ELEC374 - Lab 3

Naod Dereje - 20103501, Thierry Jones - 20108349

April 12, 2021

Contents

1 Components

The purpose of this lab was to design, simulate, implement and verify a simple RISC computer(Mini SRC), including the circuits associated with the logic behind: Select and Encode, Memory Subsystem, CON FF and Input/Output ports. Verilog was chosen over VHDL as it is better for more complex simulations. The verilog code for each of these components can be found in the Appendices.

1.1 Select and Encode

Select and Encode logic was used for load and store instruction as well as add, and, or instructions. The opcode that is read into Select and Encode is used to create the outputs for Ra, Rb , and Rc, in order to generate the GRa, GRb, GRc, which is the encoded to R0in-R15in and R0out-R15out registers. Their is also support for a sign-extended C value in the lower 19 bits of the instrcution address.

1.2 Memory Subsystem

The memory subsystem was created using the Megafunction in Quartus to generate inputs and outputs for the RAM accordingly. In order to define the memory subsystem within our project, an instance of the RAM was created on our datapath, and was wired to the datapath using existing wires created in the previous lab. The RAM's memory is accessed through the corresponding hex file in the project, Which contains the opcodes for the instructions at the specififed memory slots. A detailed breakdown of the memory subsystem can be found at Appendix C.1.

1.3 CON FF

The conff logic is created to ensure that conidtional logic such as the branch insturction are able to be executed. The instructions assocaited with the conditional logic are stored in I-formatting, with the second register holding the branching condition. The components for the con ff logic can be found at Appendix D.

1.4 Input/Output Ports

The input and output ports in the datapath allow for the bus to receive values from an input unit, as well as output values from the bus onto the output unit. In a real implmentation of an I/O port instruction, the logic for these ports would be used to read from some input given to port, and give feedback to the system through some output such as updating the result of an arithmetic instruction, or the bit pattern after a bit shift or rotation was completed. The code for the input and output ports can be found at Appendix E.1 and Appendix E.2 respectively.

2 Circuitry Demonstration

To demonstrate the success of the RAM and memory interface logic, multiple testbenches were created to simulate each operation. The testbenches all have their own control sequences unique to every operation and can be found in the appendices. The following subsections of the cicuitry demonstration will detail the changes made in the test benches due to the control sequence changing as well as the waveforms generated by each simulation. Each instruction used to demonstrate the circuitry requires a specific OP Code. The following table shows the OP Code values stored in RAM initially, their address in RAM and also the instruction itself. For a full version of the initial RAM state, please see Appendix **??**. Furthermore, in an effort to save paper, only the relevant signals were included for each instruction.
*Note that these instructions were tested before the rest, thus, the .hex file hadn't be completed. Thus, Appendix **??** and Appendix **??** have empty values in RAM where the OP Codes are for later tests.

Address (hex)	Instruction	Op code (hex)
28	ld R1, 85	00800085
29	ld R1, $35(R0)	00800023
30	ldi R1,$85	08800085
31	ldi R1, $35(R0)	08800023
1*	st $90, R1	10080090
1*	st $90(R1), R1	10080090
8	addi R2, R1, $-5	590FFFFB
9	andi R2, R1, $26	61080026
a	ori R2, R1, $26	69080026
30	in R1	A8800000
31	out R1	B0800000
20	brzr R2, 35	91000023
21	brnz R2, 35	91200023
22	brpl R2, 35	91400023
23	brmi R2, 35	91600023
18	jal R1	A0800000
19	jr R1	98800000

Figure 1: Table of relevant values in RAM initially

2.1 Load Instructions

Two load instructions were tested to ensure that the values are able to load from the RAM onto its appropriate register. The control sequences for *ld R1,$85*, *ld R0, $35(R1)*, *ldi R1,$85*, *ldi R0, $35(R1)* can be found in Appendices F.3, F.4, F.5, and F.6 respectively. The value 85_{16} and 35_{16} was preloaded into the memory initialization file in slots 85_{16} and 35_{16} for these instructions respectively. The specific opcodes for these instructions can be found in Figure 2, and the relevant address was loaded into the PC register. The value of 0_{16} was preloaded into register 1.

