19. Cache wrapup & Mordern Processors

EECS 370 – Introduction to Computer Organization - Winter 2016

Profs. Valeria Bertacco & Reetu Das

EECS Department
University of Michigan in Ann Arbor, USA

Recap: 3 C's of cache misses

- First reference to an address
  - **Compulsory** miss
    - Also sometimes called a **"cold start"** miss
    - First reference to any block will always miss
- Cache is too small to hold all the data
  - **Capacity** miss
    - Would have had a hit with a large enough cache
- Replaced it from a busy set
  - **Conflict** miss
    - Would have had a hit with a fully associative cache

Recap: Classifying cache misses

- Can you classify all cache misses?
  - Compulsory miss?
  - Capacity miss?
  - Conflict miss?
- Yes! (with a cache simulator)
  - Simulate with a cache of unlimited size (cache size = memory size) - Any misses must be compulsory misses
  - Simulate again with a fully associative cache of the intended size - Any new misses must be capacity misses
  - Simulate a third time, with the actual intended cache - Any new misses must be conflict misses

How did we get to 3C's formulas?

1. Compulsory – Misses (Infinite cache)
   - First references to a cache block
2. Capacity – Misses(Fully Associative) – Compulsory
   - Fully associative cache of same size as the original cache given
3. Conflict – Misses(Given Cache) – Misses(Fully Associative)

3 C's Practice Problem

Consider a cache with the following configuration: write-allocate, total size is 16 bytes, block size is 4 bytes, and 2-way associative. The memory address size is 16 bits and byte-addressable. The replacement policy is LRU. The cache is empty at the start.

For the following memory accesses, indicate whether the reference is a hit or miss, and the type of a miss (compulsory, conflict, capacity)

3 C's Practice Problem – Compulsory Misses

Address	Type
0x2A	Load
0x23	Load
0x14	Load
0x29	Store
0x18	Store
0x21	Load
0x2B	Store
0x1C	Load
0x2D	Store
0x17	Store
0x19	Store
0x1D	Load

3 C's Practice Problem – Fully Associative Cache

Address	Type
0x2A	Load
0x23	Load
0x14	Load
0x29	Store
0x18	Store
0x21	Load
0x2B	Store
0x1C	Load
0x2D	Store
0x17	Store
0x19	Store
0x1D	Load

3 C's Practice Problem – Given Set Associative Cache

Address	Type
0x2A	Load
0x23	Load
0x14	Load
0x29	Store
0x18	Store
0x21	Load
0x2B	Store
0x1C	Load
0x2D	Store
0x17	Store
0x19	Store
0x1D	Load

3 C's Practice Problem – Classify Misses

Address	Type	Infinite	FA	SA
0x2A	Load	M	M	M
0x23	Load	M	M	M
0x14	Load	M	M	M
0x29	Store	H	H	H
0x18	Store	M	M	M
0x21	Load	H	H	M
0x2B	Store	H	H	M
0x1C	Load	M	M	M
0x2D	Store	M	M	M
0x17	Store	H	M	M
0x19	Store	H	M	M
0x1D	Load	H	H	M

3 C's Practice Problem

Consider a cache with the following configuration: write-allocate, total size is 16 bytes, block size is 4 bytes, and 2-way associative. The memory address size is 16 bits and byte-addressable. The replacement policy is LRU. The cache is empty at the start. For the following memory accesses, indicate whether the reference is a hit or miss, and the type of a miss (compulsory, conflict, capacity):

Compare the number of bytes read and written from/to the memory when the cache is **write- back verses write-through**, you **do not need to drain the write-back cache to memory** at the end of the memory access sequence.

3 C's Practice Problem – Stores

Address	Type
0x2A	Load
0x23	Load
0x14	Load
0x29	Store
0x18	Store
0x21	Load
0x2B	Store
0x1C	Load
0x2D	Store
0x17	Store
0x19	Store
0x1D	Load

Final Cache State

18, 19, 20, 21 - Dirty	Set 0
28, 29, 2A, 2B - Dirty	
1C, 1D, 1E, 1F	Set 1
14, 15, 16, 17 - Dirty	

