

Exercise Session 7

Cache coerency, Tomasulo, Scoreboard + Expl. Reg. Renaming

Advanced Computer Architectures

Politecnico di Milano

May 21st, 2025

Alessandro Verosimile <alessandro.verosimile@polimi.it>



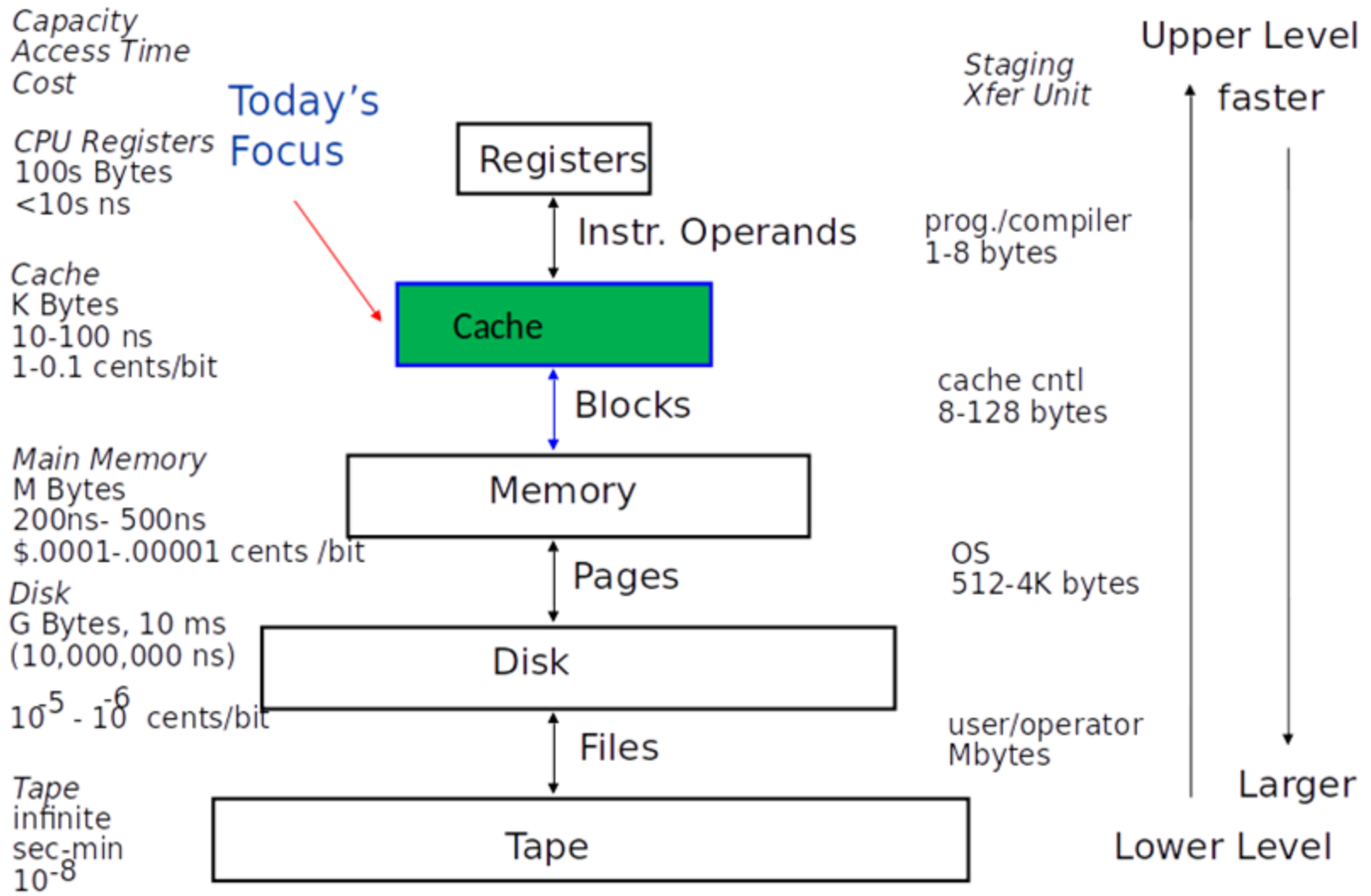
POLITECNICO
MILANO 1863

POLITECNICO MILANO 1863
NECST
laboratory

Recall: Locality Principles

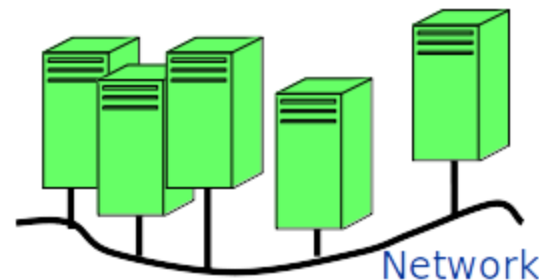
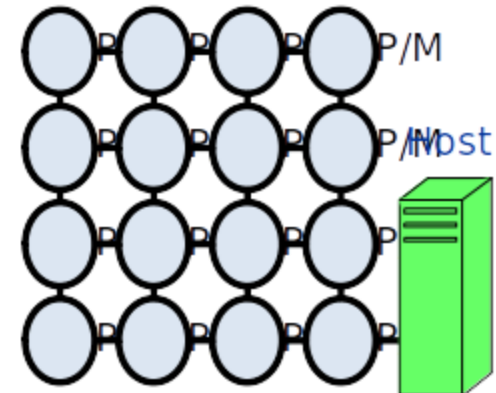
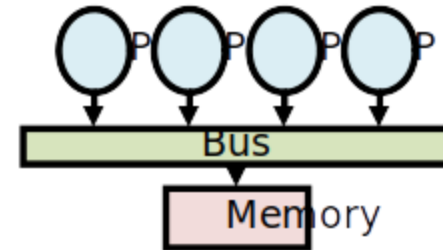
- Programs access a small proportion of their address space at any time
- **Temporal** locality
 - Items accessed recently are likely to be accessed again soon
 - e.g., instructions in a loop, induction variables
- **Spatial** locality
 - Items near those accessed recently are likely to be accessed soon
 - E.g., sequential instruction access, array data

Recall: Memory hierarchy



Recall: MIMD Machines

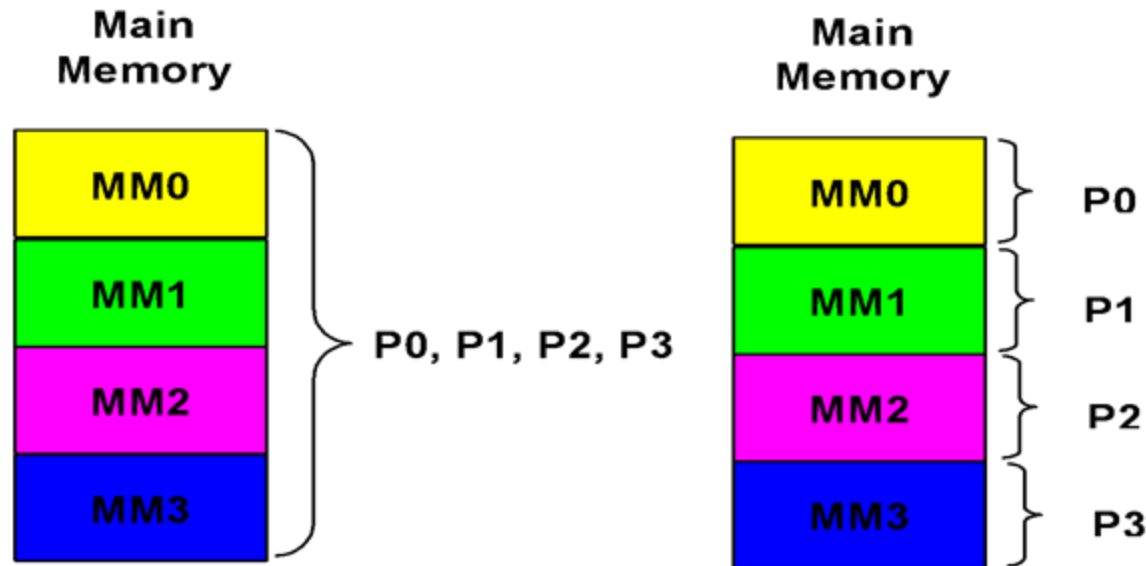
- **Symmetric Multiprocessor**
 - Multiple processors in box with shared memory communication
 - Current MultiCore chips like this
 - Every processor runs copy of OS
- **Non-uniform shared-memory** with separate I/O through host
 - Multiple processors
 - Each with local memory
 - general scalable network
 - Extremely light “OS” on node provides simple services
 - Scheduling/synchronization
 - Network-accessible host for I/O
- **Cluster**
 - Many independent machine connected with general network
 - Communication through messages



Recall: Memory Address Space Model

Single logically shared
address space

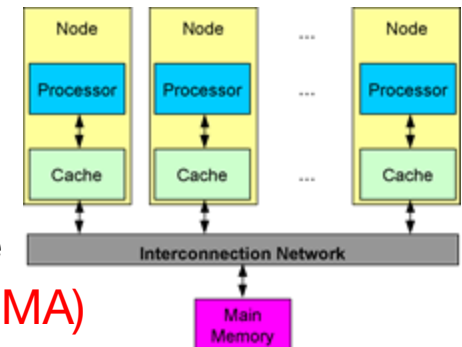
Multiple and private
address spaces



Recall: Physical Memory Organization

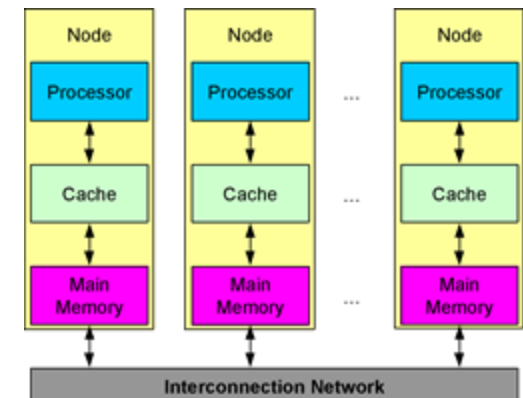
Centralized shared-memory architectures

- at most few dozen processor chips (< 100 cores)
- Large caches, single memory multiple banks
- Often called symmetric multiprocessors (SMP) and the style of architecture called **Uniform Memory Access (UMA)**



Distributed memory architectures

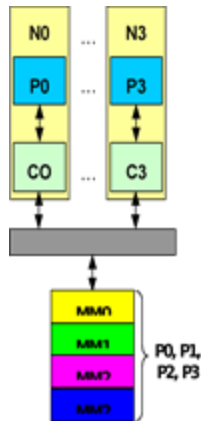
- To support large processor counts
- Requires high-bandwidth interconnect
- Cons: data communication among processors
- **Non Uniform Memory Access (NUMA)**



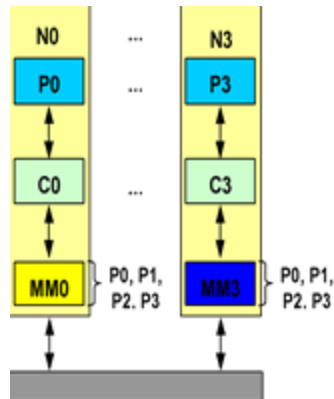
Recall: Address Space vs. Physical Memory Org.

