POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

POLITECNICO MILANO 1863 NECST laboratory

Exercise Session 10

Cache Coherency, Exam Simulation (CC, Tomasulo, VLIW), Extras in Solved version

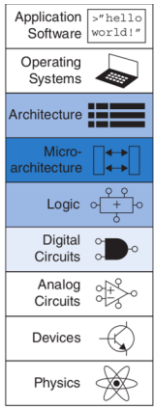
Advanced Computer Architectures

21 May 2025

Davide Conficconi ‹davide.conficconi@polimi.it›

POLITECNICO MILANO 1863 | DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

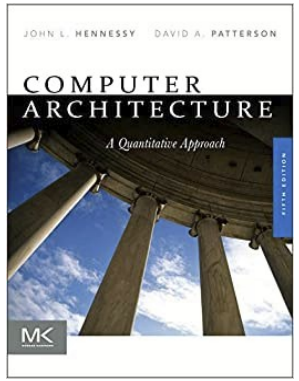
Recall: Material (EVERYTHING OPTIONAL)



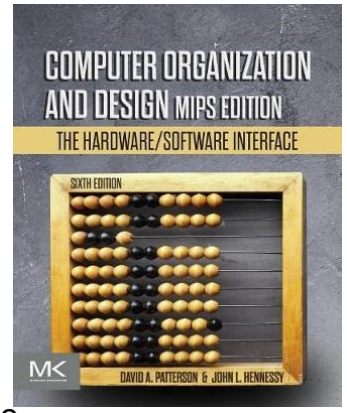
https://webeep.polimi.it/course/view.php?id=14754

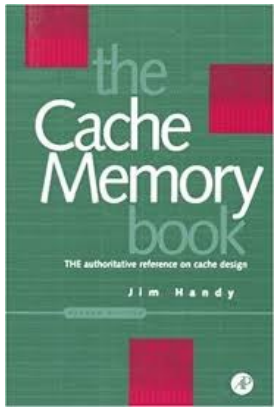
https://tinyurl.com/aca-grid25

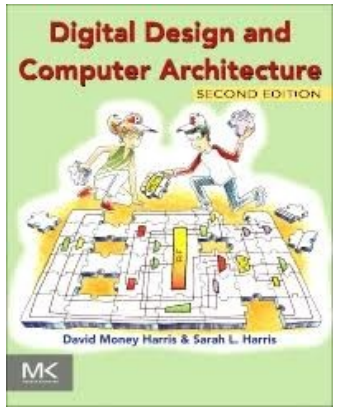
Textbook: Hennessy and Patterson, Computer Architecture: A Quantitative Approach

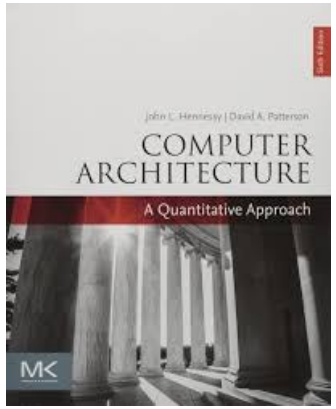


Other Interesting Reference



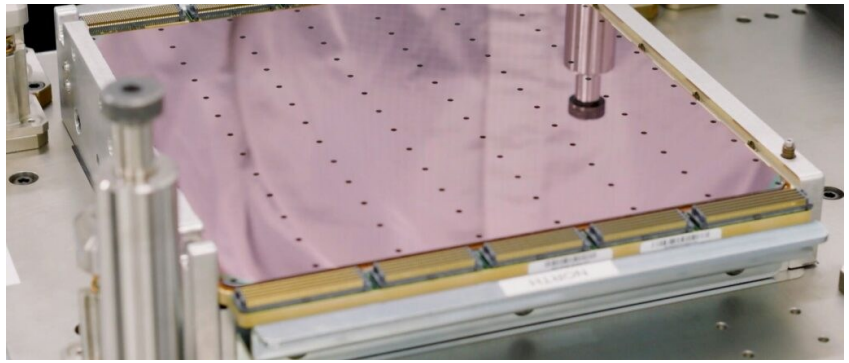


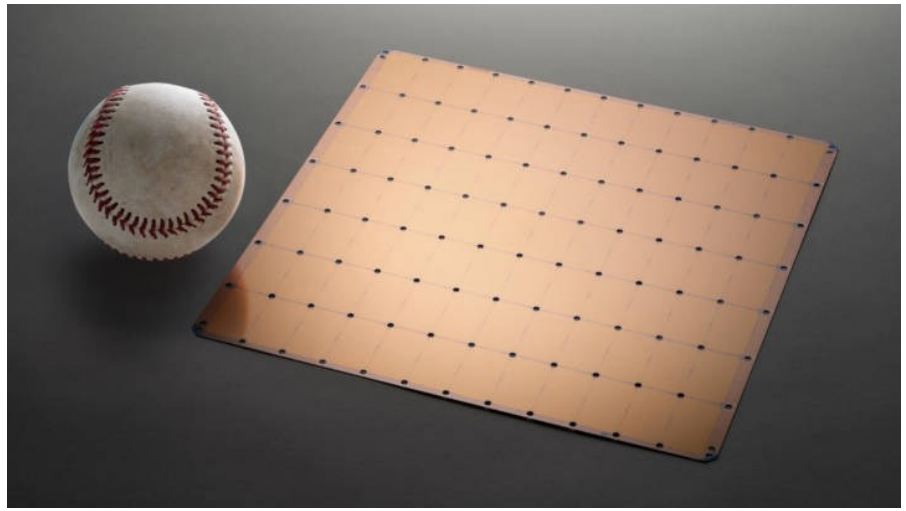


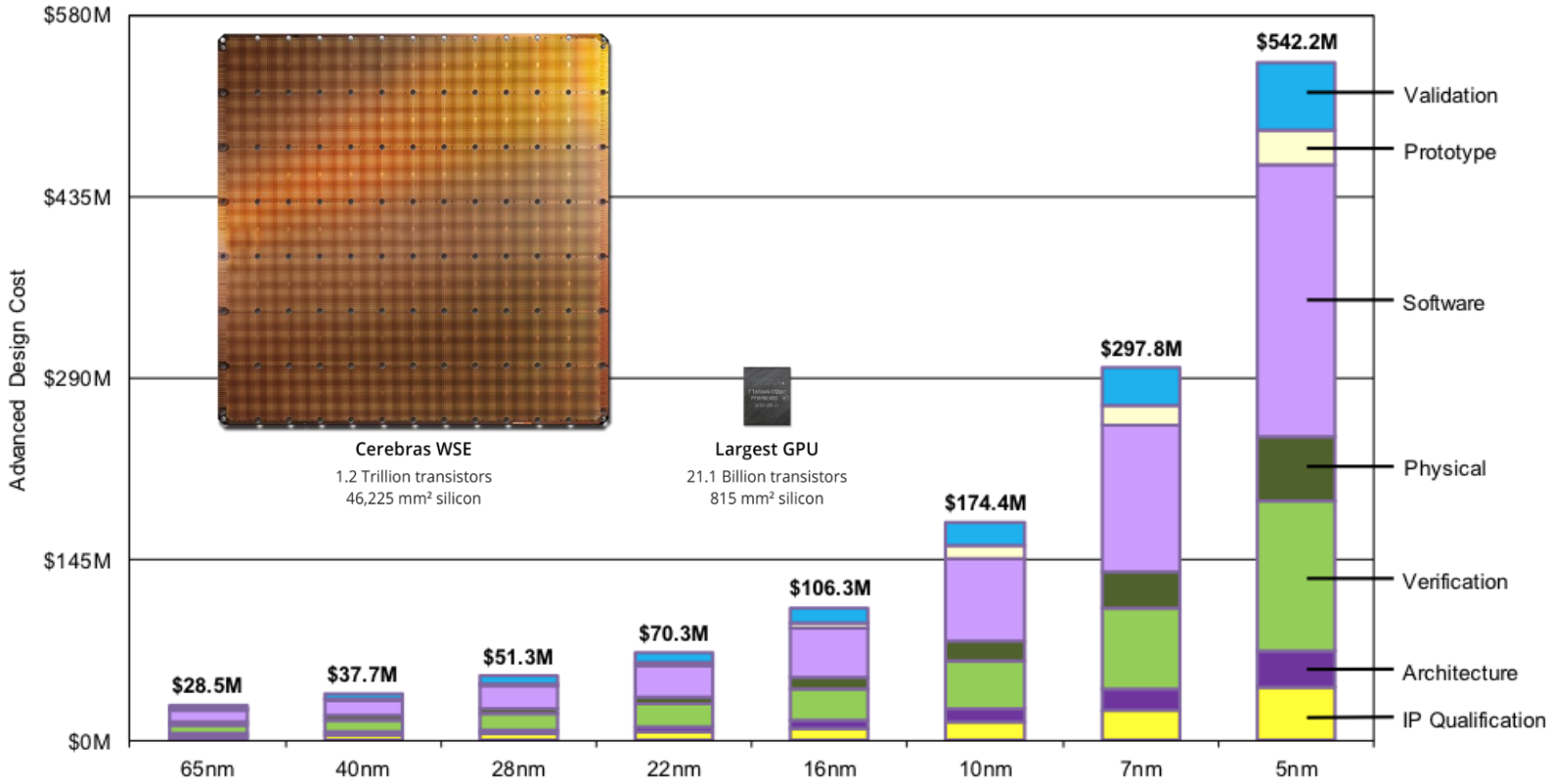


Recall 2024: News from the outer world: Waferscale Computing

https://www.nextplatform.com/2024/05/15/one-cerebras-wafer-beats-an-exascale-super-at-molecular-dynamics/







"The reason to think about wafer-scale, and making it work, is that it is a way around the chief barrier to higher computer performance: **off-chip communication**."

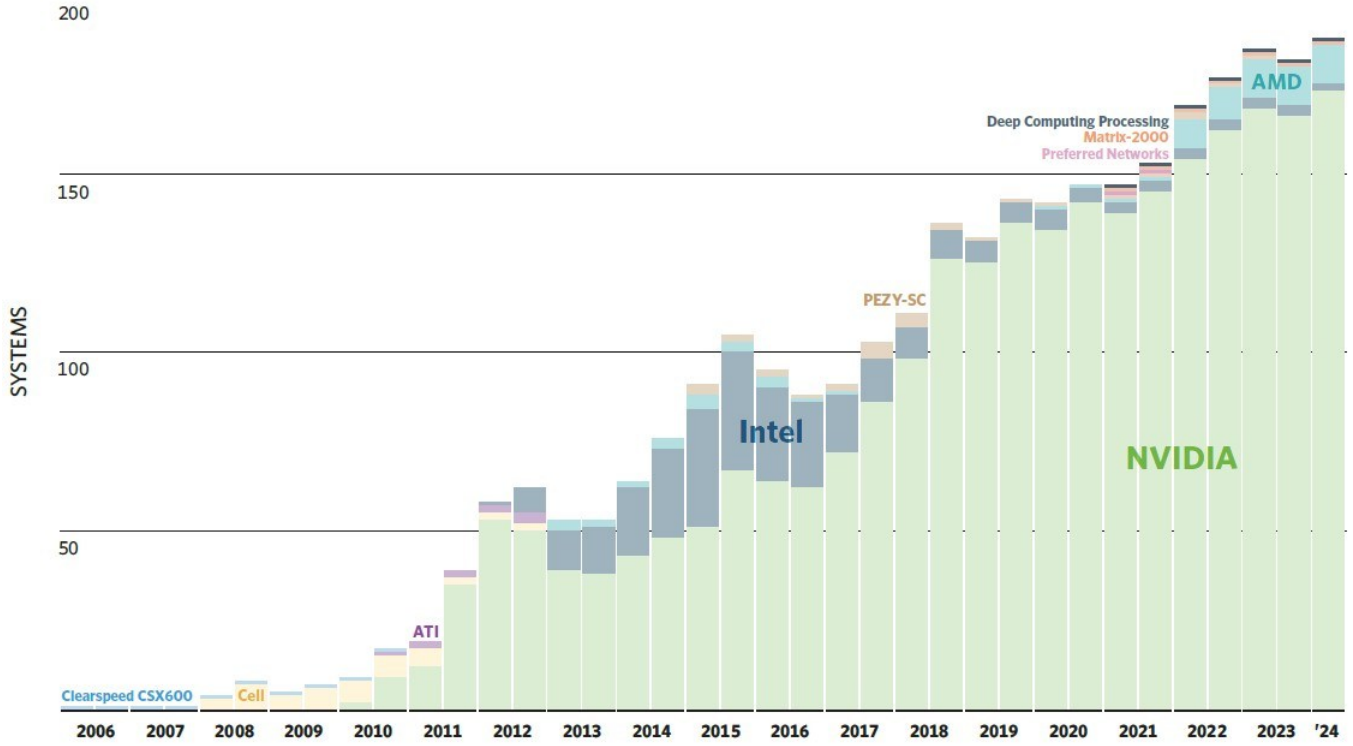
Recall 2024: News from the outer world: Top500 Nov 24 List

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
5	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461
6	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
7	**Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cray OS, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	2,121,600	434.90	574.84	7,124
8	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
9	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN EuroHPC/CINECA Italy	1,824,768	241.20	306.31	7,494
10	**Tuolumne** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	1,161,216	208.10	288.88	3,387

https://top500.org/lists/top500/2024/11/

Recall 2024: News from the outer world: Top500 Supercomputers at ICS'24

https://www.nextplatform.com/2024/05/13/top500-supers-this-is-peak-nvidia-for-accelerated-supercomputers/