3 C's Practice Problem - Solution

Address	Type	Hit or Miss	Compulsory	Capacity	Conflict
0x2A	Load	Miss	X		
0x23	Load	Miss	X		
0x14	Load	Miss	X		
0x29	Store	Hit			
0x18	Store	Miss	X		
0x21	Load	Miss			X
0x2B	Store	Miss			X
0x1C	Load	Miss	X		
0x2D	Store	Miss	X		
0x17	Store	Miss		X	
0x19	Store	Miss		X	
0x1D	Load	Miss			X

Number of bytes read and written from/to memory when the cache is *write through:*

#Bytes read: 44

#Bytes written: 6

Number bytes read and written from/to memory when the cache is a *write back*:

#Bytes read: 44

#Bytes written: 12

Caches – Practice Problem 1

Given the following cache configuration:

Cache size = 16 bytes	Write allocate
Block size = 2 bytes	LRU replacement policy
Direct mapped	Address size = 16 bits, byte addressable
memory	

1. Given the address 0xF19C, what are the tag and index values (specify in hex)?

Answer: Tag = 0xF19, Set index = 0x6

Caches - Practice Problem 1

2. What is the average memory latency (measured in cycles) for the next reference stream if hits take 1 cycle and the miss penalty is 10 cycles (note: miss penalty excludes bringing the data from the cache to the register)?

4	7	21	22	23	20	5	7	36	6	4
M	M	M	M	H	H	H	H	M	H	H

Avg latency = ((#hits * hit_latency) + (#misses * miss_latency)) /#accesses
= ((6 * 1) + (5 * (10 + 1)) / 11
= 61 / 11 = 5.55

Caches – Practice Problem 1

3. Given that you want the reference stream hit/miss behavior to be the following, What is the minimum cache associativity which will result in this behavior (all other parameters, e.g. size, block size, LRU are unchanged)?

4	7	21	22	23	20	5	7	36	6	4
M	M	M	M	H	H	H	H	M	H	H

Answer: 2-way associative

Caches – Practice Problem 2

- Given a same 200 MHz processor with 8KB instruction and data caches, with memory access latency of 20 cycles. Both caches are 2-way associative. A programming running on this processor has a 95% icache hit rate and a 90% dcache hit rate. On average, 30% of the instructions are loads or stores. The CPI of this system, if caches were ideal would be 1. Clock cycle is 5ns.

  Suppose you have 2 options for the next generation processor, which do you pick?

  - Option 1: Double the clock frequency, this will increase your memory latency to 50 cycles. Assume a base CPI of 1 can still be achieved after this change.
  - Option 2: Double the size of your caches, this will increase the instruction cache hit rate to 98% and the data cache hit rate to 95%. Assume the hit latency is still 1 cycle.

Caches – Practice Problem 2

Answer to question on previous slide

Option 1 (double clock freq, base cycle time is 5 ns, so new cycle time is 2.5 ns)
CPI = baseCPI + IcacheStallCPI + DcacheStallCPI
CPI = 1.0 + 0.05*50 + 0.3*0.1*50 = 5.0
Execution time = 5.0 * Ninstrs * 2.5ns = 12.5 * Ninstrs

