

15. Basic Processor Design – Finishing Pipelining and Performance

EECS 370 – Introduction to Computer Organization – Winter 2023

**EECS Department
University of Michigan in Ann Arbor, USA**

Review: Control hazards

beq	1	1	10
add	3	4	5

time



beq fetch decode execute memory writeback

add fetch decode execute

Review: Approaches to handling control hazards

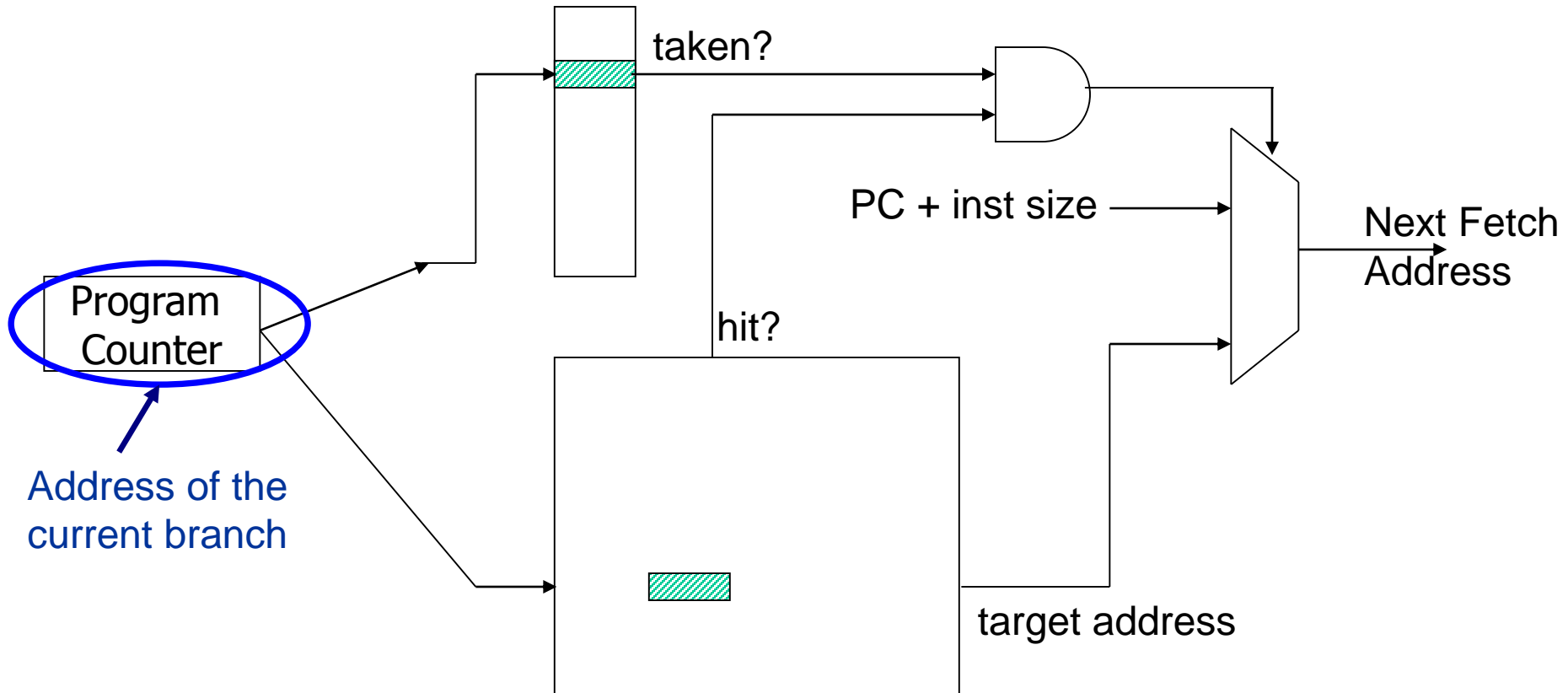
- ❑ Avoid
 - Make sure there are no hazards in the code.
- ❑ Detect and stall
 - Delay fetch until branch resolved.
- ❑ **Speculate and Squash-if-Wrong aka Branch Prediction**
 - Go ahead and fetch more instructions in case it is correct, but stop them if they shouldn't have been executed.

Review: Branch Prediction

- Predict the next fetch address (to be used in the next cycle)
- Requires three things to be predicted at fetch stage:
 - Whether the fetched instruction is a branch
 - Branch direction (if conditional)
 - Branch target address (if direction is taken)
- Observation: Target address remains the same for a conditional direct branch across dynamic instances
 - Store the target address from previous instance and access it with the PC
 - Called Branch Target Buffer (BTB) or Branch Target Address Cache