Single Logically Shared Address Space (Shared-Memory Architectures)

Centralized Memory

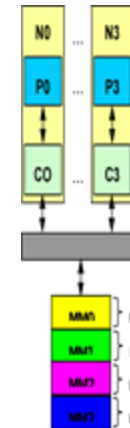


Distributed Memory

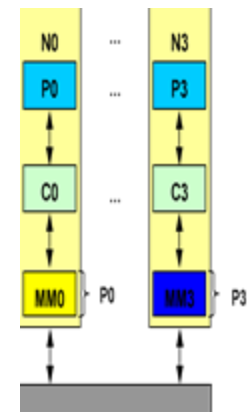


Multiple and Private Address Space (Message Passing Architectures)

Centralized Memory



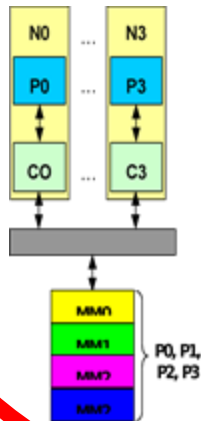
Distributed Memory



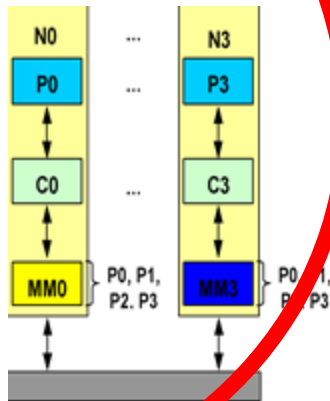
Recall: Address Space vs. Physical Memory Org.

Single Logically Shared Address Space (Shared-Memory Architectures)

Centralized Memory

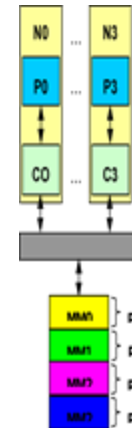


Distributed Memory

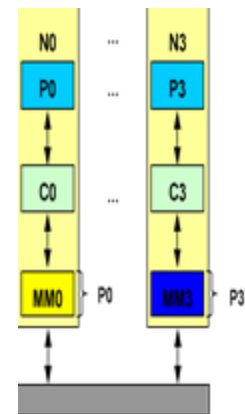


Multiple and Private Address Space (Message Passing Architectures)

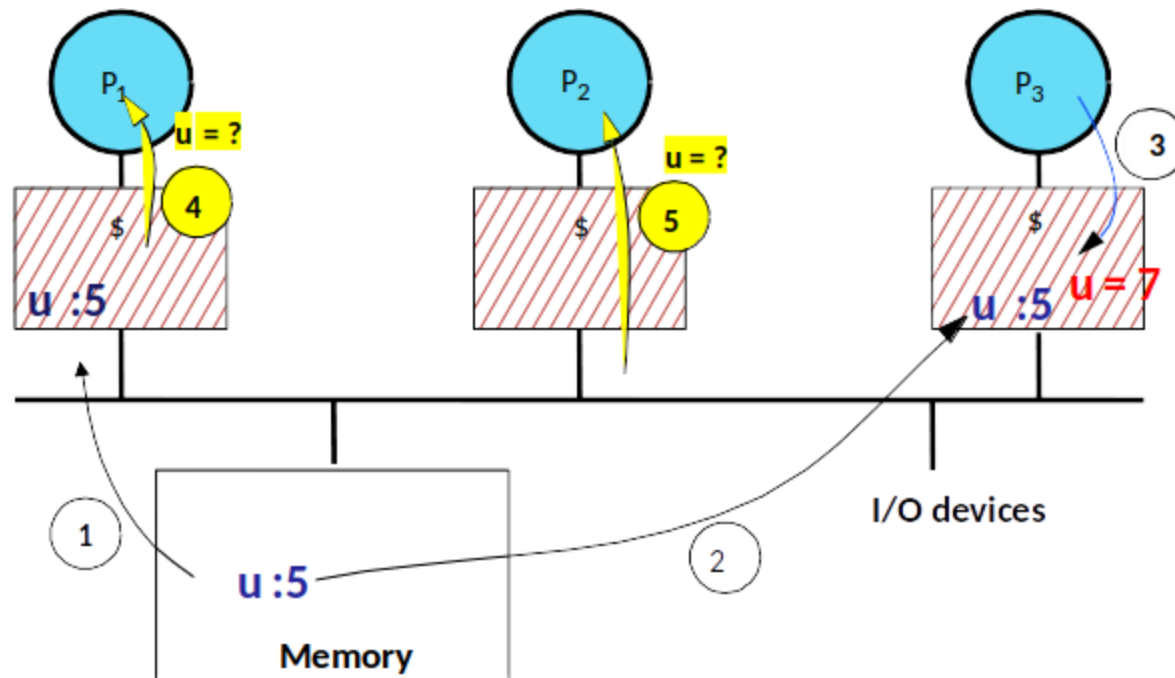
Centralized Memory



Distributed Memory



Recall: Example Cache Coherency Problem



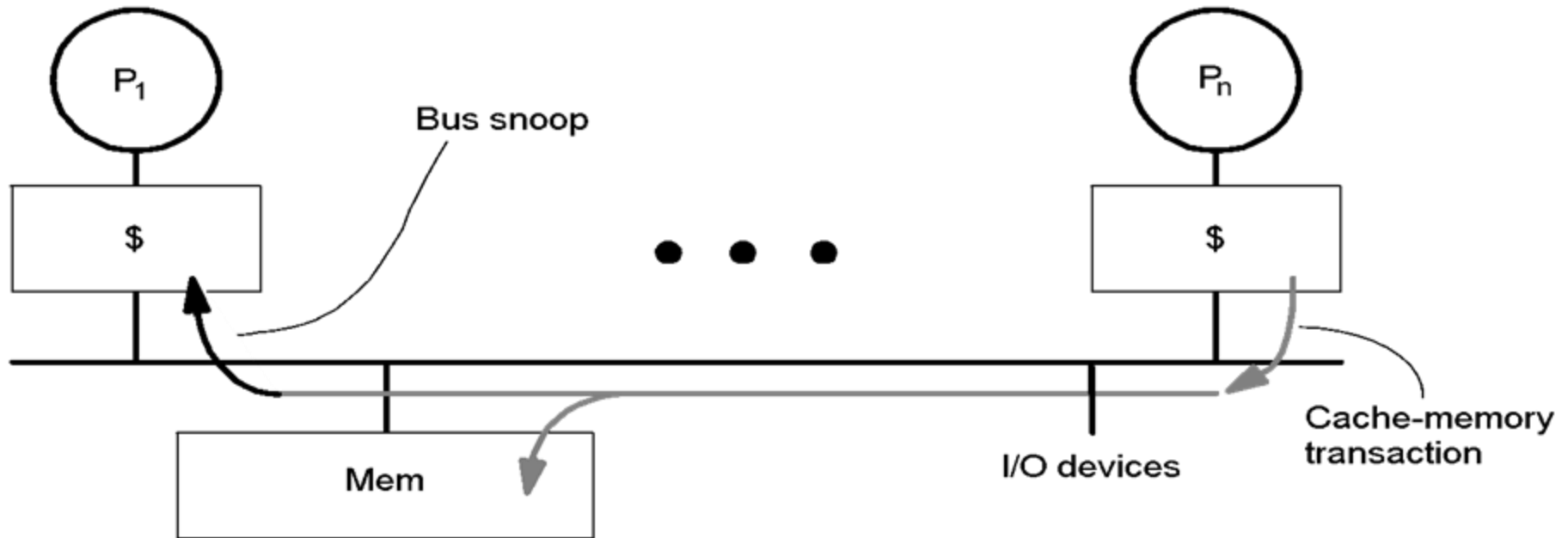
Things to note:

- Processors see different values for *u* after event 3
- With write back caches, value written back to memory depends on happenstance of which cache flushes or writes back value
 - Processes accessing main memory may see very stale value
- Unacceptable to programs, and frequent!

