CS 152: Discussion Section 2

Pipelining Review

Yue Dai, Albert Ou
02/07/2020

Administrivia

- PS1 due on **Monday 10:30am**; solutions will be released next Friday
- PS2 will be released next Wednesday
- Lab 1 due on Wednesday, Feb 19th

Agenda

- Iron Law
- Pipelining
  - Hazards
  - Pipeline operation
  - Alternative pipeline organizations
- Exceptions

Iron Law (Important!)

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

Q: List three different techniques to improve each term.

Iron Law

Case 1: In a classic RISC pipeline, modifying the ISA (and thus the microarchitecture) to **use hardware interlocking instead of software interlocking** for both branch delay slots and load-use delay slots. [Quiz 1, 2011]

Instruction/program:

CPI:

Cycle Time:

Iron Law

Case 2: Remove hardware floating-point instructions and instead **use software subroutines** for floating-point arithmetic. [Quiz 1, 2013]

Instruction/program:

CPI:

Cycle Time:

Pipeline - Hazard

- Structural hazard
  - Q: Are there any structural hazards in a classic 5-stage RISC pipeline?
  - Q: What modifications to the pipeline may introduce a structure hazard?
- Data hazard
- Control hazard

Pipeline - Data Hazard

- Read-After-Write (RAW)

```
ADD x1, x2, x3
SW  x4, 4(x1)
```

- Write-After-Read (WAR)

```
ADD x1, x2, x3
SUB x2, x4, x5
```

- Write-After-Write (WAW)

```
ADD x1, x2, x3
SUB x1, x4, x5
```

Q: Why is RAW the only true dependency in a classic 5-stage RISC pipeline?

Pipeline - Exercise

Label all data hazards:

```
ADDI    x1, x0, 4
SW      x1, 8(x2)
SLLI    x3, x1, 1
ADD     x3, x2, x3
LW      x1, 0(x3)
```

Pipeline - Exercise

Q: What does the following code do? How many iterations does it run?

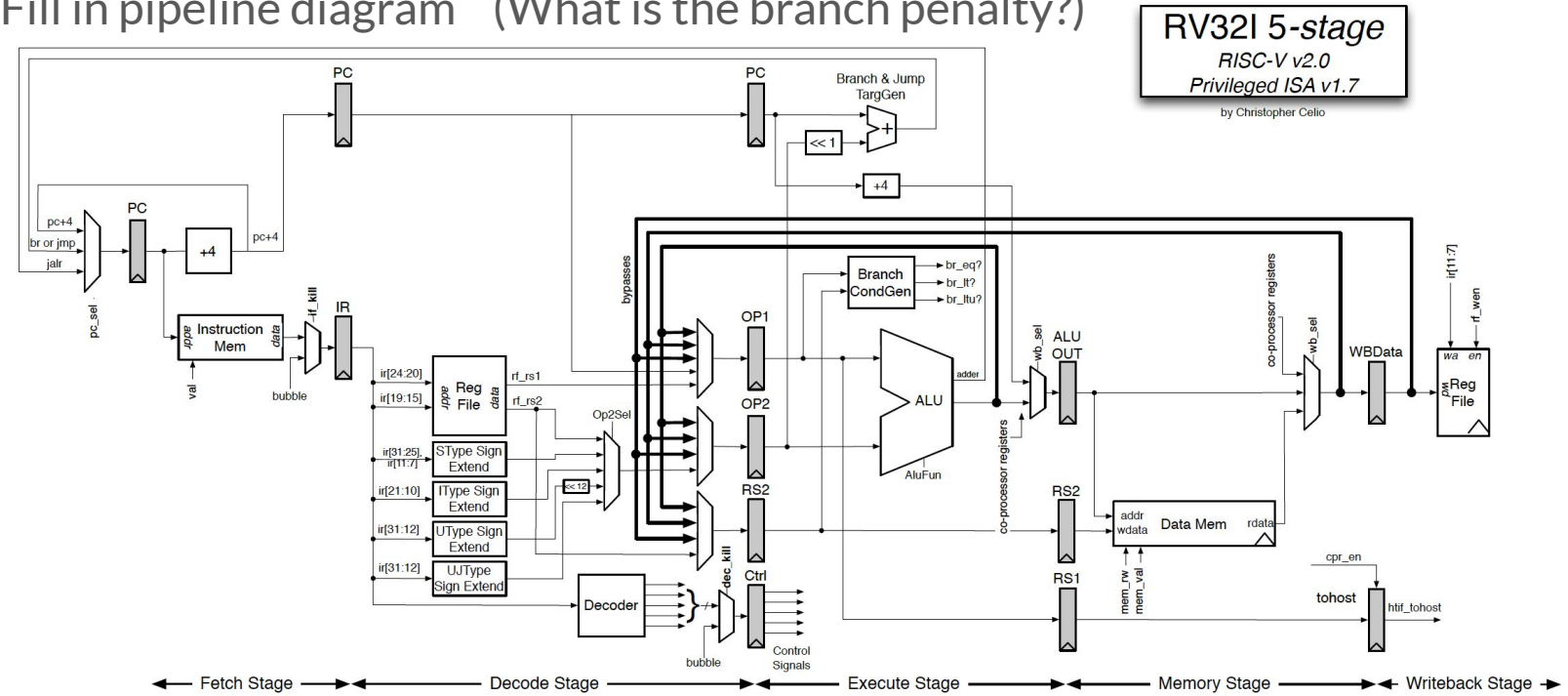
```
        ADD   x1, x0, x0
        ADDI  x2, x0, 0x800
LOOP:   LW    x2, 4(x2)
        ADD   x1, x2, x1
        BNEQ  x2, x0, LOOP
```

Memory:

0x400: 0x0
0x404: 0xD40
...
0x800: 0x9F0
0x804: 0x400
...
0x9F0: 0x400
0x9F4: 0x0
...
0xD40: 0x0
0xD44: 0x9F0

Pipeline - Classic RISC (Load-Use Interlock)

Fill in pipeline diagram (What is the branch penalty?)



