1'b1;
193     aluop = 2'b00;
194     alushop = 2'b11;
195     wmem = 1'b1;
196     wa = 1'b1;
197     wb = 1'b1;
198     lalu = 1'b0;
199 end

201 10: begin
202     Mux1 = 1'b0;
203     Mux2 = 2'b11;
204     Mux3 = 3'b111;
205     Mux4 = 2'b11;
206     Mux5 = 2'b11;
207     Mux6 = 2'b11;
208     Mux7 = 2'b11;
209     Mux11 = 1'b1;

210     wrf = 1'b1;
211     wpc = 1'b1;
212     wir = 1'b1;
213     lccr = 1'b1;
214     aluop = 2'b11;
215     alushop = 2'b11;
216     wmem = 1'b1;
217     wa = 1'b0;
218     wb = 1'b1;
219     lalu = 1'b1;
220 end

221 11: begin
222     Mux1 = 1'b1;
223     Mux2 = 2'b11;
224     Mux3 = 3'b111;
225     Mux4 = 2'b11;
226     Mux5 = 2'b11;
227     Mux6 = 2'b11;
228     Mux7 = 2'b01;
229     Mux11 = 1'b1;

230     wrf = 1'b1;
231     wpc = 1'b1;
232     wir = 1'b1;
233     lccr = 1'b1;
234     aluop = 2'b11;
235     alushop = 2'b11;
236     wmem = 1'b0;

```

```

241     wa = 1'b1;
242     wb = 1'b1;
243     lalu = 1'b1;
244 end
245
246 12: begin
247     Mux1 = 1'b1;
248     Mux2 = 2'b11;
249     Mux3 = 3'b001;
250     Mux4 = 2'b00;
251     Mux5 = 2'b11;
252     Mux6 = 2'b11;
253     Mux7 = 2'b11;
254     Mux11 = 1'b1;
255
256     wrf = 1'b1;
257     wpc = 1'b1;
258     wir = 1'b1;
259     lccr = 1'b1;
260     aluop = 2'b00;
261     alushop = 2'b11;
262     wmem = 1'b1;
263     wa = 1'b1;
264     wb = 1'b1;
265     lalu = 1'b0;
266 end
267
268 13: begin
269     Mux1 = 1'b1;
270     Mux2 = 2'b11;
271     Mux3 = 3'b111;
272     Mux4 = 2'b11;
273     Mux5 = 2'b11;
274     Mux6 = 2'b11;
275     Mux7 = 2'b01;
276     Mux11 = 1'b1;
277
278     wrf = 1'b1;
279     wpc = 1'b1;
280     wir = 1'b1;
281     lccr = 1'b1;
282     aluop = 2'b11;
283     alushop = 2'b11;
284     wmem = 1'b1;
285     wa = 1'b1;
286     wb = 1'b1;
287     lalu = 1'b1;
288 end

```



```

289 14: begin
      Mux1 = 1'b1;
291      Mux2 = 2'b00;
      Mux3 = 3'b111;
293      Mux4 = 2'b10;
      Mux5 = 2'b10;
295      Mux6 = 2'b11;
      Mux7 = 2'b01;
297      Mux11 = 1'b0;

      wrf = 1'b0;
      wpc = 1'b1;
301      wir = 1'b1;
      lccr = 1'b0;
303      aluop = 2'b00;
      alushop = 2'b11;
305      wmem = 1'b1;
      wa = 1'b1;
307      wb = 1'b1;
      lalu = 1'b1;
309 end

311 15: begin
      Mux1 = 1'b1;
313      Mux2 = 2'b11;
      Mux3 = 3'b010;
315      Mux4 = 2'b01;
      Mux5 = 2'b11;
317      Mux6 = 2'b11;
      Mux7 = 2'b11;
319      Mux11 = 1'b1;

      wrf = 1'b1;
      wpc = 1'b1;
323      wir = 1'b1;
      lccr = 1'b0;
325      aluop = 2'b00;
      alushop = 2'b11;
327      wmem = 1'b1;
      wa = 1'b1;
329      wb = 1'b1;
      lalu = 1'b0;
331 end

333 16: begin
      Mux1 = 1'b1;
335      Mux2 = 2'b01;
      Mux3 = 3'b111;
337      Mux4 = 2'b11;

```

