

Final Project Paper

Instructions: A 2-page report in pdf describing the development process, like how you designed your code including your choice of data structures and how you implemented the important control modules and the final simulation results (number of cycles, register contents, number of instructions, etc.). Also, the group members' names have to be included.

Development Process

TABLE OF CONTENTS

- [Development Process](#)
- [TABLE OF CONTENTS](#)
- [SETUP](#)
- [PLANNING](#)
- [PROCESS IMPROVEMENT](#)
- [RESULTS](#)
- [DEBUGGING](#)
- [APPENDIX](#)
- [ORGANIZED CODE EXAMPLE](#)
- [CLOCK CYCLES EXAMPLE](#)
- [REGISTER FILE CONTENTS](#)
- [DEBUGGING USING WAVEFORMS](#)

SETUP

We began developing our processor on Tuesday April 5th. Our initial time estimate was that the project would take approximately 10 hours to complete. We scheduled a couple of meetings in the future to try to space out the work times. Nick already had modelsim on his computer, but Chris was using a lab computer. We decided it would be best to use github to collaborate and keep track of all our changes to the files. All our code can be found at:

<https://github.com/chriscanal/examples>

Collaborating in github is great for setting up a development process. The problem with using git was that the computers in the Snell Engineering computer lab do not have git and do not allow users to install new software. Chris only had a mac computer so we had to figure out another way for Chris to use both modelSim and git so that Chris and Nick could collaborate efficiently. Chris choose to use AWS Cloud Workspaces (see <https://aws.amazon.com/workspaces/>). He started up a cheap instance of a windows machine with 1 processor and 4 gigs of ram, then he installed both git and modelsim on the machine. Everything worked great. Despite this small roadbump Chris and nick were still fairly confident that they could finish the project in about ten more hours.

Final Project Paper

PLANNING

The first order of business for us was to schedule all of our meeting times to work on the project together. The list of dates below includes all of the time we spent working on the project. We originally paced ourselves for a 10 hour project.

Tue Apr 5	5:00pm – 6:00pm	⊕ Meet with Nick in Library to work on Project - Library to work on Project 📅 👤
Fri Apr 8	1:30pm – 3:00pm	⊕ Chris and Nick work on Computer Architecture Project - Library 📅 👤
Sun Apr 10	10:30am – 3:30pm	⊕ Chris and Nick do Computer Architecture Project - Library 📅 👤
Wed Apr 13	11:30am – 1:00pm	⊕ Chris and Nick work on Comp arch final 📅
Thu Apr 14	3:00pm – 4:00pm	⊕ Chris and Nick meet Linbin for Computer Architecture help - Basement of Hayden 📅 👤
Fri Apr 15	2:00pm – 3:00pm	⊕ Meet with Nick Kanaian for Computer Arch 📅 👤
	5:00pm – 6:30pm	⊕ Meet with Nick Kanaian for Computer Arch - library 📅 👤
Sun Apr 17	9:00am – 10:30pm	⊕ Meet with Nick Kanaian for Computer Arch - library 📅 👤
Mon Apr 18	12:00pm – 4:00pm	⊕ Chris and Nick finish comp arch project - Library 📅 👤
	12:00pm – 4:00pm	⊕ Chris and Nick Finish Comp Arch Final Project - Library 📅 👤

The first meeting was entirely dedicated to set up and planning, the rest were work sessions. As you can see in the table below, the project took us more than 30 hours total. This was due to a lot of time spent debugging our many files.

PROCESS IMPROVEMENT

Now that our environments were set up, we ready to start developing. We each created our own branches and started dividing up assignments. Chris started by creating the new paths in the control, ALU control and ALU, while nick began working on the processor. Chris finished this part quickly. Most of the work for this project went into the processor.v file. Since we wanted to make the best use of our time, we decided to use collabedit (see <http://collabedit.com/sknns>). This site allows us to edit the same file real time. It is very similar to google docs, however it has the addition of line numbers and an interpreter that color code the text to make the code easier to read. We continued in this manner while simultaneously using github to keep track of other small changes that we made in other files.

Final Project Paper

DEBUGGING

Debugging was the majority of time for this project. Collectively, we spent more than 30% of the time debugging our code. We attribute this failure mainly due to poor naming conventions at the onset of this project. We eventually developed a strict naming convention that helped us easily follow the datapath of various instructions. The convention we choose was:

(Module Name)(Variable Name)(Input or Output)

An example of this naming convention is the opcode input to the control module, this results in: controlOpcodeInput. This strict adherence to this naming convention helped us know what variables represented each register or wire from each module without having to look it up. This also allowed us to make sure that none of the variables had names that conflicted. Another helpful convention that we followed was labeling our blocks of code. This helps keep large code files easy to read(see Appendix: [ORGANIZED CODE EXAMPLE](#)).Unfortunately, we still had some bugs in the form of miss matched inputs. We approached these bugs using modelsim's waveform generator tool.

