ECE 411 Midterm 2

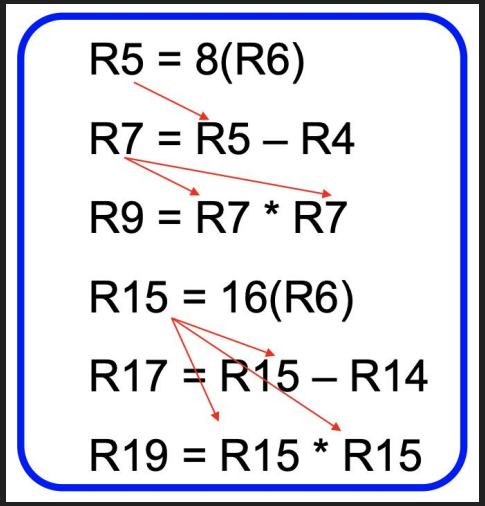
Srijan Chakraborty, Abishek Venkit,
Keshav Harisrikanth

Agenda

- ILP and Dynamic Scheduling
- Tomasulo
- Prefetching
- Energy Efficient Computing
- DRAM and Memory
- Multicore and SMT
- Cache Coherence
- SIMD + GPU
- Questions

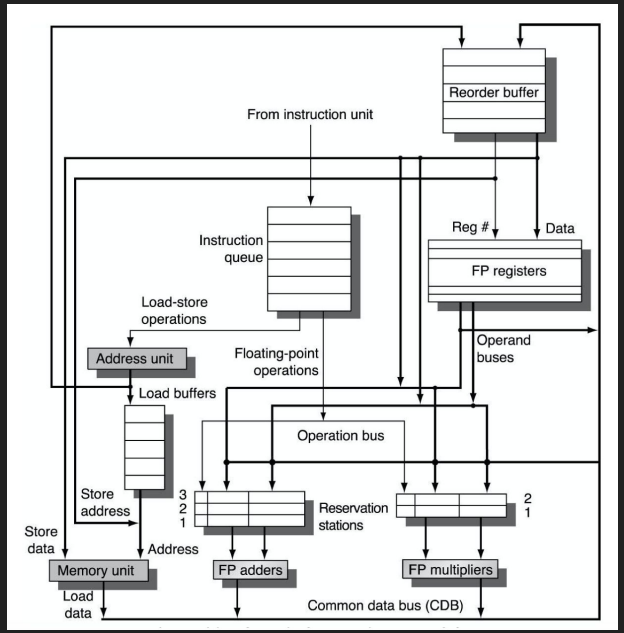
ILP and Dynamic Scheduling

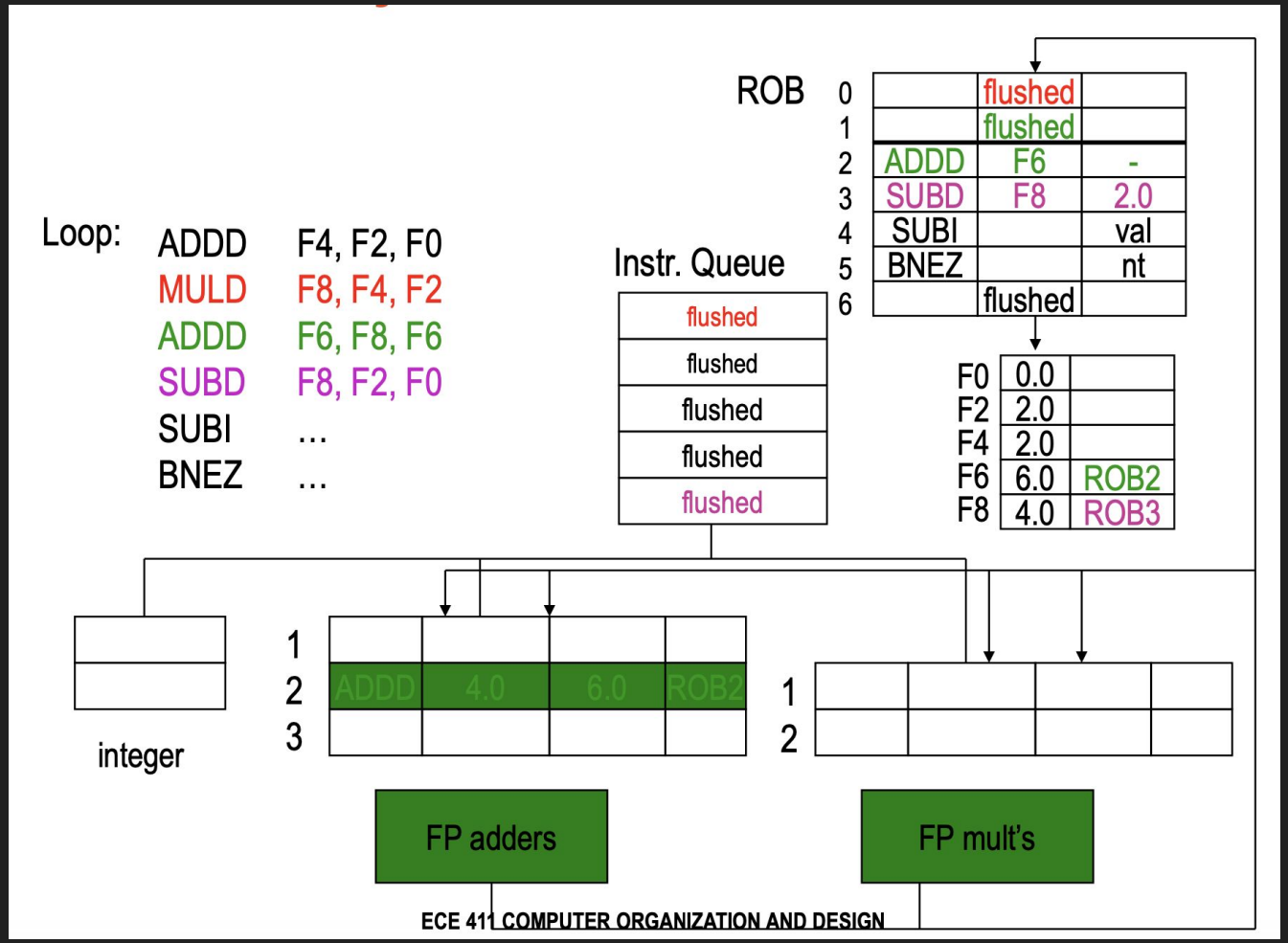
- ILP = measure of inter-instruction dependency
- Larger window = more parallelism
- Remove dependencies by reordering instructions or renaming registers
- Loads after stores must occur in order if they are operating on the same data
- Know what RAW, WAR, and WAW dependencies are and when they appear