Recall: Potential Solutions

- HW-based solutions to maintain coherency:
Cache-Coherence Protocols
- Key issues to implement a cache coherent protocol in multiprocessors is tracking the status of any sharing of a data block.
- Two classes of protocols:
 - **Snooping Protocols**
 - Directory-Based Protocols

Recall: Snooping protocols



Recall: Basic Snooping Protocols

Snooping Protocols are of two types depending on what happens on a write operation:

Write-Invalidate Protocol

Write-Update or Write-Broadcast Protocol

Recall: MESI Protocol

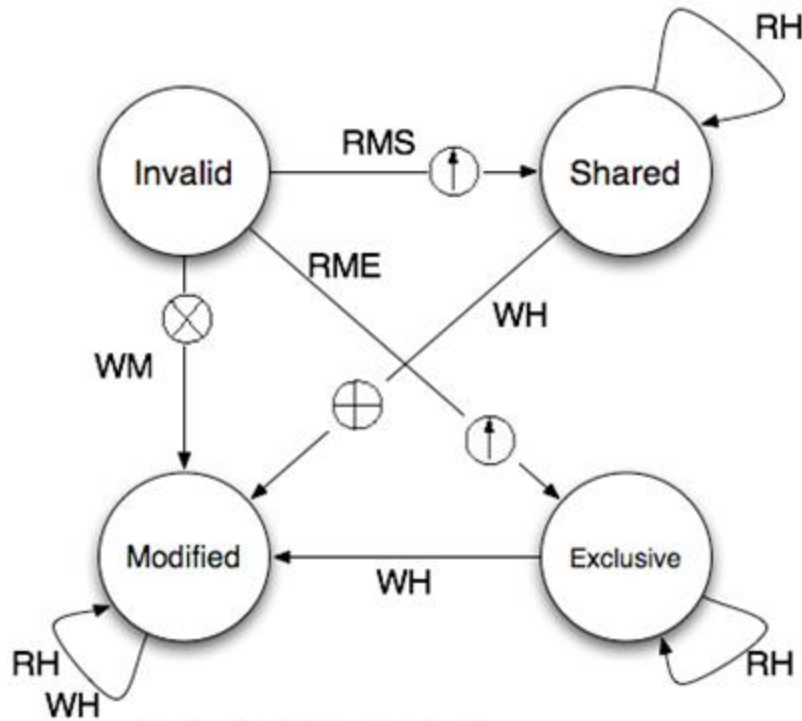
MESI Protocol: Write-Invalidate

Each cache block can be in one of **four** states:

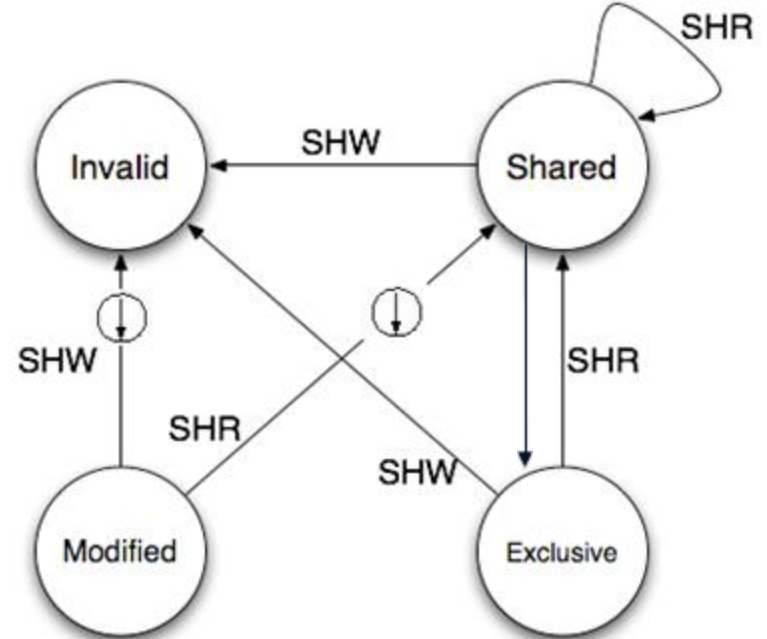
- **Modified**: the block is dirty and cannot be shared; cache has only copy, its writeable.
- **Exclusive**: the block is clean and cache has only copy;
- **Shared**: the block is clean and other copies of the block are in cache;
- **Invalid**: block contains no valid data

Add exclusive state to distinguish exclusive (writable) and owned (written)

Recall: MESI State Transition Diagram







Cache line in the "acting" processor



Transaction due to events snooped on the common BUS

BUS Transactions

RH = Read Hit
RMS = Read Miss, Shared
RME = Read Miss, Exclusive
WH = Write Hit
WM = Write Miss
SHR = Snoop Hit on a Read
SHW = Snoop Hit on a Write or
Read-with-Intent-to-Modify

-  = Snoop Push
-  = Invalidate Transaction
-  = Read-with-Intent-to-Modify
-  = Cache Block Fill

Exe1: Cache Coherency

Consider the following access pattern on a two-processor system with a direct-mapped, write-back cache with **one cache block and a two cache block memory**.

Assume the MESI protocol is used, with **write-back caches, write-allocate, and invalidation** of other caches on write (instead of updating the value in the other caches).

Recall: Write Policy

- Write-through: all writes update cache and underlying memory/cache
 - Can always discard cached data - most up-to-date data is in memory
 - Cache control bit: only a *valid* bit
 - Write-back: all writes simply update cache
 - Can't just discard cached data - may have to write it back to memory
 - Cache control bits: both *valid* and *dirty* bits
- How to identify the most recent data value of a cache block in case of cache miss?
 - It can be in a cache rather in a memory
 - Can use the same snooping scheme both for cache misses and writes
 - Each processor snoops every address placed on the bus
 - If a processor finds that it has a dirty copy of the requested cache block, it provides the cache block in response to the read request
 - memory access is aborted

Recall: Write Policy

- **What happens on write miss?**
- Write allocate: allocate new cache line in cache
 - Usually means that you have to do a “read miss” to fill in rest of the cache-line!
 - Alternative: per/word valid bits
- Write non-allocate (or “write-around”):
 - Simply send write data through to underlying memory/cache - don’t allocate new cache line!

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0				
2	P0: write block 1				
3	P0: write block 0				
4	P1: read block 0				
5	P1: write block 0				
6	P0: read block 1				
7	P1: read block 1				
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1				
3	P0: write block 0				
4	P1: read block 0				
5	P1: write block 0				
6	P0: read block 1				
7	P1: read block 1				
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0				
4	P1: read block 0				
5	P1: write block 0				
6	P0: read block 1				
7	P1: read block 1				
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0				
5	P1: write block 0				
6	P0: read block 1				
7	P1: read block 1				
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0				
6	P0: read block 1				
7	P1: read block 1				
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1				
7	P1: read block 1				
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1				
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

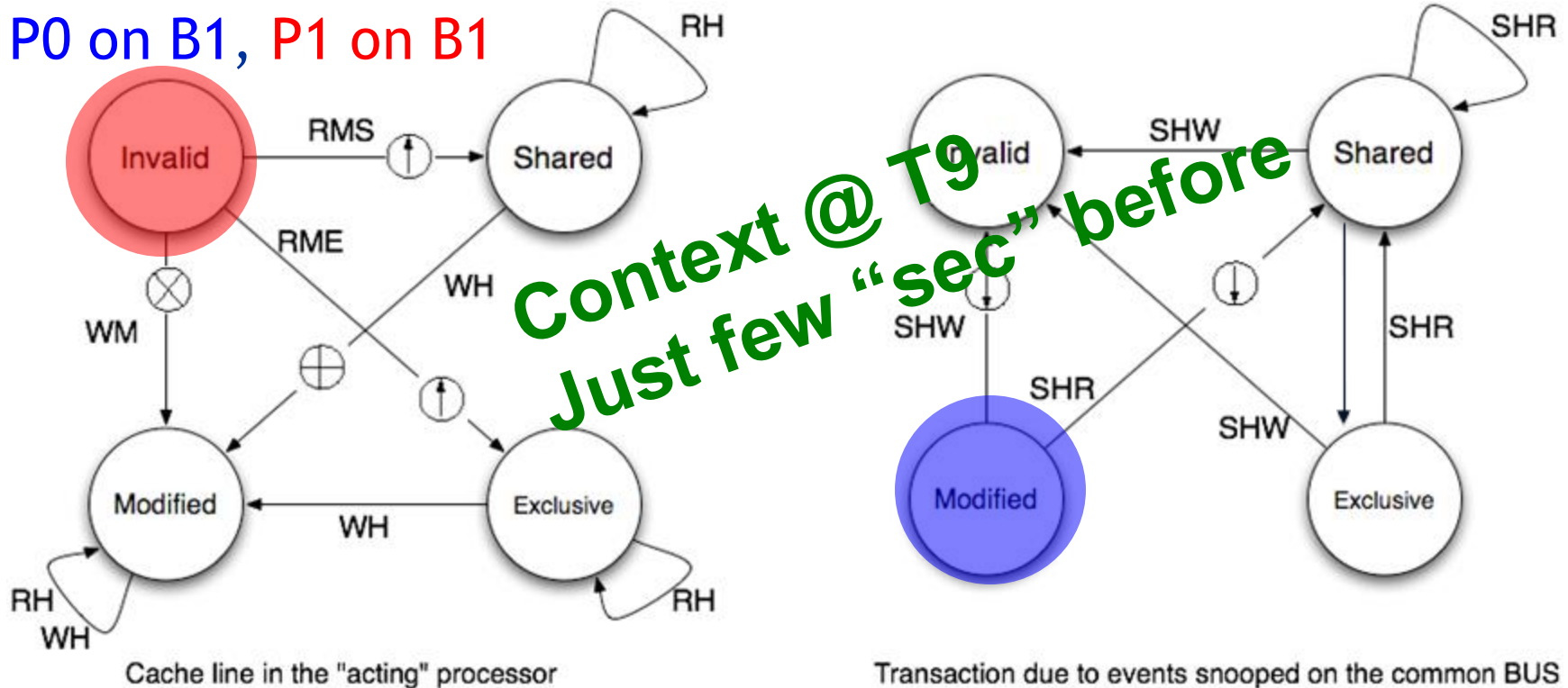
Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1				
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1	Modified (1)	Invalid	Yes	No
9	P1: write block 1				
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

@T9: P1: write block 1

P0 on B1, P1 on B1



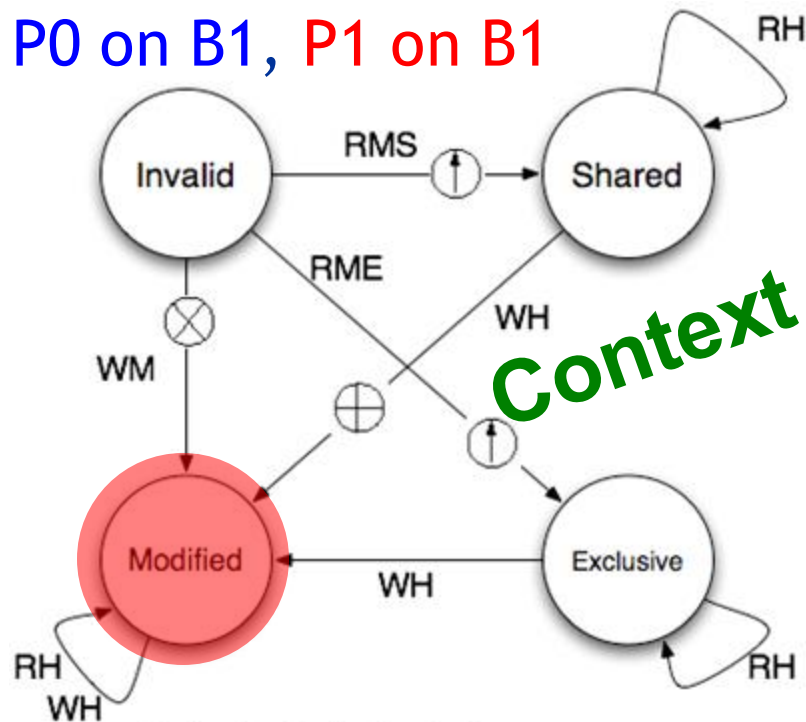
BUS Transactions

RH = Read Hit
 RMS = Read Miss, Shared
 RME = Read Miss, Exclusive
 WH = Write Hit
 WM = Write Miss
 SHR = Snoop Hit on a Read
 SHW = Snoop Hit on a Write or
 Read-with-Intent-to-Modify