A General Components

A.1 Control Unit

```
module control_unit(

output reg Gra,Grb, Grc, Rin, Rout, BAout, Cout, Zloout, Zhighout// here, you will define the inputs and outputs to your Control Unit
Yin, Zin,PCout, IncPC, MARin,Read, Write, Clear, ADD, AND, SHR,

input[31:0] IR,
input Clock, Reset, Stop, Con_FF);

parameter Initialize_Reset = -1 , Reset_state = 0 , fetch0 = 1 , fetch1 = 2 , fetch2 = 3 , // basic
 // add
 add3 = 4 , add4 = 5 , add5 = 6 ,
 // sub
 sub3 = 7 , sub4 = 8 , sub5 = 9 ,
 // shr
 shr3 = 10 , shr4 = 11 , shr5 = 12 ,
 // shl
 shl3 = 13 , shl4 = 14 , shl5 = 15 ,
 // ror
 ror3 = 16 , ror4 = 17 , ror5 = 18 ,
 // rol
 rol3 = 19 , rol4 = 20 , rol5 = 21 ,
 // and
 and3 = 22 , and4 = 23 , and5 = 24 ,
 // or
 or3 = 25 , or4 = 26 , or5 = 27 ,
 // addi
addi3 = 28 , addi4 = 29 , addi5 = 30 ,
 // andi
 andi3 = 31 , andi4 = 32 , andi5 = 33 ,
 // ori
 ori3 = 34 , ori4 = 35 , ori5 = 36 ,
 // mul
 mul3 = 37 , mul4 = 38 , mul5 = 39 , mul6 = 40 ,
 // div
 div3 = 41 , div4 = 42 , div5 = 43 , div6 = 44 ,
 // neg
neg3 = 45 , neg4 = 46 ,
 // not
 not3 = 47 , not4 = 48 ,
 // ld
 ld3 = 49 , ld4 = 50 , ld5 = 51 , ld6 = 52 , ld7 = 53 ,
 // ldi
 ldi3 = 54 , ldi4 = 55 , ldi5 = 56 ,
 // st
 st3 = 57 , st4 = 58 , st5 = 59 , st6 = 60 , st7 = 61 ,
 // bracnh
 branch3 = 62 , branch4 = 63 , branch5 = 64 , branch6 = 65 ,
 // jr
 jr3 = 76 ,
 // jal
 jal3 = 77 , jal4 = 78 ,
 // in
 in3 = 79 ,
```

```
54  // out
55  out3 = 80 ,
56  // mfhi
57  mfhi3 = 81 ,
58  // mflo
59  mflo3 = 82 ,
60  // nop
61  nop3 = 83 ,
62  // halt
63  halt3 = 84;
64
65  integer Present_state = Reset_state ;
66
67  always@ ( posedge Clock , posedge Reset , posedge Stop )
68   begin
69     if( Reset ) Present_state = #50 Initialize_Reset ;
70     if( Stop ) Present_state = halt3 ;
71     else case ( Present_state )
72       Initialize_Reset : #40 Present_state = Reset_state ;
73       Reset_state : #50 Present_state = fetch0 ;
74       fetch0 : #50 Present_state = fetch1 ;
75       fetch1 : #50 Present_state = fetch2 ;
76       fetch2 : #50 begin
77         case (IR [31:27])
78         5 ' b00000 : Present_state = ld3;
79         5 ' b00001 : Present_state = ldi3;
80         5 ' b00010 : Present_state = st3;
81         5 ' b00011 : Present_state = add3;
82         5 ' b00100 : Present_state = sub3;
83         5 ' b00101 : Present_state = shr3;
84         5 ' b00110 : Present_state = shl3;
85         5 ' b00111 : Present_state = ror3;
86         5 ' b01000 : Present_state = rol3;
87         5 ' b01001 : Present_state = and3;
88         5 ' b01010 : Present_state = or3;
89         5 ' b01011 : Present_state = addi3 ;
90         5 ' b01100 : Present_state = andi3 ;
91         5 ' b01101 : Present_state = ori3;
92         5 ' b01110 : Present_state = mul3;
93         5 ' b01111 : Present_state = div3;
94         5 ' b10000 : Present_state = neg3;
95         5 ' b10001 : Present_state = not3;
96         5 ' b10010 : Present_state =branch3;
97         5 ' b10011 : Present_state = jr3;
98         5 ' b10100 : Present_state = jal3;
99         5 ' b10101 : Present_state = in3;
100        5 ' b10110 : Present_state = out3;
101        5 ' b10111 : Present_state =mfhi3;
102        5 ' b11000 : Present_state =mflo3;
103        5 ' b11001 : Present_state = nop3;
104        5 ' b11010 : Present_state = halt3;
105        default : begin end
106      endcase
107    end
108    add3 : #50 Present_state = add4 ;
109    add4 : #50 Present_state = add5 ;
110    add5 : #50 Present_state = fetch0 ;
111    sub3 : #50 Present_state = sub4 ;
112    sub4 : #50 Present_state = sub5 ;
```

```
113     sub5 : #50 Present_state = fetch0 ;
114     shr3 : #50 Present_state = shr4 ;
115     shr4 : #50 Present_state = shr5 ;
116     shr5 : #50 Present_state = fetch0 ;
117     shl3 : #50 Present_state = shl4 ;
118     shl4 : #50 Present_state = shl5 ;
119     shl5 : #50 Present_state = fetch0 ;
120     ror3 : #50 Present_state = ror4 ;
121     ror4 : #50 Present_state = ror5 ;
122     ror5 : #50 Present_state = fetch0 ;
123     rol3 : #50 Present_state = rol4 ;
124     rol4 : #50 Present_state = rol5 ;
125     rol5 : #50 Present_state = fetch0 ;
126     and3 : #50 Present_state = and4 ;
127     and4 : #50 Present_state = and5 ;
128     and5 : #50 Present_state = fetch0 ;
129     or3 : #50 Present_state = or4 ;
130     or4 : #50 Present_state = or5 ;
131     or5 : #50 Present_state = fetch0 ;
132     addi3 : #50 Present_state = addi4 ;
133     addi4 : #50 Present_state = addi5 ;
134     addi5 : #50 Present_state = fetch0 ;
135     andi3 : #50 Present_state = andi4 ;
136     andi4 : #50 Present_state = andi5 ;
137     andi5 : #50 Present_state = fetch0 ;
138     ori3 : #50 Present_state = ori4 ;
139     ori4 : #50 Present_state = ori5 ;
140     ori5 : #50 Present_state = fetch0 ;
141     mul3 : #50 Present_state = mul4 ;
142     mul4 : #50 Present_state = mul5 ;
143     mul5 : #50 Present_state = mul6 ;
144     mul6 : #50 Present_state = fetch0 ;
145     div3 : #50 Present_state = div4 ;
146     div4 : #50 Present_state = div5 ;
147     div5 : #50 Present_state = div6 ;
148     div6 : #50 Present_state = fetch0 ;
149     neg3 : #50 Present_state = neg4 ;
150     neg4 : #50 Present_state = fetch0 ;
151     not3 : #50 Present_state = not4 ;
152     not4 : #50 Present_state = fetch0 ;
153     ld3 : #50 Present_state = ld4 ;
154     ld4 : #50 Present_state = ld5 ;
155     ld5 : #50 Present_state = ld6 ;
156     ld6 : #50 Present_state = ld7 ;
157     ld7 : #50 Present_state = fetch0 ;
158     ldi3 : #50 Present_state = ldi4 ;
159     ldi4 : #50 Present_state = ldi5 ;
160     ldi5 : #50 Present_state = fetch0 ;
161     st3 : #50 Present_state = st4 ;
162     st4 : #50 Present_state = st5 ;
163     st5 : #50 Present_state = st6 ;
164     st6 : #50 Present_state = st7 ;
165     st7 : #50 Present_state = fetch0 ;
166     branch3 : #50 Present_state = branch4 ;
167     branch4 : #50 Present_state = branch5 ;
168     branch5 : #50 Present_state = branch6 ;
169     branch6 : #50 Present_state = fetch0 ;
170     jr3 : #50 Present_state = fetch0 ;
171     jal3 : #50 Present_state = jal4 ;
```

```
172       jal4 : #50 Present_state = fetch0 ;
173       in3 : #50 Present_state = fetch0 ;
174       out3 : #50 Present_state = fetch0 ;
175       mfhi3 : #50 Present_state = fetch0 ;
176       mflo3 : #50 Present_state = fetch0 ;
177       nop3 : #50 Present_state = fetch0 ;
178       default : begin end
179     endcase
180   end
181
182   always@ ( Present_state )
183   begin
184      case ( Present_state )
185     Initialize_Reset : Clear <= 0;
186     Reset_state : begin
187       #10 Clear <= 0;
188       #15 Clear <= 1;
189       PCout <= 0; Zlowout <= 0; Zhighout <= 0;
190       MDRout <= 0; // initialize the signals
191       MARin <= 0; Zlowin <= 0; Zhighin <= 0;
192       PCin <= 0; MDRin <= 0; IRin <= 0; Yin <= 0;
193       IncPC <= 0; Read <= 0; Write <= 0;
194       highin <= 0; lowin <= 0;
195       Cout <= 0; outPortIn <= 0; inPortOut <= 0;
196       con_in <= 0;
197       highout <= 0; lowout <= 0;
198       ADD <= 0; SUB <= 0; SHR <= 0; SHL <= 0; ROR
199       <= 0; ROL <= 0; AND <= 0; OR <= 0;
200       MUL <= 0; DIV <= 0; NEG <= 0; NOT <= 0;
201       BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0;
202       Grb <= 0; Grc <= 0;
203       R15_enable <= 0; PC_enable <= 0;
204       Run <= 1;
205       end
206     fetch0 : begin
207       #10 PCout <= 1; MARin <= 1; IncPC <= 1;
208       Zlowin <= 1;
209       #15 PCout <= 0; MARin <= 0; IncPC <= 0;
210       Zlowin <= 0;
211       end
212     fetch1 : begin
213       #10 Zlowout <= 1; PCin <= 1; Read <= 1; MDRin<= 1;
214       #15 Zlowout <= 0; PCin <= 0; Read <= 0; MDRin<= 0;
215       end
216     fetch2 : begin
217       #10 MDRout <= 1; IRin <= 1;
218       #15 MDRout <= 0; IRin <= 0;
219       end
220     add3 : begin
221       #10 Grb <= 1; Rout <= 1; Yin <= 1;
222       #15 Grb <= 0; Rout <= 0; Yin <= 0;
223       end
224     add4 : begin
225       #10 Grc <= 1; Rout <= 1; Zlowin <= 1; ADD <=1;
226       #15 Grc <= 0; Rout <= 0; Zlowin <= 0; ADD <=0;
227       end
228     add5 : begin
229       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
230       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
```

```
231       end
232     sub3 : begin
233       #10 Grb <= 1; Rout <= 1; Yin <= 1;
234       #15 Grb <= 0; Rout <= 0; Yin <= 0;
235       end
236     sub4 : begin
237       #10 Grc <= 1; Rout <= 1; Zlowin <= 1; SUB <=1;
238       #15 Grc <= 0; Rout <= 0; Zlowin <= 0; SUB <=0;
239       end
240     sub5 : begin
241       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
242       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
243       end
244     shr3 : begin
245       #10 Grb <= 1; Rout <= 1; Yin <= 1;
246       #15 Grb <= 0; Rout <= 0; Yin <= 0;
247       end
248     shr4 : begin
249       #10 Grc <= 1; Rout <= 1; Zlowin <= 1; SHR <=1;
250       #15 Grc <= 0; Rout <= 0; Zlowin <= 0; SHR <=0;
251       end
252     shr5 : begin
253       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
254       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
255       end
256     shl3 : begin
257       #10 Grb <= 1; Rout <= 1; Yin <= 1;
258       #15 Grb <= 0; Rout <= 0; Yin <= 0;
259       end
260     shl4 : begin
261       #10 Grc <= 1; Rout <= 1; Zlowin <= 1; SHL <=1;
262       #15 Grc <= 0; Rout <= 0; Zlowin <= 0; SHL <=0;
263       end
264     shl5 : begin
265       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
266       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
267       end
268     ror3 : begin
269       #10 Grb <= 1; Rout <= 1; Yin <= 1;
270       #15 Grb <= 0; Rout <= 0; Yin <= 0;
271       end
272     ror4 : begin
273       #10 Grc <= 1; Rout <= 1; ROR <= 1; Zlowin <=1;
274       #15 Grc <= 0; Rout <= 0; ROR <= 0; Zlowin <=0;
275       end
276     ror5 : begin
277       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
278       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
279       end
280     rol3 : begin
281       #10 Grb <= 1; Rout <= 1; Yin <= 1;
282       #15 Grb <= 0; Rout <= 0; Yin <= 0;
283       end
284     rol4 : begin
285       #10 Grc <= 1; Rout <= 1; ROL <= 1; Zlowin <=1;
286       #15 Grc <= 0; Rout <= 0; ROL <= 0; Zlowin <=0;
287       end
288     rol5 : begin
289       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
```

```
290       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
291       end
292     and3 : begin
293       #10 Grb <= 1; Rout <= 1; Yin <= 1;
294       #15 Grb <= 0; Rout <= 0; Yin <= 0;
295       end
296     and4 : begin
297       #10 Grc <= 1; Rout <= 1; AND <= 1; Zlowin <=1;
298       #15 Grc <= 0; Rout <= 0; AND <= 0; Zlowin <=0;
299       end
300     and5 : begin
301       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
302       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
303       end
304     or3 : begin
305       #10 Grb <= 1; Rout <= 1; Yin <= 1;
306       #15 Grb <= 0; Rout <= 0; Yin <= 0;
307       end
308     or4 : begin
309       #10 Grc <= 1; Rout <= 1; OR <= 1; Zlowin <=1;
310       #15 Grc <= 0; Rout <= 0; OR <= 0; Zlowin <=0;
311       end
312     or5 : begin
313       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
314       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
315       end
316     addi3 : begin
317       #10 Grb <= 1; Rout <= 1; Yin <= 1;
318       #15 Grb <= 0; Rout <= 0; Yin <= 0;
319       end
320     addi4 : begin
321       #10 Cout <= 1; Zlowin <= 1; ADD <= 1;
322       #15 Cout <= 0; Zlowin <= 0; ADD <= 0;
323       end
324     addi5 : begin
325       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
326       #20 Zlowout <= 0; Gra <= 0; Rin <= 0;
327       end
328     andi3 : begin
329       #10 Grb <= 1; Rout <= 1; Yin <= 1;
330       #15 Grb <= 0; Rout <= 0; Yin <= 0;
331       end
332     andi4 : begin
333       #10 Cout <= 1; Zlowin <= 1; AND <= 1;
334       #15 Cout <= 0; Zlowin <= 0; AND <= 0;
335       end
336     andi5 : begin
337       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
338       #20 Zlowout <= 0; Gra <= 0; Rin <= 0;
339       end
340     ori3 : begin
341       #10 Grb <= 1; Rout <= 1; Yin <= 1;
342       #15 Grb <= 0; Rout <= 0; Yin <= 0;
343       end
344     ori4 : begin
345       #10 Cout <= 1; Zlowin <= 1; OR <= 1;
346       #15 Cout <= 0; Zlowin <= 0; OR <= 0;
347       end
348     ori5 : begin
```

```
349       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
350       #20 Zlowout <= 0; Gra <= 0; Rin <= 0;
351       end
352     mul3 : begin
353       #10 Grb <= 1; Rout <= 1; Yin <= 1;
354       #15 Grb <= 0; Rout <= 0; Yin <= 0;
355       end
356     mul4 : begin
357       #10 Gra <= 1; Rout <= 1; MUL <= 1; Zlowin <=1; Zhighin <= 1;
358       #35 Gra <= 0; Rout <= 0; MUL <= 0; Zlowin <=0; Zhighin <= 0;
359       end
360     mul5 : begin
361       #10 Zlowout <= 1; lowin <= 1;
362       #15 Zlowout <= 0; lowin <= 0;
363       end
364     mul6 : begin
365       #10 Zhighout <= 1; highin <= 1;
366       #15 Zhighout <= 0; highin <= 0;
367       end
368     div3 : begin
369       #10 Gra <= 1; Rout <= 1; Yin <= 1;
370       #15 Gra <= 0; Rout <= 0; Yin <= 0;
371       end
372     div4 : begin
373       #10 Grb <= 1; Rout <= 1; DIV <= 1; Zlowin <=1; Zhighin <= 1;
374       #35 Grb <= 0; Rout <= 0; DIV <= 0; Zlowin <=0; Zhighin <= 0;
375       end
376     div5 : begin
377       #10 Zlowout <= 1; lowin <= 1;
378       #15 Zlowout <= 0; lowin <= 0;
379       end
380     div6 : begin
381       #10 Zhighout <= 1; highin <= 1;
382       #15 Zhighout <= 0; highin <= 0;
383       end
384     neg3 : begin
385       #10 Grb <= 1; Rout <= 1; Zlowin <= 1; NEG <=1;
386       #15 Grb <= 0; Rout <= 0; Zlowin <= 0; NEG <=0;
387       end
388     neg4 : begin
389       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
390       #20 Zlowout <= 0; Gra <= 0; Rin <= 0;
391       end
392     not3 : begin
393       #10 Grb <= 1; Rout <= 1; Zlowin <= 1; NOT <=1;
394       #15 Grb <= 0; Rout <= 0; Zlowin <= 0; NOT <=0;
395       end
396     not4 : begin
397       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
398       #20 Zlowout <= 0; Gra <= 0; Rin <= 0;
399       end
400     ld3 : begin
401       #10 Grb <= 1; BAout <= 1; Yin <= 1;
402       #15 Grb <= 0; BAout <= 0; Yin <= 0;
403       end
404     ld4 : begin
405       #10 Cout <= 1; Zlowin <= 1; ADD <= 1;
406       #15 Cout <= 0; Zlowin <= 0; ADD <= 0;
407       end
```

```
408     ld5 : begin
409       #10 Zlowout <= 1; MARin <= 1;
410       #15 Zlowout <= 0; MARin <= 0;
411       end
412     ld6 : begin
413       #10 Read <= 1; MDRin <= 1;
414       #15 Read <= 0; MDRin <= 0;
415       end
416     ld7 : begin
417       #10 MDRout <= 1; Gra <= 1; Rin <= 1;
418       #15 MDRout <= 0; Gra <= 0; Rin <= 0;
419       end
420     ldi3 : begin
421       #10 Grb <= 1; BAout <= 1; Yin <= 1;
422       #15 Grb <= 0; BAout <= 0; Yin <= 0;
423       end
424     ldi4 : begin
425       #10 Cout <= 1; Zlowin <= 1; ADD <= 1;
426       #15 Cout <= 0; Zlowin <= 0; ADD <= 0;
427       end
428     ldi5 : begin
429       #10 Zlowout <= 1; Gra <= 1; Rin <= 1;
430       #15 Zlowout <= 0; Gra <= 0; Rin <= 0;
431       end
432     st3 : begin
433       #10 Grb <= 1; Rout <= 1; Yin <= 1; BAout <=1;
434       #15 Grb <= 0; Rout <= 0; Yin <= 0; BAout <=0;
435       end
436     st4 : begin
437       #10 Cout <= 1; Zlowin <= 1; ADD <= 1;
438       #15 Cout <= 0; Zlowin <= 0; ADD <= 0;
439       end
440     st5 : begin
441       #10 Zlowout <= 1; MARin <= 1;
442       #15 Zlowout <= 0; MARin <= 0;
443       end
444     st6 : begin
445       #10 Gra <= 1; Rout <= 1; MDRin <= 1;
446       #15 Gra <= 0; Rout <= 0; MDRin <= 0;
447       end
448     st7 : begin
449       #10 MDRout <= 1; Write <= 1;
450       #15 MDRout <= 0; Write <= 0;
451       end
452     branch3 : begin
453       #10 Gra <= 1; Rout <= 1; con_in <= 1;
454       #15 Gra <= 0; Rout <= 0; con_in <= 0;
455       end
456     branch4 : begin
457       #10 PCout <= 1; Yin <= 1;
458       #10 PCout <= 0; Yin <= 0;
459       end
460     branch5 : begin
461       #10 Cout <= 1; Zlowin <= 1; ADD <= 1;
462       #15 Cout <= 0; Zlowin <= 0; ADD <= 0;
463       end
464     branch6 : begin
465       if ( Con_FF ) begin
466         #10 Zlowout <= 1; PC_enable <= 1;
```

```
467         #15 Zlowout <= 0; PC_enable <= 0;
468         end
469       end
470     jr3 : begin
471       #10 Gra <= 1; Rout <= 1; PCin <= 1;
472       #15 Gra <= 0; Rout <= 0; PCin <= 0;
473       end
474     jal3 : begin
475       #10 R15_enable <= 1; PCout <= 1;
476       #15 R15_enable <= 0; PCout <= 0;
477       end
478     jal4 : begin
479       #10 Gra <= 1; Rout <= 1; PCin <= 1;
480       #15 Gra <= 0; Rout <= 0; PCin <= 0;
481       end
482     in3 : begin
483       #10 Gra <= 1; Rin <= 1; inPortOut <= 1;
484       #15 Gra <= 0; Rin <= 0; inPortOut <= 0;
485       end
486     out3 : begin
487       #10 Gra <= 1; Rout <= 1; outPortIn <= 1;
488       #15 Gra <= 0; Rout <= 0; outPortIn <= 0;
489       end
490     mfhi3 : begin
491       #10 Gra <= 1; Rin <= 1; highout <= 1;
492       #15 Gra <= 0; Rin <= 0; highout <= 0;
493       end
494     mflo3 : begin
495       #10 Gra <= 1; Rin <= 1; lowout <= 1;
496       #15 Gra <= 0; Rin <= 0; lowout <= 0;
497       end
498     nop3 : begin end // nothing
499     halt3 : Run <= 0; // no Run
500         default : begin end
501       endcase
502     end
503   endmodule
```

