Jyresa Mae M. Amboang
Frances Nicole S. Gigataras
Crishelle A. Agopitac
Erica Dianne Q. Canillo
Laboratory Submitted: June 13, 2022
Laboratory Hours: 504 hours

Final Project Part II
Multicycle ARM Processor (Part II)

Multicycle Processor Completion

In this Final Project Part II, you will complete your own multicycle ARM processor! Please refer to your Final Project Part I handout for an overview of the processor. In the Final Project Part I, you created the control unit. In this lab you will design the datapath and test your completed ARM multicycle processor. This lab uses the same test bench and generic parts as in laboratory activity 8. Copy them from arm_single_xx.sv from laboratory activity 8. Copy your alu_xx.sv file from laboratory activity 5 to your lab11_xx directory. Also copy your arm_multi_xx.sv file containing your controller from Final Project Part I.

Table 1. Expected Instruction Trace

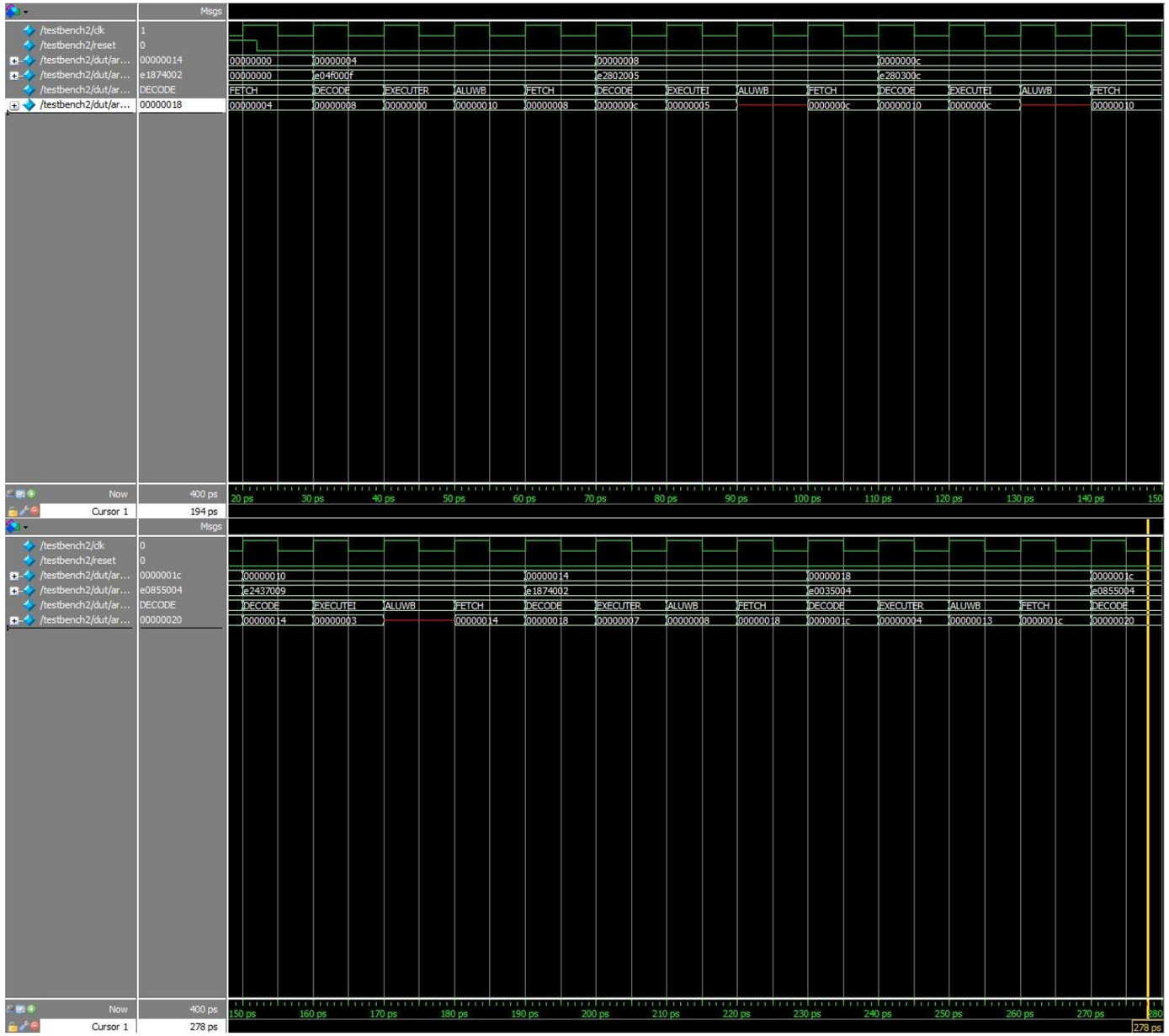
Cycle	Reset	PC	Instr	(FSM) state	SrcA	SrcB	ALUResult
1	1	00	0	FETCH	0	4	4
2	0	04	SUB E04F000F	DECODE	4	4	8
3	0	04		EXECUTER	8	8	0
4	0	04		ALUWB	x	x	x
5	0	04		FETCH	4	4	8
6	0	08	ADD E2802005	DECODE	8	4	C
7	0	08		EXECUTEI	0	5	5
8	0	08		ALUWB	x	x	x
9	0	08		FETCH	8	4	C
10	0	0C	ADD E280300C	DECODE	C	4	10
11	0	0C		EXECUTEI	0	C	C
12	0	0C		ALUWB	x	x	x
13	0	0C		FETCH	C	4	10
14	0	10	SUB E2437009	DECODE	10	4	14
15	0	10		EXECUTEI	C	9	3
16	0	10		ALUWB	x	x	x
17	0	10		FETCH	10	4	14
18	0	14	ORR E1874002	DECODE	14	4	18
19	0	14		EXECUTER	3	5	7
20	0	14		ALUWB	x	x	x
21	0	14		FETCH	14	4	18
22	0	18	AND E0035004	DECODE	18	4	1C
23	0	18		EXECUTER	C	7	4
24	0	18		ALUWB	x	x	x
25	0	18		FETCH	18	4	1C
26	0	1C	ADD E0855004	DECODE	1C	4	20
27	0	1C		EXECUTER	4	7	B