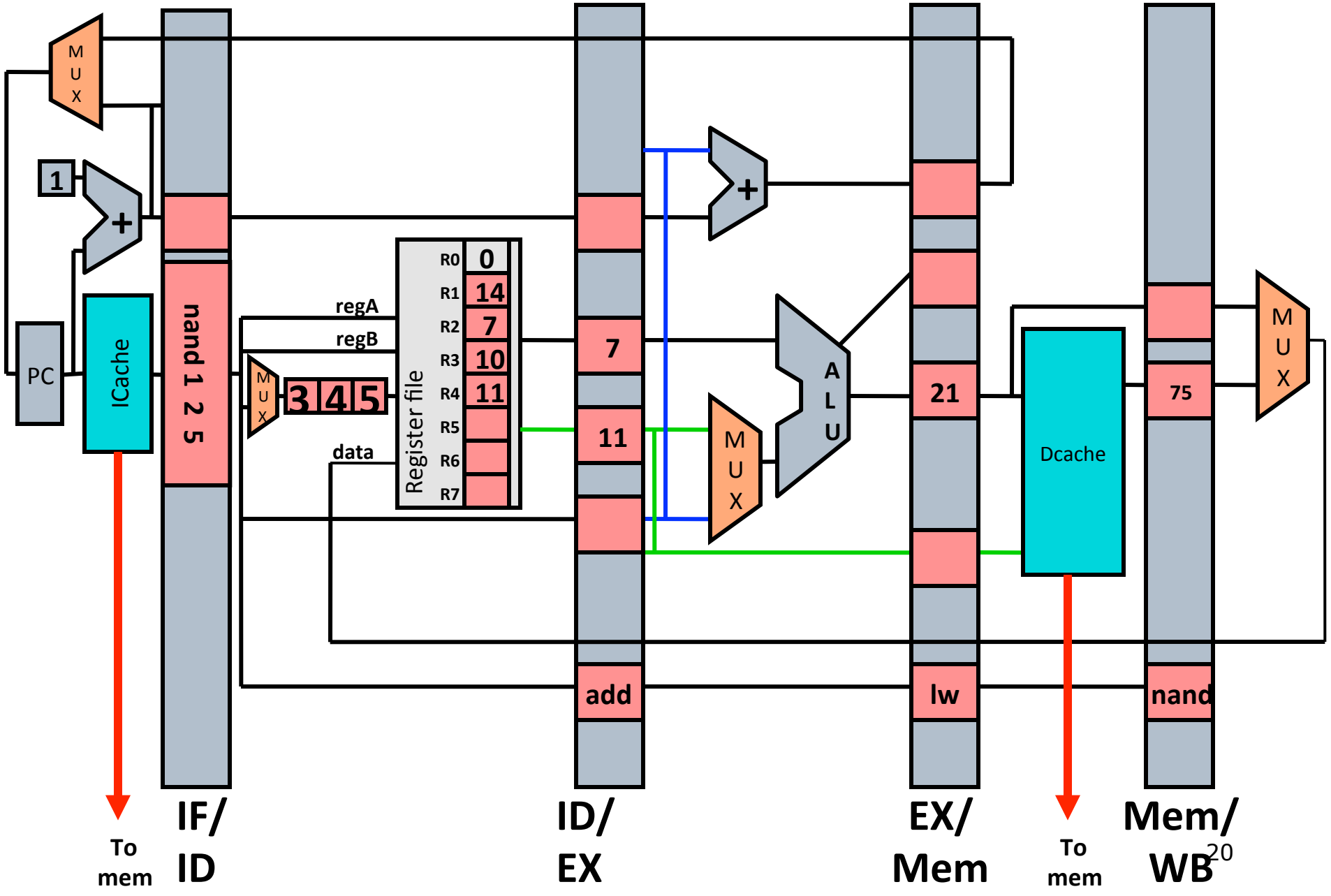
Option 2 (icache/dcache miss rates lowered to 2% and 5%)
CPI = baseCPI + IcacheStallCPI + DcacheStallCPI
CPI = 1.0 + 0.02*20 + 0.3*0.05*20 = 1.7
Execution time = 1.7 * Ninstrs * 5ns = 8.5 * Ninstrs

Therefore, Option 2 is the better choice

Recap: Integrating Caches into a Pipeline

- How are caches integrated into a pipelined implementation?
  - Replace instruction memory with Icache
  - Replace data memory with Dcache
- Issues
  - Memory accesses now have variable latency
  - Both caches may miss at the same time

LC2K Pipeline with Caches



Putting It Together & More on Today's Processors

- What's inside modern processors?
  - Deep memory hierarchy
  - Processors which can hide memory latency
  - More and more parallelism!
  - Lets take a peak