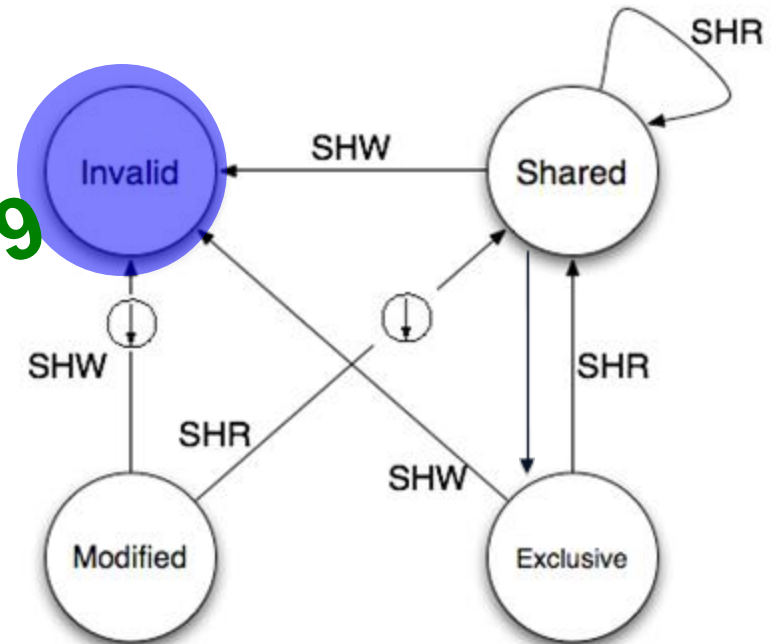
↓ = Snoop Push
 ⊗ = Invalidate Transaction
 ⊕ = Read-with-Intent-to-Modify
 ↑ = Cache Block Fill

@T9: P1: write block 1

P0 on B1, P1 on B1



Cache line in the "acting" processor



Transaction due to events snooped on the common BUS

BUS Transactions

RH = Read Hit
 RMS = Read Miss, Shared
 RME = Read Miss, Exclusive
 WH = Write Hit
 WM = Write Miss
 SHR = Snoop Hit on a Read
 SHW = Snoop Hit on a Write or Read-with-Intent-to-Modify

⬇️ = Snoop Push
 ⊗ = Invalidate Transaction
 ⊕ = Read-with-Intent-to-Modify
 ⬆️ = Cache Block Fill

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1	Modified (1)	Invalid	Yes	No
9	P1: write block 1	Invalid	Modified (1)	Yes	No
10	P0: read block 0				
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1	Modified (1)	Invalid	Yes	No
9	P1: write block 1	Invalid	Modified (1)	Yes	No
10	P0: read block 0	Exclusive (0)	Modified (1)	Yes	No
11	P1: write block 1				
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1	Modified (1)	Invalid	Yes	No
9	P1: write block 1	Invalid	Modified (1)	Yes	No
10	P0: read block 0	Exclusive (0)	Modified (1)	Yes	No
11	P1: write block 1	Exclusive (0)	Modified (1)	Yes	No
12	P1: read block 1				
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1	Modified (1)	Invalid	Yes	No
9	P1: write block 1	Invalid	Modified (1)	Yes	No
10	P0: read block 0	Exclusive (0)	Modified (1)	Yes	No
11	P1: write block 1	Exclusive (0)	Modified (1)	Yes	No
12	P1: read block 1	Exclusive (0)	Modified (1)	Yes	No
13	P0: read block 1				
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1	Modified (1)	Invalid	Yes	No
9	P1: write block 1	Invalid	Modified (1)	Yes	No
10	P0: read block 0	Exclusive (0)	Modified (1)	Yes	No
11	P1: write block 1	Exclusive (0)	Modified (1)	Yes	No
12	P1: read block 1	Exclusive (0)	Modified (1)	Yes	No
13	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes
14	P1: write block 1				

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: write block 0	Modified (0)	Invalid	No	Yes
4	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes
5	P1: write block 0	Invalid	Modified (0)	No	Yes
6	P0: read block 1	Exclusive (1)	Modified (0)	No	Yes
7	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
8	P0: write block 1	Modified (1)	Invalid	Yes	No
9	P1: write block 1	Invalid	Modified (1)	Yes	No
10	P0: read block 0	Exclusive (0)	Modified (1)	Yes	No
11	P1: write block 1	Exclusive (0)	Modified (1)	Yes	No
12	P1: read block 1	Exclusive (0)	Modified (1)	Yes	No
13	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes
14	P1: write block 1	Invalid	Modified (1)	Yes	No



Exe2: Cache Coherency

Consider the following access pattern on a two-processor system with a direct-mapped, write-back cache with **one cache block and a two cache block memory**.

Assume the MESI protocol is used, with **write-back caches, write-allocate, and invalidation** of other caches on write (instead of updating the value in the other caches).

Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: read block 1	Modified (1)	Exclusive (0)	Yes	No
4	P1: read block 0	Modified (1)	Exclusive (0)	Yes	No
5	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
6	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
7	P1: read block 0	Shared (1)	Exclusive (0)	Yes	Yes
8	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes
9	P0: write block 1	Modified (1)	Invalid (1)	Yes	No
10	P1: write block 0	Modified (1)	Modified (0)	No	No
11	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes

Exe 1: Cache Coherency

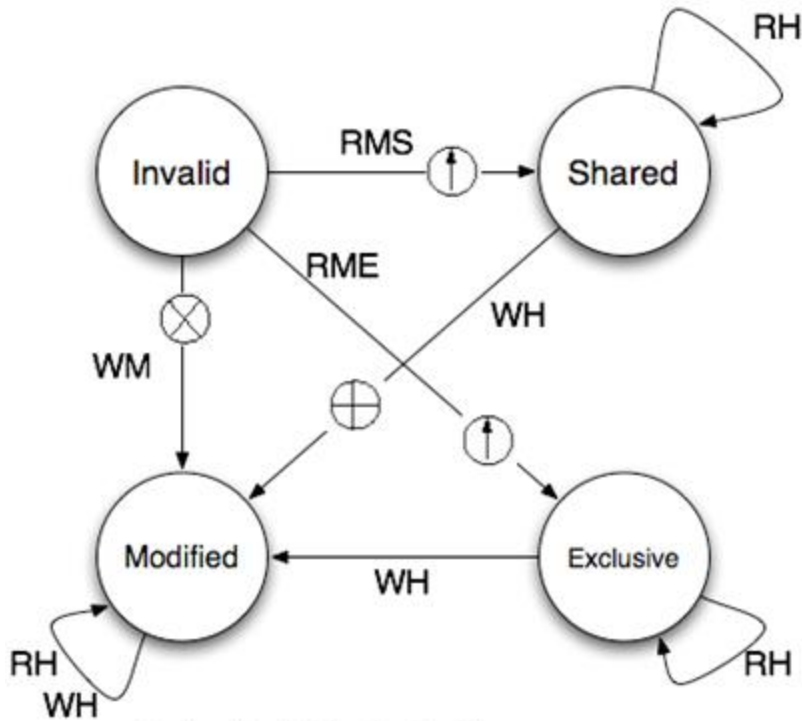
Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: read block 1	Modified (1)	Exclusive (0)	Yes	No
4	P1: read block 0	Modified (1)	Exclusive (0)	Yes	No
5	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
6	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
7	P1: read block 0	Shared (1)	Exclusive (0)	Yes	Yes
8	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes
9	P0: write block 1	Modified (1)	Invalid (1)	Yes	No
10	P1: write block 0	Modified (1)	Modified (0)	No	No
11	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes

Is this ok?

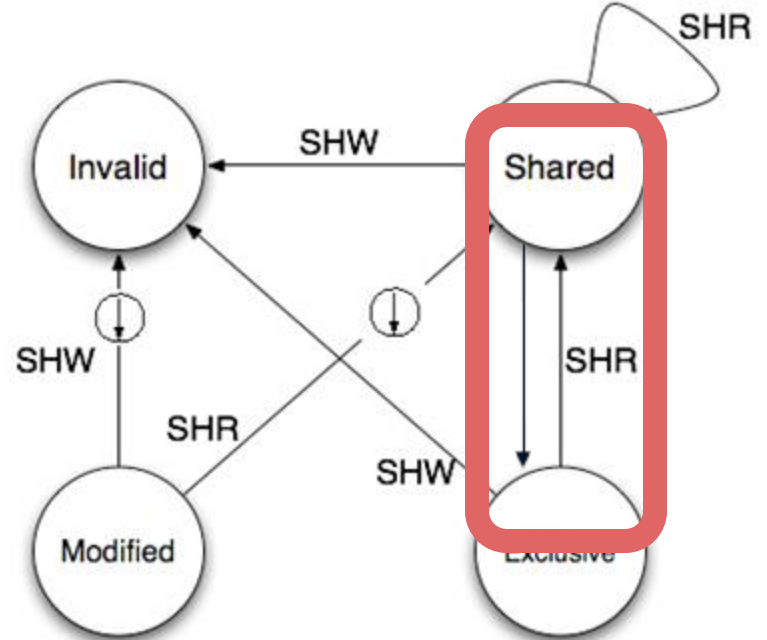
Exe 1: Cache Coherency

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: read block 1	Modified (1)	Exclusive (0)	Yes	No
4	P1: read block 0	Modified (1)	Exclusive (0)	Yes	No
5	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
6	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
7	P1: read block 0	Shared (1)	Exclusive (0)	Yes	Yes
8	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes
9	P0: write block 1	Modified (1)	Invalid (1)	Yes	No
10	P1: write block 0	Modified (1)	Modified (0)	No	No
11	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes