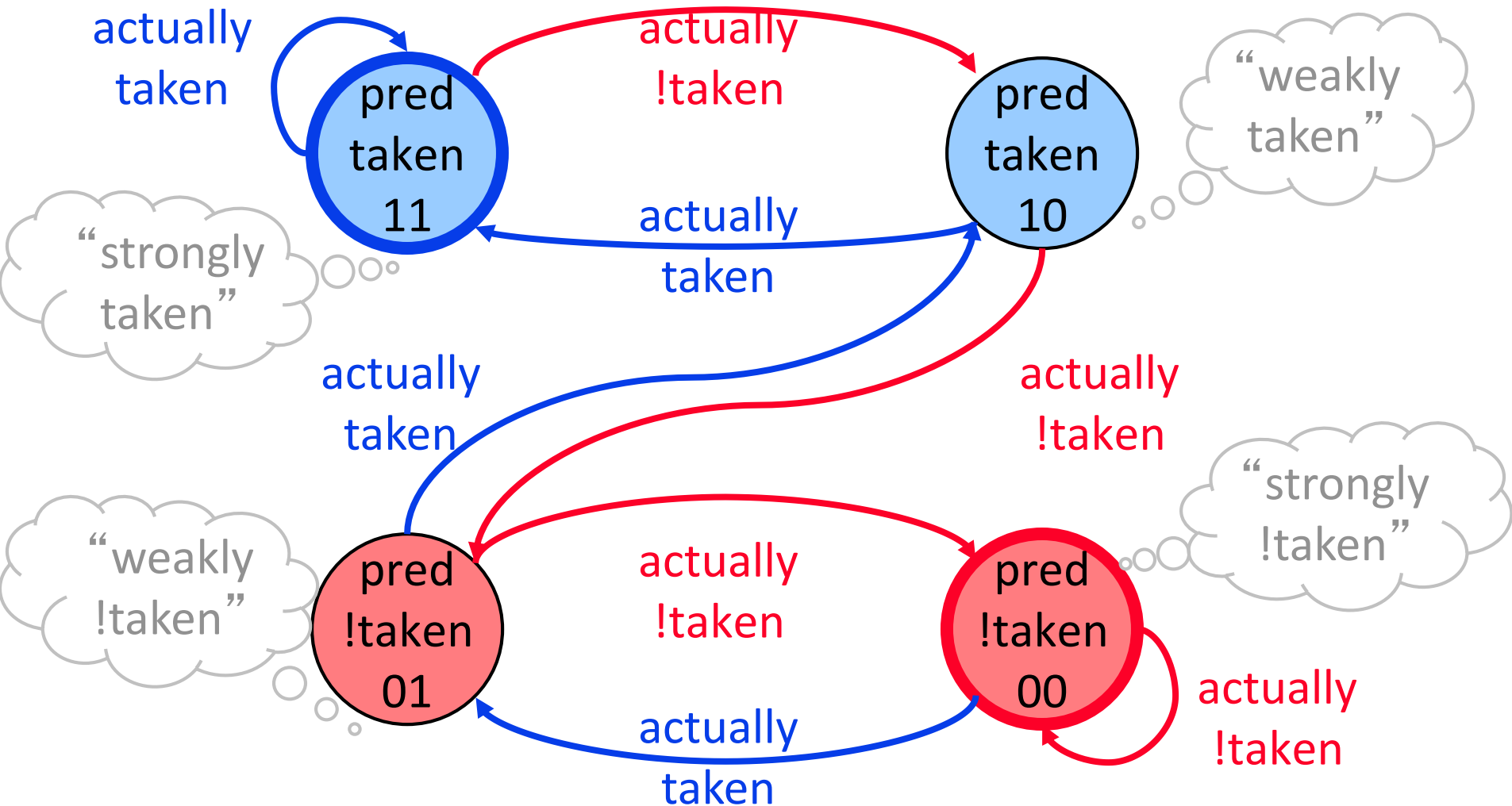
Review: Fetch Stage with Branch Prediction

Direction predictor (2-bit counters)



Cache of Target Addresses (BTB: Branch Target Buffer)

Review: State Machine for 2-bit Saturating Counter



Branch prediction

- ❑ Predict not taken: ~50% accurate.
- ❑ Predict backward taken: ~65% accurate.
- ❑ Predict same as last time: ~80% accurate.

- ❑ Pentium: ~85% accurate.
- ❑ Pentium Pro: ~92% accurate.
- ❑ Best paper designs: ~96% accurate.

What can go wrong?

- ❑ **Data hazards:** since register reads occur in stage 2 and register writes occur in stage 5 it is possible to read the wrong value if is about to be written.
- ❑ **Control hazards:** A branch instruction may change the PC, but not until stage 4. What do we fetch before that?
- ❑ **Exceptions:** How do you handle exceptions in a pipelined processor with 5 instructions in flight?

Exceptions

- ❑ Exception: when something unexpected happens during program execution.
 - Example: divide by zero.
 - The situation is more complex than the hardware can handle
 - So the hardware branches to a function, an “exception handler” which is code to try to deal with the problem.
- ❑ The exact way to set up such an exception handler will vary by ISA.
 - With C on x86 you would use `<signal.h>` functions to handle the “SIGFPE” signal.
 - There is a pretty good [Hackaday article](#) on this if you want to learn more.

Exceptions and Pipelining

- ❑ The hardware branches to the “exception handler”
 - This means that any instruction which can “throw” an exception *could* be a branch.
 - Throwing an exception should be rare (“exceptional”)

- ❑ So we would treat it much like a branch we predicted as “not taken”
 - Squash instructions behind it and then branch
 - It will introduce stalls, but since it should be rare, we don’t worry about it.
 - “Make the common case fast”.