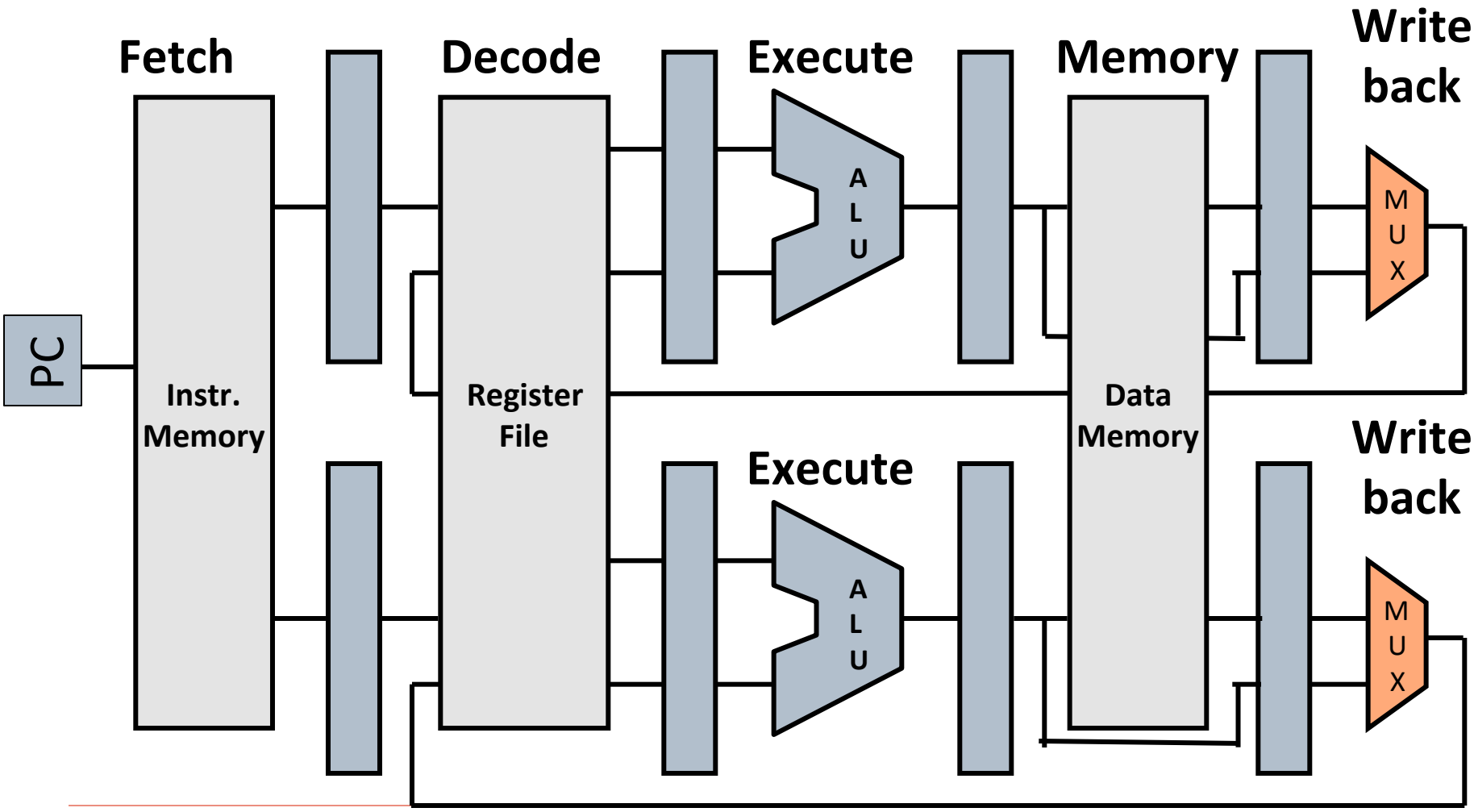
Building with Pipelines

- 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

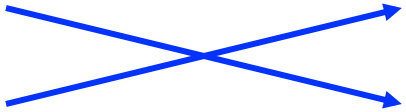
ILP Techniques: Superscalar



Other Techniques for ILP: Out of Order Execution

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