Recall: MESI State Transition Diagram







Cache line in the "acting" processor



Transaction due to events snooped on the common BUS

BUS Transactions

RH = Read Hit
RMS = Read Miss, Shared
RME = Read Miss, Exclusive
WH = Write Hit
WM = Write Miss
SHR = Snoop Hit on a Read
SHW = Snoop Hit on a Write or
Read-with-Intent-to-Modify

-  = Snoop Push
-  = Invalidate Transaction
-  = Read-with-Intent-to-Modify
-  = Cache Block Fill

Recall

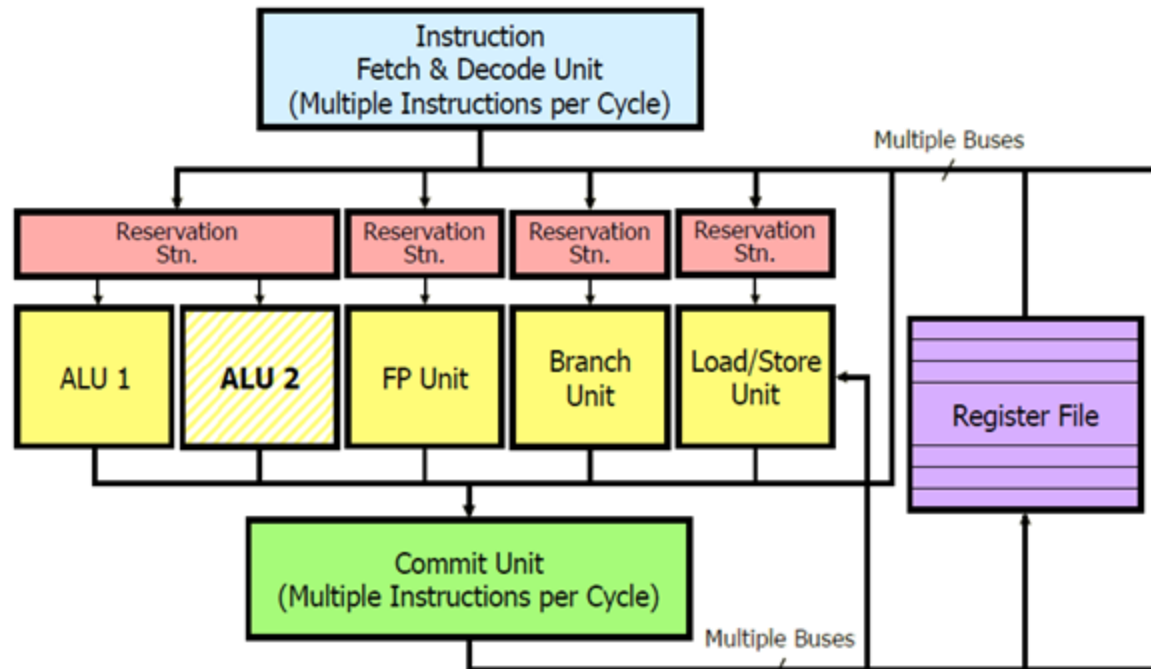
States of cache lines with MESI

© Politecnico di Milano

	Modified	Exclusive	Shared	Invalid
Line valid?	Yes	Yes	Yes	No
Copy in memory...	Has to be updated	Valid	Valid	-
Other copies in other caches?	No	No	Maybe	Maybe
A write on this line...	Access the BUS	Access the BUS	Access the BUS and Update the cache	Direct access to the BUS



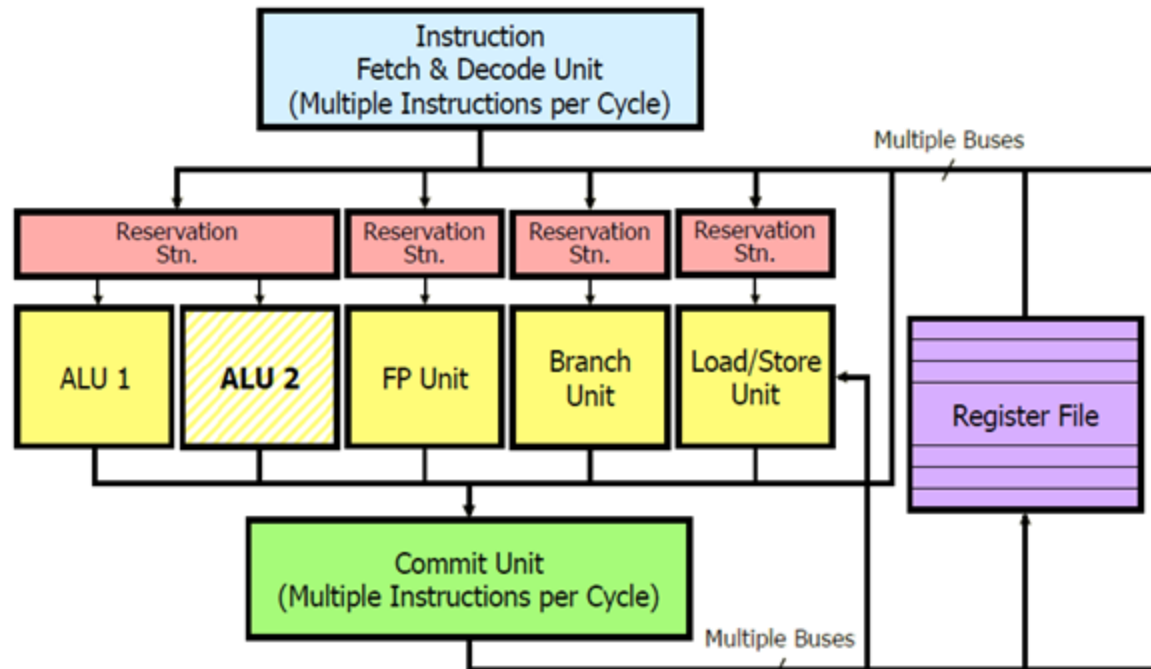
The Context: ILP limits



Class of dependencies:

- Name Dependencies → WAR/WAW
- Data Dependencies → RAW
- Control Dependencies → Conditional Branches

The Context: ILP limits



Class of dependencies:

Name Dependencies

→ WAR/WAW

- Data Dependencies

→ RAW

- Control Dependencies

→ Conditional Branches

Explicit Register Renaming

```
ld x1, (x3)
addi x3, x1, #4
sub x6, x7, x9
add x3, x3, x6
ld x6, (x1)
add x6, x6, x3
sd x6, (x1)
ld x6, (x11)
```

```
ld P1, (Px)
addi P2, P1, #4
sub P3, Py, Pz
add P4, P2, P3
ld P5, (P1)
add P6, P5, P4
sd P6, (P1)
ld P7, (Pw)
```

Physical RF

P0	32bit
P1	
P2	
P3	
...	
P62	
P63	

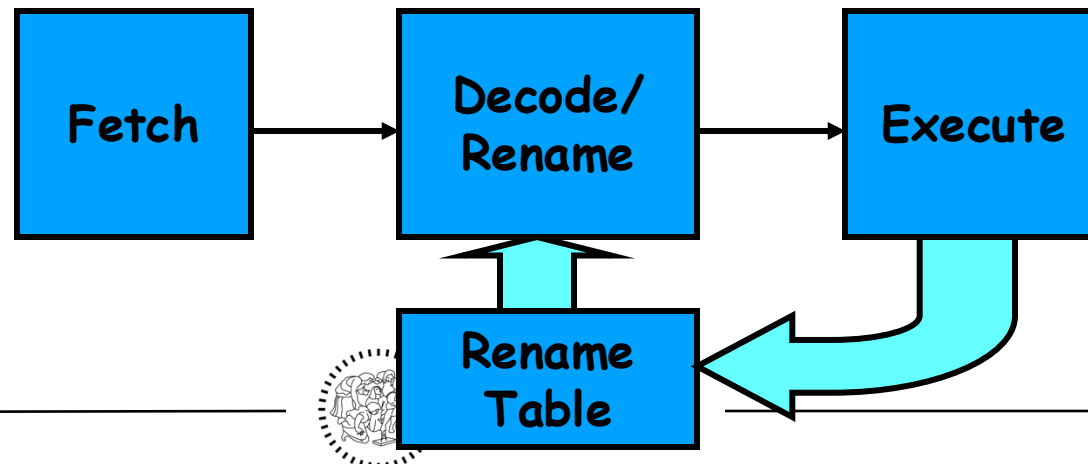
Explicit Register Renaming

```
ld x1, (x3)
addi x3, x1, #4
sub x6, x7, x9
add x3, x3, x6
ld x6, (x1)
add x6, x6, x3
sd x6, (x1)
ld x6, (x11)
```

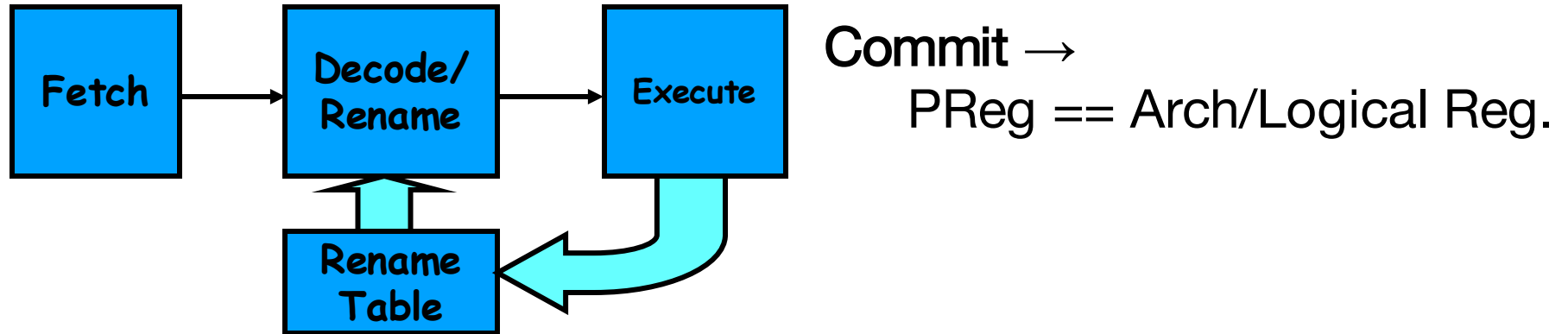
```
ld P1, (Px)
addi P2, P1, #4
sub P3, Py, Pz
add P4, P2, P3
ld P5, (P1)
add P6, P5, P4
sd P6, (P1)
ld P7, (Pw)
```