We noticed various errors in the waveforms mainly due to programmer fatigue and ignorance of basic modelsim/verilog skills. The waveform generating tool allowed us to follow the datapath throughout the processor and check to make sure that the data matched up from module to module. We could follow the datapath by checking which variables were set equal to each other within a cycle and then finding each of those waveforms and comparing to check equality. We also used this method to check the outputs of the various modules. (see Appendix: [DEBUGGING USING WAVEFORM](#) for an example of our debugging method).

RESULTS

After more than 30 hours we got our test bench to execute all instructions correctly and end with the correct register outputs. We printed the register contents in the transcript for quick easy reading of the testbench. We used a 1Mhz clock and the processor ran for 149 cycles (see Appendix: [CLOCK CYCLES EXAMPLE](#) for our proof of clock cycles). The way that we set up our processor, one instruction executes every clock cycle; therefore, 149 instructions were executed in total. The total run time of our program was just under 150 microseconds. The register contents can be seen in Appendix: [REGISTER FILE CONTENTS](#).

Final Project Paper

APPENDIX

ORGANIZED CODE EXAMPLE

```
5 module Processor(  
6     clk,  
7     instruction,  
8     memoryDataOutOutput_processorInput,  
9     writtenRegAddressOutput,  
10    writtenRegDataOutput,  
11    memoryInstAddrInput_processorOutput,  
12    memoryDataAddrInput_processorOutput,  
13    memoryDataInInput_processorOutput,  
14    memoryMemReadInput_processorOutput,  
15    memoryMemWriteInput_processorOutput  
16 );  
17  
18 /*-----DECLERATIONS-----*/  
19  
20  
21 /*-----ALU-----*/  
22 //INPUTS  
23 reg [31:0] ALUPort0Input;  
24 reg [31:0] ALUPort1Input;  
25 reg [3:0] ALUControlInput;  
26  
27 //OUTPUTS  
28 wire [31:0] ALUResultOutput;  
29 wire ALUZeroResultOutput; //Set to 1 if the result of the result of the ALU operation is '0'  
30 /*-----END ALU-----*/  
31  
32  
33  
34  
35 /*-----ALUControl-----*/  
36 //INPUTS  
37 reg [5:0] ALUControlFunctInput;  
38 reg [1:0] ALUControlOpInput;  
39  
40 //OUTPUTS  
41 wire [3:0] ALUControlOperationOutput;  
42 /*-----END ALUControl-----*/  
43  
44  
45  
46  
47 /*-----AndGate-----*/  
48 //INPUTS  
49 reg andGateInput1;  
50 reg andGateInput2;  
51  
52 //OUTPUTS  
53 wire andGateOutput;  
54 /*-----End AndGate-----*/  
55
```

Final Project Paper

```
347 /*-----PCAdd-----*/
348 ALU myPCAdd(.ALUControl(PCAddControlInput), .DataIn0(PCAddPort0Input), .DataIn1(PCAddPort1Input), .DataOut(PCAddResultOutput), .ZeroOut(PCAddZeroResultOutput) );
349 /*-----END PCAdd-----*/
350
351
352
353
354 /*-----RegFile-----*/
355 regFile myRegFile(.readAddress0(regFileReadReg0Input), .readAddress1(regFileReadReg1Input), .writeAddress(regFileWriteRegInput), .writeData(regFileWriteRegDataInput) );
356 /*-----END RegFile-----*/
357
358
359
360
361 /*-----SignExtend-----*/
362 Sign_extend mySignExtend(.out(signExtendOutput), .in(signExtendInput) );
363 /*-----END SignExtend-----*/
364
365
366 /*=====END MODULE INSTANCES=====*/
367
368
369
370
371 /*=====CONTROL CONNECTIONS=====*/
372
373
374 always@(controlRegDstOutput, controlALUSrcOutput, controlMemtoRegOutput, controlRegWriteOutput, controlMemReadOutput, controlMemWriteOutput, controlBranchOutput, controlJumpOutput)
375 begin
376     /*-----MUX control lines-----*/
377
378     MUXALUSrcControlInput = controlALUSrcOutput;
379     MUXBranchControlInput = controlBranchOutput;
380     MUXJumpControlInput = controlJumpOutput;
381     MUXMemtoRegControlInput = controlMemtoRegOutput;
382     MUXRegDstControlInput = controlRegDstOutput;
383
384     /*-----End MUX control lines-----*/
385
386     /*-----Other module control lines-----*/
387
388     regFileRegWriteInput = controlRegWriteOutput;
389     memoryMemReadInput_processorOutput = controlMemReadOutput;
390     memoryMemWriteInput_processorOutput = controlMemWriteOutput;
391     ALUControlOpInput = controlALUOpOutput;
392
393     /*-----End Other module control lines-----*/
394 end
395
396 /*=====END CONTROL CONNECTIONS=====*/
397
398
399
```