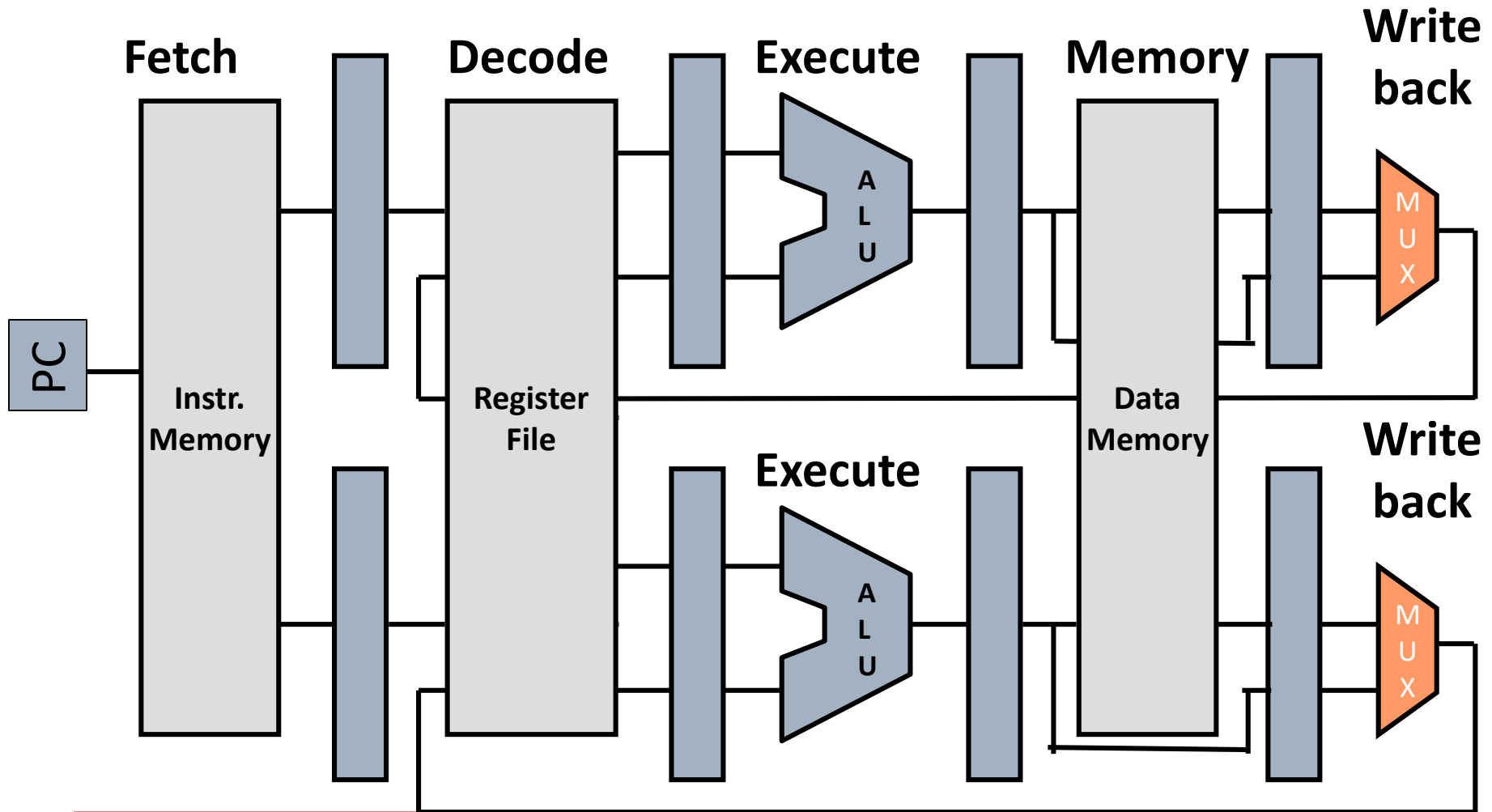
Building with Pipelines

- ❑ CPI for pipelining:
 - 1 (ideal case - no stalls)
 - > 1 (reality, depends on program)
- ❑ What if we want to improve performance more?
 - Want CPI as low as possible – lower than 1
- ❑ Use Parallelism
 - Instruction Level Parallelism (ILP) – Within task
 - Thread Level Parallelism (TLP) – Having many tasks
 - Data Level Parallelism (DLP) – Many tasks with same instructions

Creating more pipelines

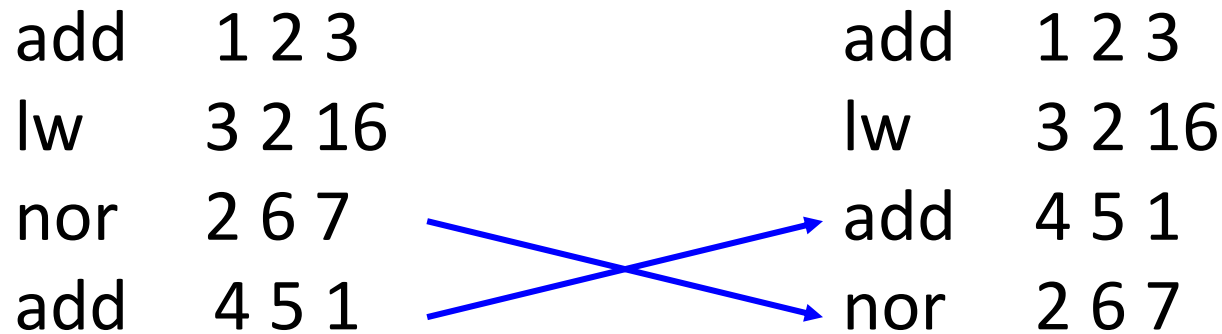
- ❑ Instruction Level Parallelism – Superscalar Pipeline
 - Have two or more pipelines in same processor
 - pipelines need to work in tandem to improve single program performance
- ❑ Thread Level Parallelism – Multi-core
 - Have two or more processors (Independent Pipelines)
 - Need more programs or a parallel program
 - does not improve single program performance
- ❑ Data Level Parallelism – Single Inst. Multiple Data (SIMD)
 - Have two or more execution pipelines (ID->WB)
 - Share the same fetch and control pipeline to save power (IF+cont.)
 - Similar to GPU's