Physical RF

P0	32bit
P1	
P2	
P3	
...	
P62	
P63	

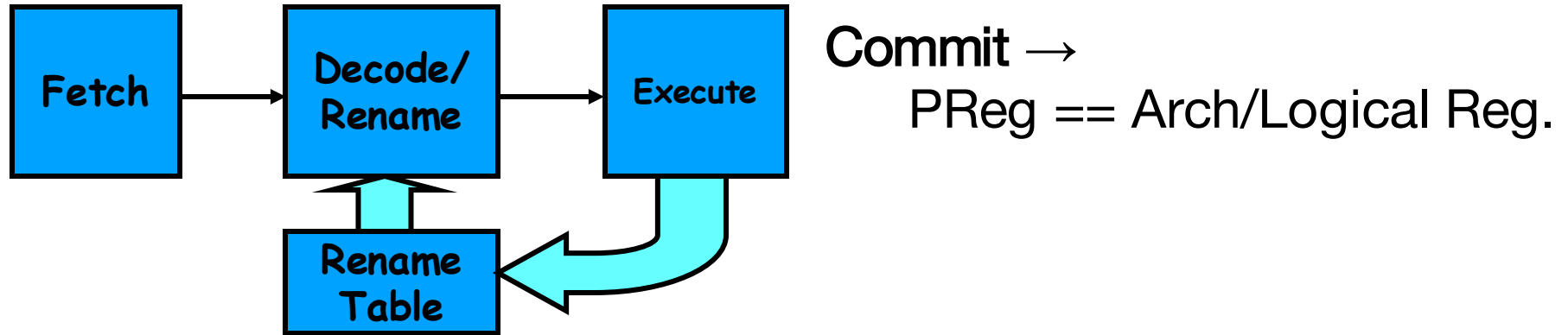


EXTRA: What is the problem?



When can we reuse a physical register?
a.k.a. Physical Registers Lifetime

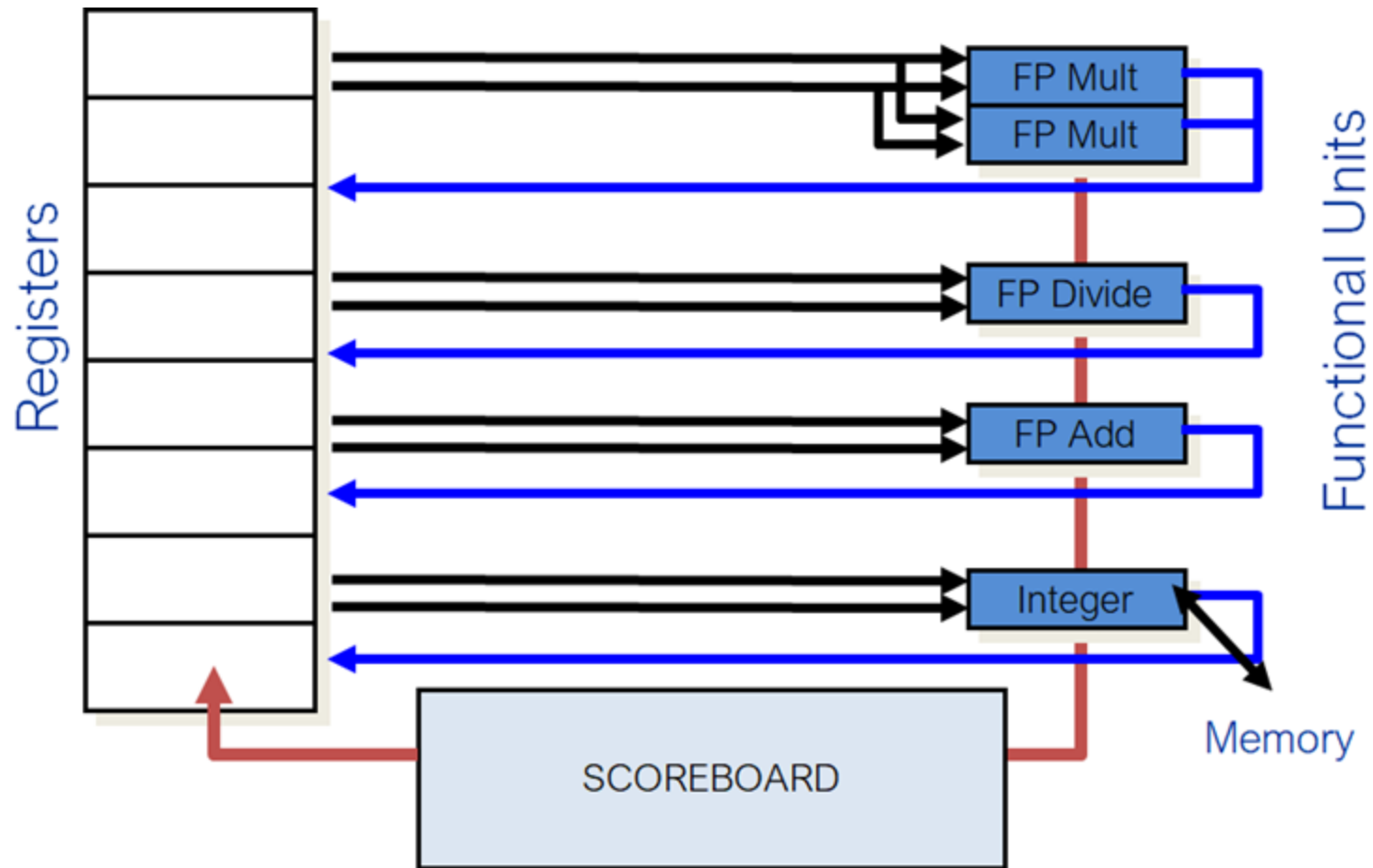
EXTRA: What is the problem?



**When can we reuse a physical register?
a.k.a. Physical Registers Lifetime**

When next writer of same architectural register commits
(MOST of NOWADAYS Designs <https://doi.org/10.1109/HPCA56546.2023.10071122>)

Exe 3 Scoreboard



[Parallel operation in the control data 6600](#)

Recall: the Scoreboard pipeline

ISSUE	READ OPERAND	EXE COMPLETE	WB
Decode instruction;	Read operands;	Operate on operands;	Finish exec;
Structural FUs check; WAW checks	RAW check; WAR if need to read	Notify Scoreboard on completion;	WAR & Struct check (FUs will hold results); Can overlap issue/read&write;

Recall: the Scoreboard pipeline with Register Renaming

ISSUE	READ OPERAND	EXE COMPLETE	WB
Decode instruction; allocate new physical register for result	Read operands;	Operate on operands;	Finish exec;
Structural FUs check; WAW checks; free physical registers check	RAW check; WAR if need to read	Notify Scoreboard on completion;	WAR & Struct check (FUs will hold results); Can overlap issue/read&write;

No checks for WAR or WAW hazards!

Exe 3 Scoreboard: the Code

```
I1:  LD  F6 32+ R2
I2:  ADDD F2 F6 F4
I3:  MULTD F0 F4 F2
I4:  SUBD F12 F2 F6
I5:  ADDD F0 F12 F2
```

Exe 3.1 Scoreboard: Conflicts

I1: LD F6 32+ R2
I2: ADDD F2 F6 F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2

Exe 3.1 Scoreboard: Conflicts

RAW **F6** I1-I2

I1: LD **F6** B2+ R2
I2: ADDD F2 **F6** F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2

Exe 3.1 Scoreboard: Conflicts

I1: LD **F6** R2+ R2
I2: ADDD F2 **F6** F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 **F6**
I5: ADDD F0 F12 F2

RAW **F6** I1-I2

RAW **F6** I1-I4

Exe 3.1 Scoreboard: Conflicts

I1: LD **F6** R2+ R2
I2: ADDD **F2** **F6** F4
I3: MULTD F0 F4 **F2**
I4: SUBD F12 F2 **F6**
I5: ADDD F0 F12 F2

RAW **F6** I1-I2

RAW **F6** I1-I4

RAW **F2** I2-I3

Exe 3.1 Scoreboard: Conflicts

I1: LD F6 B2+ R2
I2: ADDD F2 F6 F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2

RAW **F6** I1-I2

RAW **F6** I1-I4

RAW **F2** I2-I3

RAW **F2** I2-I4

Exe 3.1 Scoreboard: Conflicts

I1: LD F6 B2+ R2
I2: ADDD ~~F2~~ F6 F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2

RAW **F6** I1-I2

RAW **F6** I1-I4

RAW **F2** I2-I3

RAW **F2** I2-I4

RAW **F2** I2-I5

Exe 3.1 Scoreboard: Conflicts

I1: LD F6 B2+ R2
I2: ADDD ~~F2~~ F6 F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2

RAW **F6** I1-I2

RAW **F6** I1-I4

RAW **F2** I2-I3

RAW **F2** I2-I4

RAW **F2** I2-I5

RAW **F12** I4-I5

Exe 3.1 Scoreboard: Conflicts

I1: LD F6 B2+ R2
I2: ADDD F2 F6 F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2

RAW **F6** I1-I2

RAW **F6** I1-I4

RAW **F2** I2-I3

RAW **F2** I2-I4

RAW **F2** I2-I5

RAW **F12** I4-I5

WAW **F0** I3-I5

Exe 3.3 Scoreboard: CC 0

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2						
I2	ADDD F2 F6 F4						
I3	MULTD F0 F4 F2						
I4	SUBD F12 F2 F6						
I5	ADDD F0 F12 F2						

F0	F2	F4	F6	F8	F10	F12	...	F30
P0	P2	P4	P6	P8	P10	P12	...	P30

Initialized Rename Table – registers from P32 in the free list

4 FPALU 3 cc latency, single write port for the pool

1 MEM 2 cc latency

RAW **F6** I1-I2
 RAW **F6** I1-I4
 RAW **F2** I2-I3
 RAW **F2** I2-I4
 RAW **F2** I2-I5
 RAW **F12** I4-I5

Exe 3.3 Scoreboard: CC 1

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1					MU
I2	ADDD F2 F6 F4						
I3	MULTD F0 F4 F2						
I4	SUBD F12 F2 F6						
I5	ADDD F0 F12 F2						

F0	F2	F4	F6	F8	F10	F12	...	F30
P0	P2	P4	P32	P8	P10	P12	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