A.2 Datapath

```
1
2  module datapath (input Clock, Reset, Stop );
3
4   wire Rin , Rout , BAout , Cout , Gra , Grb , Grc;
5   wire PCOut , MDRout , Zhighout , Zlowout , Zhighin , Zlowin , highin ,
       lowin ,
6  highout , lowout ;
7   wire MARin , PCin , MDRin , IRin , Yin , con_in , IncPC ;
8   wire outPortIn , inPortOut , inPortIn ;
9   wire Read , Write , Clear , Run , PC_enable , R15_enable ;
10  wire ADD , SUB , SHR , SHL , ROR , ROL , AND , OR , MUL , DIV , NEG , NOT;
11  wire [11:0] CONTROL ;
12  assign CONTROL [0] = AND;
13  assign CONTROL [1] = OR;
14  assign CONTROL [2] = ADD;
15  assign CONTROL [3] = SUB;
16  assign CONTROL [4] = NEG;
17  assign CONTROL [5] = NOT;
18  assign CONTROL [6] = SHL;
```

```
19  assign CONTROL [7] = SHR;
20  assign CONTROL [8] = ROL;
21  assign CONTROL [9] = ROR;
22  assign CONTROL [10] = MUL;
23  assign CONTROL [11] = DIV;
24
25  wire [31:0] Bus_Mux_Out ; // output of bus
26  wire [31:0] BusMuxIn_R0 , BusMuxIn_R1 , BusMuxIn_R2 , BusMuxIn_R3 ,
27          BusMuxIn_R4 , BusMuxIn_R5 , BusMuxIn_R6 , BusMuxIn_R7 , BusMuxIn_R8
                ,
28          BusMuxIn_R9 , BusMuxIn_R10 , BusMuxIn_R11 , BusMuxIn_R12 ,
                BusMuxIn_R13 ,
29          BusMuxIn_R14 , BusMuxIn_R15 , BusMuxIn_Z_HI , BusMuxIn_Z_LO ,
                BusMuxIn_HI ,
30          BusMuxIn_LO , BusMuxIn_PC , BusMuxIn_IR , BusMuxIn_IN_PORT ,
31          BusMuxIn_OUT_PORT , BusMuxIn_MDR ; // register " storage "
32
33  wire [31:0] ZOut_HI , ZOut_LO ; // ALU output
34
35  wire [31:0] Y_contents ;
36
37  wire [31:0] CSignExtended ;
38
39
40  wire [8:0] mar_out ;
41
42  wire R15in ;
43  assign R15in = R15_enable | RXin [15];
44
45  wire PCin_or_enable ;
46  assign PCin_or_enable = PCin | PC_enable ;
47
48  wire [31:0] R0_out ;
49  assign R0_out = BAout ? 0 : BusMuxIn_R0 ;
50
51  // 15 Registers
52  register_zero #(0) R0 (Clock , Clear , BAout , Bus_Mux_Out , RXin [0] ,
        BusMuxIn_R0 );
53  Register #(0) R1 (Clock , Clear , Bus_Mux_Out , RXin [1] , BusMuxIn_R1 );
        //preloads 133 into R1 (hex 85)
54  Register #(0) R2 (Clock , Clear , Bus_Mux_Out , RXin [2] , BusMuxIn_R2 );
        //preloads 1 into R2
55  Register #(0) R3 (Clock , Clear , Bus_Mux_Out , RXin [3] , BusMuxIn_R3 );
56  Register #(0) R4 (Clock , Clear , Bus_Mux_Out , RXin [4] , BusMuxIn_R4 );
57  Register #(0) R5 (Clock , Clear , Bus_Mux_Out , RXin [5] , BusMuxIn_R5 );
58  Register #(0) R6 (Clock , Clear , Bus_Mux_Out , RXin [6] , BusMuxIn_R6 );
59  Register #(0) R7 (Clock , Clear , Bus_Mux_Out , RXin [7] , BusMuxIn_R7 );
60  Register #(0) R8 (Clock , Clear , Bus_Mux_Out , RXin [8] , BusMuxIn_R8 );
61  Register #(0) R9 (Clock , Clear , Bus_Mux_Out , RXin [9] , BusMuxIn_R9 );
62  Register #(0) R10 (Clock , Clear , Bus_Mux_Out , RXin [10] , BusMuxIn_R10 )
        ;
63  Register #(0) R11 (Clock , Clear , Bus_Mux_Out , RXin [11] , BusMuxIn_R11 )
        ;
64  Register #(0) R12 (Clock , Clear , Bus_Mux_Out , RXin [12] , BusMuxIn_R12 )
        ;
65  Register #(0) R13 (Clock , Clear , Bus_Mux_Out , RXin [13] , BusMuxIn_R13 )
        ;
66  Register #(0) R14 (Clock , Clear , Bus_Mux_Out , RXin [14] , BusMuxIn_R14 )
        ;
```

```
67   Register #(0) R15 (Clock , Clear , Bus_Mux_Out , R15in , BusMuxIn_R15 );
68
69   // High and Low Register Used in Multiplication and Division
70   Register HI (Clock , Clear , Bus_Mux_Out , highin , BusMuxIn_HI );
71   Register LO (Clock , Clear , Bus_Mux_Out , lowin , BusMuxIn_LO );
72
73   Register Z_HI (Clock , Clear , ZOut_HI , Zhighin , BusMuxIn_Z_HI );
74   Register Z_LO (Clock , Clear , ZOut_LO , Zlowin , BusMuxIn_Z_LO );
75
76   /* PC IR Y */
77   Register #(0) PC (Clock , Clear , Bus_Mux_Out , PCin_or_enable ,
         BusMuxIn_PC ); //preload address 1 in PC
78   Register #(0) IR (Clock , Clear , Bus_Mux_Out , IRin , BusMuxIn_IR );
79   Register #(0) Y (Clock , Clear , Bus_Mux_Out , Yin , Y_contents );
80
81   /* I/O */
82   inputPort #(0) IN_PORT (Clock , Clear , inPortIn , from_input_unit,
         BusMuxIn_IN_PORT );
83   outputPort OUT_PORT (Clock , Clear , outPortIn , Bus_Mux_Out ,
         to_output_unit )
84
85
86    Bus bus (.R0_out(RXout[0]), .R1_out(RXout [1]), .R2_out(RXout [2]) , .
         R3_out(RXout [3]), .R4_out(RXout[4]), .R5_out(RXout [5]), .R6_out(
         RXout[6]),
87    .R7_out(RXout[7]), .R8_out(RXout [8]), .R9_out(RXout[9]) , .R10_out(RXout
         [10]), .R11_out(RXout[11]), .R12_out(RXout [12]), .R13_out(RXout[13]),
88    .R14_out ( RXout [14]) , .R15_out(RXout[15]), .HI_out(highout), .LO_out(
         lowout), .Z_high_out(Zhighout), . Z_low_out(Zlowout), .PC_out(PCOut),
89    .MDR_out(MDRout), .In_Portout ( inPortOut ), .C_out(Cout), .BusMuxIn_R0(
         R0_out), .BusMuxIn_R1(BusMuxIn_R1), .BusMuxIn_R2(BusMuxIn_R2), .
         BusMuxIn_R3(BusMuxIn_R3),
90    .BusMuxIn_R4(BusMuxIn_R4), .BusMuxIn_R5(BusMuxIn_R5), .BusMuxIn_R6(
         BusMuxIn_R6), .BusMuxIn_R7(BusMuxIn_R7), .BusMuxIn_R8(BusMuxIn_R8), .
         BusMuxIn_R9(BusMuxIn_R9),
91    .BusMuxIn_R10(BusMuxIn_R10), . BusMuxIn_R11(BusMuxIn_R11), .BusMuxIn_R12(
         BusMuxIn_R12), .BusMuxIn_R13(BusMuxIn_R13), .BusMuxIn_R14(BusMuxIn_R14
         ), .BusMuxIn_R15(BusMuxIn_R15),
92    .BusMuxIn_HI(BusMuxIn_HI), .BusMuxIn_LO(BusMuxIn_LO), .BusMuxIn_Z_HI(
         BusMuxIn_Z_HI), .BusMuxIn_Z_LO(BusMuxIn_Z_LO), .BusMuxIn_PC(
         BusMuxIn_PC), .BusMuxIn_MDR(BusMuxIn_MDR),
93    .BusMuxIn_IN_PORT(BusMuxIn_IN_PORT), .C_Sign_Extended(CSignExtended ), .
         BusMuxOut(Bus_Mux_Out));
94
95   ALU alu (.A( Y_contents ), .B( Bus_Mux_Out ), . C_LO ( ZOut_LO ), . C_HI (
       ZOut_HI ), .cntrl(CONTROL), .IncPC(IncPC));
96
97   MDR mdr (. Read ( Read ), .clk ( Clock ), .clr ( Clear ), . MDRin ( MDRin ),
        . BusMuxOut(Bus_Mux_Out ), . Mdatain ( mdr_data_in ), . MDRout (
       BusMuxIn_MDR ));
98
99
100  select_and_encode IR_select (Gra , Grb , Grc , Rin , Rout , BAout ,
       BusMuxIn_IR , RXin , RXout , CSignExtended );
101
102  con_ff con_logic (con_in , BusMuxIn_IR [22:19] , Bus_Mux_Out , toControlUnit
       );
103
104  MAR mar ( Bus_Mux_Out , MARin , Clock , Clear , mar_out );
```

```
105
106 ram ram_inst (. address ( mar_out ), . clock ( Clock ), . data (
        BusMuxIn_MDR ), .wren ( Write ), .q( mdr_data_in ));
107
108 ControlUnit CPU (. Read ( Read ), . Write ( Write ), .Run(Run), . Clear (
        Clear ), .PC_enable ( PC_enable ), . R15_enable ( R15_enable ), .Gra(Gra
        ), .Grb(Grb), .Grc(Grc), .Rin(Rin), .Rout(Rout), .PCout(PCOut), .MDRout(
        MDRout), .Zhighout(Zhighout), .Zlowout(Zlowout), .highout(highout), .
        lowout(lowout), .Zhighin(Zhighin), . Zlowin ( Zlowin ), . highin (
        highin ), . lowin (
109 lowin ), . PCin ( PCin ), . IRin ( IRin ), .Yin(Yin), . MDRin ( MDRin ), .
        MARin (
110 MARin ), . outPortIn ( outPortIn ), . inPortOut ( inPortOut ), . Cout ( Cout
        ), .
111 BAout ( BAout ), . con_in ( con_in ), . IncPC ( IncPC ),
112 .ADD(ADD), .SUB(SUB), .SHR(SHR), .SHL(SHL), .ROR(ROR), .ROL(ROL), .AND(AND
113 ), .OR(OR), .MUL(MUL), .DIV(DIV), .NEG(NEG), .NOT(NOT),
114 .IR( BusMuxIn_IR ), . Clock ( Clock ), . Reset ( Reset ), . Stop ( Stop ), .
        Con_FF (
115 toControlUnit ));
```

