CS152/252A
Fall 2026

Pipelining
Discussion 2

1 Iron Law

1.1 For each term in the Iron Law, give at least three techniques that improve that term.

Instructions/program:

(a) Better algorithms

(b) Better compiler optimizations

(c) Different programming languages

(d) Complex instructions (RISC to CISC)

(e) Remove branch delay slots

Cycles/instruction:

(a) From multi-cycle implementation to pipelined implementation

(b) Optimize microcode

(c) Bypassing (forwarding)

(d) Adding branch delay slots

(e) Decoupling

(f) Branch prediction

(g) Prefetching

(h) CISC to RISC

Time/cycle:

(a) Shorten critical path

(b) Minimizing logic depth; less logic complexity; less wire congestion

(c) Deeper pipelining

(d) Better VLSI technology

(e) Different clock domains

1.2 Explain how each term changes given the proposed modification (increase / decrease / no effect).

(a) [**Quiz 1, 2013**] Remove hardware floating-point instructions (and the pipelined FP-Unit) and instead use software subroutines for floating-point arithmetic

Instructions/program: Increase (need more instructions to emulate floating-point operations)

Cycles/instruction: Decrease (integer operations usually lower latency)

Time/cycle: No effect (FPU is usually pipelined to not be on the critical path)
OR
Slight decrease (less logic)

(b) [**Sp24 MT1 Q2.2**] Suppose we extended the RISC-V Base ISA with a conditional move instruction which copies `rs1` to `rd` if and only if `rs2` is nonzero.

Instructions/program: Same or decrease (short branch sequences could be replaced by 1 instruction)

Cycles/instruction: Ambiguous (replaced branches could have been easy or hard for the branch predictor to predict)

Time/cycle: Slight increase (more control circuitry), or no effect if still overshadowed by the critical path

Tip: Iron Law questions might be ambiguous (by design), therefore you must state your assumptions in your answer to receive full credit!

2 Pipeline Hazards

2.1 Define the three types of pipeline hazards.

Data: An instruction depends on data produced by an earlier instruction

Structural: An instruction needs a resource being used by another instruction in the pipeline

Control: An instruction depends on the next-instruction address produced by an earlier instruction

2.2 Which type(s) of data hazard is/are possible in a classic RISC-V pipeline?

```
ADD x1, x2, x3
SW  x4, 4(x1)
```
Read-After-Write (RAW)

```
ADD x1, x2, x3
SUB x2, x4, x5
```
Write-After-Read (WAR)

```
ADD x1, x2, x3
SUB x1, x4, x5
```
Write-After-Write (WAW)

Only RAW is possible, as instructions travel in-order from D (reg read) to W (reg write).
- WAR: need HW that can issue a younger instruction's write before an older instruction's read
- WAW: need HW that can commit writes out of order; everything in the classic pipeline is in-order

2.3 Data Hazards: Identify all potential hazards

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

RAW	WAR	WAW
addi -> sw	sw <- add	addi -> lw
addi -> slli	lw <- add	addi -> sub
addi -> add	lw <- slli	lw -> sub
slli -> add	lw <- sw	
slli -> lw	sub <- sw	

RAW	WAR	WAW
	`sub <- slli`	
	`sub <- add`	

2.4 Structural Hazards

(a) Are there any structural hazards in a classic 5-stage RISC pipeline?

No, hardware resources are not shared between pipeline stages

(b) Are there any structural hazards in the single-bus microcoded machine?

The shared bus

(c) What modifications to a RISC pipeline may introduce a structural hazard?

- Single-ported memory (combining IF and MEM)
- Functional units / RF write ports in superscalar implementation
- Multi-cycle unpipelined functional units / memory (iterative multiplier/divider, blocking cache)

3 Data Hazard Mitigations

3.1 From lecture, what are the two strategies for handling data hazards?

Interlock: Wait for the hazard to clear by holding the dependent instruction in the issue (decode) stage

Bypass: Resolve the hazard earlier by bypassing (forwarding) the value as soon as it is available

Consider the following linked-list traversal program. For the pipeline diagrams below, assume:

(a) The register file is write-through (an instruction in D can read the value being written back in W in the same cycle)
(b) Branches are always predicted taken; squash if wrong
(c) Memory is always single cycle
(d) The loop runs for 2 iterations

```
      ADDI  x2, x0, 0x700
LOOP: ADD   x1, x2, x0
      LW    x2, 4(x2)
      BNE   x2, x0, LOOP
      SW    x3, 4(x1)
```

Case 1: Interlock

3.2 Identify the hazards.

- `addi -> add` (RAW on `x2`)
- `addi -> lw` (RAW on `x2`)
- `lw -> bne` (RAW on `x2`)
- bne -> control

3.3 Fill out the pipeline diagram for the interlocked pipeline (no bypassing).

```
      ADDI  x2, x0, 0x700
LOOP: ADD   x1, x2, x0
      LW    x2, 4(x2)
      BNE   x2, x0, LOOP
      SW    x3, 4(x1)
```

Case 1: Interlock pipeline

3.1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
addi	F	D	X	M	W																									
(1) add		F	D	D	D	X	M	W																						
lw			F	F	F	D	X	M	W																					
bne						F	D	D	D	X	M	W																		
(2) add							F	F	F	D	X	M	W																	
lw										F	D	X	M	W																
bne											F	D	D	D	X	M	W													
(3) add												F	F	F	D	-														
lw															F	-														
sw																F	D	X	M	W										

3.4 Interlock questions

- How many bubbles were needed?

  8; 6 from stalls (2 each for the first `add`, the first `bne`, and the second `bne`) and 2 from the branch misprediction at the end.