RAW **F6** I1-I2
RAW **F6** I1-I4
RAW **F2** I2-I3
RAW **F2** I2-I4
RAW **F2** I2-I5
RAW **F12** I4-I5

Exe 3.3 Scoreboard: CC 2

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2				MU
I2	ADDD F2 F6 F4	2					FPU1
I3	MULTD F0 F4 F2						
I4	SUBD F12 F2 F6						
I5	ADDD F0 F12 F2						

F0	F2	F4	F6	F8	F10	F12	...	F30
P0	P33	P4	P32	P8	P10	P12	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

RAW **F6** I1-I2
RAW **F6** I1-I4
RAW **F2** I2-I3
RAW **F2** I2-I4
RAW **F2** I2-I5
RAW **F12** I4-I5

Exe 3.3 Scoreboard: CC 3

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2				MU
I2	ADDD F2 F6 F4	2				RAW F6	FPU1
I3	MULTD F0 F4 F2	3					FPU2
I4	SUBD F12 F2 F6						
I5	ADDD F0 F12 F2						

F0	F2	F4	F6	F8	F10	F12	...	F30
P34	P33	P4	P32	P8	P10	P12	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

RAW F6 I1-I2
RAW F6 I1-I4
RAW F2 I2-I3
RAW F2 I2-I4
RAW F2 I2-I5
RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 4

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4			MU
I2	ADDD F2 F6 F4	2				RAW F6	FPU1
I3	MULTD F0 F4 F2	3				RAW F2	FPU2
I4	SUBD F12 F2 F6	4					FPU3
I5	ADDD F0 F12 F2						

F0	F2	F4	F6	F8	F10	F12	...	F30
P34	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

RAW F6 I1-I2
 RAW F0 I1-I4
 RAW F2 I2-I3
 RAW F2 I2-I4
 RAW F2 I2-I5
 RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 5

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2				RAW F6	FPU1
I3	MULTD F0 F4 F2	3				RAW F2	FPU2
I4	SUBD F12 F2 F6	4				RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5					FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

RAW F6 I1-I2
RAW F6 I1-I4
RAW F2 I2-I3
RAW F2 I2-I4
RAW F2 I2-I5
RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 5

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2				RAW F6	FPU1
I3	MULTD F0 F4 F2	3				RAW F2	FPU2
I4	SUBD F12 F2 F6	4				RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5					FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

WAW F0 I3-I5 solved by register renaming
(even not listed in rename table)

RAW F6 I1-I2

RAW F6 I1-I4

RAW F2 I2-I3

RAW F2 I2-I4

RAW F2 I2-I5

RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 6

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6			RAW F6	FPU1
I3	MULTD F0 F4 F2	3				RAW F2	FPU2
I4	SUBD F12 F2 F6	4				RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
 RAW F2 I2-I3
 RAW F2 I2-I4
 RAW F2 I2-I5
 RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 9

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9		RAW F6	FPU1
I3	MULTD F0 F4 F2	3				RAW F2	FPU2
I4	SUBD F12 F2 F6	4				RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
 RAW F2 I2-I3
 RAW F2 I2-I4
 RAW F2 I2-I5
 RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 10

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3				RAW F2	FPU2
I4	SUBD F12 F2 F6	4				RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
~~RAW F2 I2-I3~~
~~RAW F2 I2-I4~~
~~RAW F2 I2-I5~~
RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 11

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3	11			RAW F2	FPU2
I4	SUBD F12 F2 F6	4	11			RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
~~RAW F2 I2-I3~~
~~RAW F2 I2-I4~~
~~RAW F2 I2-I5~~
 RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 14

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3	11	14		RAW F2	FPU2
I4	SUBD F12 F2 F6	4	11	14		RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
~~RAW F2 I2-I3~~
~~RAW F2 I2-I4~~
~~RAW F2 I2-I5~~
 RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 14

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULD F0 F4 F2	3	11	14		RAW F4	FPU2
I4	SULD F12 F2 F6	4	11	14		RAW F2, F6	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

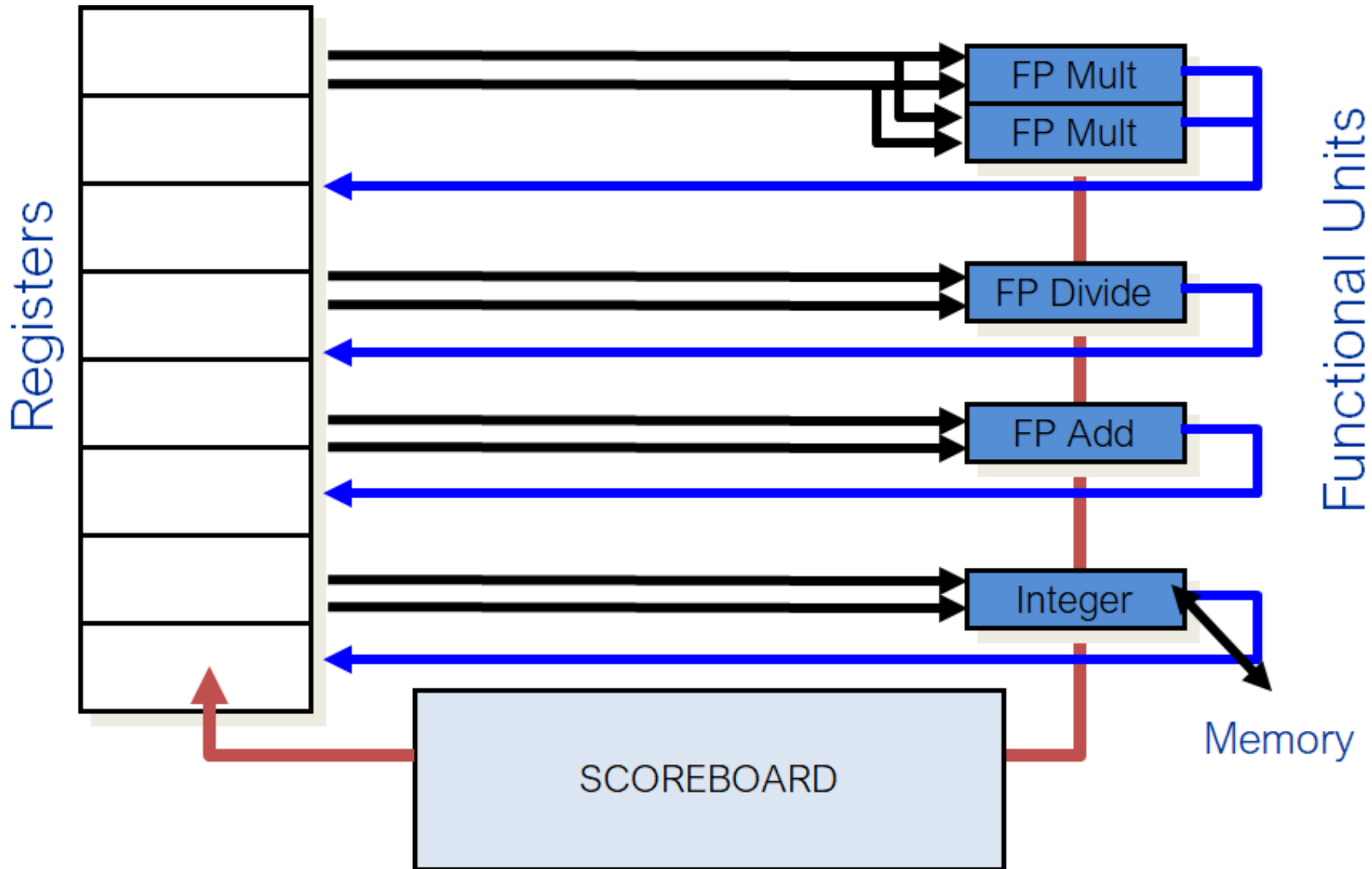
SINGLE WRITE PORT?

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

- ~~RAW F6 I1-I2~~
- ~~RAW F6 I1-I4~~
- ~~RAW F2 I2-I3~~
- ~~RAW F2 I2-I4~~
- ~~RAW F2 I2-I5~~
- RAW F12 I4-I5

Exe 3.3 Scoreboard



Exe 3.3 Scoreboard: CC 15

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3	11	14	15	RAW F2	FPU2
I4	SUBD F12 F2 F6	4	11	14		RAW F2, F6 + Struct RF	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
~~RAW F2 I2-I3~~
~~RAW F2 I2-I4~~
~~RAW F2 I2-I5~~
 RAW F12 I4-I5

Exe 3.3 Scoreboard: CC 16

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3	11	14	15	RAW F2	FPU2
I4	SUBD F12 F2 F6	4	11	14	16	RAW F2, F6 + Struct RF	FPU3
I5	ADDD F0 F12 F2	5				RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
~~RAW F2 I2-I3~~
~~RAW F2 I2-I4~~
~~RAW F2 I2-I5~~
~~RAW F12 I4-I5~~

Exe 3.3 Scoreboard: CC 17

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3	11	14	15	RAW F2	FPU2
I4	SUBD F12 F2 F6	4	11	14	16	RAW F2, F6 + Struct RF	FPU3
I5	ADDD F0 F12 F2	5	17			RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
~~RAW F2 I2-I3~~
~~RAW F2 I2-I4~~
~~RAW F2 I2-I5~~
~~RAW F12 I4-I5~~

Exe 3.3 Scoreboard: CC 21

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3	11	14	15	RAW F2	FPU2
I4	SUBD F12 F2 F6	4	11	14	16	RAW F2, F6 + Struct RF	FPU3
I5	ADDD F0 F12 F2	5	17	20	21	RAW F2, F12	FPU4

F0	F2	F4	F6	F8	F10	F12	...	F30
P36	P33	P4	P32	P8	P10	P35	...	P30

4 FPALU 3 cc latency, single write port for the pool
1 MEM 2 cc latency

~~RAW F6 I1-I2~~
~~RAW F6 I1-I4~~
~~RAW F2 I2-I3~~
~~RAW F2 I2-I4~~
~~RAW F2 I2-I5~~
~~RAW F12 I4-I5~~



Thank you for your attention

Questions?