```

339     Mux5 = 2'b11;
340     Mux6 = 2'b11;
341     Mux7 = 2'b11;
342     Mux11 = 1'b0;

343     wrf = 1'b0;
344     wpc = 1'b1;
345     wir = 1'b1;
346     lccr = 1'b1;
347     aluop = 2'b11;
348     alushop = 2'b11;
349     wmem = 1'b1;
350     wa = 1'b1;
351     wb = 1'b1;
352     lalu = 1'b1;
353 end

354 default: begin
355     Mux1 = 1'b0;
356     Mux2 = 2'b00;
357     Mux3 = 3'b000;
358     Mux4 = 2'b00;
359     Mux5 = 2'b00;
360     Mux6 = 2'b00;
361     Mux7 = 2'b00;
362     Mux11 = 1'b0;

363     wrf = 1'b1;
364     wpc = 1'b1;
365     wir = 1'b1;
366     lccr = 1'b1;
367     aluop = 2'b00;
368     alushop = 2'b00;
369     wmem = 1'b1;
370     wa = 1'b1;
371     wb = 1'b1;
372     lalu = 1'b1;
373 end
374 endcase
375 end
376 end
377 end

```

where, the control signals mean the following:

No	Signal Name	Purpose
1	Mux1	Mux1 Control Signal
2	Mux2	Mux2 Control Signal
3	Mux3	Mux3 Control Signal
4	Mux4	Mux4 Control Signal
5	Mux5	Mux5 Control Signal
6	Mux6	Mux6 Control Signal
7	Mux7	Mux7 Control Signal
8	Mux11	Mux11 Control Signal
9	wrf	Write to Register File
10	wpc	Write to Program Counter
11	wir	Write to Instruction Register
12	lccr	Write to Condition Code Register
13	aluop	ALU Operation
14	alushop	Shift Operation
15	wmem	Write to Memory
16	wa	Write to Temporary Register A
17	wb	Write to Temporary Register B
18	lalu	Load ALU Register

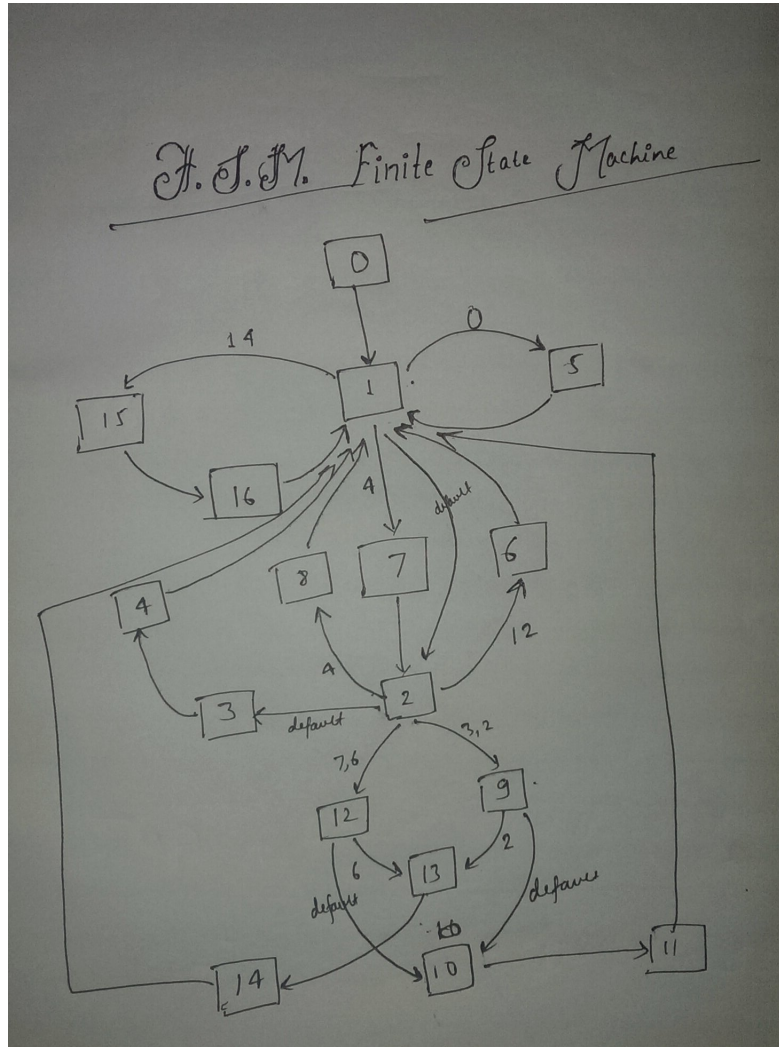
Mux8-Mux10 receive their inputs directly via combinational logic from IR. The logic is:

```

1 Mux8  = IR [ 5 ] ;
  Mux9  = ( ( IR[11]&&N ) || ( IR[10]&&Z ) || ( IR[9]&&P ) ) ;
3 Mux10 = IR [ 1 1 ] ;

```

The next state logic is summarised in the following diagram:



For each instruction the state sequence is shown below:

ALU Operation (Add, And, XOR, Not, Shifts):  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

Branch:  $1 \rightarrow 5$

Call Subroutine:  $1 \rightarrow 7 \rightarrow 2 \rightarrow 8$

Load Byte:  $1 \rightarrow 2 \rightarrow 9 \rightarrow 13 \rightarrow 14$

Load Word:  $1 \rightarrow 2 \rightarrow 12 \rightarrow 13 \rightarrow 14$

Load Immediate Address:  $1 \rightarrow 15 \rightarrow 16$

Store Byte: 1 → 2 → 9 → 10 → 11  
Store Word: 1 → 2 → 12 → 10 → 11

## 4 Simulation

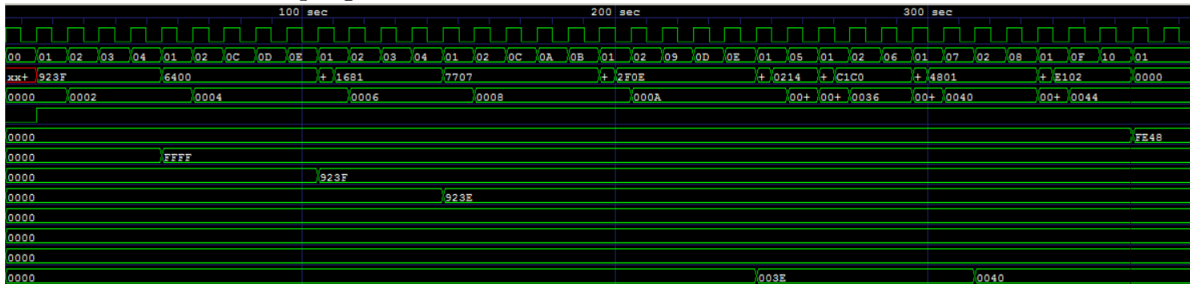
We simulated the implementation using a test code that contained all the necessary functionality. In assembly language as per the LC-3b document, the code is as follows:

```

1 NOT R1,R0
  LDW R2,R0,#0
3 ADD R3,R2,R1
  STW R3,R4,#7
5 LDB R7,R4,#14
  BRp label
7
  ORG 0x34
9 label:
  JMP R7;
11 JSR label1
13
  ORG 0x66
  label1:

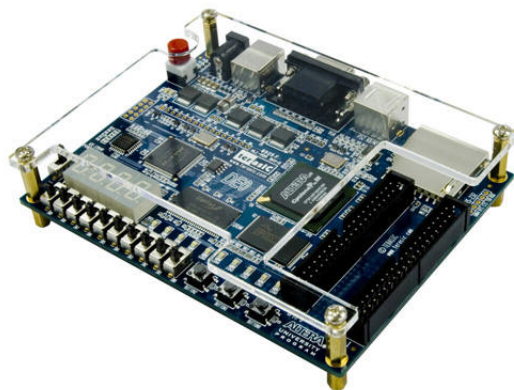
```

The simulation was carried out using GTKWave and iVerilog. The result for the above program were as shown below.



## 5 Testing

The implementation was tested on an Altera DE0-nano board having a Cyclone-IV FPGA.



The signal-tap tool was used to get states of the elements inside the FPGA at various points in time. For convenience in debugging we gave clock using a push-button on-board. The output was well as expected.