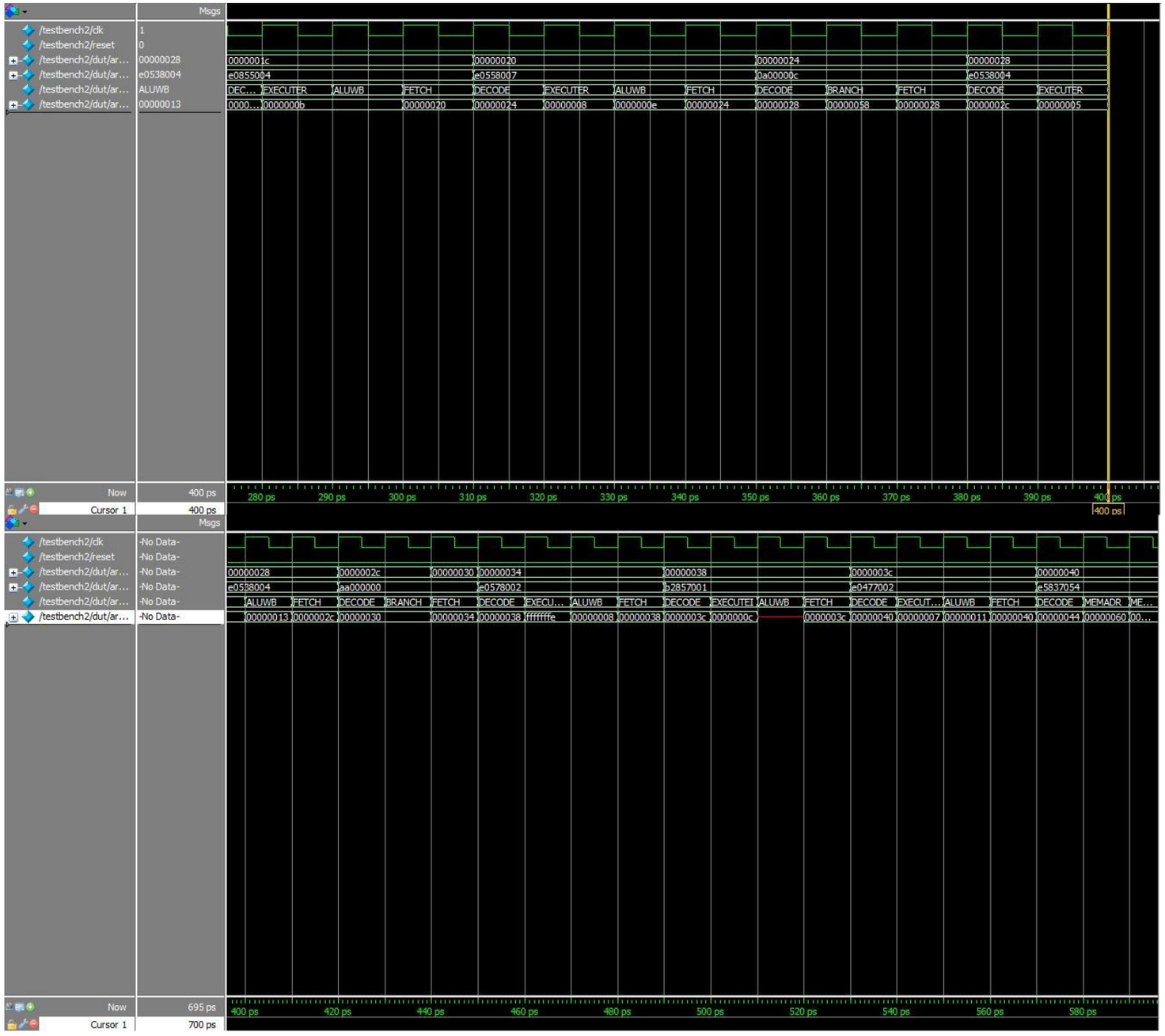
28	0	1C		ALUWB	x	x	x
29	0	1C		FETCH	1C	4	20
30	0	20	SUBS E0558007	DECODE	20	4	24
31	0	20		EXECUTER	B	3	8
32	0	20		ALUWB	x	x	x
33	0	20		FETCH	20	4	24
34	0	24	BEQ 0A00000C	DECODE	24	4	28
35	0	24		BRANCH	28	30	58
36	0	24		FETCH	24	4	28
37	0	28	SUBS E0538004	DECODE	28	4	2C
38	0	28		EXECUTER	C	7	5
39	0	28		ALUWB	x	x	x
40	0	28		FETCH	28	4	2C
41	0	2C	BGE AA000000	DECODE	2C	4	30
42	0	2C		BRANCH	30	0	30
43	0	30		FETCH	30	4	34
44	0	34	SUBS E0578002	DECODE	34	4	38
45	0	34		EXECUTER	3	5	-2
46	0	34		ALUWB	x	x	x
47	0	34		FETCH	34	4	38
48	0	38	ADDLT B2857001	DECODE	38	4	3C
49	0	38		EXECUTEI	B	1	C
50	0	38		ALUWB	x	x	x
51	0	38		FETCH	38	4	3C
52	0	3C	SUB E0477002	DECODE	3C	4	40
53	0	3C		EXECUTER	C	5	7
54	0	3C		ALUWB	x	x	x
55	0	3C		FETCH	3C	4	40
56	0	40	STR E5837054	DECODE	40	4	44
57	0	40		MEMADR	C	54	60
58	0	40		MEMWRITE	x	x	x
59	0	40		FETCH	40	4	44
60	0	44	LDR E5902060	DECODE	44	4	48
61	0	44		MEMADR	0	60	60
62	0	44		MEMREAD	x	x	x
63	0	44		MEMWB	x	x	x
64	0	44		FETCH	44	4	48
65	0	48	ADD E08FF000	DECODE	48	4	4C
66	0	48		EXECUTER	4C	0	4C
67	0	48		ALUWB	x	x	x
68	0	4C		FETCH	4C	4	50
69	0	50	B EA000001	DECODE	50	4	54
70	0	50		BRANCH	54	4	58
71	0	58		FETCH	58	4	5C
72	0	5C	STR E5802064	DECODE	5C	4	60
73	0	5C		MEMADR	0	64	64
74	0	5C		MEMWRITE	x	x	x