A.3 Bus

```
1  module Bus (R0_out, R1_out, R2_out, R3_out, R4_out, R5_out, R6_out, R7_out,
       R8_out, R9_out,  R10_out, R11_out, R12_out, R13_out, R14_out, R15_out,
       HI_out, LO_out, Z_high_out, Z_low_out, PC_out, MDR_out, In_Portout,
       C_out, BusMuxIn_R0, BusMuxIn_R1, BusMuxIn_R2, BusMuxIn_R3, BusMuxIn_R4,
       BusMuxIn_R5, BusMuxIn_R6, BusMuxIn_R7, BusMuxIn_R8, BusMuxIn_R9,
       BusMuxIn_R10, BusMuxIn_R11, BusMuxIn_R12, BusMuxIn_R13, BusMuxIn_R14,
       BusMuxIn_R15, BusMuxIn_HI, BusMuxIn_LO, BusMuxIn_Z_HI, BusMuxIn_Z_LO,
       BusMuxIn_PC, BusMuxIn_MDR,  BusMuxIn_IN_PORT, C_Sign_Extended, BusMuxOut
       );
2
3      input wire R0_out, R1_out, R2_out, R3_out, R4_out, R5_out, R6_out,
           R7_out, R8_out, R9_out,  R10_out, R11_out, R12_out, R13_out, R14_out
           , R15_out, HI_out, LO_out, Z_high_out, Z_low_out, PC_out, MDR_out,
           In_Portout, C_out;
4
5      input [31:0] BusMuxIn_R0, BusMuxIn_R1, BusMuxIn_R2, BusMuxIn_R3,
           BusMuxIn_R4, BusMuxIn_R5, BusMuxIn_R6, BusMuxIn_R7, BusMuxIn_R8,
           BusMuxIn_R9, BusMuxIn_R10, BusMuxIn_R11, BusMuxIn_R12, BusMuxIn_R13,
            BusMuxIn_R14, BusMuxIn_R15, BusMuxIn_HI, BusMuxIn_LO, BusMuxIn_Z_HI
           , BusMuxIn_Z_LO, BusMuxIn_PC, BusMuxIn_MDR, BusMuxIn_IN_PORT,
           C_Sign_Extended;
6
7      output reg [31:0] BusMuxOut;
8
9      reg [4:0] encoderOut;
10
11     always @ (*)
12     begin
13         if (C_out) BusMuxOut = C_Sign_Extended; else
14         if (In_Portout) BusMuxOut = BusMuxIn_IN_PORT; else
15         if (MDR_out)    BusMuxOut = BusMuxIn_MDR; else
16         if (PC_out)     BusMuxOut = BusMuxIn_PC; else
17         if (Z_low_out)  BusMuxOut = BusMuxIn_Z_LO; else
18         if (Z_high_out) BusMuxOut = BusMuxIn_Z_HI; else
19         if (LO_out)     BusMuxOut = BusMuxIn_LO; else
20         if (HI_out)     BusMuxOut = BusMuxIn_HI; else
```

```
21          if (R15_out)     BusMuxOut = BusMuxIn_R15; else
22          if (R14_out)     BusMuxOut = BusMuxIn_R14; else
23          if (R13_out)     BusMuxOut = BusMuxIn_R13; else
24          if (R12_out)     BusMuxOut = BusMuxIn_R12; else
25          if (R11_out)     BusMuxOut = BusMuxIn_R11; else
26          if (R10_out)     BusMuxOut = BusMuxIn_R10; else
27          if (R9_out)      BusMuxOut = BusMuxIn_R9; else
28          if (R8_out)      BusMuxOut = BusMuxIn_R8; else
29          if (R7_out)      BusMuxOut = BusMuxIn_R7; else
30          if (R6_out)      BusMuxOut = BusMuxIn_R6; else
31          if (R5_out)      BusMuxOut = BusMuxIn_R5; else
32          if (R4_out)      BusMuxOut = BusMuxIn_R4; else
33          if (R3_out)      BusMuxOut = BusMuxIn_R3; else
34          if (R2_out)      BusMuxOut = BusMuxIn_R2; else
35          if (R1_out)      BusMuxOut = BusMuxIn_R1; else
36          if (R0_out)      BusMuxOut = BusMuxIn_R0; else
37                           BusMuxOut = C_Sign_Extended;
38      end
39
40  endmodule
```

