BTSB-ISA
Updated

9-bit Pipelined CPU

Ezequiel Herrera-Ortiz
Richard Bull
Cody Heiner

1. Have you made any changes to your ISA from lab 1? What were they? Why did you make them?
A: For this lab we haven't changed anything. From lab 3 we noticed that we had to rewrite a few instructions for program 1 and program 3 as they did not run properly. For program 3 the old instruction did not take into account that we need the smallest value that occurs the most. In program 2 the shifting was done opposite to the correct way and the implicit registers were changed. However the changes to program 1's MFL and MFH instruction were partially for convenience. We also needed to do immense error checking and corrections to almost all instructions implemented and as a side-effect of these changes some extra functionality is now available, however it is unused. We also have realized one of our instructions, SM which was intended for Program 1 is unneeded. If we really needed a new instruction we could remove this one". We also noticed that our branching by address stored in register would be more of a headache than it was worth since our pipeline would require special routing for a branch to be done as soon as possible. So we removed 1 bit from our branching description and now we only have Branch Equals To and Branch Not Equals to with a signed 5 bit offset from the current PC. We also noticed that none of our programs are very long so a branching distance of roughly +/- 16 lines is almost an entire program.

2. Explain how you deal with each relevant data hazard.
A: Each pipeline forwards the register it executed to the previous pipelines. A pipeline will check if it's register destination matches any of the dependent registers forwarded. If it sees a match it will use the data of the most recently updated part of the pipeline (closer to the current pipeline stage means it's newer). For our standard data hazards we've introduced stalls based on again dependent values. We check all 4 stages of the pipeline and if a stage has a value we need and it still hasn't completed, we stall however cycles it takes for it to complete. Then we simply let forwarding handle the rest. We also had to change the way some of our functions we implemented. Some of our modules, such as the regfile, executed logic that should've been done in the ALU. So to follow the flow of the pipeline we had to break up the logic and spread it throughout modules.

3. Explain how you deal with each relevant branch hazard.
A: We essentially always guess our branch will be taken and check in the EX stage if that was true. Since our programs are almost entirely loop based, 99% of the time we guess correctly and nothing needs to be done to correct our prediction. However when our loop ends, or if we simply just don't take a branch, we flush the last 2 cycles and update our PC to reflect the right line.

4. What are your dynamic instruction counts for program 1? program 2? program 3?
A:
P1: 57 Instructions (Excluding Flushes)

P2: 272

P3: 274

5. What are your cycle counts for program 1? program 2? program 3?
A:
P1: 63

P2: 276

P3: 278

6. What is your CPI for each program? What did you do to reduce CPI? How would you reduce CPI further?
A:
P1: CPI = 1.10
Unless we had a very sophisticated branch predictor, this is probably as good as it's going to get.

P2: CPI = 1.01

P3: CPI = 1.01

7. What could you have done differently (e.g., ISA changes) to better optimize for dynamic instruction count?
A:
Our instruction count is pretty ideal. Any more optimization would just lead to an unnecessary amount of logic that would greatly increase the complexity of our machine as well as increase our CT.

8. What did you do in your ISA design that made pipelining easy? What did you do in your ISA that made pipelining hard? Give examples.
A:
Having only 5 pipeline stages and having the execute stage in the 4th pipe. This greatly reduced the amount of lookahead that was required. Having different register addressing doubled the amount of lookahead we had to do.

9. How successful were you at minimizing cycle time? What could you have done better?
A: Since we used the standard 5 stage pipeline and most of our logic was done in the EX and the ID stage, I would say we did a horrible job at decreasing our cycle time.

10. If you were to make a deeper pipeline, what would you change? Would performance be better or worse?
A: It would probably be wise to break up the EX stage into several parts, a long with ID and IF. The register file adds a lot of delay to ID, branch prediction adds more delay than it should to IF, and finally the EX stage is hindered by doing almost all of the computation.

11. How easy/difficult would it be to extend your design to a superscalar implementation? Give examples.
A: This would greatly increase the difficulty. We would need to need to change our data forwarding logic to check on the instruction as well. We would need twice as many checks, and branch prediction would also get very ugly for such cases like if we see 2 branches.

12. What is the cycle time of your machine? Which pipeline stage is the bottleneck of your design? What do you think would be the cycle time if you eliminated that bottleneck (i.e., what is the second longest stage – give your best guess, don't have to prove it)?
A:

13. What is the total execution time for each of the three programs?
A:
P1: 655ns

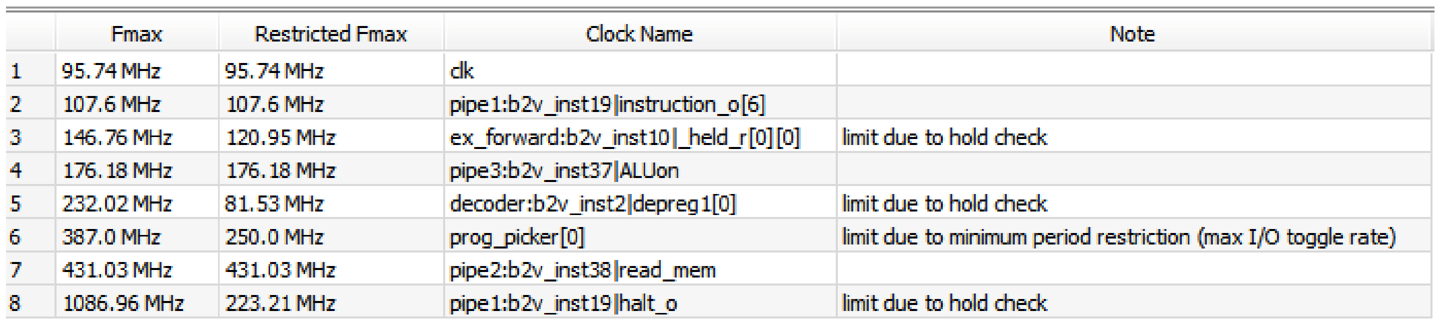
P2: 2869ns

P3: 2891ns

14. Which would be easiest to improve in your design, IC, CPI, or CT? Why?
A: To improve performance we would most likely be able to get the most performance out of CT. Our pipeline stages are fairly unbalanced, IF and wb carry less workload than our execute or reg files, so we would have to split those pipelines into smaller ones.

15. If you were to start the entire design over (including ISA), what is the most important change you would make?
A: If we were to start the entire design over, we probably wouldn't change much from our original design since we already know how to do it. We would however, try to use some of the more advance techniques to improve the performance such as a branch predictor and possibly extra pipeline stages. It would also probably be wise to thoroughly examine how much we need to run each program and over-specialize instead of trying to generalize and have specialized instructions as a side-thought. I believe in the end that bit us badly.

Timing Analyzer Results:



	Fmax	Restricted Fmax	Clock Name	Note
1	95.74 MHz	95.74 MHz	clk	
2	107.6 MHz	107.6 MHz	pipe1:b2v_inst19|instruction_o[6]	
3	146.76 MHz	120.95 MHz	ex_forward:b2v_inst10|_held_r[0][0]	limit due to hold check
4	176.18 MHz	176.18 MHz	pipe3:b2v_inst37|ALUon	
5	232.02 MHz	81.53 MHz	decoder:b2v_inst2|depreg1[0]	limit due to hold check
6	387.0 MHz	250.0 MHz	prog_picker[0]	limit due to minimum period restriction (max I/O toggle rate)
7	431.03 MHz	431.03 MHz	pipe2:b2v_inst38|read_mem	
8	1086.96 MHz	223.21 MHz	pipe1:b2v_inst19|halt_o	limit due to hold check