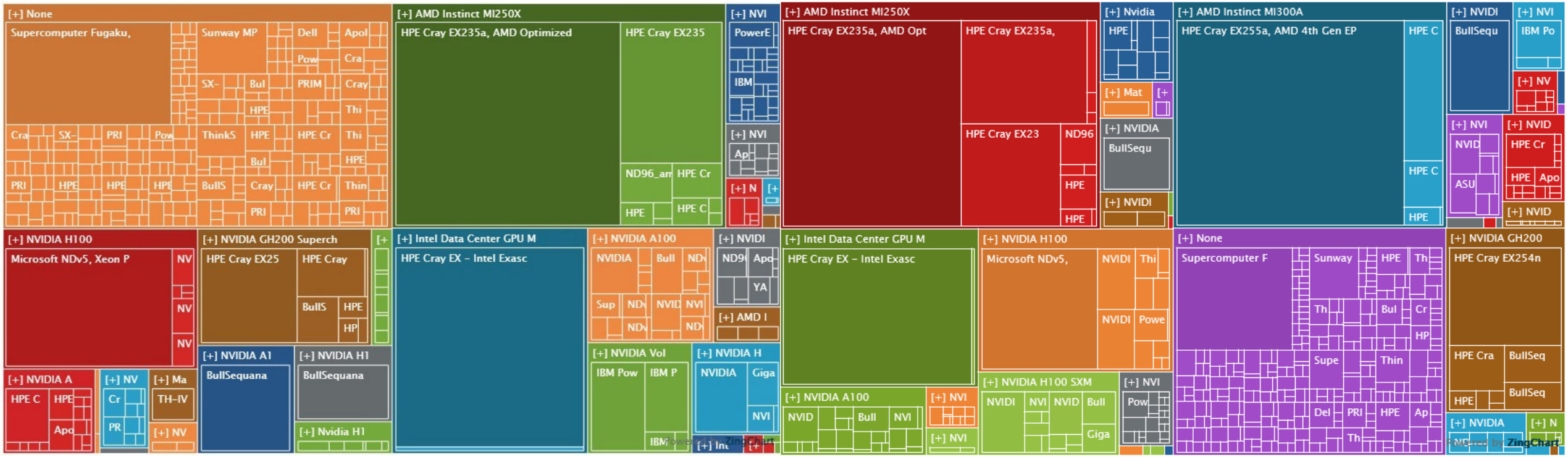


If you look at the supercomputing machines that are accelerated

TOP 500 The List.

https://top500.org/

Recall 2024: News from the outer world: Top500 May 24 vs Nov 24 Accelerators/Co-processors



distribution of Rpeak of accelerated and non-accelerated machines on the list.

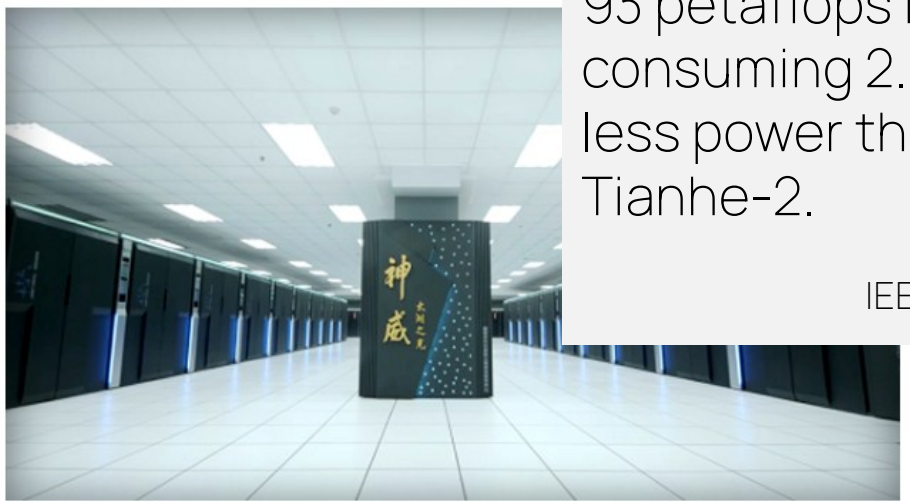
https://top500.org/

News from the outer world: Pay attention to the Optimization Bias!

Meet the new world's fastest supercomputer: China's TaihuLight

By Jamie Lendino on June 20, 2016 at 9:49 am | 77 Comments

1.1K shares



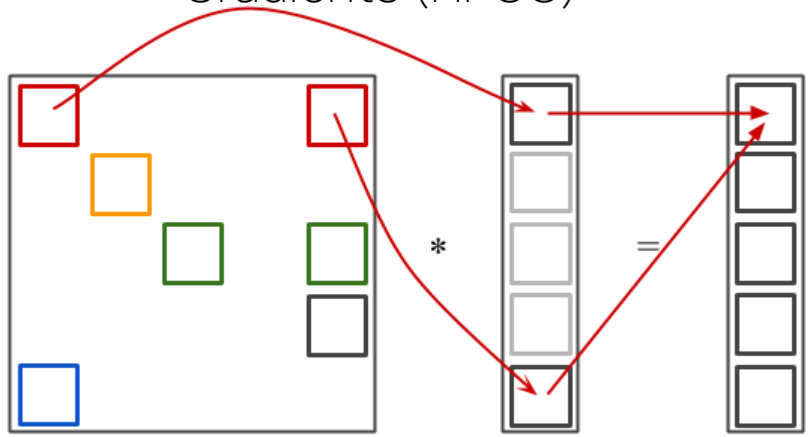
China has done it again — except this time with a brand new supercomputer. The Sunway TaihuLight is now the fastest system in the world, according to the twice-per-year TOP500 list, with a stunning Linpack benchmark result of 93 petaflops. That makes it three times faster than the prior champion, China's Tianhe-2, which we've covered numerous times on ExtremeTech and had sat on top of the list since it first went online in 2013.

What's even more interesting this time around is what's under TaihuLight's hood: a locally developed ShenWei processor and custom interconnect, instead of parts sourced elsewhere. The ShenWei 26010 is a 260-core, 64-bit RISC chip that exceeds 3 teraflops at

TaihuLight achieves
93 petaflops in Linpack
consuming 2.4 megawatts
less power than
Tianhe-2.

Rachel Courtland in
IEEE Spectrum, August 2016

High-Performance Conjugate Gradients (HPCG)



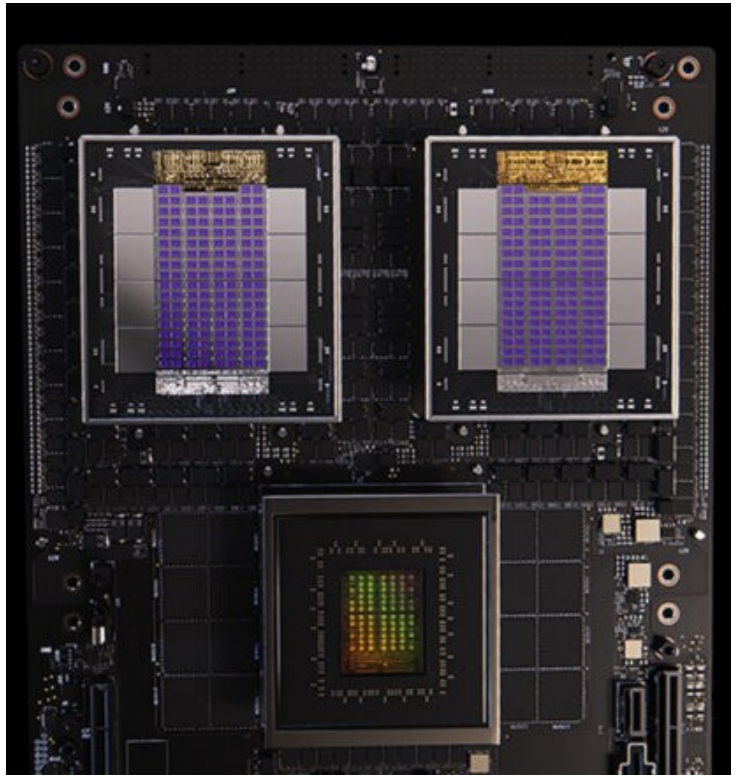
- Sparse matrix-vector multiplication.
- Sparse triangular solver.
- Vector updates.
- Global dot products.
- Local symmetric Gauss-Seidel smoother.

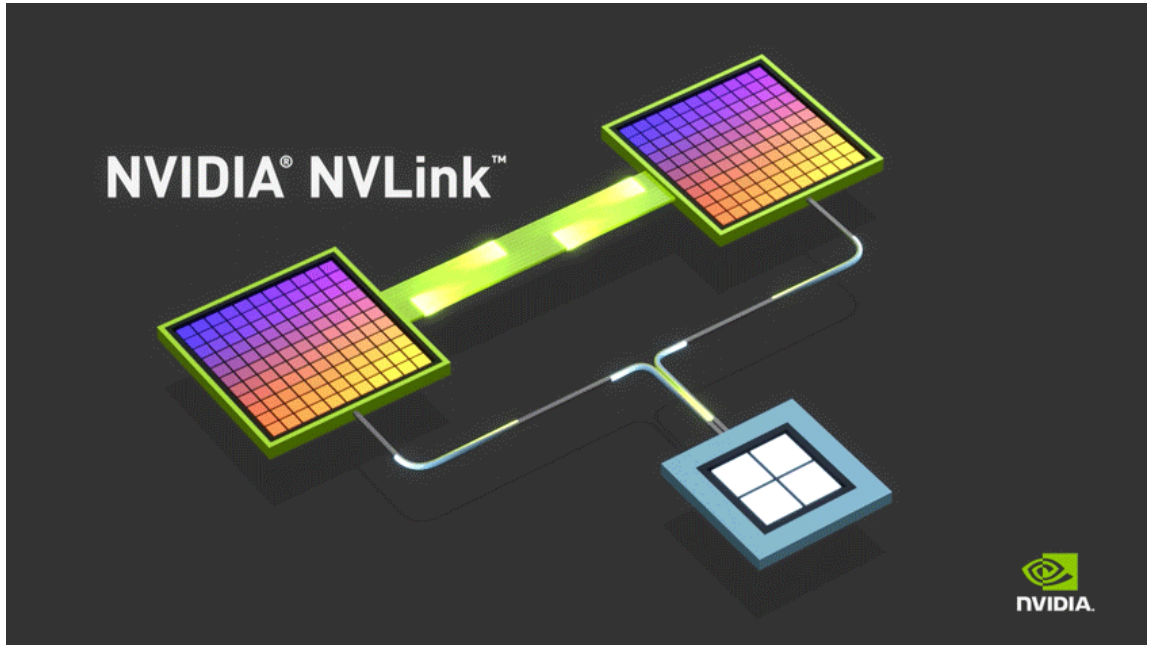
TaihuLight achieves 0.3 percent of its theoretical peak performance in HPCG.

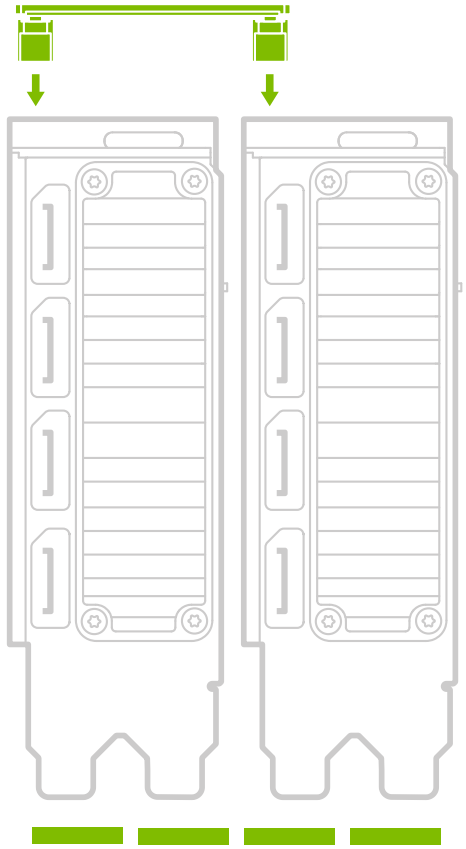
Rachel Courtland, IEEE Spectrum, Aug. 2016.