A.4 Register Zero

```
1  module register_zero #(parameter VAL = 0)(input clk, clr, BAout, input
       [31:0] BusMuxOut, input R0in, output reg [31:0] BusMuxIn_R0);
2      always@(posedge clk or negedge clr)
3      begin
4          if(clr == 0) BusMuxIn_R0 = 0;
5          else if (BAout == 0) BusMuxIn_R0 = 0;
6          else if(R0in) BusMuxIn_R0 <= BusMuxOut;
7      end
8      initial BusMuxIn_R0 = VAL; // assigns initial value
9  endmodule
```

A.5 Arithmetic Logic Unit

```
1  module ALU (input[31:0] A, B, output reg[31:0] C_LO, C_HI, input wire [3:0]
       cntrl, input IncPC);
2
3    wire[63:0] div_quotient;
4
5    wire [63:0] booth_result;
6
7    div restoring_div(.quotient_and_remainder(div_quotient), .dividend(A), .
         divisor(B));
8
9    mul booth (booth_result, A, B);
10
11   always @(*) begin
12     C_LO = 0;
13     C_HI = 0;
14     if (IncPC) C_LO = B + 1; // increases program counter
15     else begin
16       case(cntrl)
17         11  : begin // division
18               C_LO = div_quotient[31:0];
19               C_HI = div_quotient[63:32];
20             end
21         10  :   begin // multiplication
```

```
22                C_LO = booth_result[31:0];
23                C_HI = booth_result [63:32];
24              end
25          9   :   C_LO = A >> B | A << (32 - B); // rotate right
26          8   : C_LO = A << B | A >> (32 - B); // rotate left
27          7   :   C_LO = A >>> B; // right arithmetic shift - A = how many
                shifts, B = the number you want to shift
28          6   :   C_LO = A <<< B; // left arihtmatic shift - A = how many
                shifts, B = the number you want to shift
29          5   :   C_LO = ~B; // logical not
30          4   :   C_LO = -B; //negation function
31          3   :   C_LO = A - B;
32          2   :   C_LO = A + B;
33          1   :   C_LO = A | B;
34          0   :   C_LO = A & B;
35          default : begin end
36        endcase
37      end
38    end
39  endmodule
```

B Select and Encode

```
1   module select_and_encode (Gra, Grb, Grc, Rin, Rout, BAout, IR, RXin, RXout,
        CSignExtended);
2
3       input Gra, Grb, Grc, Rin, Rout, BAout; // from CPU
4       input [31:0] IR;
5       output reg [15:0] RXin, RXout; // Registers 0 - 15 in/out
6       output [31:0] CSignExtended;
7
8       reg [3:0] selected_register;
9
10      always @(Gra, Grb, Grc, IR, BAout, Rin, Rout) // are these the right
            thingsin the sensitivity list
11      begin
12          RXout = 0;
13          RXin = 0;
14          selected_register = 0;
15
16          // Select
17          if (Gra == 1) selected_register = IR[26:23]; else
18          if (Grb == 1) selected_register = IR[22:19]; else
19          if (Grc == 1) selected_register = IR[18:15];
20
21          if (BAout == 1|| Rout == 1) begin
22              RXout[selected_register] = 1;
23          end else
24          if (Rin) begin
25              RXin[selected_register] = 1;
26          end
27
28      end
29
30      assign CSignExtended = $signed(IR[18:0]);
31  endmodule
```

C Memory Subsystem

C.1 RAM

```
// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: ram.v
// Megafunction Name(s):
//      altsyncram
//
// Simulation Library Files(s):
//      altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2013 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors.  Please refer to the
//applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module ram (
  address,
  clock,
  data,
  wren,
  q);

  input [8:0]  address;
  input   clock;
  input [31:0]  data;
  input   wren;
  output  [31:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
  tri1    clock;
`ifndef ALTERA_RESERVED_QIS
```

```
56  // synopsys translate_on
57  `endif
58
59    wire [31:0] sub_wire0;
60    wire [31:0] q = sub_wire0[31:0];
61
62    altsyncram  altsyncram_component (
63          .address_a (address),
64          .clock0 (clock),
65          .data_a (data),
66          .wren_a (wren),
67          .q_a (sub_wire0),
68          .aclr0 (1'b0),
69          .aclr1 (1'b0),
70          .address_b (1'b1),
71          .addressstall_a (1'b0),
72          .addressstall_b (1'b0),
73          .byteena_a (1'b1),
74          .byteena_b (1'b1),
75          .clock1 (1'b1),
76          .clocken0 (1'b1),
77          .clocken1 (1'b1),
78          .clocken2 (1'b1),
79          .clocken3 (1'b1),
80          .data_b (1'b1),
81          .eccstatus (),
82          .q_b (),
83          .rden_a (1'b1),
84          .rden_b (1'b1),
85          .wren_b (1'b0));
86    defparam
87      altsyncram_component.clock_enable_input_a = "BYPASS",
88      altsyncram_component.clock_enable_output_a = "BYPASS",
89  `ifdef NO_PLI
90      altsyncram_component.init_file = "ram.rif"
91  `else
92      altsyncram_component.init_file = "ram.hex"
93  `endif
94  ,
95      altsyncram_component.intended_device_family = "Cyclone␣III",
96      altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
97      altsyncram_component.lpm_type = "altsyncram",
98      altsyncram_component.numwords_a = 512,
99      altsyncram_component.operation_mode = "SINGLE_PORT",
100     altsyncram_component.outdata_aclr_a = "NONE",
101     altsyncram_component.outdata_reg_a = "UNREGISTERED",
102     altsyncram_component.power_up_uninitialized = "FALSE",
103     altsyncram_component.read_during_write_mode_port_a = "DONT_CARE",
104     altsyncram_component.widthad_a = 9,
105     altsyncram_component.width_a = 32,
106     altsyncram_component.width_byteena_a = 1;
107
108
109 endmodule
110
111 // ============================================================
112 // CNX file retrieval info
113 // ============================================================
114 // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
```

```
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone III"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "ram.hex"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "512"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "2"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "9"
// Retrieval info: PRIVATE: WidthData NUMERIC "32"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "ram.hex"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone III"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "512"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING "DONT_CARE
    "
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "9"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "32"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 9 0 INPUT NODEFVAL "address[8..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 32 0 INPUT NODEFVAL "data[31..0]"
// Retrieval info: USED_PORT: q 0 0 32 0 OUTPUT NODEFVAL "q[31..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 9 0 address 0 0 9 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 32 0 data 0 0 32 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 32 0 @q_a 0 0 32 0
```

```
// Retrieval info: GEN_FILE: TYPE_NORMAL ram.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_inst.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL ram_bb.v FALSE
// Retrieval info: LIB_FILE: altera_mf
```

C.2 Memory Address Register

```
module MAR(input [31:0] BusMuxOut, input MARin, clk, clr, output reg [8:0]
    Address);
    always@(posedge clk or negedge clr)
    begin
        if(clr == 0) Address <= 0;
        else if(MARin) Address <= BusMuxOut[8:0];
    end
endmodule
```