ILP Techniques: Superscalar



Other Techniques for ILP: Out of Order Execution

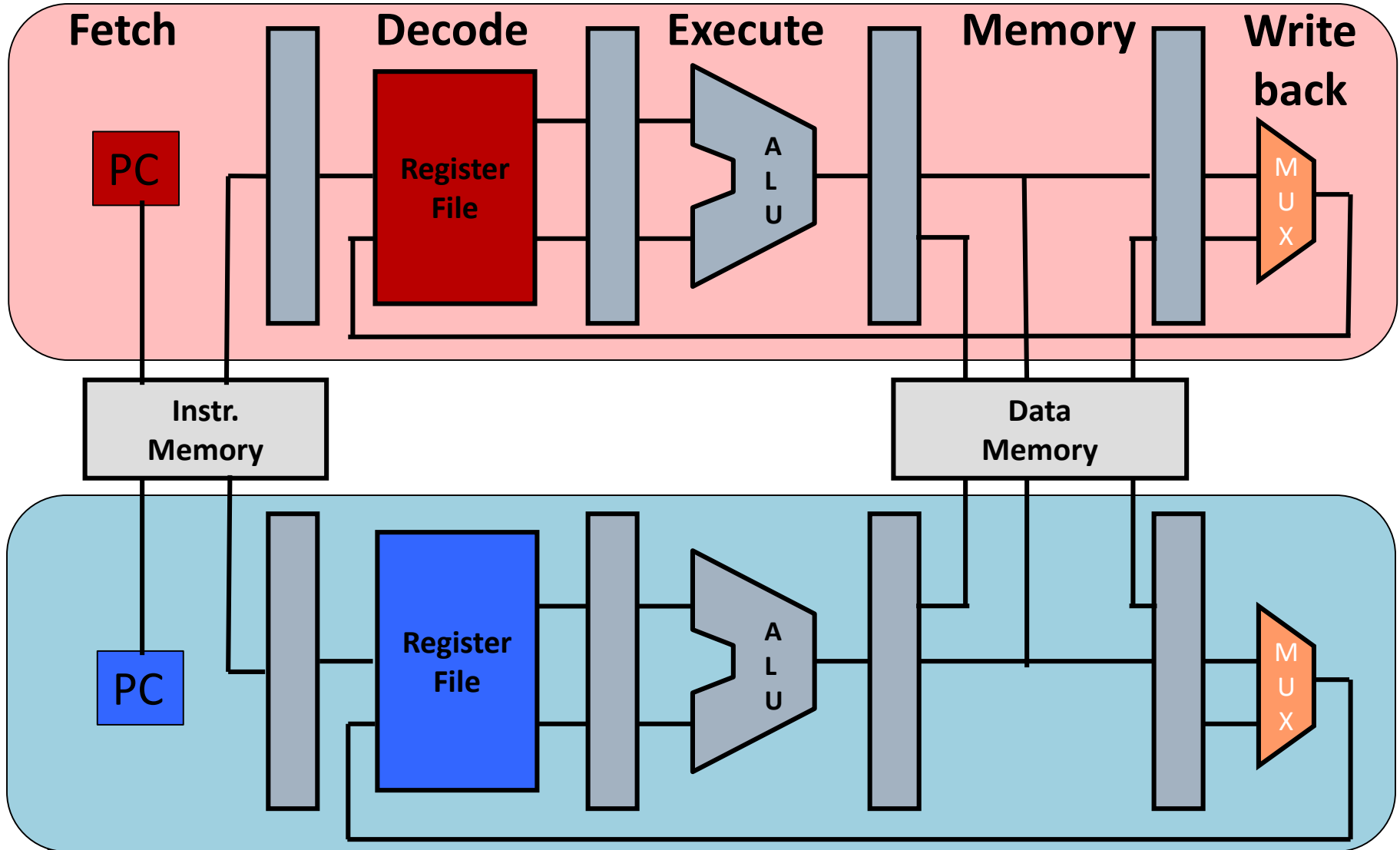
- ❑ Eliminating stall conditions decreases CPI
- ❑ Reorder instructions to avoid stalls
- ❑ Example (5-stage LC2K pipeline):