News from the outer world:
Nvidia Licenses NVLink Memory Ports To CPU And Accelerator Makers

https://www.nextplatform.com/2025/05/19/nvidia-licenses-nvlink-memory-ports-to-cpu-and-accelerator-makers/







https://www.nvidia.com/it-it/design-visualization/nvlink-bridges/

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

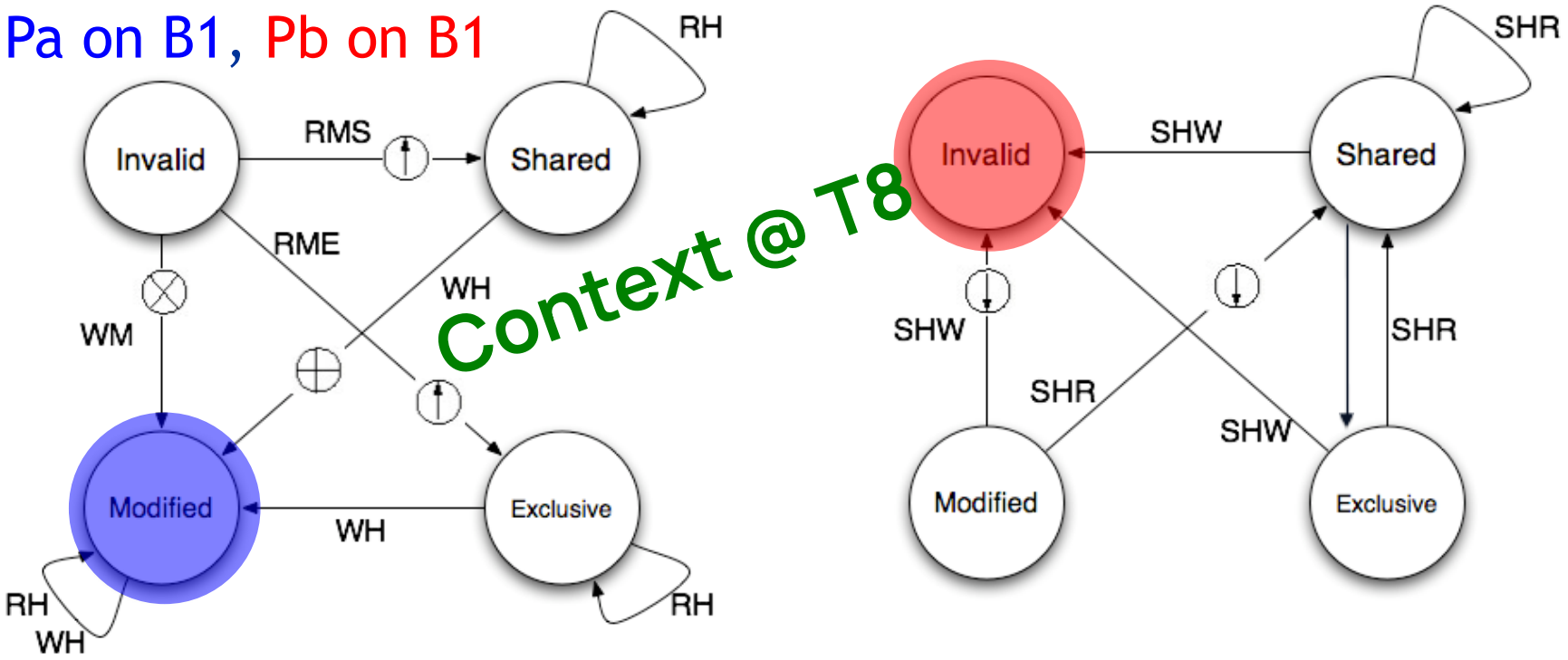
Exe 1: Cache Coherency

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

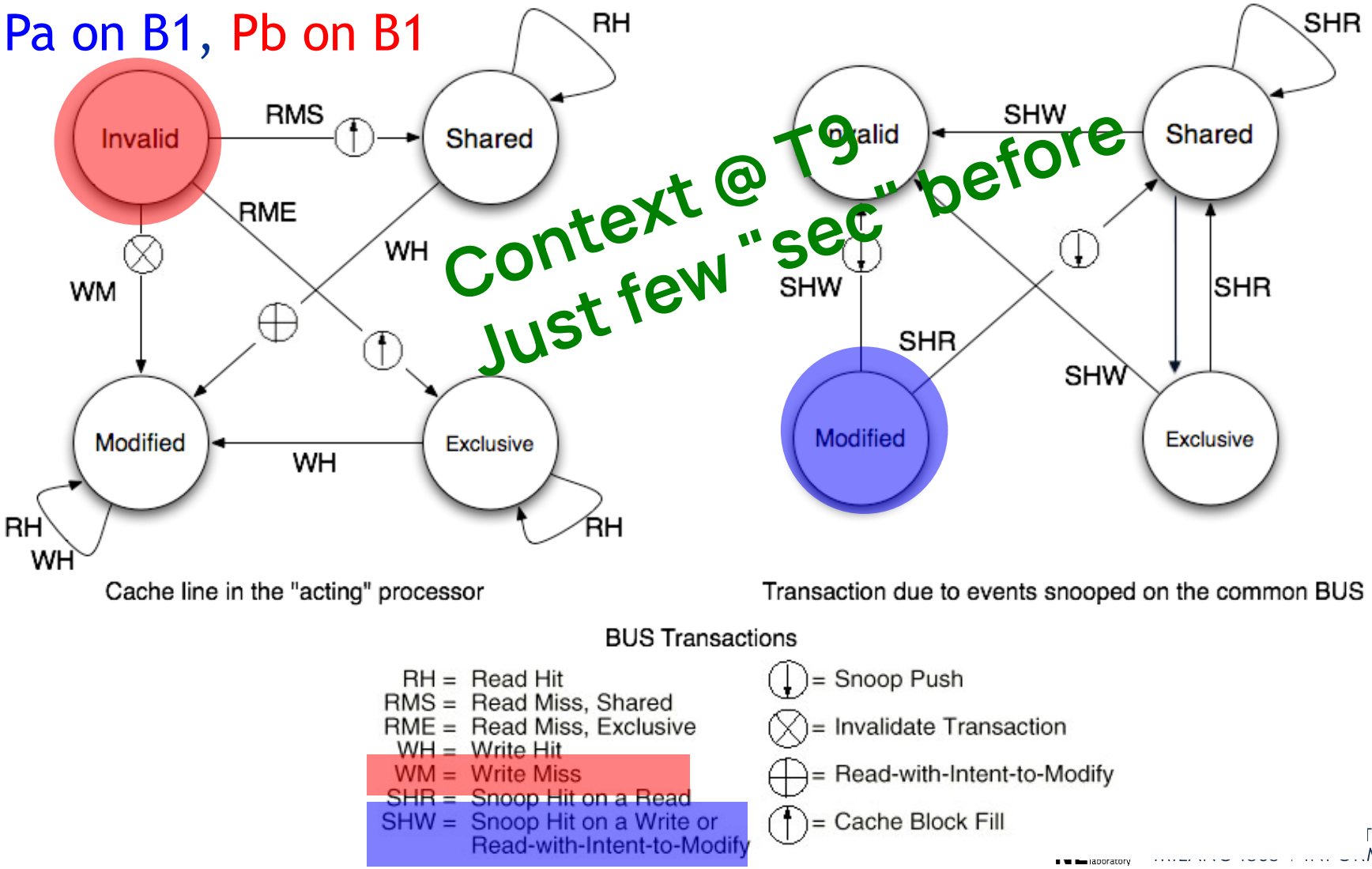
Exe 1: Cache Coherency

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

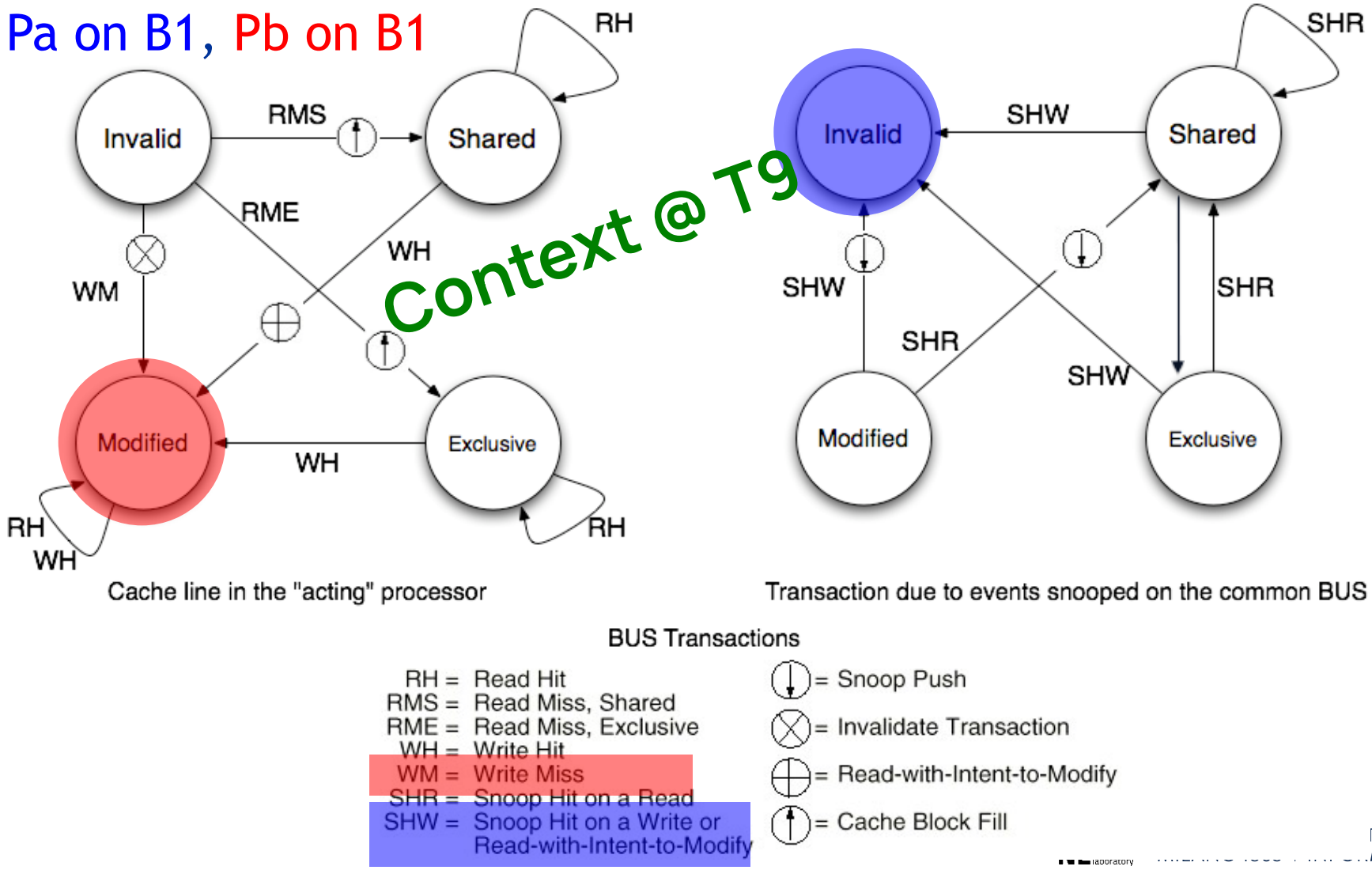
@T9: P1: write block 1



@T9: P1: write block 1



@T9: P1: write block 1



Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe 1: Cache Coherency

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

Exe: 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 : 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

Exe : Cache Coherency

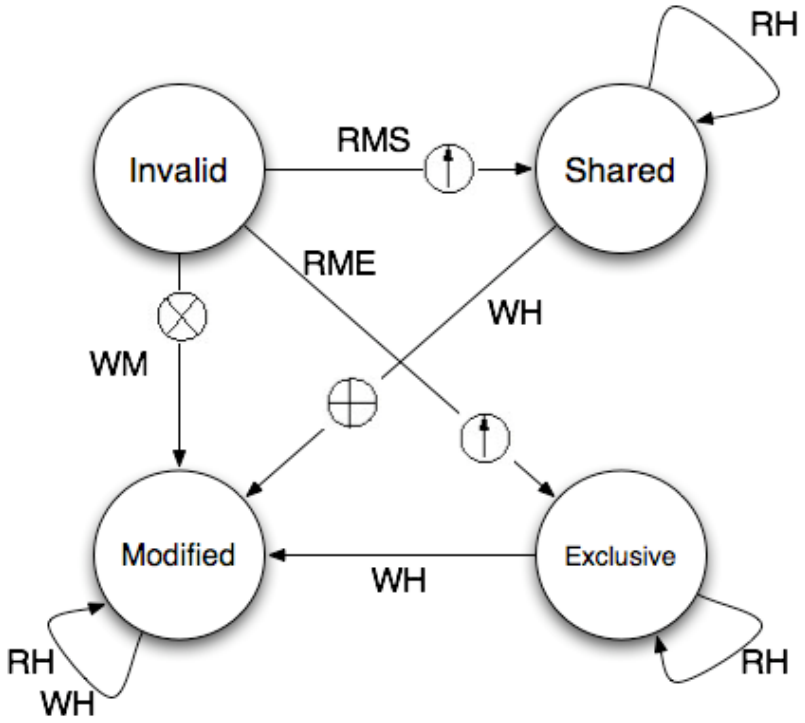
Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: read block 1	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?