C.3 Memory Data Register

```
module MDR (input Read, clr, clk, MDRin, input [31:0] BusMuxOut, Mdatain,
    output reg [31:0] MDRout);

    always@(posedge clk or negedge clr)
    begin
        if(clr == 0) MDRout <= 0; // 32'h0000_0000 zero also works
        else if(MDRin) MDRout <= Read ? Mdatain : BusMuxOut;
    end

endmodule
```

D CON FF

```
module con_ff (con_in, IR, BusMuxOut, toControlUnit);

    input con_in; // enable
    input [3:0] IR;
    input [31:0] BusMuxOut;
    output reg toControlUnit; // this is PC + 1 + C (signExtended)

    function triple_nor(input [31:0] BusMuxOut);
        // as long as 1 bit is 1, return true. Since this is then put
            through a NOT gate, it means
        // as long as 1 bit is 1, return false
        // that is, if zero, return true
        triple_nor = (BusMuxOut == 0) ? 1 : 0;
    endfunction

    always @(posedge con_in) begin

    // Note: Nor with same input is a not. Negated again, it is the original
        .
        if (con_in) begin
            case (IR[3:2])
                0: toControlUnit = triple_nor(BusMuxOut); // branch if zero
                1: toControlUnit = !triple_nor(BusMuxOut); // branch if non
                    zero
```

```
22                  2: toControlUnit = !BusMuxOut[31]; // branch if greater than
                        zero
23                  3: toControlUnit = BusMuxOut[31]; // branch if less than
                        zero
24                  default: begin end // otherwise do nothing
25              endcase
26          end else toControlUnit <= 0; // set 0
27      end
28
29  endmodule
```

E Input/Output Ports

E.1 Input Port

```
1  module inputPort #(parameter VAL = 0)(input clk, clr, strobe, input [31:0]
       inputUnit, output reg [31:0] busMuxIn_In_PortIn);
2      always@(posedge clk or negedge clr)
3      begin
4          if(clr == 0) busMuxIn_In_PortIn <= 0;
5          else if(strobe) busMuxIn_In_PortIn <= inputUnit;
6      end
7      initial busMuxIn_In_PortIn = VAL; // assigns initial value
8  endmodule
```

E.2 Output Port

```
1  module outputPort #(parameter VAL = 0) (input clk, clr, outPortIn, input
       [31:0] BusMuxOut, output reg [31:0] outputUnit);
2      always@(posedge clk or negedge clr)
3      begin
4          if(clr == 0) outputUnit <= 0;
5          else if(outPortIn) outputUnit <= BusMuxOut;
6      end
7      initial outputUnit = VAL; // assigns initial value
8  endmodule
```

F Control Sequences

F.1 st $90, R1

```
1  always @(Present_state)// do the required job ineach state
2  begin
3      case (Present_state) //assert the required signals in each clock cycle
4          Default: begin //initialize the signals
5              PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                   initialize the signals
6              MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7              PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8              IncPC <= 0; Read <= 0; CONTROL <= 0;
9              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10             Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                   <= 0;
11             Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12             BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                   ram_enable <= 0;
13             R15_enable <= 0; PC_enable <= 0;
14         end
15         T0: begin
```

```
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin <= 1; // Mdatain
                    <= 32'h10080090; // opcode for st 90 r1
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1;
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Cout<= 1; MARin <= 1;
29              #10 Cout<= 0; MARin <= 0;
30          end
31          T4: begin
32              #10 Grb <= 1; BAout<= 1; Rout <= 1; MDRin <= 1;
33              #15 Grb <= 0; BAout<= 0; Rout <= 0; MDRin <= 0;
34          end
35          T5: begin
36              #10 MDRout <= 1; ram_enable <= 1;  // output of RDR written to
                    RAM
37              #15 MDRout <= 0; ram_enable <= 0;
38          end
39      endcase
40  end
```

F.2 st $90(R1), R1

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
13              R15_enable <= 0; PC_enable <= 0;
14          end
15          T0: begin
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin <= 1; // Mdatain
                    <= 32'h590FFFFB; opcode for ADDI R2, R1, -5 01011 0010 0001
                    111 1111 1111 1111 1011 in init file
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1;
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
```

```
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Grb <= 1; Rout<= 1; Yin <= 1;
29              #15 Grb <= 0; Rout<= 0; Yin <= 0;
30          end
31          T4: begin
32              #10 Zlowin <= 1; CONTROL <= 2; // add, result is stored in ZLO
33              #15 Zlowin <= 0;
34          end
35          T5: begin
36              #10 Zlowout <= 1; MARin <= 1;  // store in MAR
37              #15 Zlowout <= 0; MARin <= 0;
38          end
39          T6: begin
40              #10 Cout<= 1; Read <= 0; MDRin <= 1; // read from CSignExtended,
                     store in MDR
41              #20 Cout<= 0; MDRin <= 0;
42          end
43          T7: begin // output of MDR written to RAM
44              #10 MDRout <= 1; ram_enable <= 1;
45              #15 MDRout <= 0; ram_enable <= 0;
46          end
47      endcase
48  end
49
50
51
52  always @(Present_state)// do the required job ineach state
53  begin
54      case (Present_state) //assert the required signals in each clock cycle
55          Default: begin //initialize the signals
56              PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
57              MARin <= 0; Zlowin <= 0; Zhighin <= 0;
58              PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
59              IncPC <= 0; Read <= 0; CONTROL <= 0;
60              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
61              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
62              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
63              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
64          end
65          T0: begin
66              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin <= 1; // Mdatain
                    <= 32'h190FFFFB; opcode for // opcode for st 90 r1
67              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
68          end
69          T1: begin
70              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1;
71              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
72          end
73          T2: begin
74              #10 MDRout<= 1; IRin <= 1;
75              #15 MDRout<= 0; IRin <= 0;
76          end
77          T3: begin // read from R1, store in Z_LO
78              #10 Grb <= 1; Rout <= 1; Zlowin <= 1;
```

```
79              #15 Grb <= 0; Rout <= 0; Zlowin <= 0;
80          end
81          T4: begin // write address from instruction to Y register
82              #10 Cout <= 1; Yin <= 1;
83              #15 Cout <= 0; Yin <= 0;
84          end
85          T5: begin // Add Y and Z_LO, send to MAR
86              #10 Zlowout <= 1; MARin <= 1; CONTROL <= 2;
87              #15 Zlowout <= 0; MARin <= 0;
88          end
89          T6: begin // read from Z_LO, write to MDR
90              #10 Zlowout<= 1; MDRin <= 1;
91              #15 Zlowout<= 0; MDRin <= 0;
92          end
93          T7: begin // output of MDR written to RAM
94              #10 MDRout <= 1; ram_enable <= 1;
95              #15 MDRout <= 0; ram_enable <= 0;
96          end
97      endcase
98  end
```

F.3 ld R1, $85), R1

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
13              R15_enable <= 0; PC_enable <= 0;
14          end
15          T0: begin
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                     32'h00800085; //opcode for ld 90 r1
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Grb <= 1; BAout<= 1; Yin <= 1;
29              #15 Grb <= 0; BAout<= 0; Yin <= 0;
30          end
31          T4: begin
```

```
32              #10 Cout<= 1;  CONTROL<= 2; Zlowin <= 1;
33              #15 Cout<= 0;  Zlowin <= 0;
34          end
35          T5: begin
36              #10 Zlowout<= 1; MARin <= 1;
37              #15 Zlowout<= 0; MARin <= 0;
38          end
39          T6: begin
40              #10 Read <= 1; MDRin <= 1;
41              #15 Read <= 0; MDRin <= 0;
42          end
43          T7: begin
44              #10 MDRout <= 1; Gra <= 1; Rin <= 1;
45              #15 MDRout <= 0; Gra <= 0; Rin <= 0;
46          end
47      endcase
48  end
```

F.4 ld R0, $35(R1)

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
13              R15_enable <= 0; PC_enable <= 0;
14          end
15          T0: begin
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                     32'h00800023; //opcode for ld 90 r1
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Grb <= 1; BAout<= 1; Yin <= 1;
29              #15 Grb <= 0; BAout<= 0; Yin <= 0;
30          end
31          T4: begin
32              #10 Cout<= 1;  CONTROL<= 2; Zlowin <= 1;
33              #15 Cout<= 0; Zlowin <= 0;
34          end
```

```
35          T5: begin
36              #10 Zlowout<= 1; MARin <= 1;
37              #15 Zlowout<= 0; MARin <= 0;
38          end
39          T6: begin
40              #10 Read <= 1; MDRin <= 1;
41              #15 Read <= 0; MDRin <= 0;
42          end
43          T7: begin
44              #10 MDRout <= 1; Gra <= 1; Rin <= 1;
45              #15 MDRout <= 0; Gra <= 0; Rin <= 0;
46          end
47      endcase
48  end
```

F.5 ldi R1, $85

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
13              R15_enable <= 0; PC_enable <= 0;
14          end
15          T0: begin
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                     32'h08800085; //Reading from RAM 0000101000000
                    xxxxxxxxxxxxxxxx
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Grb <= 1; BAout<= 1; Yin <= 1;
29              #15 Grb <= 0; BAout<= 0; Yin <= 0;
30          end
31          T4: begin
32              #10 Cout<= 1; Zlowin <= 1; CONTROL <= 2;
33              #15 Cout<= 0; Zlowin <= 0;
34          end
35          T5: begin
36              #10 Zlowout<= 1; Gra <= 1; Rin <= 1;
```

```
37              #15 Zlowout<= 0; Gra <= 0; Rin <= 0;
38          end
39      endcase
40  end
```

F.6 ldi R0, $35(R1)