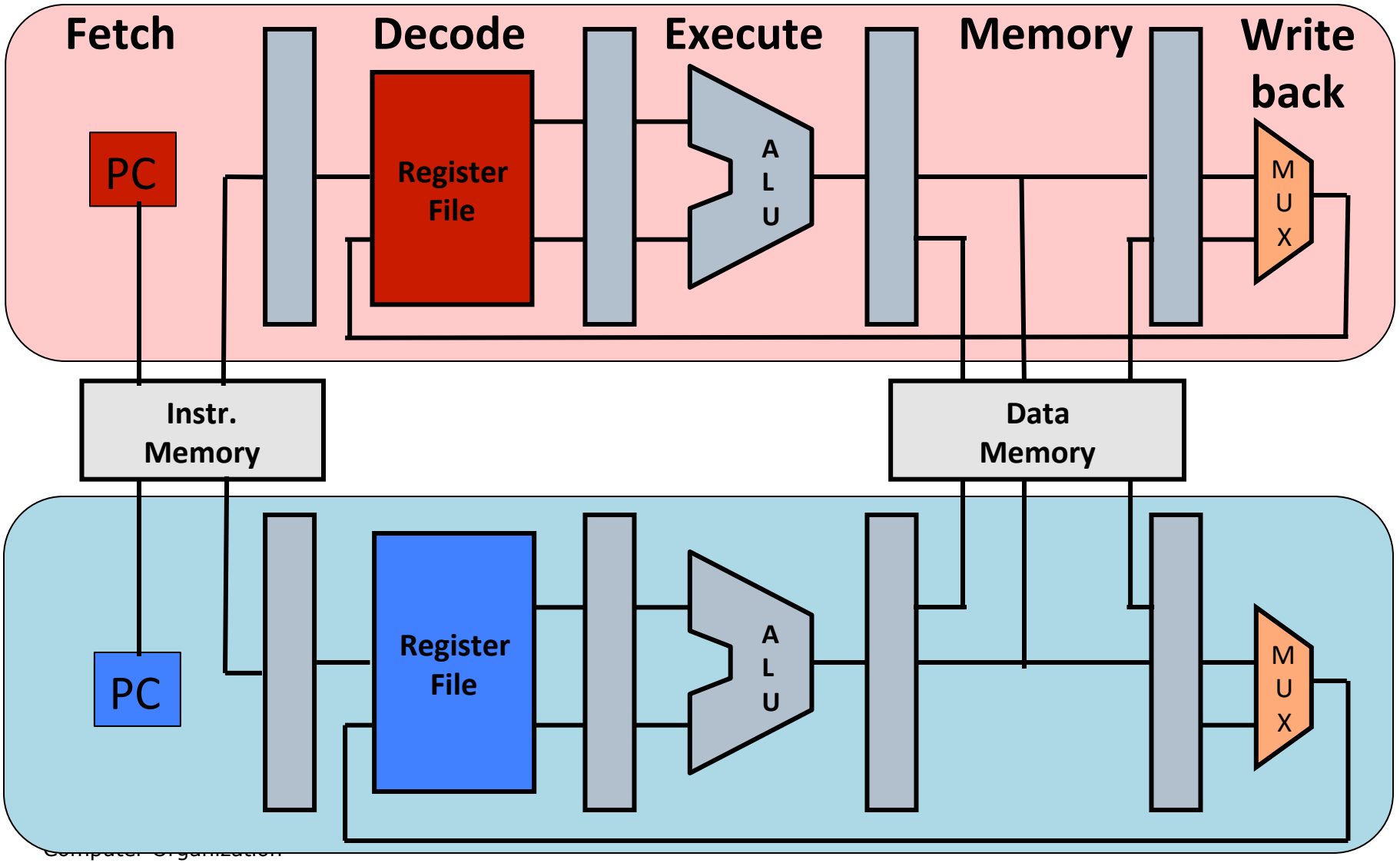
```
add   1 2 3          add   1 2 3
lw    3 2 16         lw    3 2 16
nor   2 6 7          add   4 5 1
add   4 5 1          nor   2 6 7
```



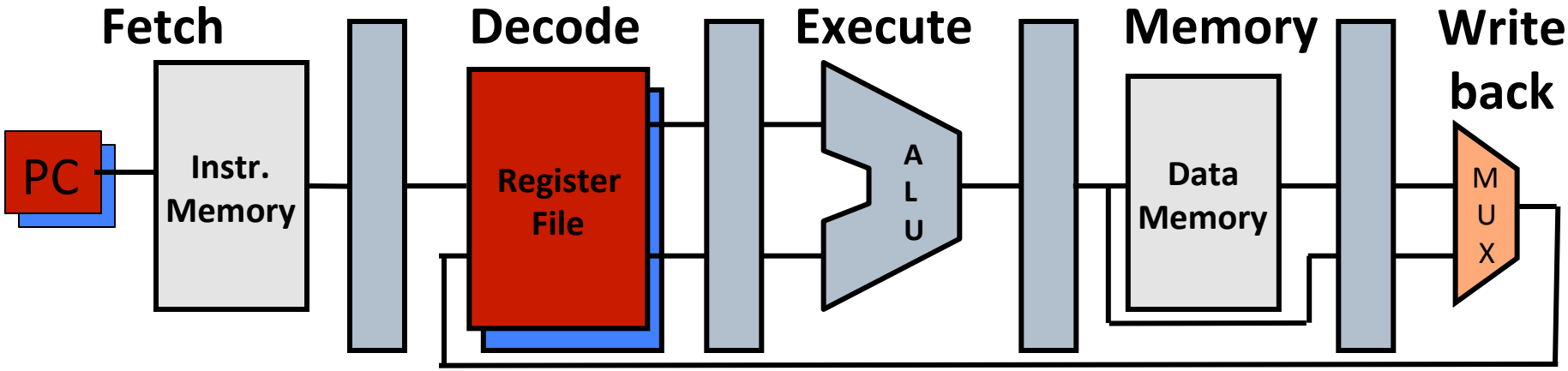
Why Use Out of Order Execution?