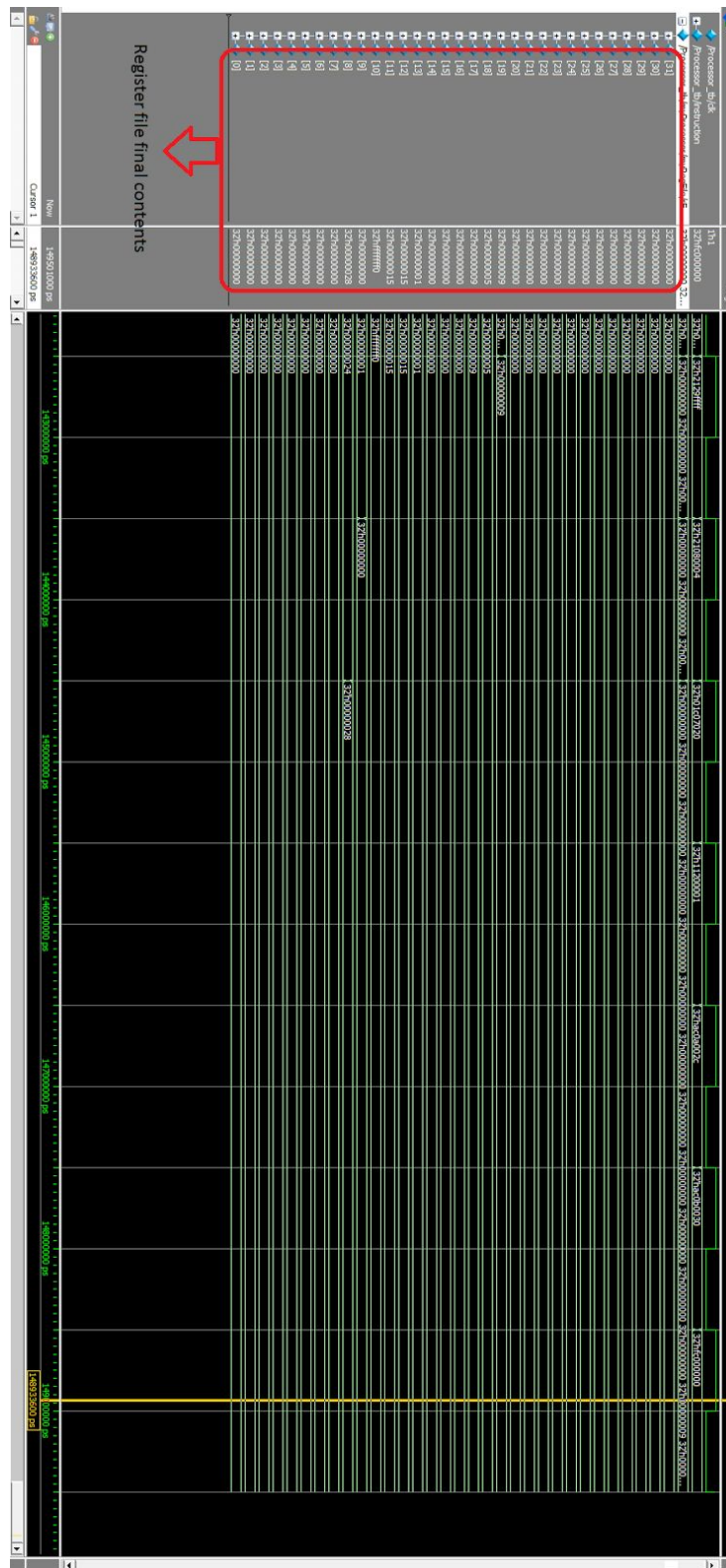
Final Project Paper

CLOCK CYCLES EXAMPLE



Final Project Paper

REGISTER FILE CONTENTS



Register file final contents

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14	0x00000000
R15	0x00000000
R16	0x00000000
R17	0x00000000
R18	0x00000000
R19	0x00000000
R20	0x00000000
R21	0x00000000
R22	0x00000000
R23	0x00000000
R24	0x00000000
R25	0x00000000
R26	0x00000000
R27	0x00000000
R28	0x00000000
R29	0x00000000
R30	0x00000000
R31	0x00000000

DEBUGGING USING WAVEFORMS