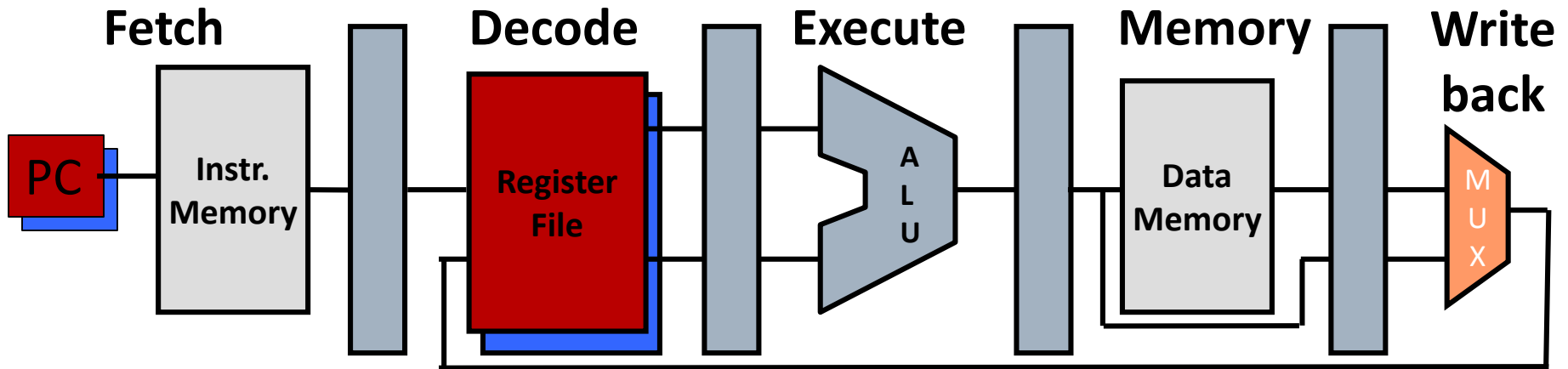
Why Use Out of Order Execution?

- ❑ Some instructions take a long time to execute
 - Floating point operations
 - Some loads and stores (more when we talk about memory hierarchy)
- ❑ Options:
 - Increase cycle time
 - Increase number of pipeline stages
 - Execute other instructions while you wait