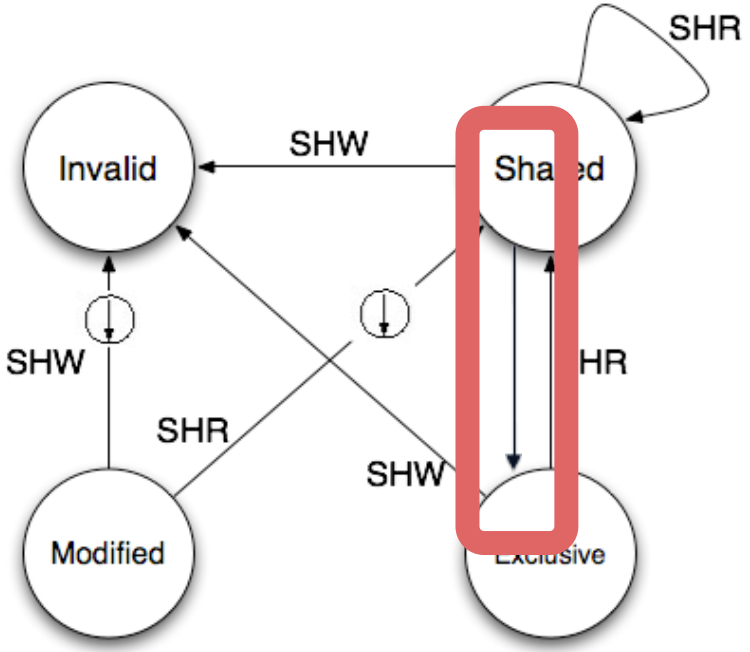
Exe : Cache Coherency

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

33

States of cache lines with MESI

	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

Exam Simulation for Practical Exe part

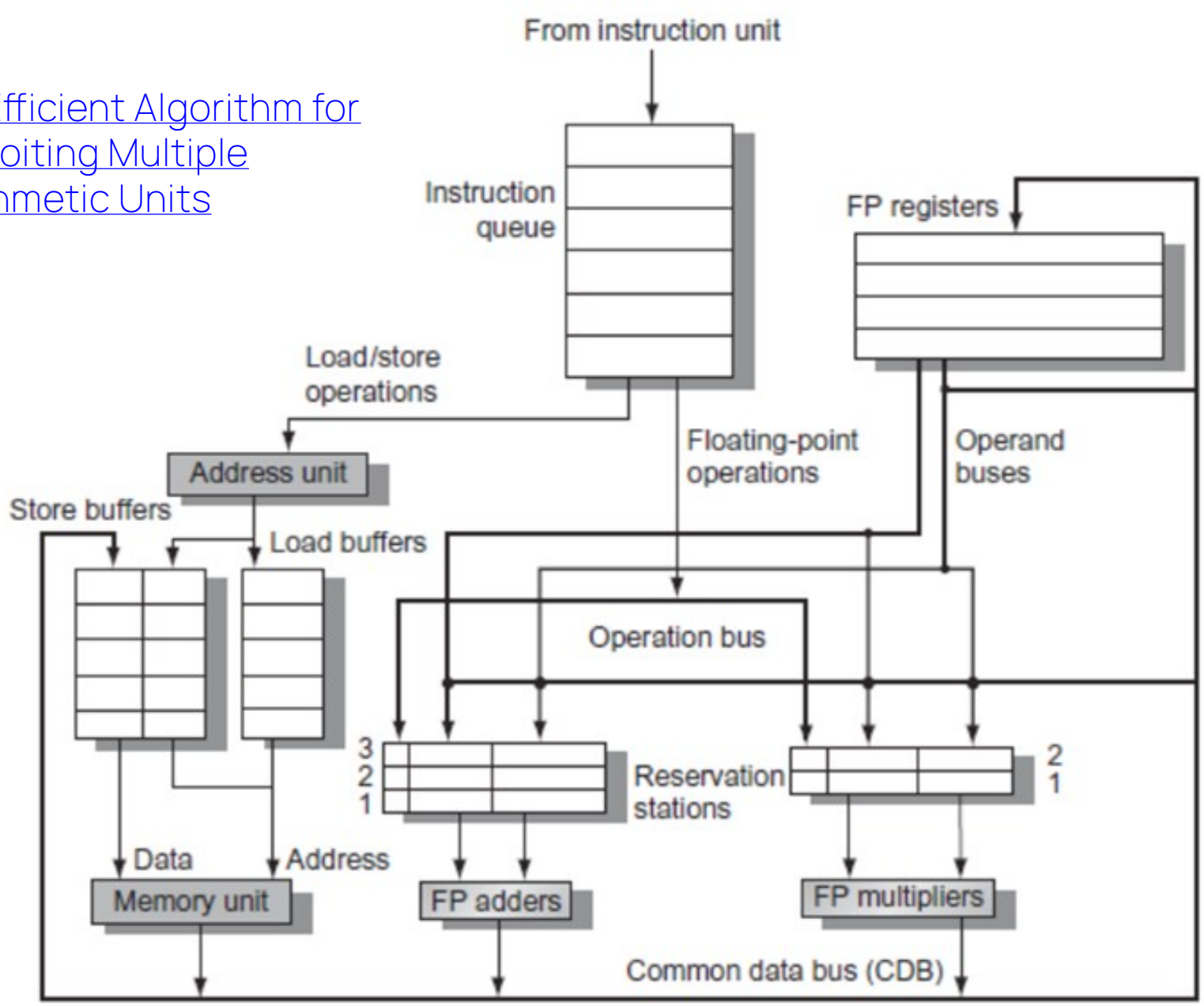
Problem 1: Tomasulo execution

Problem 2: Cache Coherency

Problem 3: VLIW

Exe Tomasulo

[An Efficient Algorithm for Exploiting Multiple Arithmetic Units]



Recall: the Tomasulo pipeline

ISSUE	EXECUTION	WRITE
Get Instruction from Queue and Rename Registers	Execute and Watch CDB;	Write on CDB;
Structural RSs check; WAW and WAR solved by Renaming (!!!in-order-issue!!!);	Check for Struct on FUs; RAW delaying; Struct check on CDB;	(FUs will hold results unless CDB free) RSs/FUs marked free

Exe .1 Tomasulo: Code

```
I1:  lw $f1, 0($r0)
I2:  faddi $f1, $f1, C1
I3:  faddi $f2, $f1, C2
I4:  sw $f2, 0($r0)
I5:  lw $f2, 4($r0)
I6:  fadd $f2, $f2, $f2
I7:  sw $f2, 4($r0)
```

Exe .1 Tomasulo: Conflicts

```
I1:   lw $f1, 0($r0)
I2:   faddi $f1, $f1, C1
I3:   faddi $f2, $f1, C2
I4:   sw $f2, 0($r0)
I5:   lw $f2, 4($r0)
I6:   fadd $f2, $f2, $f2
I7:   sw $f2, 4($r0)
```

RAW f1 I1-I2
RAW f1 I2-I3
RAW f2 I3-I4
RAW f2 I5-I6
RAW f2 I6-I7

WAW f2 I5-I6
WAW f2 I5-I3
WAW f1 I2-I1
WAW f2 I6-I3
WAR f2 I4-I5
WAR f2 I4-I6

Exe 3.2 Tomasulo

- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 4
- 2 RESERVATION STATIONS (RS3, RS4) + 1 ALU/BR FUs (ALU1) with latency 2

Instruction	ISSUE	START EXE	WB	Hazards Type	RSi	Unit
I1:`lw $f1, 0($r0)`						
I2:`faddi $f1, $f1, C1`						
I3:`faddi $f2, $f1, C2`						
I4:`sw $f2, 0($r0)`						
I5:`lw $f2, 4($r0)`						
I6:`fadd $f2, $f2, $f2`						
I7:`sw $f2, 4($r0)`						

RAW f1 I1-I2 **RAW f1 I2-I3** **RAW f2 I3-I4** **RAW f2 I5-I6** **RAW f2 I6-I7**

Exe 3.2 Tomasulo

- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 4
- 2 RESERVATION STATIONS (RS3, RS4) + 1 ALU/BR FUs (ALU1) with latency 2

Instruction	ISSUE	START EXE	WB	Hazards Type	RSi	Unit
I1:`lw $f1, 0($r0)`	1	2	6		RS1	LDU1
I2:`faddi $f1, $f1, C1`	2	7	9	RAW $f1	RS3	ALU1
I3:`faddi $f2, $f1, C2`	3	10	12	RAW $f1 (struct ALU1)	RS4	ALU1
I4:`sw $f2, 0($r0)`	4	13	17	RAW $f2 (struct LDU1)	RS2	LDU1
I5:`lw $f2, 4($r0)`	7	18	22	struct RS1 + struct LDU1	RS1	LDU1
I6:`fadd $f2, $f2, $f2`	10	23	25	struct RS3 + RAW $f2 (struct ALU1)	RS3	ALU1
I7:`sw $f2, 4($r0)`	18	26	30	struct RS2 + RAW $f2 (struct LDU1)	RS2	LDU1

RAW f1 I1-I2 **RAW f1 I2-I3** **RAW f2 I3-I4** **RAW f2 I5-I6** **RAW f2 I6-I7**

Exe 3.2 Tomasulo

- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 4
- 2 RESERVATION STATIONS (RS3, RS4) + 1 ALU/BR FUs (ALU1) with latency 2

Instruction	ISSUE	START EXE	WB	Hazards Type	RSi	Unit
I1:`lw $f1, 0($r0)`	1	2	6		RS1	LDU1
I2:`faddi $f1, $f1, C1`	2	7	9	RAW $f1	RS3	ALU1
I3:`faddi $f2, $f1, C2`	3	10	12	RAW $f1 (struct ALU1)	RS4	ALU1
I4:`sw $f2, 0($r0)`	4	13	17	RAW $f2 (struct LDU1)	RS2	LDU1
I5:`lw $f2, 4($r0)`	7	18	22	struct RS1 + struct LDU1	RS1	LDU1
I6:`fadd $f2, $f2, $f2`	10	23	25	struct RS3 + RAW $f2 (struct ALU1)	RS3	ALU1
I7:`sw $f2, 4($r0)`	18	26	30	struct RS2 + RAW $f2 (struct LDU1)	RS2	LDU1

RAW f1 I1-I2 **RAW f1 I2-I3** **RAW f2 I3-I4** **RAW f2 I5-I6** **RAW f2 I6-I7**

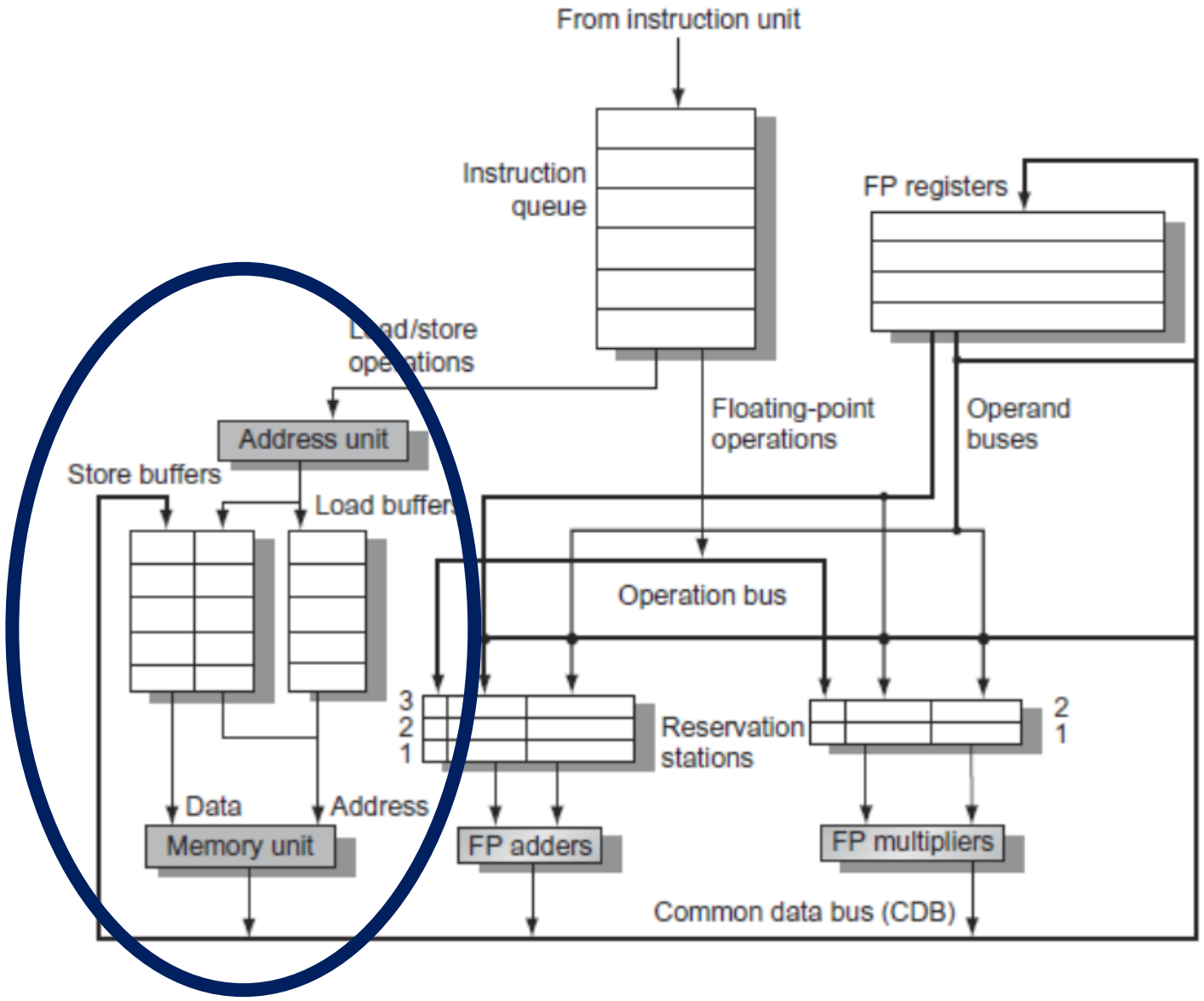
Recall: 3-stages of the Tomasulo Algorithm: EXECUTION

- When both **operands** are **ready** then execute.
- If not ready, **watch the common data bus** for results.
  - By delaying execution until operands are available, RAW hazards are avoided.
    Notice that several instructions could become ready in the same clock cycle for the same FU.
- *Load and stores: two-step process*
  - *First step: **compute effective address**, **place** it in load or store **buffer**.*
  - *Loads in Load Buffer execute as soon as memory unit is available*
  - *Stores in store buffer wait for the value to be stored before being sent to memory unit.*

*Load and stores **are maintained in program order through the effective address calculation***

→to prevent hazard in memory

Tomasulo Algorithm Another View



Exe 3.2 Tomasulo

- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 4
- 2 RESERVATION STATIONS (RS3, RS4) + 1 ALU/BR FUs (ALU1) with latency 2

Instruction	ISSUE	START EXE	WB	Hazards Type	RSi	Unit
I1:`lw $f1, 0($r0)`						
I2:`faddi $f1, $f1, C1`						
I3:`faddi $f2, $f1, C2`						
I4:`sw $f2, 0($r0)`						
I5:`lw $f2, 4($r0)`						
I6:`fadd $f2, $f2, $f2`						
I7:`sw $f2, 4($r0)`						

RAW f1 I1-I2 **RAW f1 I2-I3** **RAW f2 I3-I4** **RAW f2 I5-I6** **RAW f2 I6-I7**

Exe 3.2 Tomasulo

- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 4
- 2 RESERVATION STATIONS (RS3, RS4) + 1 ALU/BR FUs (ALU1) with latency 2

Instruction	ISSUE	START EXE	WB	Hazards Type	RSi	Unit
I1:`lw $f1, 0($r0)`	1	2	6		RS1	LDU1
I2:`faddi $f1, $f1, C1`	2	7	9	RAW $f1	RS3	ALU1
I3:`faddi $f2, $f1, C2`	3	10	12	RAW $f1 (struct ALU1)	RS4	ALU1
I4:`sw $f2, 0($r0)`	4	14	18	RAW $f2 + struct LDU1	RS2	LDU1
I5:`lw $f2, 4($r0)`	7	8	13	struct RS1 + struct CDB	RS1	LDU1
I6:`fadd $f2, $f2, $f2`	10	14	16	struct RS3 + RAW $f2 (struct ALU1)	RS3	ALU1
I7:`sw $f2, 4($r0)`	14	19	23	struct RS1 + RAW $f2 + struct LDU1	RS1	LDU1