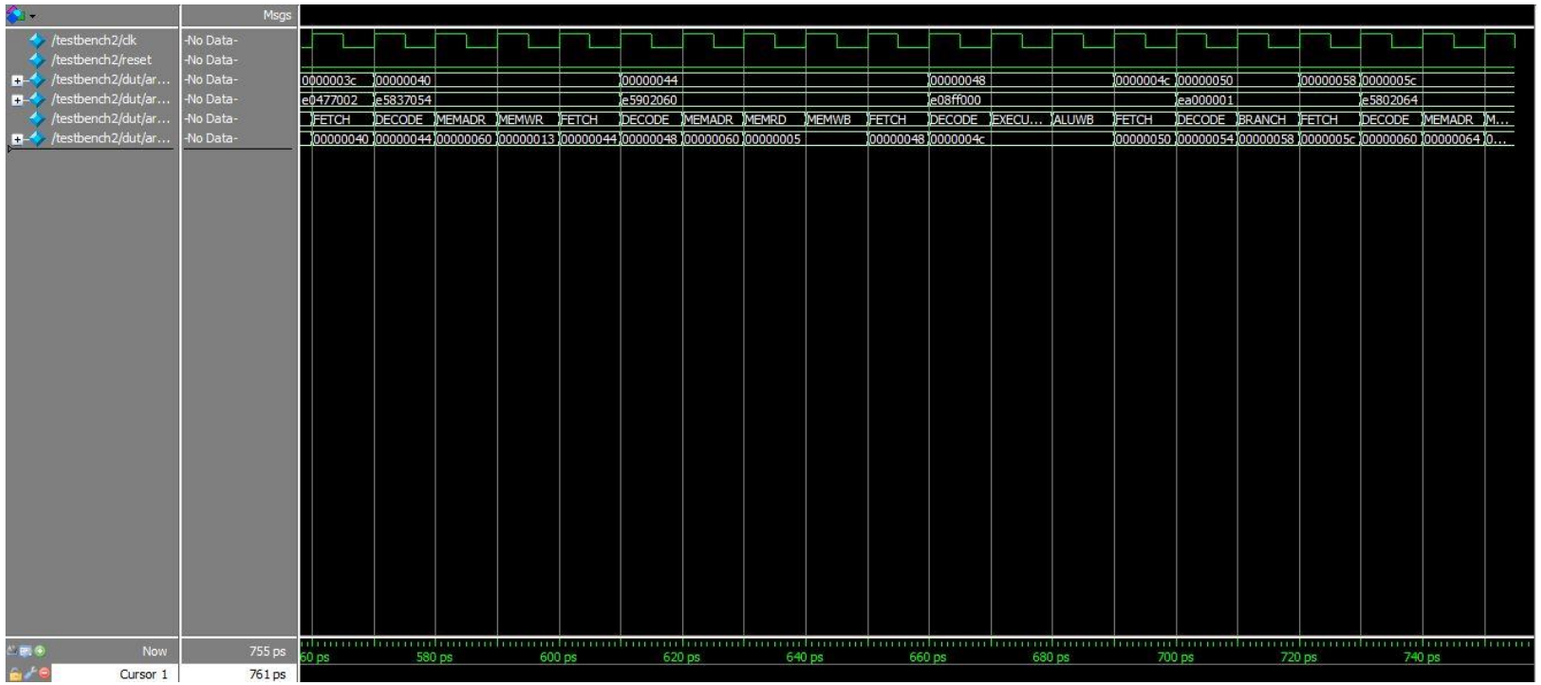
SystemVerilog Code for Datapath

```
module datapath(input  logic        clk, reset,
                output logic [31:0] Adr, WriteData,
                input  logic [31:0] ReadData,
                output logic [31:0] Instr,
                output logic [3:0]  ALUFlags,
                input  logic        PCWrite, RegWrite,
                input  logic        IRWrite,
                input  logic        AdrSrc,
                input  logic [1:0]  RegSrc,
                input  logic [1:0]  ALUSrcA, ALUSrcB, ResultSrc,
                input  logic [1:0]  ImmSrc, ALUControl);

  logic [31:0] PCNext, PC;
  logic [31:0] ExtImm, SrcA, SrcB, Result;
  logic [31:0] Data, RD1, RD2, A, ALUResult, ALUOut;
  logic [3:0]  RA1, RA2;

  // next PC logic
  flopenr #(32) pcreg(clk, reset, PCWrite, Result, PC);

  // memory logic
  mux2 #(32)    adrmux(PC, ALUOut, AdrSrc, Adr);
  flopenr #(32) ir(clk, reset, IRWrite, ReadData, Instr);
  flopr   #(32) datareg(clk, reset, ReadData, Data);

  // register file logic
  mux2 #(4)   ra1mux(Instr[19:16], 4'b1111, RegSrc[0], RA1);
  mux2 #(4)   ra2mux(Instr[3:0], Instr[15:12], RegSrc[1], RA2);
  regfile     rf(clk, RegWrite, RA1, RA2,
                 Instr[15:12], Result, Result,
                 RD1, RD2);
  flopr #(32) srcareg(clk, reset, RD1, A);
  flopr #(32) wdreg(clk, reset, RD2, WriteData);
  extend      ext(Instr[23:0], ImmSrc, ExtImm);

  // ALU logic
  mux3 #(32)  srcamux(A, PC, ALUOut, ALUSrcA, SrcA);
  mux3 #(32)  srcbmux(WriteData, ExtImm, 32'd4, ALUSrcB, SrcB);
  alu         alu(SrcA, SrcB, ALUControl, ALUResult, ALUFlags);
  flopr #(32) aluoutreg(clk, reset, ALUResult, ALUOut);
  mux3 #(32)  resmux(ALUOut, Data, ALUResult, ResultSrc, Result);
endmodule
```