TLP Techniques: Multiprocessors

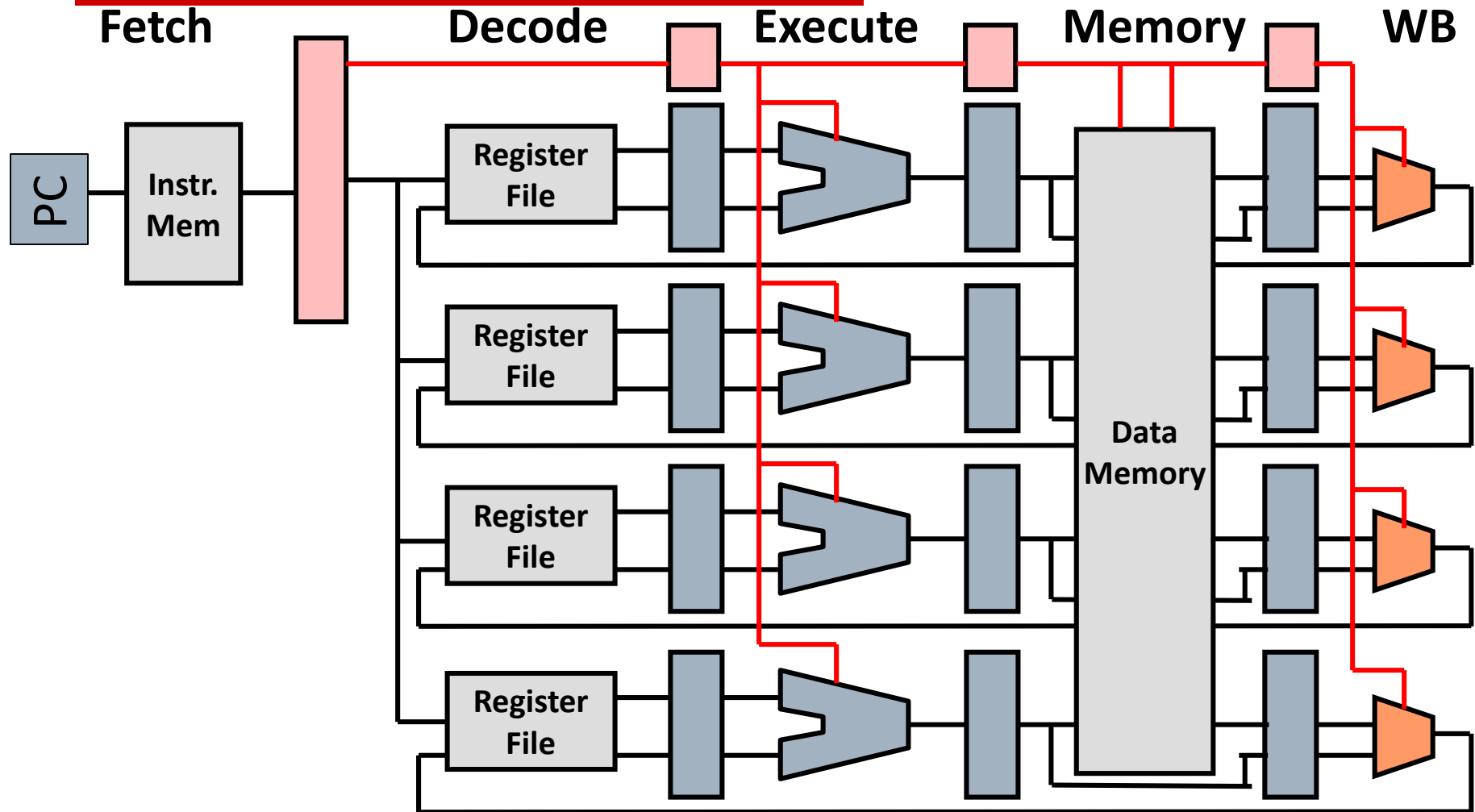


Other Techniques for TLP: Multi-Threading

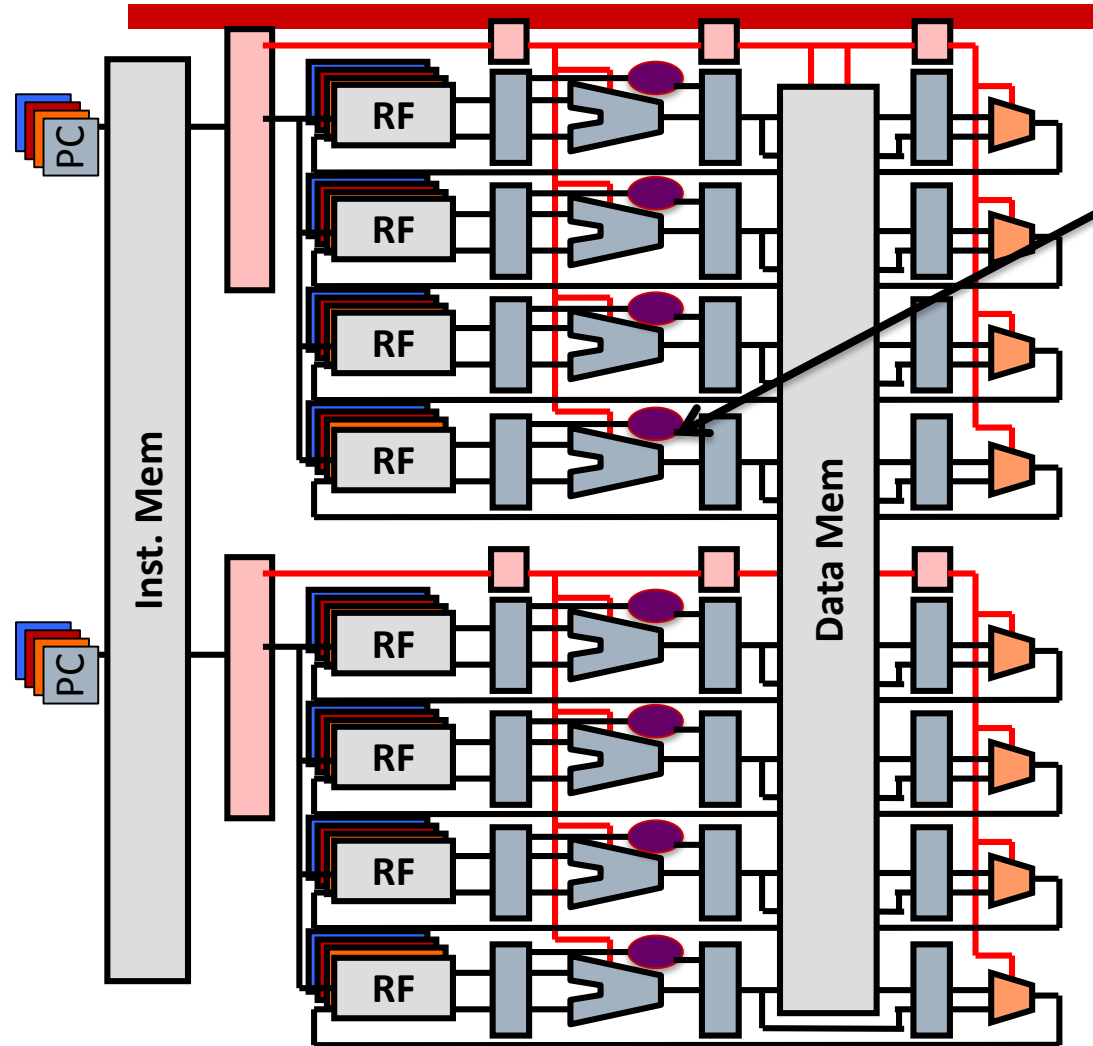


- ❑ Virtual Multiprocessor (Multi-Threading or HyperThreading)
 - Duplicate the state (PC, Registers) but time share hardware
 - User/Operating system see 2 cores, but only one execution
 - Used to hide long latencies (i.e. memory access to disk)

DLP Techniques: Single Instr. Multiple Data (SIMD)



Building a GPU



- ❑ Add special functional units in EX
- ❑ Combine Techniques
 - SIMD + MP + MT = SIMT
- ❑ MT used to hide memory latencies
- ❑ SIMD used to decrease power of fetch/control
- ❑ MP used to improve throughput

LC2k Pipeline Summary

Fetch	Decode	Execute	Memory	WB
PC read		Need register values	Branches resolved	Register values produced

- ❑ Data hazards
 - ❑ Hazard exists if producer-consumer of a register within a 2 instruction window
 - ❑ Note for project, the window is 3 instructions
 - ❑ Detect and stall – insert enough noops to separate producer and consumer by > 2 instructions (> 3 instructions for project)
 - ❑ Detect and forward
 - ❑ Handles all cases except LW-USE, need 1 noop here
- ❑ Control hazards
 - ❑ Detect and stall – needs 3 noops inserted after each branch
 - ❑ Predict and squash
 - ❑ Zero noops if predict correctly
 - ❑ 3 if predict incorrectly

Review: basic performance equation

- ❑ Execution time (Time/Program) =
 - # of instr (I/P) \times CPI (C/I) \times cycle time (T/C)
- ❑ Multi-cycle decreases cycle time, but increases CPI.
- ❑ Pipelining decreases CPI
 - Approaches 1.0 if no stalls (hazards that are fixed by stalling).