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
13              R15_enable <= 0; PC_enable <= 0;
14          end
15          T0: begin
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; //Mdatain <=
                    32'h08800023; //Reading from RAM 0000101000000
                    xxxxxxxxxxxxxxxx
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Grb <= 1; BAout<= 1; Yin <= 1;
29              #15 Grb <= 0; BAout<= 0; Yin <= 0;
30          end
31          T4: begin
32              #10 Cout<= 1; Zlowin <= 1; CONTROL <= 2;
33              #15 Cout<= 0; Zlowin <= 0;
34          end
35          T5: begin
36              #10 Zlowout<= 1; Gra <= 1; Rin <= 1;
37              #15 Zlowout<= 0; Gra <= 0; Rin <= 0;
38          end
39      endcase
40  end
```

F.7 brzr R2, 35

```
1   //uses ConFF to determine what branch instruction to use op code is
        different however
2   // opcode for brnr PC <- PC + 1 + C (10010xxxx-00xxxxxxxxxxxxx)//
```

```
3
4  always @(Present_state)// do the required job ineach state
5  begin
6      case (Present_state) //assert the required signals in each clock cycle
7          Default: begin //initialize the signals
8              PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                   initialize the signals
9              MARin <= 0; Zlowin <= 0; Zhighin <= 0;
10             PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
11             IncPC <= 0; Read <= 0; CONTROL <= 0;
12             Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
13             Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                   <= 0;
14             Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
15             BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                   ram_enable <= 0;
16             R15_enable <= 0; PC_enable <= 0;
17         end
18         T0: begin
19             #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;
20             #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
21         end
22         T1: begin
23             #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                    32'h91000023; // opcode for brzr R2, 35  (10010 0010 0000
                   0000 000 0000 0010 0011)//
24             #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
25         end
26         T2: begin
27             #10 MDRout<= 1; IRin <= 1;
28             #15 MDRout<= 0; IRin <= 0;
29         end
30         T3: begin
31             #10 Gra <= 1; Rout <= 1; con_in <= 1;
32             #15 Gra <= 0; Rout <= 0; con_in <= 0;
33         end
34         T4: begin
35             #10 PCout<= 1; Yin <= 1;  // put PC in Y register
36             #10 PCout<= 0; Yin <= 0;
37         end
38         T5: begin
39             #10 Cout<= 1; CONTROL <= 2; Zlowin <= 1;  // add PC with branch,
                    store in Z_LO
40             #15 Cout<= 0; Zlowin <= 0;
41         end
42         T6: begin // enable the PC
43             #10 Zlowout <= 1; PC_enable <= 1; //Reading from PCin
44             #15 Zlowout <= 0; PC_enable <= 0;
45         end
46     endcase
47 end
```

F.8 brnz R2, 35

```
1  //uses ConFF to determine what branch instruction to use op code is
       different however
2  // opcode for brnr PC <- PC + 1 + C (10010xxxx-01xxxxxxxxxxxxx)//
3
4  always @(Present_state)// do the required job ineach state
```

```
begin
    case (Present_state) //assert the required signals in each clock cycle
        Default: begin //initialize the signals
            PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                initialize the signals
            MARin <= 0; Zlowin <= 0; Zhighin <= 0;
            PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
            IncPC <= 0; Read <= 0; CONTROL <= 0;
            Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
            Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                <= 0;
            Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
            BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                ram_enable <= 0;
            R15_enable <= 0; PC_enable <= 0;
        end
        T0: begin
            #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;
            #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
        end
        T1: begin
            #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; //Mdatain <=
                32'h91200023; // opcode for brnz R2, 35 (10010 0010 0100
                0000 000 0000 0010 0011)//
            #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
        end
        T2: begin
            #10 MDRout<= 1; IRin <= 1;
            #15 MDRout<= 0; IRin <= 0;
        end
        T3: begin
            #10 Gra <= 1; Rout <= 1; con_in <= 1;
            #15 Gra <= 0; Rout <= 0; con_in <= 0;
        end
        T4: begin
            #10 PCout<= 1; Yin <= 1;  // put PC in Y register
            #10 PCout<= 0; Yin <= 0;
        end
        T5: begin
            #10 Cout<= 1; CONTROL <= 2; Zlowin <= 1;  // add PC with branch,
                 store in Z_LO
            #15 Cout<= 0; Zlowin <= 0;
        end
        T6: begin // enable the PC
            #10 Zlowout <= 1; PC_enable <= 1; //Reading from PCin
            #15 Zlowout <= 0; PC_enable <= 0;
        end
    endcase
end
```

F.9 brpl R2, 35

```
//uses ConFF to determine what branch instruction to use op code is
    different however
// opcode for brpl PC <- PC + 1 + C (10010xxxx-10xxxxxxxxxxxxx)//

always @(Present_state)// do the required job ineach state
begin
    case (Present_state) //assert the required signals in each clock cycle
```

```
7           Default: begin //initialize the signals
8               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
9               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
10              PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
11              IncPC <= 0; Read <= 0; CONTROL <= 0;
12              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
13              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
14              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
15              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
16              R15_enable <= 0; PC_enable <= 0;
17          end
18          T0: begin
19              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;
20              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
21          end
22          T1: begin
23              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                     32'h91400023; // opcode for brpl R2, 35 (10010 0010 1000
                    0000 000 0000 0010 0011)//
24              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
25          end
26          T2: begin
27              #10 MDRout<= 1; IRin <= 1;
28              #15 MDRout<= 0; IRin <= 0;
29          end
30          T3: begin
31              #10 Gra <= 1; Rout <= 1; con_in <= 1;
32              #15 Gra <= 0; Rout <= 0; con_in <= 0;
33          end
34          T4: begin
35              #10 PCout<= 1; Yin <= 1;  // put PC in Y register
36              #10 PCout<= 0; Yin <= 0;
37          end
38          T5: begin
39              #10 Cout<= 1; CONTROL <= 2; Zlowin <= 1;  // add PC with branch,
                     store in Z_LO
40              #15 Cout<= 0; Zlowin <= 0;
41          end
42          T6: begin // enable the PC
43              #10 Zlowout <= 1; PC_enable <= 1; //Reading from PCin
44              #15 Zlowout <= 0; PC_enable <= 0;
45          end
46      endcase
47  end
```

F.10 brmi R2, 35

```
1   //uses ConFF to determine what branch instruction to use op code is
        different however
2   // opcode for brmi PC <- PC + 1 + C (10010xxxx-11xxxxxxxxxxxxx)//
3
4   always @(Present_state)// do the required job ineach state
5   begin
6       case (Present_state) //assert the required signals in each clock cycle
7           Default: begin //initialize the signals
8               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
```

```
                initialize the signals
9               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
10              PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
11              IncPC <= 0; Read <= 0; CONTROL <= 0;
12              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
13              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
14              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
15              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
16              R15_enable <= 0; PC_enable <= 0;
17          end
18          T0: begin
19              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;
20              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
21          end
22          T1: begin
23              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                     32'h91600023; // opcode for brmi R2, 35 (10010 0010 1100
                    0000 000 0000 0010 0011)//
24              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
25          end
26          T2: begin
27              #10 MDRout<= 1; IRin <= 1;
28              #15 MDRout<= 0; IRin <= 0;
29          end
30          T3: begin
31              #10 Gra <= 1; Rout <= 1; con_in <= 1;
32              #15 Gra <= 0; Rout <= 0; con_in <= 0;
33          end
34          T4: begin
35              #10 PCout<= 1; Yin <= 1;  // put PC in Y register
36              #10 PCout<= 0; Yin <= 0;
37          end
38          T5: begin
39              #10 Cout<= 1; CONTROL <= 2; Zlowin <= 1;  // add PC with branch,
                     store in Z_LO
40              #15 Cout<= 0; Zlowin <= 0;
41          end
42          T6: begin // enable the PC
43              #10 Zlowout <= 1; PC_enable <= 1; //Reading from PCin
44              #15 Zlowout <= 0; PC_enable <= 0;
45          end
46      endcase
47  end
```

F.11 jal R1

```
1   //uses ConFF to determine what branch instruction to use op code is
        different however
2   // opcode for jal R1 (10100xxxx----------------------)//   if R15 it is
        for precedure return
3
4   always @(Present_state)// do the required job ineach state
5   begin
6       case (Present_state) //assert the required signals in each clock cycle
7           Default: begin //initialize the signals
8               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
```

```
9               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
10              PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
11              IncPC <= 0; Read <= 0; CONTROL <= 0;
12              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
13              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
14              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
15              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
16              R15_enable <= 0; PC_enable <= 0;
17          end
18          T0: begin
19              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin <= 1;  //
                    initialize the signals
20              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin <= 0;
21          end
22          T1: begin
23              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; //Mdatain <=
                    32'hA0800000; // opcode for jal R1
                    (10100000100000000000000000) 000//
24              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
25          end
26          T2: begin
27              #10 MDRout<= 1; IRin <= 1;
28              #15 MDRout<= 0; IRin <= 0;
29          end
30          T3: begin
31              #10 R15_enable <= 1; PCout <= 1;
32              #15 R15_enable <= 0; PCout <= 0;
33          end
34          T4: begin
35              #10 Gra <= 1; Rout <= 1; PCin <= 1;
36              #15 Gra <= 0; Rout <= 0; PCin <= 0;
37          end
38      endcase
39  end
```

F.12 jr R1

```
1   //uses ConFF to determine what branch instruction to use op code is
        different however
2   // opcode for jr R1 (10011xxxx-----------------------)//   if R15 it is for
         precedure return
3
4   always @(Present_state)// do the required job ineach state
5   begin
6       case (Present_state) //assert the required signals in each clock cycle
7           Default: begin //initialize the signals
8               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
9               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
10              PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
11              IncPC <= 0; Read <= 0; CONTROL <= 0;
12              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
13              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
14              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
15              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
```