RAW f1 I1-I2 **RAW f1 I2-I3** **RAW f2 I3-I4** **RAW f2 I5-I6** **RAW f2 I6-I7**

Exe 3.2 Tomasulo

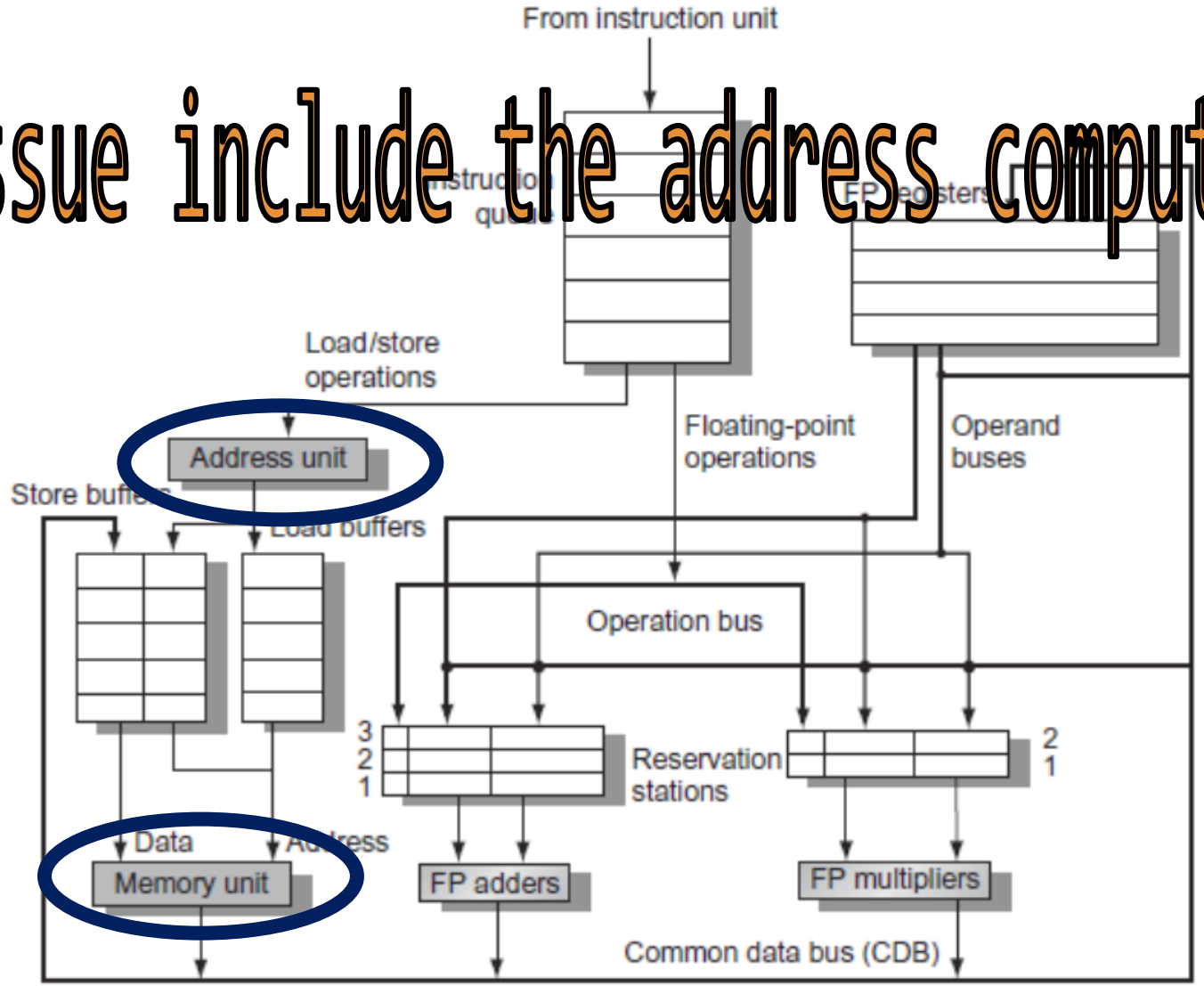
- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 4
- 2 RESERVATION STATIONS (RS3, RS4) + 1 ALU/BR FUs (ALU1) with latency 2

Instruction	ISSUE	START EXE	WB	Hazards Type			RSi	Unit
I1:`lw $f1, 0($r0)`	1	2	6	1	2	6	RS1	LDU1
I2:`faddi $f1, $f1, C1`	2	7	9	2	7	9	RS3	ALU1
I3:`faddi $f2, $f1, C2`	3	10	12	3	10	12	RS4	ALU1
I4:`sw $f2, 0($r0)`	4	14	18	4	13	17	RS2	LDU1
I5:`lw $f2, 4($r0)`	7	8	13	7	18	22	RS1	LDU1
I6:`fadd $f2, $f2, $f2`	10	14	16	10	23	25	RS3	ALU1
I7:`sw $f2, 4($r0)`	14	19	23	18	26	30	RS1	LDU1
				+ struct LDU1				

RAW f1 I1-I2 **RAW f1 I2-I3** **RAW f2 I3-I4** **RAW f2 I5-I6** **RAW f2 I6-I7**

Tomasulo Algorithm Another View

Does the Issue include the address computation or not?



Tomasulo algorithm (1/3)

- Loads and stores go through a **functional unit** for effective **address computation** before proceeding to effective load and store buffers;
- Loads take a **second execution step** to **access memory**, **then** go to **Write Result** to send the value **from memory to RF and/or RS**;
- **Stores complete** their execution in their **Write Result stage** (writes data to memory)
- All writes occur in Write Result – simplifying Tomasulo algorithm.

Tomasulo algorithm (2/3)

- A Load and a Store can be done in **different order**, provided they access **different memory locations**
- Otherwise, a WAR (interchange load-store sequence) or a RAW (interchange store-load sequence) may result (WAW if two stores are interchanged).

Loads can be reordered freely.

- To detect such hazards: **data memory addresses** associated with any **earlier memory operation** must have been **computed** by the **CPU**

(i.e., **address computation executed in program order**)

Tomasulo algorithm (3/3)

- **Load** executed **out of order** with previous **store**: assume **address computed** in program order. When Load address has been computed, it can be compared with A fields in active Store buffers: in the case of a **match**, **Load is not sent** to Load buffer until conflicting store completes.
- **Stores** must check for **matching addresses** in both **Load** and **Store buffers** (dynamic disambiguation, alternative to static disambiguation performed by the compiler)
- Drawback: **amount of hardware** required.
- Each **RS** must contain a **fast associative buffer**; **single CDB** may limit performance.

Exe 3.2 Tomasulo

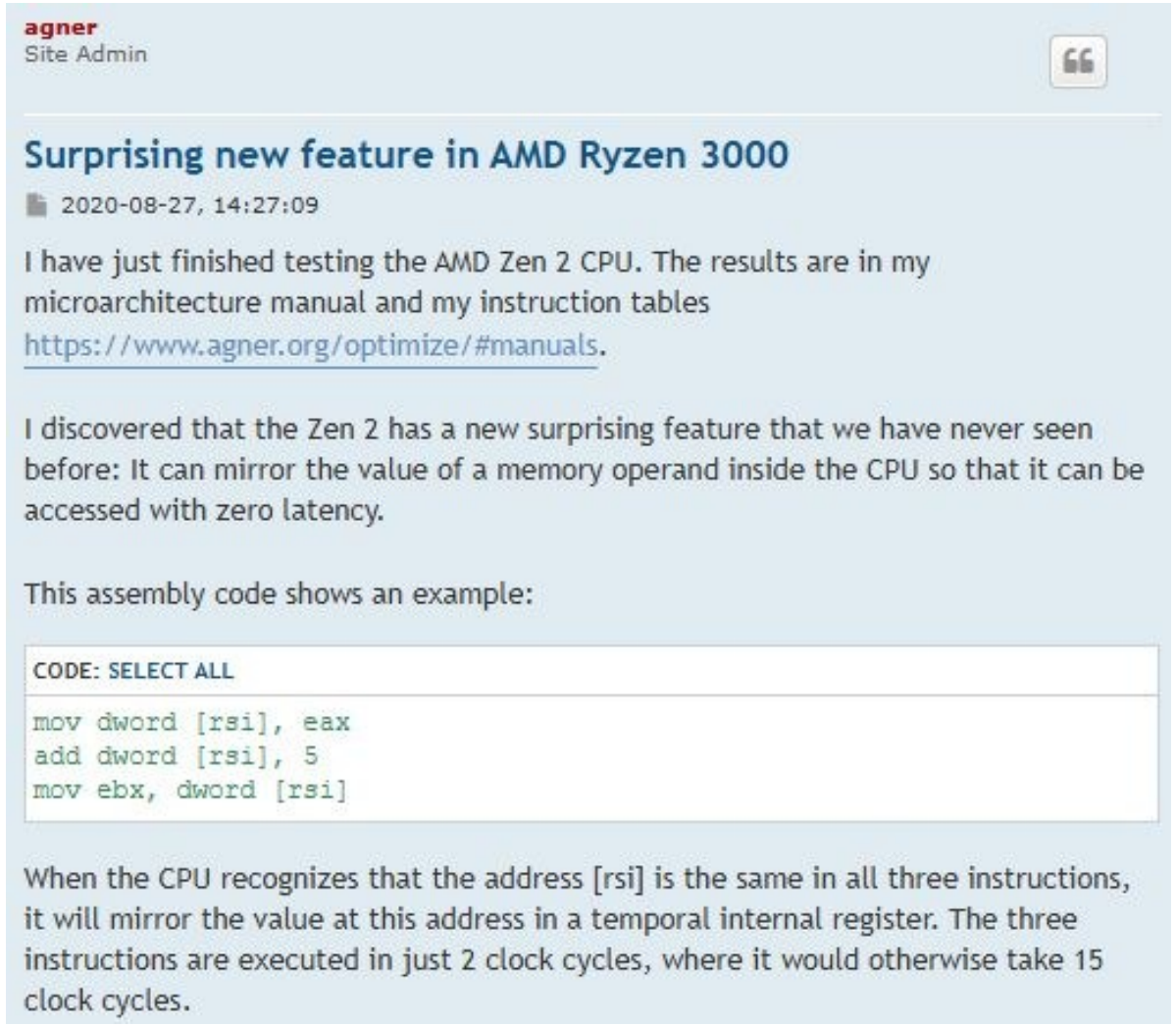
- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 4
- 2 RESERVATION STATIONS (RS3, RS4) + 1 ALU/BR FUs (ALU1) with latency 2

Instruction	ISSUE	START EXE	WB	Hazards Type			RSi	Unit
I1:`lw $f1, 0($r0)`	1	2	6	1	2	6	RS1	LDU1
I2:`faddi $f1, $f1, C1`	2	7	9	2	7	9	RS3	ALU1
I3:`faddi $f2, $f1, C2`	3	10	12	3	10	12	RS4	ALU1
I4:`sw $f2, 0($r0)`	4	14	18	4	13	17	RS2	LDU1
I5:`lw $f2, 4($r0)`	7	8	13	7	18	22	RS1	LDU1
I6:`fadd $f2, $f2, $f2`	10	14	16	10	23	25	RS3	ALU1
I7:`sw $f2, 4($r0)`	14	19	23	18	26	30	RS1	LDU1
				+ struct LDU1				

RAW f1 I1-I2 **RAW f1 I2-I3** **RAW f2 I3-I4** **RAW f2 I5-I6** **RAW f2 I6-I7**

News from the outer world: AMD Memory Renaming (not yet documented)

https://www.agner.org/forum/viewtopic.php?f=1&t=41



agner
Site Admin

Surprising new feature in AMD Ryzen 3000

2020-08-27, 14:27:09

I have just finished testing the AMD Zen 2 CPU. The results are in my microarchitecture manual and my instruction tables https://www.agner.org/optimize/#manuals.

I discovered that the Zen 2 has a new surprising feature that we have never seen before: It can mirror the value of a memory operand inside the CPU so that it can be accessed with zero latency.

This assembly code shows an example:

CODE: SELECT ALL

```
mov dword [rsi], eax
add dword [rsi], 5
mov ebx, dword [rsi]
```

When the CPU recognizes that the address [rsi] is the same in all three instructions, it will mirror the value at this address in a temporal internal register. The three instructions are executed in just 2 clock cycles, where it would otherwise take 15 clock cycles.

Memory renaming is a microarchitectural optimization that recognizes dataflow through a store and a load and reassigns the communication to a physical register.

The latency of the load then becomes truly ZERO cycles as the producing instruction can forward directly to the consuming instruction.

Of course, the CPU has to recover from the possibility that an aliased pointer may touch the same storage, and AMD's implementation does detect this correctly.

https://www.linkedin.com/posts/prof-todd-austin_computerarchitecture-microarchitecture-activity-7297717375209017344-tUEq/

[1] Basically the same as register renaming, but instead of using the register file to rename architectural registers, it can rename memory instead.

https://news.ycombinator.com/item?id=24302057

Exe 2: Cache Coherency

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

Assume the **MESI protocol** is used, with **write-back** caches, **write-allocate,** and **write-invalidate** of other caches.