Pipeline - Pointer Chasing Example

Q: For the prior code and pipeline what is the CPI?

(CPI is measured from when first instruction commits to when last instruction in the sequence commits)

Q: Give an expression for CPI for *K* iterations.

Q: What is the CPI with perfect branch prediction?

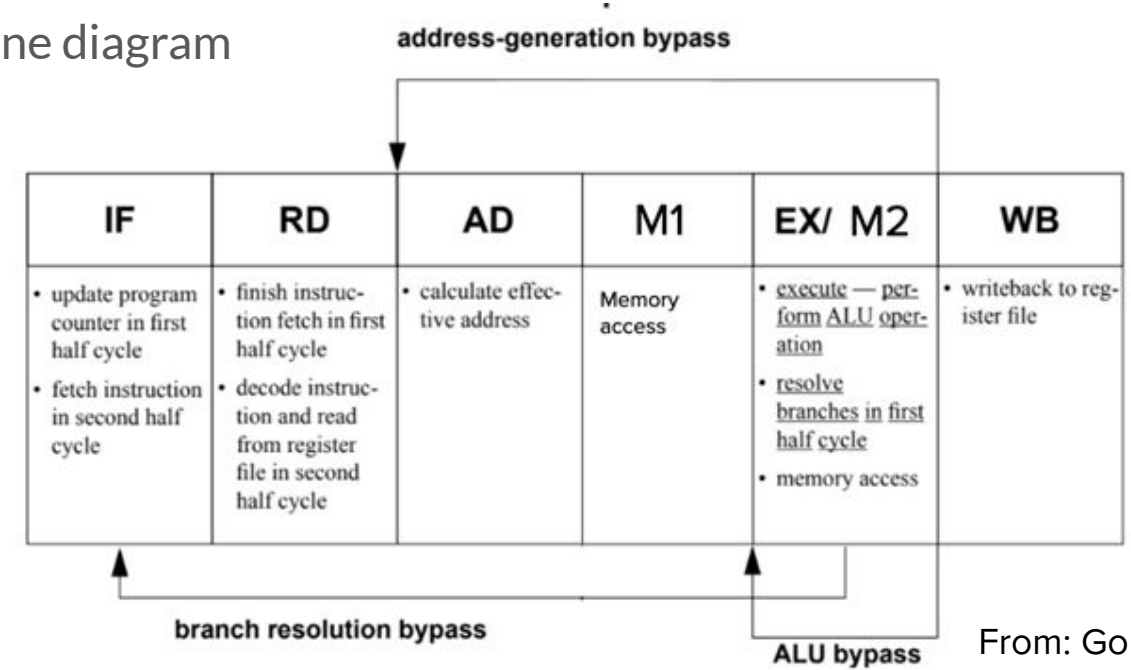
Pipeline - Impact after Modification

Q: How does CPI change if we split M into M1 and M2?

Q: How does CPI change when the M stage is made to be N stages long?

Pipeline - Address Generation Interlock

Fill in pipeline diagram



From: Golden and Mudge, *A comparison of two pipeline organizations* (1994)

Pipeline - AGI vs LUI

Q: What is the CPI for this pipeline (AGI-2)?

Q: How does CPI change when there is only one MEM stage?

Q: How does CPI change when there are *N* MEM stages?

Pipeline - Apply Iron Law

Suppose the classic RISC pipeline closes timing at 1 GHz.

Q: (For this application), at what frequency does the AGI-2 pipeline perform better? (Assume perfect branch prediction)

Q: AGI-N vs LUI-N?

Exceptions

Precise Exception Definition:

> All instructions prior to the exception in program order have committed, and none of the instructions after (and including the faulting instruction) appear to have started.

Q: Why are precise exceptions useful?

Q: Why might one not want to always implement precise exceptions?

Exception Handling in RISC-V (M-mode)

Example: Misaligned address exception on a load

1. MEPC (*exception PC)* ← PC of the load instruction
2. MCAUSE (*exception cause*) ← 0x4 (i.e., load address misaligned)
3. MTVAL ← Exception-specific metadata (i.e., faulting address)
4. MPP ← Previous privilege mode
5. MPIE ← MIE; MIE ← 0: Disable interrupts (and save the previous value)
6. PC ← MTVEC (*trap vector*)

This all happens atomically - why?

In the Exception Handler

- Preserve context
  - Swap stack pointer (x1) with MSCRATCH CSR (pointer to M-mode stack)
  - Spill as many registers as needed to memory
- Decode exception cause, dispatch to correct handler
- Emulate unaligned load
  - Issue two aligned loads and combine them together
  - Write result to the entry for *rd* in the saved register context
- Restore registers
  - Read old register values back from the stack
  - Swap MSCRATCH to restore original stack pointer

Exception Return

- Increment EPC
  - In this case, the instruction has been emulated, so no need to re-execute!
- Execute MRET to return to previous privilege mode in MPP field
  - PC ← MEPC
  - MIE ← MPIE