```
16              R15_enable <= 0; PC_enable <= 0;
17          end
18          T0: begin
19              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
20              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
21          end
22          T1: begin
23              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; //Mdatain <=
                    32'h98800000; // opcode for jr R1
                    (10011000100000000000000000) 000//
24              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
25          end
26          T2: begin
27              #10 MDRout<= 1; IRin <= 1;
28              #15 MDRout<= 0; IRin <= 0;
29          end
30          T3: begin
31              #10 Gra <= 1; Rout <= 1; PCin <= 1;
32              #15 Gra <= 0; Rout <= 0; PCin <= 0;
33          end
34      endcase
35  end
```

F.13 in R1

```
1   // Input Port
2   always @(Present_state)// do the required job ineach state
3   begin
4       case (Present_state) //assert the required signals in each clock cycle
5           Default: begin //initialize the signals
6               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
7               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
8               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
9               IncPC <= 0; Read <= 0; CONTROL <= 0;
10              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
11              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
12              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
13              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
14              R15_enable <= 0; PC_enable <= 0;
15          end
16          T0: begin
17              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
18              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
19          end
20          T1: begin
21              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                     32'h00000151; // opcode for in r1 101010001// put Mdatain
                    onto inputport
22              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
23          end
24          T2: begin
25              #10 MDRout<= 1; IRin <= 1; // read MDR into IR
26              #15 MDRout<= 0; IRin <= 0;
27          end
```

```
28          T3: begin
29              #10 Gra <= 1; Rin <= 1; inPortOut <= 1;
30              #15 Gra <= 0; Rin <= 0; inPortOut <= 0;
31        end
32      endcase
33  end
```

F.14 out R1

```
1   // Output Port
2   always @(Present_state)// do the required job ineach state
3   begin
4       case (Present_state) //assert the required signals in each clock cycle
5           Default: begin //initialize the signals
6               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
7               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
8               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
9               IncPC <= 0; Read <= 0; CONTROL <= 0;
10              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
11              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
12              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
13              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
14              R15_enable <= 0; PC_enable <= 0;
15          end
16          T0: begin
17              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
18              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
19          end
20          T1: begin
21              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                     32'h00000161; // opcode for out r1 101100001// put Mdatain
                    onto outputport
22              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
23          end
24          T2: begin
25              #10 MDRout<= 1; IRin <= 1; // read MDR into IR
26              #15 MDRout<= 0; IRin <= 0;
27          end
28          T3: begin
29              #10 Gra <= 1; Rout <= 1; outPortIn <= 1;
30              #15 Gra <= 0; Rout <= 0; outPortIn <= 0;
31        end
32      endcase
33  end
```

F.15 addi R2, R1, -5

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
```

```
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
13              R15_enable <= 0; PC_enable <= 0;
14          end
15          T0: begin
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin <= 1; // Mdatain
                    <= 32'h590FFFFB; opcode for ADDI R2, R1, -5 01011 0010 0001
                    111 1111 1111 1111 1011 in init file
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1;
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Grb <= 1; Rout<= 1; Yin <= 1;
29              #15 Grb <= 0; Rout<= 0; Yin <= 0;
30          end
31          T4: begin
32              #10 Cout<= 1; Zlowin <= 1; CONTROL <= 2; // add, result is
                    stored in Z_LO
33              #15 Cout<= 0; Zlowin <= 0;
34          end
35          T5: begin
36              #10 Zlowout<= 1; Gra <= 1; Rin <= 1; // read from Z_LO, store in
                     R2
37              #20 Zlowout<= 0; Gra <= 0; Rin <= 0;
38          end
39      endcase
40  end
```

F.16 andi R2, R1 $26

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
13              R15_enable <= 0; PC_enable <= 0;
14          end
```

```
15          T0: begin
16              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin <= 1; // Mdatain
                    <= 32'h61080026; opcode for ANDI R2, R1, $26 : 01100 0010
                    0001 000 0000 0000 0010 0110 in init file
17              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
18          end
19          T1: begin
20              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1;
21              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
22          end
23          T2: begin
24              #10 MDRout<= 1; IRin <= 1;
25              #15 MDRout<= 0; IRin <= 0;
26          end
27          T3: begin
28              #10 Grb <= 1; Rout<= 1; Yin <= 1;
29              #15 Grb <= 0; Rout<= 0; Yin <= 0;
30          end
31          T4: begin
32              #10 Cout<= 1; Zlowin <= 1; CONTROL <= 0; // and, result is
                    stored in Z_LO
33              #15 Cout<= 0; Zlowin <= 0;
34          end
35          T5: begin
36              #10 Zlowout<= 1; Gra <= 1; Rin <= 1; // read from Z_LO, store in
                     R2
37              #20 Zlowout<= 0; Gra <= 0; Rin <= 0;
38          end
39      endcase
40  end
```

F.17 ori R2, R1 $26

```
1   always @(Present_state)// do the required job ineach state
2   begin
3       case (Present_state) //assert the required signals in each clock cycle
4           Default: begin //initialize the signals
5               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
6               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
7               PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
8               IncPC <= 0; Read <= 0; CONTROL <= 0;
9               Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
10              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
11              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
12              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0; R15_enable <= 0;
13          end
14          T0: begin
15              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin <= 1; // Mdatain
                    <= 32'h59080026; opcode for ORI R2, R1, 26 01011 0010 0001
                    000 0000 0000 0010 0110 in init file
16              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
17          end
18          T1: begin
19              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1;
20              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
21          end
```

```
22          T2: begin
23              #10 MDRout<= 1; IRin <= 1;
24              #15 MDRout<= 0; IRin <= 0;
25          end
26          T3: begin
27              #10 Grb <= 1; Rout<= 1; Yin <= 1;
28              #15 Grb <= 0; Rout<= 0; Yin <= 0;
29          end
30          T4: begin
31              #10 Cout<= 1; Zlowin <= 1; CONTROL <= 1; // or, result is stored
                     in Z_LO
32              #15 Cout<= 0; Zlowin <= 0;
33          end
34          T5: begin
35              #10 Zlowout<= 1; Gra <= 1; Rin <= 1; // read from Z_LO, store in
                     R2
36              #20 Zlowout<= 0; Gra <= 0; Rin <= 0;
37          end
38      endcase
39  end
```

F.18 mfhi R2

```
1   //uses ConFF to determine what branch instruction to use op code is
        different however
2   // opcode for mfhi R2 (10100xxxx------------------------)//   if R15 it is
        for procedure return
3
4   always @(Present_state)// do the required job ineach state
5   begin
6       case (Present_state) //assert the required signals in each clock cycle
7           Default: begin //initialize the signals
8               PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                    initialize the signals
9               MARin <= 0; Zlowin <= 0; Zhighin <= 0;
10              PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
11              IncPC <= 0; Read <= 0; CONTROL <= 0;
12              Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
13              Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                    <= 0;
14              Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
15              BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                    ram_enable <= 0;
16              R15_enable <= 0; PC_enable <= 0;
17          end
18          T0: begin
19              #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                    initialize the signals
20              #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
21          end
22          T1: begin
23              #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; //Mdatain <=
                    32'hB9000000; // opcode for mfhi R2
                    (10111001000000000000000000) 000//
24              #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
25          end
26          T2: begin
27              #10 MDRout<= 1; IRin <= 1;
28              #15 MDRout<= 0; IRin <= 0;
```

```
        end
        T3: begin // enable write to register by reading from HI
            #10 Gra <= 1; Rin <= 1; highout <= 1;
            #15 Gra <= 0; Rin <= 0; highout <= 0;
        end

    endcase
end
```

F.19 mflo R2

```
//uses ConFF to determine what branch instruction to use op code is
    different however
// opcode for mflo R2 (10100xxxx----------------------)//   if R15 it is
    for procedure return

always @(Present_state)// do the required job ineach state
begin
    case (Present_state) //assert the required signals in each clock cycle
        Default: begin //initialize the signals
            PCout <= 0; Zlowout <= 0; Zhighout <= 0; MDRout<= 0;  //
                initialize the signals
            MARin <= 0; Zlowin <= 0; Zhighin <= 0;
            PCin <= 0; MDRin <= 0; IRin  <= 0; Yin <= 0;
            IncPC <= 0; Read <= 0; CONTROL <= 0;
            Clear <= 1; Yout <= 0; highin <= 0; lowin <= 0;
            Cout <= 0; outPortIn <= 0; inPortOut <= 0; inPortIn<= 0; con_in
                <= 0;
            Mdatain <= 32'h00000000; highout <= 0; lowout <= 0;
            BAout <= 0; Rin <= 0; Rout <= 0; Gra <= 0; Grb <= 0; Grc <= 0;
                ram_enable <= 0;
            R15_enable <= 0; PC_enable <= 0;
        end
        T0: begin
            #10 PCout <= 1; MARin <= 1; IncPC <= 1; Zlowin<= 1;  //
                initialize the signals
            #15 PCout <= 0; MARin <= 0; IncPC <= 0; Zlowin<= 0;
        end
        T1: begin
            #10 Zlowout<= 1; PCin <= 1; Read <= 1; MDRin <= 1; // Mdatain <=
                 32'hC1000000; // opcode for mflo R2
                (11000001000000000000000000) 000//
            #15 Zlowout<= 0; PCin <= 0; Read <= 0; MDRin <= 0;
        end
        T2: begin
            #10 MDRout<= 1; IRin <= 1;
            #15 MDRout<= 0; IRin <= 0;
        end
        T3: begin
            #10 Gra <= 1; Rin <= 1; lowout <= 1;
            #15 Gra <= 0; Rin <= 0; lowout <= 0;
        end
    endcase
end
```