Exe 2: Cache Coherency

Time	After Operation	Px cache block state	Py cache block state	Pz cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?	Memory at block 2 up to date?
0	Px read block 0	Exclusive (0)	Invalid	Modified (2)	yes	yes	no
1	Py write block 2						
2	Px read block 2						
3	Pz read block 0						
4	Pz write block 2						
5	Px read block 2						
6	Py read block 2						
7	Pz write block 2						
8	Px write block 1						
9	Py read block 1						
10	Pz: read block 1						

Exe 2: Cache Coherency

Time	After Operation	Px cache block state	Py cache block state	Pz cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?	Memory at block 2 up to date?
0	Px read block 0	Exclusive (0)	Invalid	Modified (2)	yes	yes	no
1	Py write block 2	Exclusive (0)	Modified (2)	Invalid (2)	yes	yes	no
2	Px read block 2	Shared (2)	Shared (2)	Invalid (2)	yes	yes	yes
3	Pz read block 0	Shared (2)	Shared (2)	Exclusive (0)	yes	yes	yes
4	Pz write block 2	Invalid (2)	Invalid (2)	Modified (2)	yes	yes	no
5	Px read block 2	Shared (2)	Invalid (2)	Shared (2)	yes	yes	yes
6	Py read block 2	Shared (2)	Shared (2)	Shared (2)	yes	yes	yes
7	Pz write block 2	Invalid (2)	Invalid (2)	Modified (2)	yes	yes	no
8	Px write block 1	Modified (1)	Invalid (2)	Modified (2)	yes	no	no
9	Py read block 1	Shared (1)	Shared (1)	Modified (2)	yes	yes	no
10	Pz: read block 1	Shared (1)	Shared (1)	Shared (1)	yes	yes	yes

Exe3 VLIW: Architecture

Details about the **5-operations** VLIW machine with 5 fully **pipelined multi-cycle** functional units:

- 1 **Integer ALU** with **1 cycle latency**
- 2 **Memory Unit** with **2 cycles latency**
- 1 **Floating Point Unit** SUM with **3 cycles latency**
- 1 **FPU** MUL with **4 cycles latency**

- Schedule one iteration of the loop on the 4-opsVLIW machine (we don't write the NOPs) with **ASAP** and **pipelined** FUs.

- **Do not** use neither software pipelining nor loop unrolling nor modifying loop indexes
- Compute FLOPs/CC

Exe3 VLIW: Code

```
NBD:ld $f1,FORCE($fp)
    ld $f0,DY($fp)
    mul.d   $f0,$f1,$f0
    ld      $f1,FY($fp)
    add.d   $f0,$f1,$f0
    sd      $f0,FY($fp)
    ld      $f1,FORCE($fp)
    ld      $f0,DZ($fp)
    mul.d   $f0,$f1,$f0
    ld      $f1,FZ($fp)
    add.d   $f0,$f1,$f0
    sd      $f0,FZ($fp)
    lw      $2,28($fp)
    addi    $2,$2,1
    sw      $2,28($fp)
    bne     $2,$8, NBD
```

VLIW: schedule

```
BD:ld $f1,FORCE($fp)
   ld $f0,DY($fp)
   mul.d   $f0,$f1,$f0
   ld      $f1,FY($fp)
   add.d   $f0,$f1,$f0
   sd      $f0,FY($fp)
   ld      $f1,FORCE($fp)
   ld      $f0,DZ($fp)
   mul.d   $f0,$f1,$f0
   ld      $f1,FZ($fp)
   add.d   $f0,$f1,$f0
   sd      $f0,FZ($fp)
   lw      $2,28($fp)
   addi    $2,$2,1
   sw      $2,28($fp)
   bne     $2,$8, NBD
```

	Integer ALU (1 cc)	Memory Unit (2 cc)	Memory Unit (2 cc)	FPU+ (3 cc)	FPUx (4cc)
C1					
C2					
C3					
C4					
C5					
C6					
C7					
C8					
C9					
C10					
C11					
C12					
C13					
C14					
C15					
C16					
C17					
C18					
C19					

VLIW: schedule

```
BD:ld $f1,FORCE($fp)
   ld $f0,DY($fp)
   mul.d   $f0,$f1,$f0
   ld      $f1,FY($fp)
   add.d   $f0,$f1,$f0
   sd      $f0,FY($fp)
   ld      $f1,FORCE($fp)
   ld      $f0,DZ($fp)
   mul.d   $f0,$f1,$f0
   ld      $f1,FZ($fp)
   add.d   $f0,$f1,$f0
   sd      $f0,FZ($fp)
   lw      $2,28($fp)
   addi    $2,$2,1
   sw      $2,28($fp)
   bne     $2,$8, NBD
```

	Integer ALU (1 cc)	Memory Unit (2 cc)	Memory Unit (2 cc)	FPU+ (3 cc)	FPUx (4cc)
C1		`ld $f1,FORCE($fp)`	`ld $f0,DY($fp)`		
C2		`lw $2,28($fp)`			
C3		`ld $f1,FY($fp)`			`mul.d $f0,$f1,$f0`
C4	`addi $2,$2,1`				
C5		`sw $2,28($fp)`			
C6					
C7		`ld $f1,FORCE($fp)`		`add.d $f0,$f1,$f0`	
C8					
C9					
C10		`ld $f0,DZ($fp)`	`sd $f0,FY($fp)`		
C11					
C12			`ld $f1,FZ($fp)`		`mul.d $f0,$f1,$f0`
C13					
C14					
C15					
C16				`add.d $f0,$f1,$f0`	
C17					
C18					
C19	`bne $2,$8, NBD`		`sd $f0,FZ($fp)`		

VLIW: schedule v2

```
BD:ld $f1,FORCE($fp)
   ld $f0,DY($fp)
   mul.d   $f0,$f1,$f0
   ld      $f1,FY($fp)
   add.d   $f0,$f1,$f0
   sd      $f0,FY($fp)
   ld      $f1,FORCE($fp)
   ld      $f0,DZ($fp)
   mul.d   $f0,$f1,$f0
   ld      $f1,FZ($fp)
   add.d   $f0,$f1,$f0
   sd      $f0,FZ($fp)
   lw      $2,28($fp)
   addi    $2,$2,1
   sw      $2,28($fp)
   bne     $2,$8, NBD
```

	Integer ALU (1 cc)	Memory Unit (2 cc)	Memory Unit (2 cc)	FPU+ (3 cc)	FPUx (4cc)
C1		ld $f1,FORCE($fp)	ld $f0,DY($fp)		
C2		lw $2,28($fp)	ld $f1,FY($fp)		
C3					mul.d $f0,$f1,$f0
C4	addi $2,$2,1				
C5		sw $2,28($fp)			
C6		ld $f1,FORCE($fp)			
C7				add.d $f0,$f1,$f0	
C8					
C9		ld $f0,DZ($fp)			
C10		ld $f1,FZ($fp)	sd $f0,FY($fp)		
C11					mul.d $f0,$f1,$f0
C12					
C13					
C14					
C15				add.d $f0,$f1,$f0	
C16					
C17					
C18	bne $2,$8, NBD		sd $f0,FZ($fp)		
C19					

Thanks for your attention

Davide Conficconi <davide.conficconi@polimi.it>

Acknowledgements

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

Part of this material comes from:

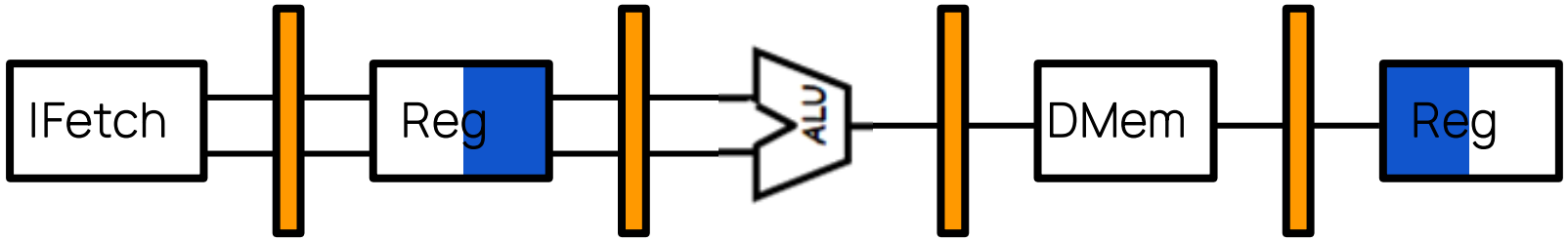
- “Computer Organization and Design” and “Computer Architecture A Quantitative Approach” Patterson and Hennessy books
- “Digital Design and Computer Architecture” Harris and Harris
- Elsevier Inc. online materials
- Papers/news cited in this lecture

Exe 3: Simple Pipelining

Exe 3 Simple Pipelining : the Code

```
I1:  addi $s3, $s2, 2
I2:  add $s5, $s4, $s3
I3:  sw $s5, 4($s3)
I4:  sub $s7, $s5, $s6
I5:  lw $s6, 4($s7)
```

Exe 3: Simple Pipelining: the Architecture



Exe 3.1 Simple Pipelining : Conflicts

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	F	D	E	M	W										
I2:	`add $s5, $s4, $s3`		F	D	E	M	W									
I3:	`sw $s5, 4($s3)`			F	D	E	M	W								
I4:	`sub $s7, $s5, $s6`				F	D	E	M	W							
I5:	`lw $s6, 4($s7)`					F	D	E	M	W						

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : solve as is

Istr	CK1	CK2	CK3	CK4	CK5	CK6	CK7	CK8	CK9	CK10	CK11
I1											
I2											
I3											
I4											
I5											
Istr	CK12	CK13	CK14	CK15	CK16	CK17	CK18	CK19	CK20	CK21	CK22
I1											
I2											
I3											
I4											
I5											

```
I1:  addi $s3, $s2, 2
I2:  sub $s4, $s3, $s1
I3:  add $s5, $s4, $s1
I4:  lw $s6, 4($s4)
I5:  sub $s7, $s4, $s6
```

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to **reduce the stalls**; Draw the pipeline schema showing all the data conflicts/dependencies.

Exe 3.3 Simple Pipelining : FWD Paths

Istr	CK1	CK2	CK3	CK4	CK5	CK6	CK7	CK8	CK9	CK10	CK11
I1											
I2											
I3											
I4											
I5											
Istr	CK12	CK13	CK14	CK15	CK16	CK17	CK18	CK19	CK20	CK21	CK22
I1											
I2											
I3											
I4											
I5											

```
I1:  addi $s3, $s2, 2
I2:  sub $s4, $s3, $s1
I3:  add $s5, $s4, $s1
I4:  lw $s6, 4($s4)
I5:  sub $s7, $s4, $s6
```

Exe 3.1 Simple Pipelining : Conflicts

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	F	D	E	M	W										
I2:	`add $s5, $s4, $s3`		F	D	E	M	W									
I3:	`sw $s5, 4($s3)`			F	D	E	M	W								
I4:	`sub $s7, $s5, $s6`				F	D	E	M	W							
I5:	`lw $s6, 4($s7)`					F	D	E	M	W						

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : Conflicts

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	F	D	E	M	W										
I2:	`add $s5, $s4, $s3`		F	D	E	M	W									
I3:	`sw $s5, 4($s3)`			F	D	E	M	W								
I4:	`sub $s7, $s5, $s6`				F	D	E	M	W							
I5:	`lw $s6, 4($s7)`					F	D	E	M	W						

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : Solve as is

	Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1:	`addi $s3, $s2, 2`	F	D	E	M	W										
I2:	`add $s5, $s4, $s3`		F	D	E	M	W									
I3:	`sw $s5, 4($s3)`			F	D	E	M	W								
I4:	`sub $s7, $s5, $s6`				F	D	E	M	W							
I5:	`lw $s6, 4($s7)`					F	D	E	M	W						

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Draw the pipeline schema showing all the conflicts/dependencies.
Solve the resulting RAW hazards without using rescheduling and path forwarding.

Exe 3.1 Simple Pipelining : solve as is

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										
3	sw $s5, 4($s3)			F	D	E	M	W									
4	sub $s7, $s5, $s6				F	D	E	M	W								
5	lw $s6, 4($s7)					F	D	E	M	W							

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.1 Simple Pipelining : solve as is

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	$D_{(S)}$	$D_{(S)}$	D	E	M	W								
3	sw $s5, 4($s3)			$F_{(S)}$	$F_{(S)}$	F	$D_{(S)}$	$D_{(S)}$	D	E	M	W					
4	sub $s7, $s5, $s6						$F_{(S)}$	$F_{(S)}$	F	D	E	M	W				
5	lw $s6, 4($s7)									F	$D_{(S)}$	$D_{(S)}$	D	E	M	W	

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to **reduce the stalls**; Draw the pipeline schema showing all the data conflicts/dependencies.