Calculating performance with no stalls

How many cycles does this code take to execute?

add	1	2	3
nor	1	4	5
add	4	6	7

What value is written to the ALU result field of the Mem/WB pipeline register at the end of cycle 5.

Calculating performance with no stalls

How many cycles does this code take to execute?

No stalls – Final WB @ cycle 7

add	1	2	3
nor	1	4	5
add	4	6	7

What value is written to the ALU result field of the Mem/WB pipeline register at the end of cycle 5.

nor result

Calculating performance with data hazards (detect and stall)

How many data hazards are there
in this code?

```
add    1 2 3
nor    3 4 5
add    3 5 6
```

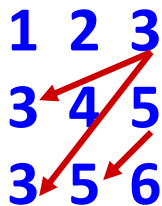
How many stall cycles if we use
detect and stall to handle the
hazards?

Time:	1	2	3	4	5	6	7	8	9	10	11
add 1 2 3	IF	ID	EX	ME	WB						
nand 3 4 5											
add 3 5 6											

Calculating performance with data hazards (detect and stall)

How many data hazards are there
in this code?

add 1 2 3
nor 3 4 5
add 3 5 6



3 data hazards

How many stall cycles if we use
detect and stall to handle the
hazards?

Time:	1	2	3	4	5	6	7	8	9	10	11
add 1 2 3	IF	ID	EX	ME	WB						
nor 3 4 5		IF	ID*	ID*	ID	EX	ME	WB			
add 3 5 6			IF*	IF*	IF	ID*	ID*	ID	EX	ME	WB

Stall : 4 cycles

Total : 11 cycles

Calculating performance with data hazards (detect and forward)

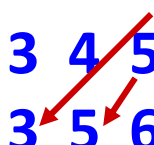
add	1	2	3
nor	3	4	5
add	3	5	6
lw	3	6	7
add	6	6	1

Where do the values for the second add instruction come from?

How many stall cycles on the LC2K pipelined datapath with data forwarding from lecture?

Calculating performance with data hazards (detect and forward)

add	1	2	3
nor	3	4	5
add	3	5	6
lw	3	6	7
add	6	6	1



Where do the values for the second add instruction come from?

From Mem/WB and EX/Mem

How many stall cycles on the LC2K pipelined datapath with data forwarding from lecture?

1 stall for lw → add

Calculating performance with control hazards (speculate and squash)

- ❑ How many cycles are saved if you perform speculate and squash for the following code (assume that branches are predicted to be not taken)?

```
add    1 2 3
beq    1 5 1
nor    6 4 1
add    3 4 5
```

- ❑ Assume the branch is taken: How many cycles to execute this code?
- ❑ Assume the branch is not taken: How many cycles execute this code?

Calculating performance with control hazards (speculate and squash)

- How many cycles are saved if you perform speculate and squash for the following code (assume that branches are predicted to be not taken)?

```
add    1 2 3
beq    1 5 1
nor    6 4 1
add    3 4 5
```

- Assume the branch is taken: How many cycles to execute this code?

3 instr + 3 stalls + 4 to empty pipe = 10 cycles

- Assume the branch is not taken: How many cycles execute this code?

4 instr + 4 to empty pipe = 8 cycles

Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

1. How many cycles does the code take assuming detect and stall for control hazards?

add	1	2	3
beq	1	5	1
lw	6	4	1
add	3	4	5
beq	5	7	-5
halt			

Assume halt is resolved in WB stage

Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

1. How many cycles does the code take assuming detect and stall for control hazards?

add	1	2	3
beq	1	5	1
lw	6	4	1
add	3	4	5
beq	5	7	-5
halt			

Instructions = $100 * (0.5 * 5 + 0.5 * 4) + 1 = 451$

Time = 451 + load stalls + branch stalls + empty pipe

Time = $451 + 100 * 0.5 * 1 +$

Time = $451 + 100 * 0.5 * 1 + (100 * 3 + 100 * 3) + 4$

Time = 1105

Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

2. How many cycles does the code take assuming speculate and squash where all branches are predicted not taken?

add	1	2	3
beq	1	5	1
lw	6	4	1
add	3	4	5
beq	5	7	-5
halt			

Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

2. How many cycles does the code take assuming speculate and squash where all branches are predicted not taken?

add	1	2	3
beq	1	5	1
lw	6	4	1
add	3	4	5
beq	5	7	-5
halt			

Instructions = $100 * (0.5 * 5 + 0.5 * 4) + 1 = 451$

Time = 451 + load stalls + branch stalls + empty pipe

Time = $451 + 100 * 0.5 * 1 + (100 * 0.5 * 3 + 99 * 3) + 4$

Time = 952

Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

3. How many cycles does the code take assuming speculate and squash where backward branches are predicted taken and forward branches not taken?