Tomasulo

- Algorithm to implement OoO execution
  - OoO execution ≠ out-of-order retirement
  - Supports precise interrupts
- Removes WAR and WAW hazards
- Minimized RAW hazards
- Implements implicit register renaming
- Review the Tomasulo execution slides and preview exams!




Loop:
ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
SUBI …
BNEZ …
Instr. Queue
flushed
flushed
flushed
flushed
flushed
ROB
0 flushed
1 flushed
2 ADDD F6 -
3 SUBD F8 2.0
4 SUBI val
5 BNEZ nt
6 flushed
F0 0.0
F2 2.0
F4 2.0
F6 6.0 ROB2
F8 4.0 ROB3
integer
1 ADDD 4.0 6.0 ROB2
2
3
1
2
FP adders
FP mult's
ECE 411 COMPUTER ORGANIZATION AND DESIGN

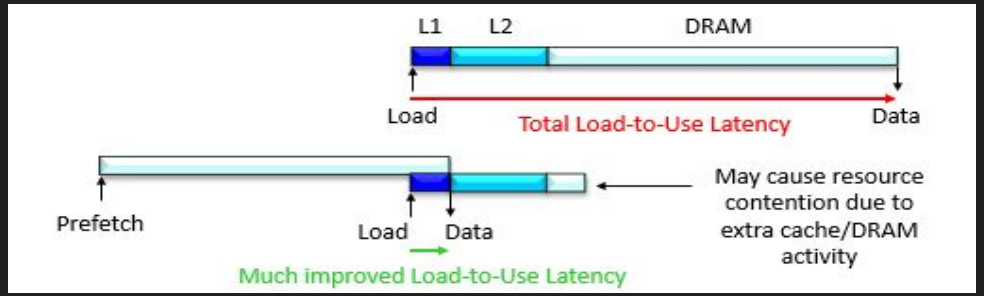
Energy Efficient Computing

- Dynamic Voltage and Frequency Scaling
  - Key for energy efficient computing
  - $P \propto V^2 f \propto V^3$
- Power gating: reduce static and dynamic power
- Clock gating: reduce dynamic power
- Energy = Power * Delay (or Execution Time)
- Energy Delay Product
  - Energy * Delay = Power * Delay^2

$$P_{tot} = P_{dyn} + P_{sta} = C_L V_{dd}^2 f + V_{dd} I_{leak}$$

Prefetching

- Load memory BEFORE it is required for operations
  - Decide when to load, what to load, how to load, and where to put it
- Can prefetch into cache or special prefetching buffer
  - In case of buffer, have to design, check, and keep it coherent
  - In case of cache directly, have to avoid polluting it
- Hardware can try to detect strides and prefetch according to them
- Software can provide prefetch instructions in ISA
  - Know where to place this in code and what to prefetch for maximum benefit
- Execution-Based: Launch separate thread for predicting cache misses