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to reduce the stalls; Draw the pipeline schema showing all the data conflicts/dependencies.

```
I1:  addi $s3, $s2, 2
I2:  add $s5, $s4, $s3
I3:  sw $s5, 4($s3)
I4:  sub $s7, $s5, $s6
I5:  lw $s6, 4($s7)
```

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling

Reschedule the instructions to reduce the stalls; Draw the pipeline schema showing all the data conflicts/dependencies.

```
I1:  addi $s3, $s2, 2
I2:  add $s5, $s4, $s3
I4:  sub $s7, $s5, $s6
I3:  sw $s5, 4($s3)
I5:  lw $s6, 4($s7)
```

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling RAW

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										
4	sub $s7, $s5, $s6			F	D	E	M	W									
3	sw $s5, 4($s3)				F	D	E	M	W								
5	lw $s6, 4($s7)					F	D	E	M	W							

RAW $s3 I1-I2

~~**RAW $s3 I1-I3**~~

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.2 Simple Pipelining : Rescheduling Execution

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D(s)	D(s)	D	E	M	W								
4	sub $s7, $s5, $s6			F(s)	F(s)	F	D(s)	D(s)	D	E	M	W					
3	sw $s5, 4($s3)						F(s)	F(s)	F	D	E	M	W				
5	lw $s6, 4($s7)									F	D(s)	D	E	M	W		

RAW $s3 I1-I2

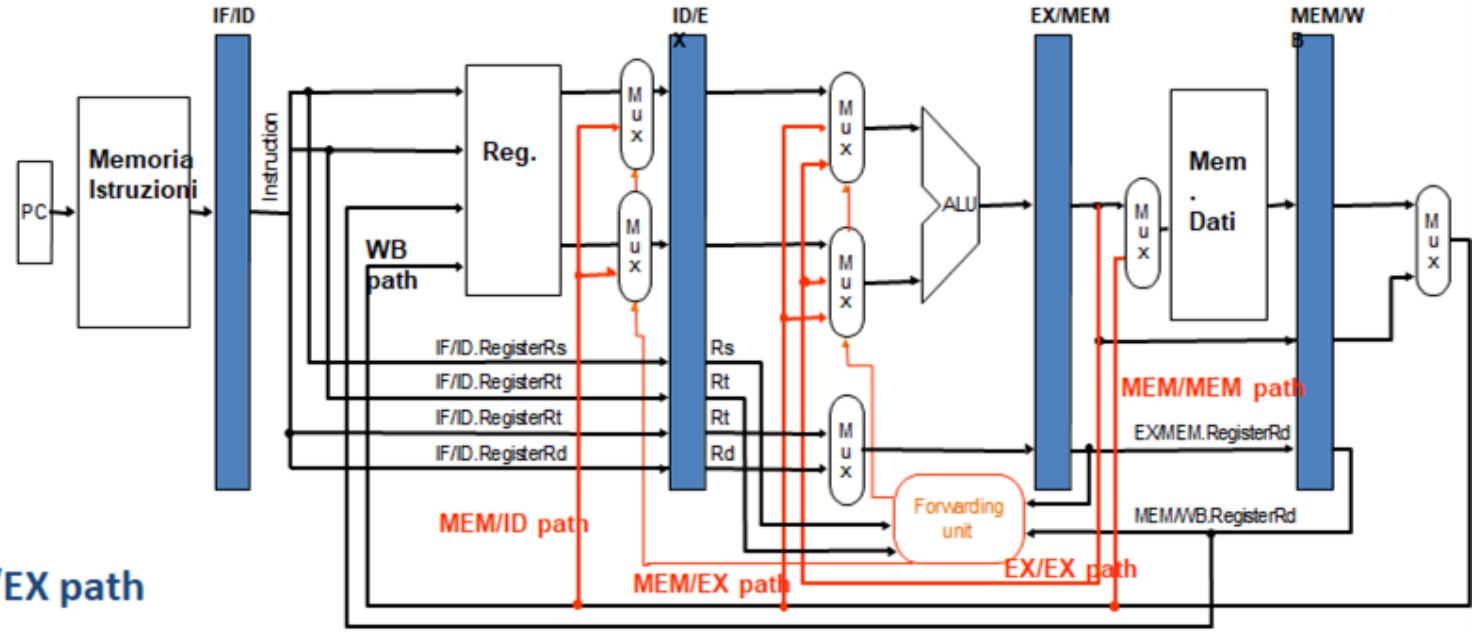
~~**RAW $s3 I1-I3**~~

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.3: Forwarding paths



- EX/EX path
- MEM/EX path
- MEM/ID path
- MEM/MEM path

The forwarding paths have been included in the pipeline. Start from the code in (a) and draw the pipeline schema showing all the forwarding paths that have to be used to solve the hazards.

Exe 3.3 Simple Pipelining : FWD Paths

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										
3	sw $s5, 4($s3)			F	D	E	M	W									
4	sub $s7, $s5, $s6				F	D	E	M	W								
5	lw $s6, 4($s7)					F	D	E	M	W							

RAW $s3 I1-I2

RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

Exe 3.3 Simple Pipelining : FWD Paths

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	FWD Path
1	addi $s3, $s2, 2	F	D	E	M	W											
2	add $s5, $s4, $s3		F	D	E	M	W										EX-EX
3	sw $s5, 4($s3)			F	D	E	M	W									M-EX M-M
4	sub $s7, $s5, $s6				F	D	E	M	W								M-EX
5	lw $s6, 4($s7)					F	D	E	M	W							EX-EX

RAW $s3 I1-I2

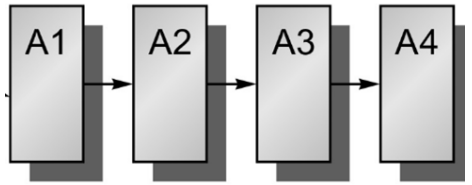
RAW $s3 I1-I3

RAW $s5 I2-I3

RAW $s5 I2-I4

RAW $s7 I4-I5

You can assume that



- All functional units are pipelined
- ALU operations take 1 cycle
- Memory operations take 3 cycles (includes time in ALU)
- Floating-point add instructions take 3 cycles
- Floating-point multiply instructions take 5 cycles
- There is no register renaming. No forwarding
- Instructions are fetched, decoded and issued in order
- The ISSUE stage is a buffer of unlimited length that holds instructions waiting to start execution
- An instruction will only enter the issue stage if it does not cause a WAR or WAW hazard
- Only one instruction can be issued at a time, and in the case multiple instructions are ready, the oldest one will go first

To simplify book-keeping for this problem we will only track instructions during a few critical stages:

- I - When an instruction enters the issue stage
- E - When an instruction starts execution (enters FU)
- C - When an instruction completes execution **(enters WB, results available this cycle)**

Exe Complex Pipeline: the Code

```
I1 L.D F2, 0(R1)
I2 ADD.D F1, F1, F2
I3 MUL.D F4, F3, F3
I4 ADDI R1, R1, 8
I5 L.D F2, 0(R1)
I6 ADD.D F1, F2, F4
```

Exe Complex Pipeline: the Conflicts

RAW F2 I1-I2
RAW R1 I4-I5
RAW F2 I5-I6
RAW F4 I3-I6
WAR R1 I1-I4
WAW F2 I2-I5
WAW F1 I2-I6
WAR F1 I2-I6
WAR F2 I2-I5

Pipeline Schema

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

CC 0

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
1	L.D F2, 0(R1)																			
2	ADD.D F1, F1, F2																			
3	MUL.D F4, F3, F3																			
4	ADDI R1, R1, 8																			
5	L.D F2, 0(R1)																			
6	ADD.D F1, F2, F4																			

RAW F2 I1-I2
RAW R1 I4-I5
RAW F2 I5-I6
RAW F4 I3-I6

WAR F2 I2-I5

WAR R1 I1-I4
WAW F2 I2-I5
WAW F1 I2-I6
WAR F1 I2-I6

Pipeline Schema

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

CC 0

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
1	L.D F2, 0(R1)	I	E			C														
2	ADD.D F1, F1, F2		I				E			C										
3	MUL.D F4, F3, F3			I				E					C							
4	ADDI R1, R1, 8					I			E		C									
5	L.D F2, 0(R1)									I		E			C					
6	ADD.D F1, F2, F4										I					E			C	

RAW F2 I1-I2
RAW R1 I4-I5
RAW F2 I5-I6
RAW F4 I3-I6

WAR F2 I2-I5

WAR R1 I1-I4
WAW F2 I2-I5
WAW F1 I2-I6
WAR F1 I2-I6

Pipeline Schema

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

CC 0

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
1	L.D F2, 0(R1)	I	E			C														
2	ADD.D F1, F1, F2		I				E			C										
3	MUL.D F1, F1, F3																			
4	ADDI R1, R1, 8					I			E		C									
5	L.D F2, 0(R1)									I		E			C					
6	ADD.D F1, F2, F4										I					E			C	

Removing the in-order issue constraint?

RAW F2 I1-I2
RAW R1 I4-I5
RAW F2 I5-I6
RAW F4 I3-I6

WAR F2 I2-I5

WAR R1 I1-I4
WAW F2 I2-I5
WAW F1 I2-I6
WAR F1 I2-I6

Pipeline Schema

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

CC 0

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19
1	L.D F2, 0(R1)	I	E			C														
2	ADD.D F1, F1, F2		I				E			C										
3	MUL.D F4, F3, F3			I	E						C									
4	ADDI R1, R1, 8					I	E	C												
5	L.D F2, 0(R1)									I	E			C						
6	ADD.D F1, F2, F4										I				E			C		

RAW F2 I1-I2
RAW R1 I4-I5
RAW F2 I5-I6
RAW F4 I3-I6

WAR F2 I2-I5

WAR R1 I1-I4
WAW F2 I2-I5
WAW F1 I2-I6
WAR F1 I2-I6

Exe Complex Pipeline: the Code

```
I1 LD F1, 0(R1)
I2 LD F2, 0(R1)
I3 ADDD F1, F1, F2
I4 ADDI R1, R1, 8
I5 MULD F1, F1, F3
I6 SD F1, 16(R1)
```

Exe Complex Pipeline: the Code

RAW F1 I1-I3
RAW F2 I2-I3
RAW R1 I4-I6
RAW F1 I3-I5
RAW F1 I5-I6
WAW F1 I5-I3
WAW F1 I3-I1
WAR F1 I5-I3
WAR R1 I4-I1
WAR R1 I4-I2

Pipeline Schema

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

CC 0

	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22
1	LD F1, 0(R1)																						
2	LD F2, 0(R1)																						
3	ADDD F1, F1, F2																						
4	ADDI R1, R1, 8																						
5	MULD F1, F1, F3																						
6	SD F1, 16(R1)																						

RAW F1 I1-I3
RAW F2 I2-I3
RAW R1 I4-I6

RAW F1 I5-I6
RAW F1 I3-I5

WAW F1 I5-I3
WAW F1 I3-I1
WAR F1 I5-I3
WAR R1 I4-I1
WAR R1 I4-I2

Pipeline Schema

ALU OP: 1 cycle
MEM OP: 3 cycles
FP ADD: 3 cycles
FP MULT: 5 cycles

CC 0

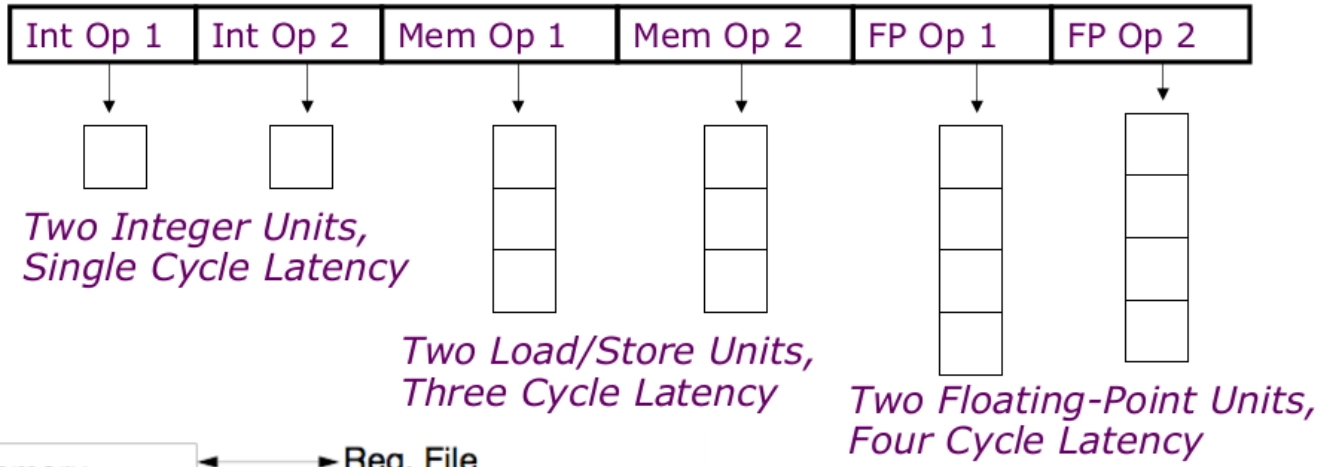
	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22
1	LD F1, 0(R1)	I	E			C																	
2	LD F2, 0(R1)		I	E			C																
3	ADDD F1, F1, F2					I		E			C												
4	ADDI R1, R1, 8						I		E	C													
5	MULD F1, F1, F3										I	E					C						
6	SD F1, 16(R1)											I						E			C		

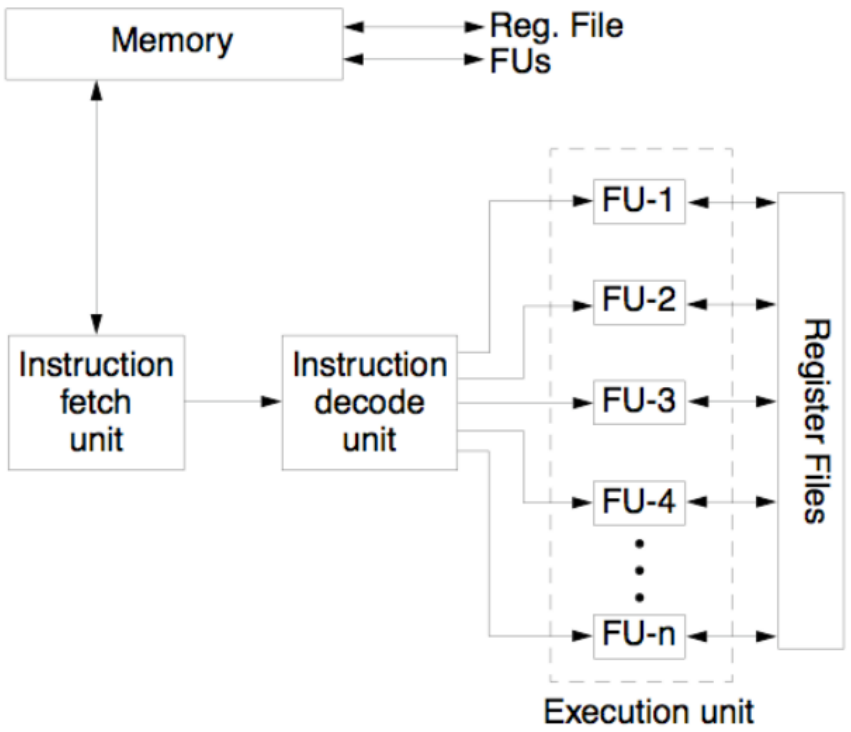
RAW F1 I1-I3
RAW F2 I2-I3
RAW R1 I4-I6

RAW F1 I5-I6
RAW F1 I3-I5

WAW F1 I5-I3
WAW F1 I3-I1
WAR F1 I5-I3
WAR R1 I4-I1
WAR R1 I4-I2

Recall VLIW and Static Scheduling





Static Scheduling: Rely on software for identifying potential parallelism (example of List Based Scheduling with ASAP)

VLIW (Very Long Instruction Word) processors expect dependency-free code.