add	1	2	3
beq	1	5	1
lw	6	4	1
add	3	4	5
beq	5	7	-5
halt			

Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

3. How many cycles does the code take assuming speculate and squash where backward branches are predicted taken and forward branches not taken, and BTB has all entries to start?

$$\# \text{ Instructions} = 100 * (0.5 * 5 + 0.5 * 4) + 1 = 451$$

$$\text{Time} = 451 + \text{load stalls} + \text{branch stalls} + \text{empty pipe}$$

$$\text{Time} = 451 + 100 * 0.5 * 1 + (100 * 0.5 * 3 + 1 * 3) + 4$$

$$\text{Time} = 658$$

add	1	2	3
beq	1	5	1
lw	6	4	1
add	3	4	5
beq	5	7	-5
halt			

Calculating performance with control hazards

Assume the first branch has the pattern **TTTN** that repeats, and the loop is iterated 100 times

4. How many cycles does the code take if a 2-bit counter BTB is used to predict each branch, how many cycles does the code take? Assume initial state of branch predictor counter is “10” (WT)

add	1	2	3
beq	1	5	1
lw	6	4	1
add	3	4	5
beq	5	7	-5
halt			

Calculating performance with control hazards

Assume the first branch has the pattern **TTTN** that repeats, and the loop is iterated 100 times

4. How many cycles does the code take if a 2-bit counter BTB is used to predict each branch, how many cycles does the code take? Assume initial state of branch predictor counter is “10” (WT)

add	1 2 3
beq	1 5 1
lw	6 4 1
add	3 4 5
beq	5 7 -5
halt	

$\checkmark \checkmark \checkmark \times \checkmark \checkmark \checkmark \times \checkmark \checkmark \checkmark \times$
beq 1 T T T N T T T N T T T N ...

beq 2 is correct 99 times, then incorrect last iter

Instructions = $100 * (0.25 * 5 + 0.75 * 4) + 1 = 426$

Time = 426 + load stalls + branch stalls + empty pipe

Time = $426 + 100 * 0.25 * 1 + 100 * 0.25 * 3 + 1 * 3 + 4$

Time = 533

Classic performance problem

- ❑ Program with following instruction breakdown:

lw	10%
sw	15%
beq	25%
R-type	50%
- ❑ Speculate “always not-taken” and squash. 80% of branches not-taken
- ❑ Full forwarding to execute stage. 20% of loads stall for 1 cycle
- ❑ What is the CPI of the program?
- ❑ What is the total execution time if cycle time is 100MHz?

Classic performance problem

- ❑ Program with following instruction breakdown:

lw	10%
sw	15%
beq	25%
R-type	50%
- ❑ Speculate “always not-taken” and squash. 80% of branches not-taken
- ❑ Full forwarding to execute stage. 20% of loads stall for 1 cycle
- ❑ What is the CPI of the program?
- ❑ What is the total execution time if cycle time is 100MHz?

$$\text{CPI} = 1 + 0.10 (\text{loads}) * 0.20 (\text{load use stall}) * 1 \\ + 0.25 (\text{branch}) * 0.20 (\text{miss rate}) * 3$$

$$\text{CPI} = 1 + 0.02 + 0.15 = 1.17$$

$$\text{Time} = 1.17 * 10\text{ns} = 11.7\text{ns per instruction}$$

Classic performance problem (cont.)

- ❑ Assume branches are resolved at Execute?
 - What is the CPI?
 - What happens to cycle time?
 - What is the total execution time?

Classic performance problem (cont.)

- ❑ Assume branches are resolved at Execute?
 - What is the CPI?
 - What happens to cycle time?

$$\text{CPI} = 1 + 0.10 (\text{loads}) * 0.20 (\text{load use stall}) * 1 \\ + 0.25 (\text{branch}) * 0.20 (\text{miss rate}) * 2$$

$$\text{CPI} = 1 + 0.02 + 0.1 = 1.12$$

Performance with deeper pipelines

- ❑ Assume the setup of the previous problem.
- ❑ What if we have a 10 stage pipeline?
 - Instructions are fetched at stage 1.
 - Register file is read at stage 3.
 - Execution begins at stage 5.
 - Branches are resolved at stage 7.
 - Memory access is complete in stage 9.
- ❑ What's the CPI of the program?
- ❑ If the clock rate was doubled by doubling the pipeline depth, is performance also doubled?

Performance with deeper pipelines

- ❑ Assume the setup of the previous problem.
- ❑ What if we have a 10 stage pipeline?
 - Instructions are fetched at stage 1.
 - Register file is read at stage 3.
 - Execution begins at stage 5.
 - Branches are resolved at stage 7.
 - Memory access is complete in stage 9.
- ❑ What's the CPI of the program?
- ❑ If the clock rate was doubled by doubling the pipeline depth, is performance also doubled?

$$\text{CPI} = 1 + 0.10 (\text{loads}) * 0.20 (\text{load use stall}) * 4 + 0.25 (\text{branch}) * 0.20 (\text{N stalls}) * 6$$

$$\text{CPI} = 1 + 0.08 + 0.30 = 1.38$$

$$\text{Time} = 1.38 * 5\text{ns} = 6.9 \text{ ns per instruction}$$