0512.4490 Advanced Computer Architecture Lab 2022

Lab #3: Pipe-lining the processor

In the previous labs, we designed a RISC processor with DMA support, suitable for inclusion in a system on a chip (SoC).

Or so we thought.

It turns out potential customers are unhappy with the low performance of our processor. Although the DMA is a unique feature, they say our baseline performance is far below our competitors.

To avoid losing the customer, management concluded that the processor should be pipelined, hoping to reduce CPI from the current 6 cycles per instruction to 1.

Further, it was estimated that the single SRAM may be a performance bottleneck due to many structural hazards, and therefore the new processor will follow the Harvard architecture, containing two single ported SRAMs, each the same size as the original: one for instructions, and one for data.

You have been in charge with the design of the new micro-architecture. You'll model it in the low-level simulator and verify it against the same high-level ISS from lab #1. To compete well, the processor should contain the following features:

1. Pipelining.
2. Branch predictor.
3. DMA support.

SP pipelined micro architecture

In contrast to labs 2 and 4, we'll now use a micro-architecture with two single ported SRAMs: `srami` for instructions, and `sramd` for data. The original control states will now be pipeline stages, active simultaneously:

F0: FETCH0: Issues read command to memory to fetch the current instruction from address PC.
F1: FETCH1: Samples memory output to the inst register.
D0: DEC0: Decode the instruction register into its fields: opcode, dst, src0, src1, and immediate (sign extended).
D1: DEC1: Prepares the ALU operands.
E0: EXEC0: Executes ALU and LD operations.
E1: EXEC1: Write backs ALU and memory (ST).

Registers (basic set, you may add additional as required):

The architectural registers and cycle_counter will remain as before:

r2 – r7: [31:0]: 6 32-bit registers, holding the ISA general registers. Different from Computer Structure course, a value written to a register in clock cycle *t*, will be available only in clock cycle *t+1*. This affects hazard analysis.
cycle_counter[31:0]: 32 bit cycle counter.

For the micro architecture registers, we will have multiple copies, where each register will now contain a prefix, depending on its pipeline stage:

fetch0_active, fetch1_active, dec0_active, dec1_active, exec0_active, exec1_active: 1-bit active signals.
fetch0_pc, fetch1_pc, dec0_pc, dec1_pc, exec0_pc, exec1_pc: [15:0] 16-bit program counter.
dec0_inst, dec1_inst, exec0_inst, exec1_inst [31:0] 32-bit instruction.
dec1_opcode, exec0_opcode, exec1_opcode [4:0]: 5-bit opcode.
dec1_dst, exec0_dst, exec1_dst[2:0]: 3 bit destination register index.
dec1_src0, exec0_src0, exec1_src0 [2:0]: 3 bit source #0 register index.
dec1_src1, exec0_src1, exec1_src1[2:0]: 3 bit source #1 register index.
dec1_immediate exec0_immediate, exec1_immediate [31:0]: sign extended immediate.
exec0_alu0, exec1_alu0 [31:0]: 32 bit alu operand #0.
exec0_alu1, exec1_alu0 [31:0]: 32 bit alu operand #1.
exec1_aluout[31:0]: 32 bit alu output.

Question 1: Harvard architecture

The new processor uses the Harvard architecture. Discuss advantages/disadvantages of that decision. Is it a good decision? What effect would using Von Neuman have.

Question 2: Structural and Data Hazards

A pipelined micro architecture introduces new hazards which didn't exist before in the simple state machine implementation. To make it easy on the programmers, we resolve the hazards in hardware.

List the hazards which exist and explain in detail how each of them is resolved in your design. Use timing diagrams to illustrate.

Question 3: Branches and branch prediction

Explain in detail how jumps are handled. Key points to consider are your branch prediction scheme, branch resolution, and pipeline flushes. Illustrate with timing diagrams.

Question 4: Low level simulator pipeline implementation

Submit a low level simulator for the SP pipelined architecture. As before, it should generate two trace files: an instruction trace which is the same as the high level simulator trace from lab 1, and a cycle trace which will be the same as the Verilog implementation, but now different (and faster) from the cycle trace in labs 2 and 4.
Notes:

- Use the provided source code template
- In this lab, you don't need to submit two separate simulators as you were requested in lab2. Instead, submit only one simulator which includes the DMA functionality.

Question 5: Low level testing: example.bin, add.bin, sqrtq.bin

Verify that the pipelined processor works with add.bin which adds two signed numbers in sign magnitude format, and the sqrtq.bin which generates quotient portion of a square root. These two example programs operate exactly as in the previous exercises. Make sure that your inst_trace and sramd outputs match the outputs of the *iss* from lab1.
Submit: For each test case, submit the two trace files (inst_trace.txt and cycle_trace.txt) and the sramd output.

Question 6: Speedup comparison, next generation improvements

In the report, answer the following questions:

1. Calculate the speedup of each test to the non-pipelined version from lab 2.
2. Is the new CPI 1 as management hoped? Explain.
3. Suggest ways to further improve the next generation design to increase the IPC even above 1.

Question 7: DMA testing in a pipeline environment

Design a test for the DMA and verify that it works well in the pipeline micro-architecture. Namely, make sure that all possible pipeline hazards are covered by the test and the processor (with the DMA) manages to overcome these hazards.
Submit:

- Attach 3 files: dma_pipe.bin; dma_pipe_cycle_trace.txt; dma_pipe_sramd_out.txt
- In the report:
  - Provide an annotated assembly code of your test.
  - Explain why the test coverage is good, and how we see from the logs that DMA works.

Note: you can begin by modifying your already existing dma.bin from lab-2. Maybe it already has a good hazard coverage...
**In your code you must indicate clearly which sections are your own. You must comment your code liberally. Copying and plagiarism will be punished since random checks will be done. Warning in the code of any kind will not be tolerated.**
**Please zip up the files like this. The program files all of the *.c *.h and makefiles including those given to you in the exercise under a subdirectory titled prog[id]. The id field is the teudat zehut number of one of the participants no**

**spaces in the subdirectory name. Another subdirectory titled other[id](no spaces) will contain the other files. In the directory name there are no brackets, the brackets just indicate that this is a field name Any deviation will not be accepted and resubmission will cost 5 points. No upper subdirectory. The top level to contain two subdirectories only!**