- What is the CPI for this program with interlocking? (CPI is measured from when the first instruction commits to when the last instruction in the sequence commits, inclusive of both: CPI = (last commit cycle − first commit cycle +1) / (# committed instructions).)

  (Last commit − first commit +1) / (# committed instructions) = $\frac{20-5+1}{8} = \frac{16}{8} = 2$
  Ideally the 8 committed instructions would commit in 8 consecutive cycles (cycles 5 to 12). With 8 bubbles, they take 16 cycles, so 2 cycles per instruction.
  Sanity check: ideal + bubbles = total cycles ($8 + 8 = 16$).

- What is the CPI assuming branches are always predicted correctly?

  2 bubbles come from the branch misprediction, so the last commit moves from cycle 20 to cycle 18: $\frac{20-5+1-2}{8} = \frac{14}{8} = 1.75$
  Sanity check: ideal + remaining bubbles $= 8 + 6 = 14$.

3.5 Interlocking & Iron Law. [**Quiz 1, 2011**] In a classic RISC pipeline, modify the ISA (and thus the microarchitecture) to use hardware interlocking instead of software interlocking for both branch delay slots and load-use delay slots.

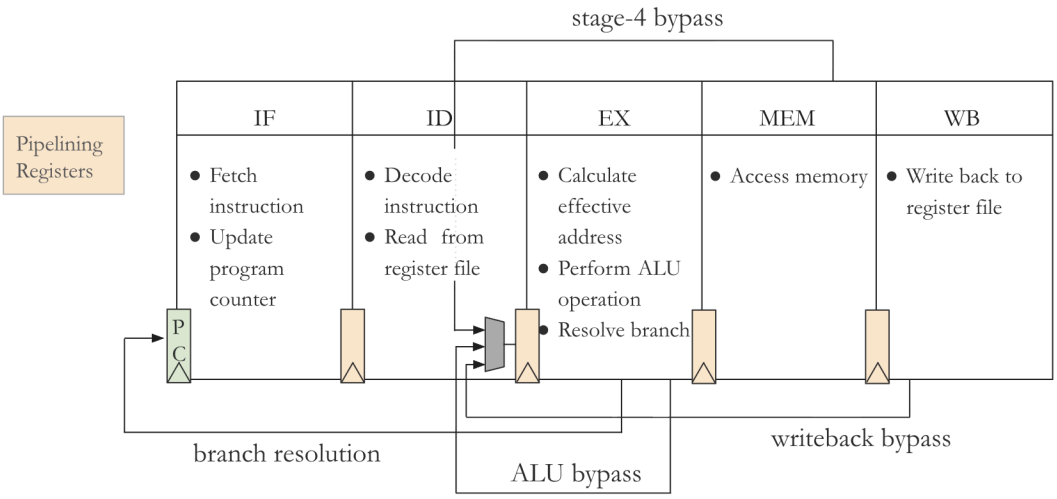
What is software interlocking? Programmers (or the compiler) have the responsibility of injecting NOPs to account for hazards.

Instructions/program: Decrease (fewer NOPs)

Cycles/instruction: Increase (NOPs count as instructions, removing NOPs with hardware interlock will cause stalls)

Time/cycle: Increase (more control logic - high fan-out stall signals)

Case 2: Bypass



3.6 Using the same code and assumptions, fill out the pipeline diagram for the bypassed pipeline shown above (ALU bypass, stage-4 bypass, and writeback bypass).

```
      ADDI  x2, x0, 0x700
LOOP: ADD   x1, x2, x0
      LW    x2, 4(x2)
      BNE   x2, x0, LOOP
      SW    x3, 4(x1)
```

Case 2: Bypass pipeline

3.3	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
addi	F	D	X	M	W																									
(1) add		F	D	X	M	W																								
lw			F	D	X	M	W																							
bne				F	D	D	X	M	W																					
(2) add					F	F	D	X	M	W																				
lw							F	D	X	M	W																			
bne								F	D	D	X	M	W																	
(3) add									F	F	D	-																		
lw											F	-																		
sw												F	D	X	M	W														

3.7 Bypass questions

- What is the CPI with bypass?

  (Last commit − first commit +1) / (# committed instructions) $= \frac{16-5+1}{8} = \frac{12}{8} = 1.5$
  Sanity check: ideal + bubbles $= 8 + (1 + 1 \text{ stalls} + 2 \text{ squashed}) = 12$.

- What is the CPI if load accesses take 2 cycles?

  $\frac{16+2-5+1}{8} = \frac{14}{8} = 1.75$
  One load per iteration, two iterations total, so 2 extra cycles (each `lw`'s extra M cycle becomes one more interlock cycle on the following `bne`, so `sw` commits at cycle 18). Note, original question stated "if all memory accesses," in which case the `sw` also takes an extra cycle: $\frac{16+3-5+1}{8} = \frac{15}{8} = 1.875$.

- What is the CPI assuming branches are always predicted correctly?

  2 bubbles come from the misprediction, so the last commit moves from cycle 16 to cycle 14: $\frac{16-5+1-2}{8} = \frac{10}{8} = 1.25$
  Sanity check: ideal + remaining bubbles $= 8 + 2 = 10$.

4 Exceptions

Exceptions: unexpected events that need to jump to an OS handler.
Sources: page faults, illegal instructions (e.g. divide by 0), timer/IO, etc.
Precise exception: all instructions before the faulting one have completed/committed, and nothing after it has.

4.1 Why are precise exceptions useful?

(a) Restartable: fix the fault, then resume

(b) Deterministic debugging: you know exactly which instruction faulted

(c) Clean restartability without exposing microarchitectural state

4.2 Why might one not want to always implement precise exceptions?

(a) Introduces hardware complexity

(b) Not always needed: some faults just kill the program (the application does not need to or cannot recover)