- 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

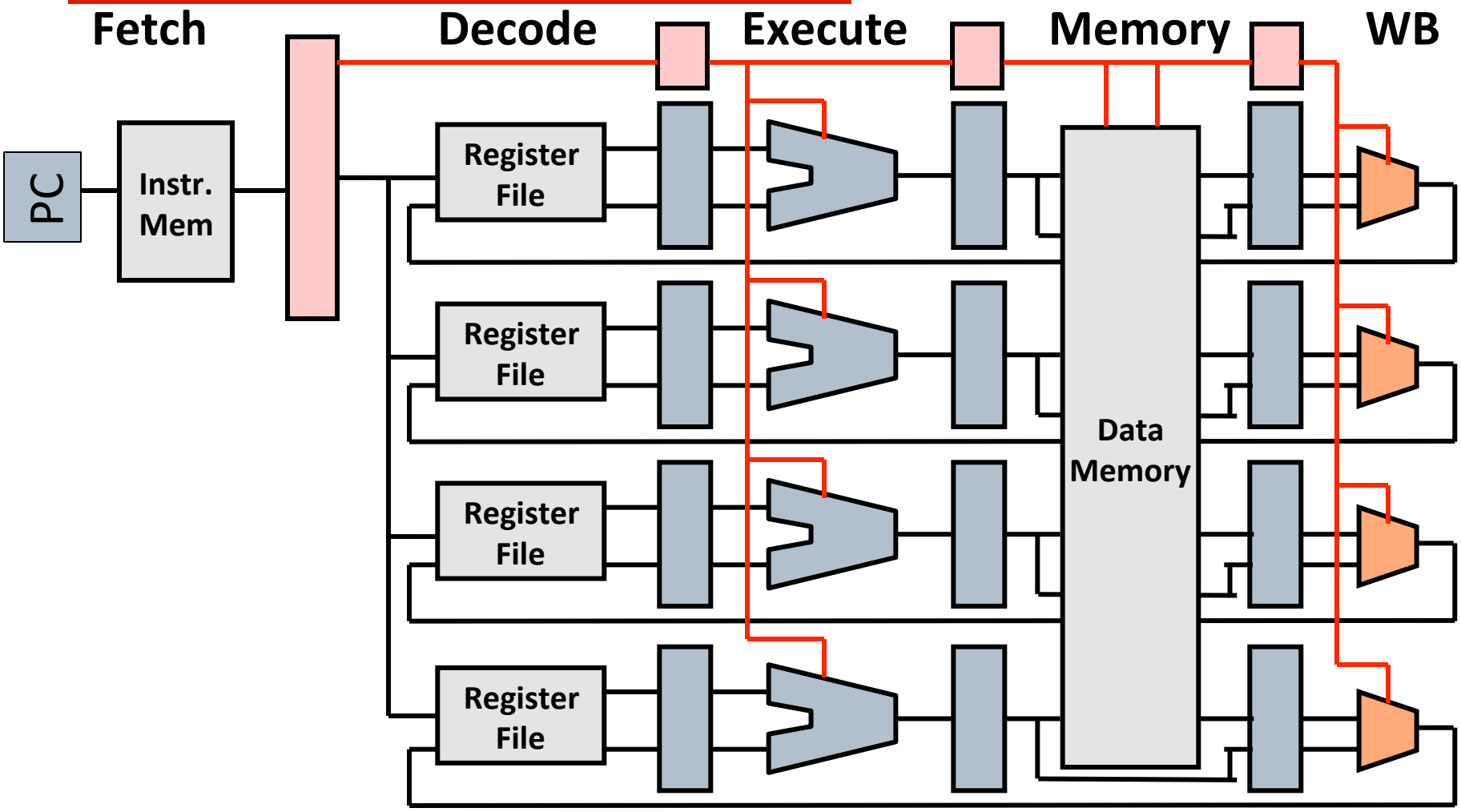


Other Techniques for TLP: Multi-Threading

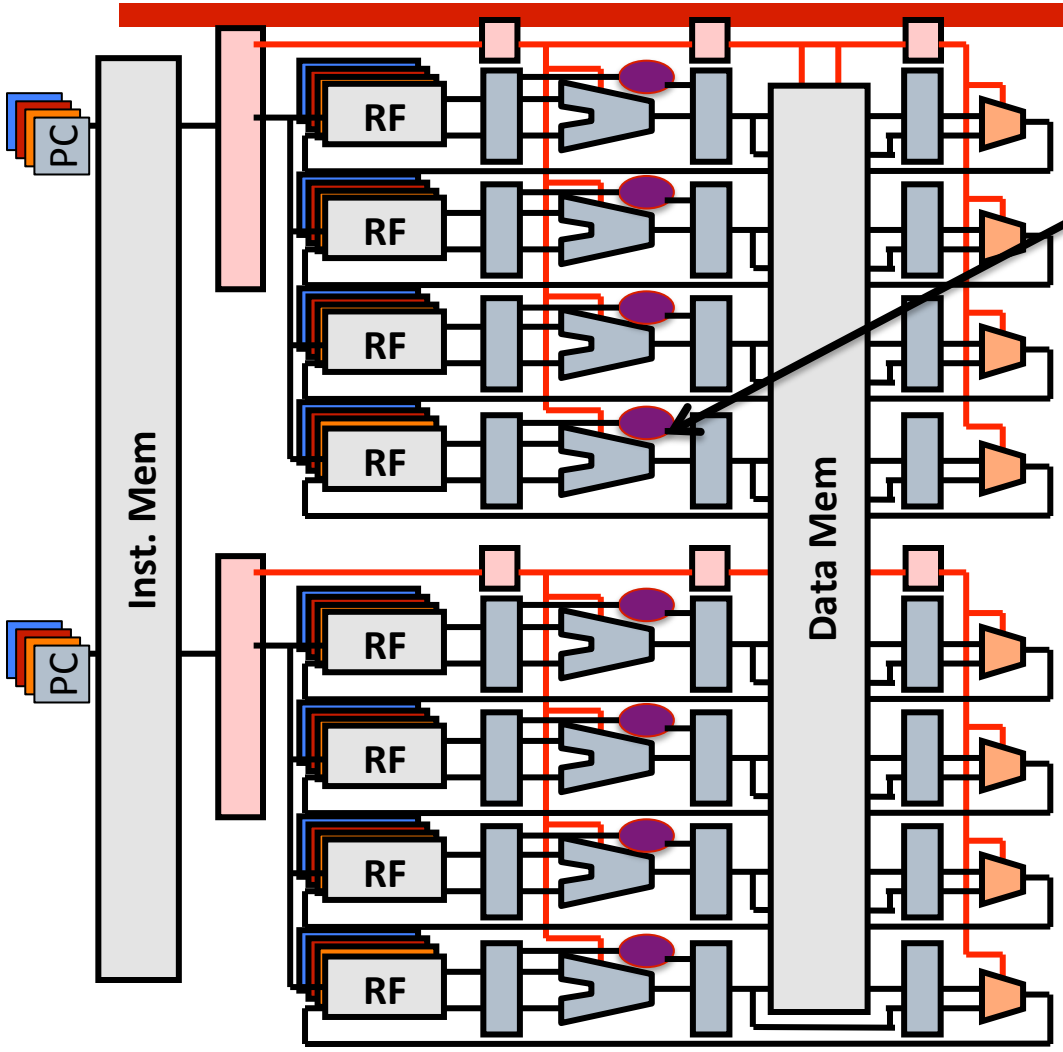


- Virtual Multiprocessor (Multi-Threading or HyperThreading)
  - Duplicate the state (PC, Registers) but time share hardware
  - User/Operating system see 2 cores, but only one execution
  - Used to hide long latencies (i.e. memory access to disk)

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