ModelSim Simulation (clk, reset, PC, Instr, state, and ALUResult)




Msgs
/testbench2/clk
/testbench2/reset
/testbench2/dut/ar...
/testbench2/dut/ar...
/testbench2/dut/ar...
/testbench2/dut/ar...
1
0
00000028
e0538004
ALUWB
00000013
0000001c
e0855004
00000020
e0558007
00000024
0a00000c
00000028
e0538004
DEC... EXECUTER ALUWB FETCH DECODE EXECUTER ALUWB FETCH DECODE BRANCH FETCH DECODE EXECUTER
0000... 0000000b 00000020 00000024 00000008 0000000e 00000024 00000028 00000058 00000028 0000002c 00000005
Now
400 ps
Cursor 1
400 ps
280 ps 290 ps 300 ps 310 ps 320 ps 330 ps 340 ps 350 ps 360 ps 370 ps 380 ps 390 ps 400 ps
400 ps
Msgs
/testbench2/clk
/testbench2/reset
/testbench2/dut/ar...
/testbench2/dut/ar...
/testbench2/dut/ar...
/testbench2/dut/ar...
-No Data-
-No Data-
-No Data-
-No Data-
-No Data-
-No Data-
00000028 0000002c 00000030 00000034 00000038 0000003c 00000040
e0538004 aa000000 e0578002 b2857001 e0477002 e5837054
ALUWB FETCH DECODE BRANCH FETCH DECODE EXECU... ALUWB FETCH DECODE EXECUTEI ALUWB FETCH DECODE EXECUT... ALUWB FETCH DECODE MEMADR ME...
00000013 0000002c 00000030 00000034 00000038 ffffffffe 00000008 00000038 0000003c 0000000c 0000003c 00000040 00000007 00000011 00000040 00000044 00000060 00...
Now
695 ps
Cursor 1
700 ps
400 ps 420 ps 440 ps 460 ps 480 ps 500 ps 520 ps 540 ps 560 ps 580 ps



Conclusion:

What did we learn or take away from performing this Final Project Part II?

Implementing the datapath for our multicycle ARM processor as part of the Final Project Part II will provide us with invaluable, hands-on expertise in digital hardware design and microarchitecture. Through this endeavor, we can deepen our understanding of the complexities of hardware design using SystemVerilog and multicycle processing with microprocessors. As a result of this effort, we will better grasp how a processor's internal components interact. The ALU, multiplexers, registers, and memory may all be seamlessly integrated into a single system thanks to the careful planning of the datapath. This work has expanded our knowledge of module instantiation, datapath signal propagation, and bus connections. As we worked through the problems with our design, this project also helped us refine our debugging abilities. We learn to backtrack to the source of an issue, spot erroneous results, and pinpoint the start of a malfunction in a circuit. This procedure improves our ability to examine complicated systems and use methodical debugging strategies. We practice checking our design's accuracy by emulating the CPU on a test bench. The simulated waveforms are analyzed predicted, and actual results are compared, and the approach is adjusted as needed. Through many iterations of simulation and verification, we learn the importance of these two attributes in digital hardware engineering. Part II of the Final Project is a chance to demonstrate what we've learned about microarchitecture, SystemVerilog, finite state machines, and logic design, and it also gives us hands-on experience with those skills and more. This helps us prepare for the issues we'll face as computer engineers in the real world, where making dependable and effective processors is a top priority. As a result of completing this work, we will have the knowledge and experience to make significant contributions to digital hardware design.