Alessandro Verosimile <alessandro.verosimile@polimi.it>

Acknowledgements

Davide Conficconi, E. Del Sozzo, Marco D. Santambrogio, D. Sciuto

Part of this material comes from:

- “Computer Organization and Design” and “Computer Architecture A Quantitative Approach” Patterson and Hennessy books
- News and paper cited throughout the lecture

and are **properties of their respective owners**

Rotating Register Files

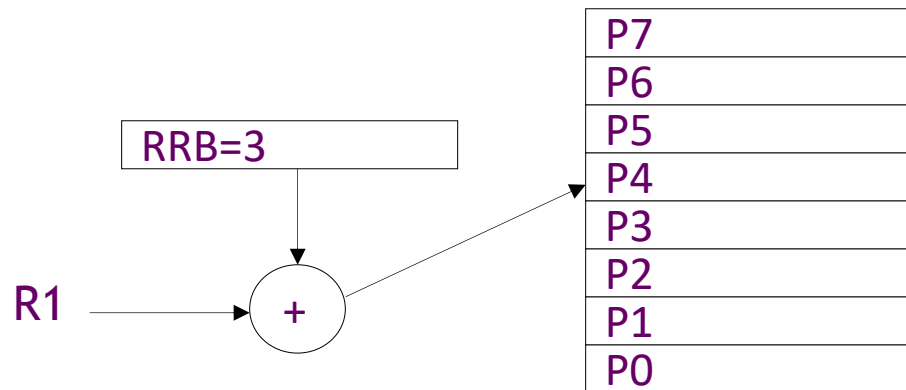
- Problems:
 - Scheduled loops require lots of registers,
 - Lots of duplicated code in prolog, epilog

Rotating Register Files

- **Problems:**
 - Scheduled loops require lots of registers,
 - Lots of duplicated code in prolog, epilog
- **Solution:**
 - Allocate new set of registers for each loop iteration

Rotating Register File

- Rotating Register Base (RRB) register points to base of current register set.
- Value added on to logical register specifier to give physical register number.



- Usually, split into rotating and non-rotating registers.

Loop Example

```
for (i=0; i<N; i++)
  B[i] = A[i] + C;
```

Compile

```
loop: ld f1, 0(r1)
      add r1, 8
      fadd f2, f0, f1
      sd f2, 0(r2)
      add r2, 8
      bne r1, r3, loop
```

loop:

Schedule

Int1 Int 2 M1 M2 FP+ FPx

add r1		ld			
				fadd	
add r2	bne	l	sd	f2	

Loop example: what we'd like to...

Int1 Int 2 M1 M2 FP+ FPx

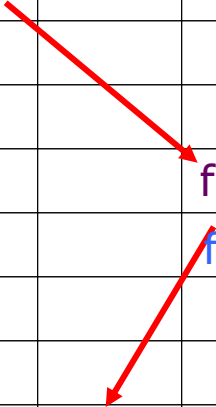
add r1		ld			
				fadd	
add r2	bne		sd		

No dep on FP in
consecutive iterations

for (i=0; i<N; i++)
 B[i] = A[i] + C;

Loop example: what we'd like to...

Int1	Int 2	M1	M2	FP+	FPx
add r1		ld			
		ld			
				fadd	
				fadd	
add r2	bne		sd		
			sd		



Loop example: what we'd like to...

Int1	Int 2	M1	M2	FP+	FPx
add r1		ld			
		ld			
		ld			
		ld		fadd	
		ld		fadd	
		ld		fadd	
		ld		fadd	
add r2	bne	ld	sd	fadd	
			sd	fadd	

Loop example: what we'd like to...

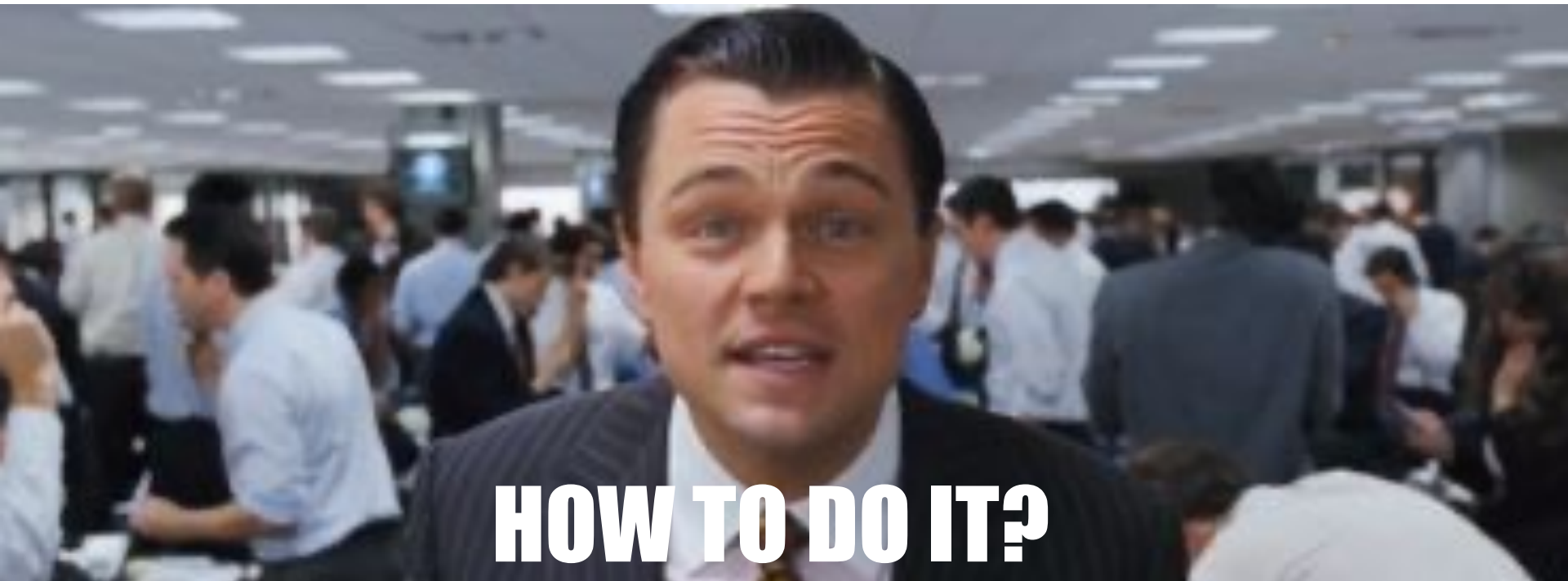
Int1 Int 2 M1 M2 FP+ FPx

add r1		ld			
		ld			
		ld			
		ld		fadd	
		ld		fadd	
		ld		fadd	
		ld		fadd	
add r2	bne	ld	sd	fadd	
			sd	fadd	

ld f1, ()	fadd fx, fy, ...	sd fz, ()	bloop
-----------	------------------	-----------	-------

Loop example: what we'd like to...

Int1 Int 2 M1 M2 FP+ FPx



HOW TO DO IT?

ld f1, ()

fadd fx, fy, ...

sd fz, ()

bloop

Rotating Register File

ld f1, ()	fadd fx, fy, ...	sd fz, ()	bloop
-----------	------------------	-----------	-------

Rotating Register File

ld f1, ()	fadd fx, fy, ...	sd fz, ()	bloop
-----------	------------------	-----------	-------

Three cycle load latency
encoded as difference of 3 in
register specifier number

Four cycle fadd latency
encoded as difference of 4 in
register specifier number

Rotating Register File

ld f1, ()	fadd fx, fy, ...	sd fz, ()	bloop
-----------	------------------	-----------	-------

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
-----------	------------------	-----------	-------

```

loop: ld f1, ...
      ...
      fadd f2, f0, f1
      sd f2, ...
      ...
  
```

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)



Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop

RRB=8

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop

RRB=8

RRB=7

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop

RRB=8

RRB=7

RRB=6

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop
ld P6, ()	fadd P10, P9,	sd P14, ()	bloop

RRB=8

RRB=7

RRB=6

RRB=5

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop
ld P6, ()	fadd P10, P9,	sd P14, ()	bloop

RRB=8

RRB=7

RRB=6

RRB=5

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop
ld P6, ()	fadd P10, P9,	sd P14, ()	bloop
ld P5, ()	fadd P9, P8,	sd P13, ()	bloop

RRB=8

RRB=7

RRB=6

RRB=5

RRB=4

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop
ld P6, ()	fadd P10, P9,	sd P14, ()	bloop
ld P5, ()	fadd P9, P8,	sd P13, ()	bloop
ld P4, ()	fadd P8, P7,	sd P12, ()	bloop

RRB=8

RRB=7

RRB=6

RRB=5

RRB=4

RRB=3

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop
ld P6, ()	fadd P10, P9,	sd P14, ()	bloop
ld P5, ()	fadd P9, P8,	sd P13, ()	bloop
ld P4, ()	fadd P8, P7,	sd P12, ()	bloop
ld P3, ()	fadd P7, P6,	sd P11, ()	bloop

RRB=8

RRB=7

RRB=6

RRB=5

RRB=4

RRB=3

RRB=2

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop
ld P6, ()	fadd P10, P9,	sd P14, ()	bloop
ld P5, ()	fadd P9, P8,	sd P13, ()	bloop
ld P4, ()	fadd P8, P7,	sd P12, ()	bloop
ld P3, ()	fadd P7, P6,	sd P11, ()	bloop
ld P2, ()	fadd P6, P5,	sd P10, ()	bloop

RRB=8

RRB=7

RRB=6

RRB=5

RRB=4

RRB=3

RRB=2

RRB=1

Rotating Register File

Three cycle load latency
encoded as difference of 3 in
register specifier number
($f1 + 3 = fy \dots y = 3+1$)

Four cycle fadd latency
encoded as difference of 4 in
register specifier number
($f5 + 4 = fz \dots z = 5+4$)

ld f1, ()	fadd f5, f4, ...	sd f9, ()	bloop
ld P9, ()	fadd P13, P12,	sd P17, ()	bloop
ld P8, ()	fadd P12, P11,	sd P16, ()	bloop
ld P7, ()	fadd P11, P10,	sd P15, ()	bloop
ld P6, ()	fadd P10, P9,	sd P14, ()	bloop
ld P5, ()	fadd P9, P8,	sd P13, ()	bloop
ld P4, ()	fadd P8, P7,	sd P12, ()	bloop
ld P3, ()	fadd P7, P6,	sd P11, ()	bloop
ld P2, ()	fadd P6, P5,	sd P10, ()	bloop

RRB=8

RRB=7

RRB=6

RRB=5

RRB=4

RRB=3

RRB=2

RRB=1

END...



From Computer Desktop Encyclopedia
Reproduced with permission.
© 2007 Tiler Corporation