Question on VLIW Architecture

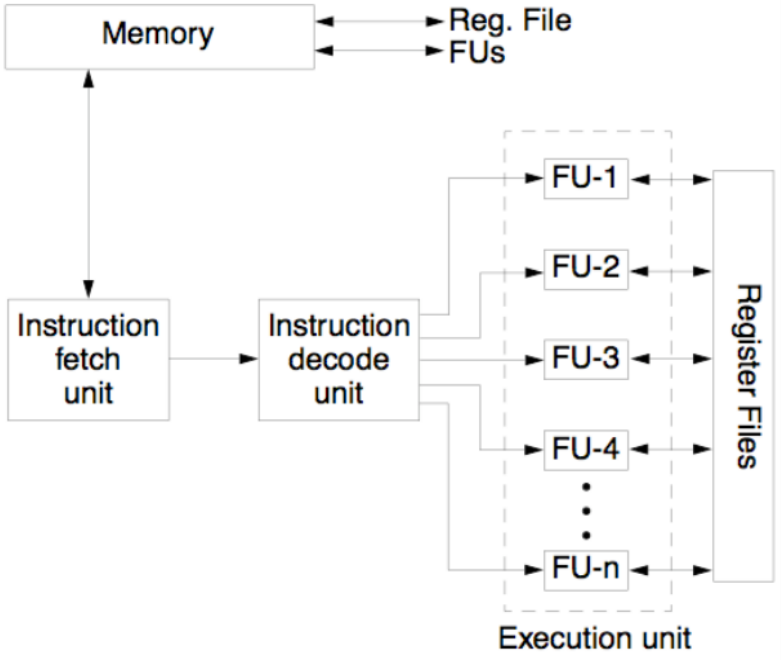
A VLIW Architecture has to have multiple Program Counters to load the necessary Multiple Data.
Given the previous statement, confirm if it is TRUE or FALSE and **effectively support** your answer.

Circle the **right** answer: True False

Question on VLIW Architecture

A VLIW Architecture has to have multiple Program Counters to load the necessary Multiple Data.
Given the previous statement, confirm if it is TRUE or FALSE and **effectively support** your answer.

Circle the **right** answer: True False



Recall: MESI Protocol

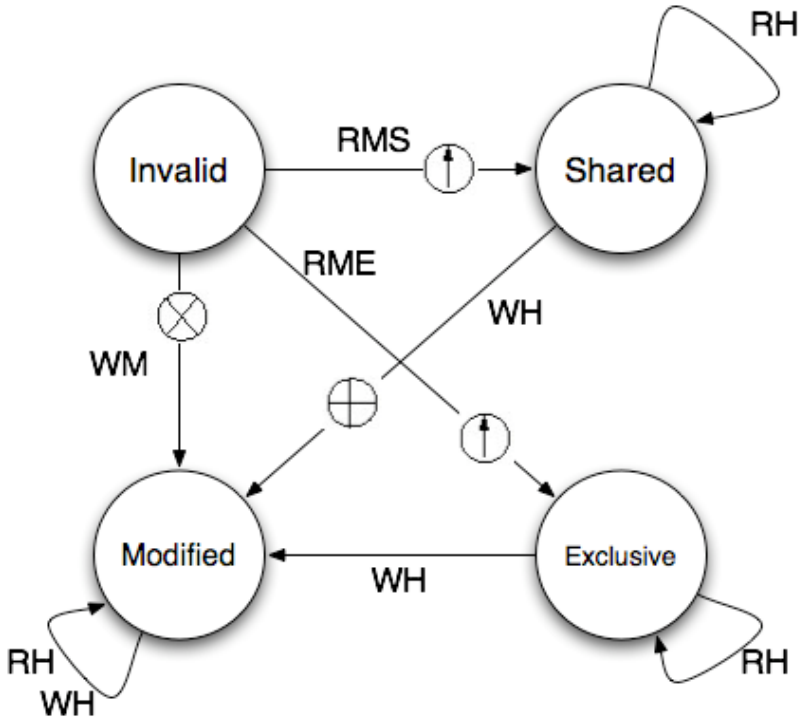
MESI Protocol: Write-Invalidate
Each cache block can be in one of four states:

- 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

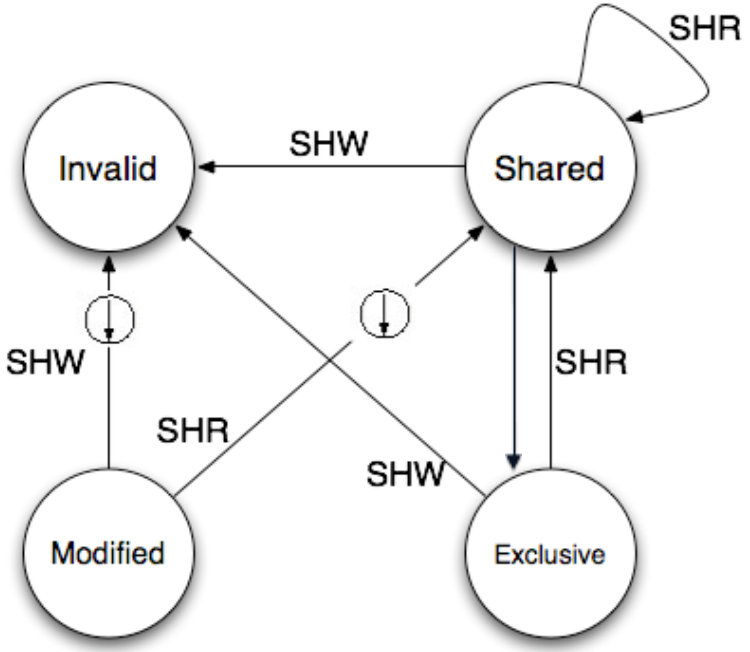
Add exclusive state to distinguish exclusive (writable) and owned (written)

104

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

105

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).

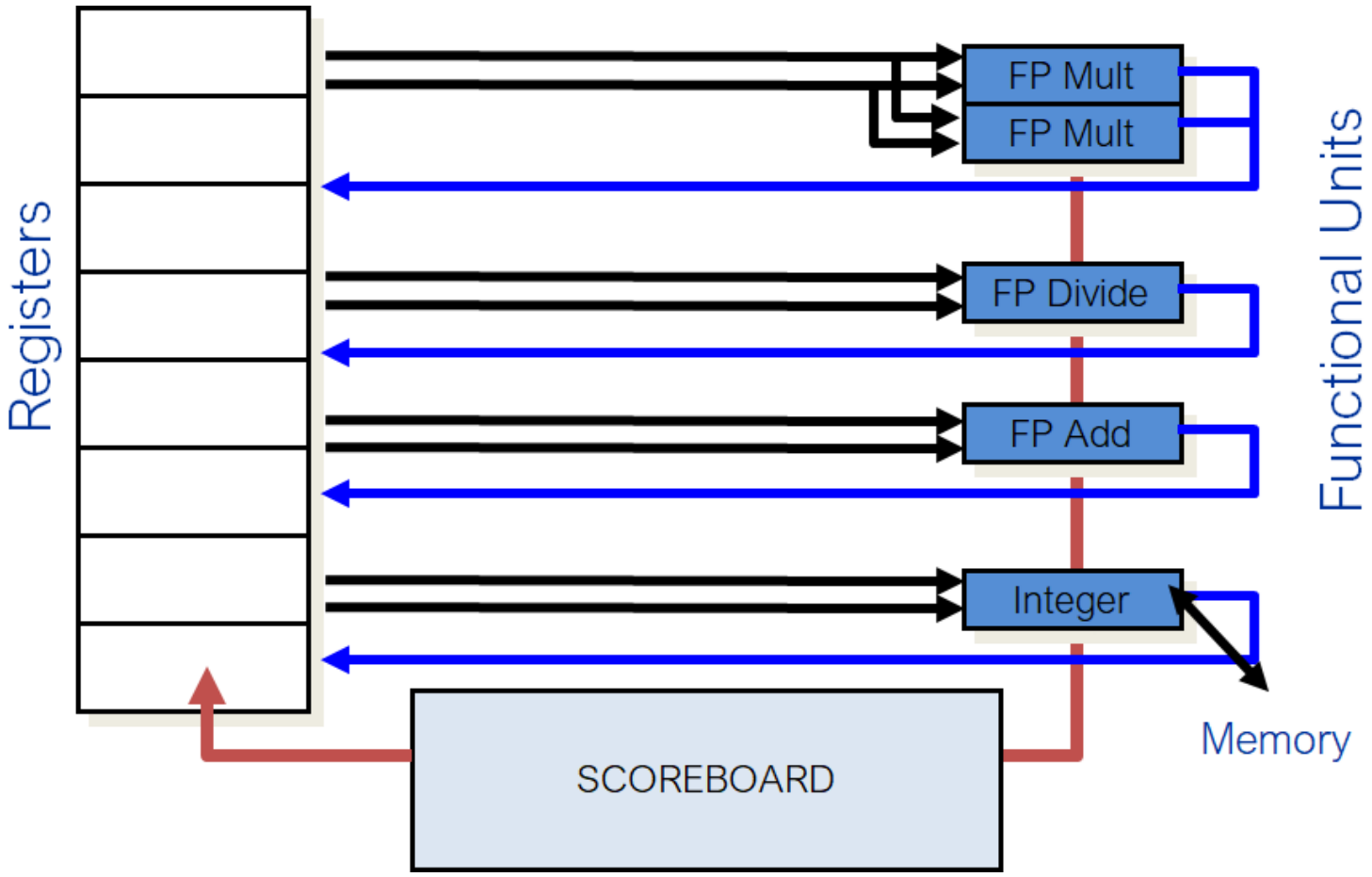
Exe 1: Cache Coherency

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	P1: write block 0				
4	P1: read block 0				
5	P1: write block 0				
6	P0: read block 0				
7	P0: read block 0				
8	P0: write block 0				
9	P1: read block 0				

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	P1: write block 0	Modified (1)	Modified (0)	No	No
4	P1: read block 0	Modified (1)	Modified (0)	No	No
5	P1: write block 0	Modified (1)	Modified (0)	No	No
6	P0: read block 0	Shared (0)	Shared (0)	Yes	Yes
7	P0: read block 0	Shared (0)	Shared (0)	Yes	Yes
8	P0: write block 0	Modified (0)	Invalid	No	Yes
9	P1: read block 0	Shared (0)	Shared (0)	Yes	Yes

Exe 1 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 4 Structural Hazard;

Exe 1 Scoreboard: the Code

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD F5, 16 (R1)	1	2	6	7
I2	ADDD F12, F5, F2	2	8	14	15
I3	MULTD F2, F4, F3	3	4	15	16
I4	DIVD F1, F12, F5	4	16	27	28
I5	SD F1, 4 (R1)	5	29	33	34
I6	SUBD F2, F12, F4	17	18	24	25

Exe 1.1 Scoreboard: Conflicts

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD F5, 16 (R1)	1	2	6	7
I2	ADDD F12, F5, F2	2	8	14	15
I3	MULTD F2, F4, F3	3	4	15	16
I4	DIVD F1, F12, F5	4	16	27	28
I5	SD F1, 4 (R1)	5	29	33	34
I6	SUBD F2, F12, F4	17	18	24	25

Exe 1.2 Scoreboard: ∃ a configuration?

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD F5, 16 (R1)	1	2	6	7
I2	ADDD F12, F5, F2	2	8	14	15
I3	MULTD F2, F4, F3	3	4	15	16
I4	DIVD F1, F12, F5	4	16	27	28
I5	SD F1, 4 (R1)	5	29	33	34
I6	SUBD F2, F12, F4	17	18	24	25

- Is there a ▒configuration▒ that can respect the shown execution?
- How many units? Which kind? What latency?

Exe 1.3 Scoreboard: if not correct, write right one

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD F5, 16 (R1)				
I2	ADDD F12, F5, F2				
I3	MULTD F2, F4, F3				
I4	DIVD F1, F12, F5				
I5	SD F1, 4 (R1)				
I6	SUBD F2, F12, F4				

If the previous table was not correct, please, write the right one and specify the number, kind and latency for each unit.

Exe 1.2 Scoreboard: ∃ a configuration?

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD F5, 16 (R1)	1	2	6	7
I2	ADDD F12, F5, F2	2	8	14	15
I3	MULTD F2, F4, F3	3	4	15	16
I4	DIVD F1, F12, F5	4	16	27	28
I5	SD F1, 4 (R1)	5	29	33	34
I6	SUBD F2, F12, F4	17	18	24	25

- Is there a "configuration" that can respect the shown execution?
- How many units? Which kind? What latency?

Exe 1.2 Scoreboard: ∃ a configuration?

	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD F5, 16 (R1)	1	2	6	7
I2	ADDD F12, F5, F2	2	8	14	15
I3	MULTD F2, F4, F3	3	4	15	16
I4	DIVD F1, F12, F5	4	16	27	28
I5	SD F1, 4 (R1)	5	29	33	34
I6	SUBD F2, F12, F4	17	18	24	25

A possible configuration consists of:

- 2 MUL/DIV units and 11 CC of latency
- 1 ADDD/SUBD unit and 6 CC of latency
- 2 Memory Unit and 4 CC of latency
- 1 write port is enough

Other solutions are possible.

Thanks for your attention to the extras

Davide Conficconi <davide.conficconi@polimi.it>

Acknowledgements

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
- “Digital Design and Computer Architecture” Harris and Harris
- Elsevier Inc. online materials
- Papers/news cited in this lecture