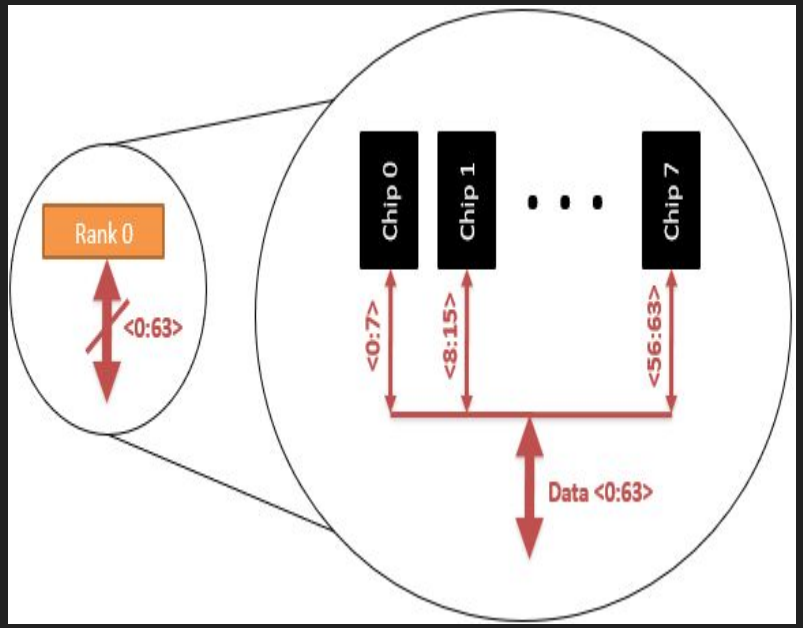


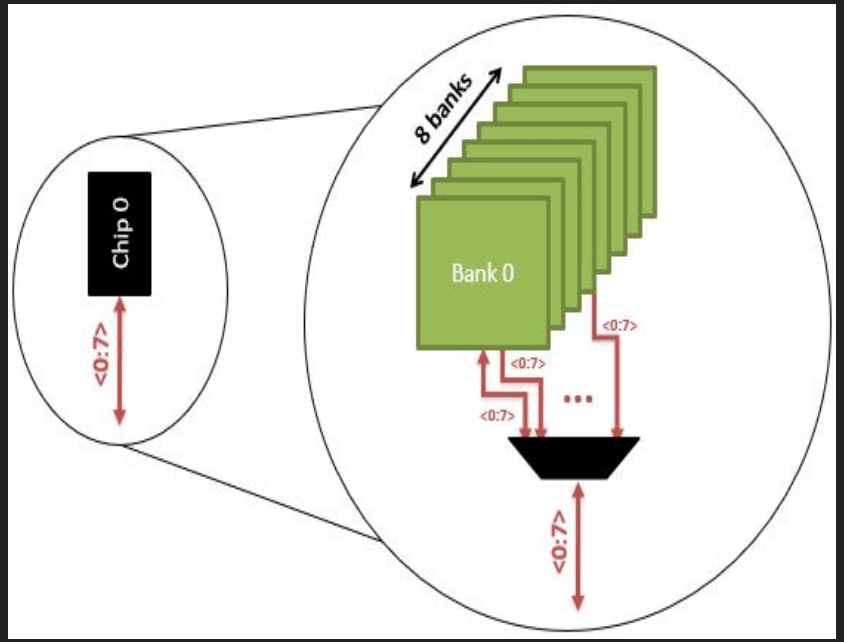
DRAM and Memory

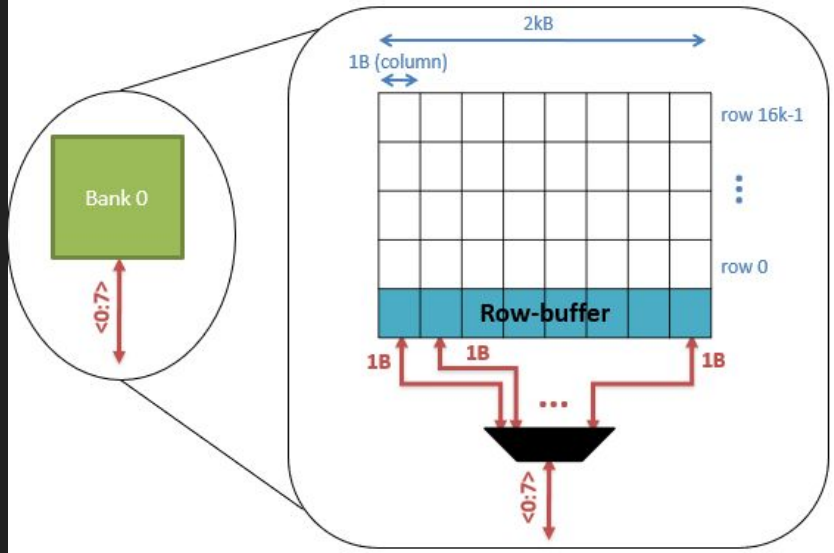
- Needs to be constantly refreshed
- Destructive reads mean we have to writeback to cells after reads
- Memory cells have row addresses and column addresses
  - Use RAS and CAS signals to read
  - To access from closed rows, need to **Activate** it, **Read or Write**, then **Precharge** for closing before next access
  - To access from open rows no activate needed
- Memory Controllers in charge of scheduling requests and buffering data
  - Schedule accesses to avoid conflicts (FCFS: First Come First Serve, FR-FCFS: First Ready First Come First Serve), also schedule refreshes
- Open Page vs. Closed Page policy

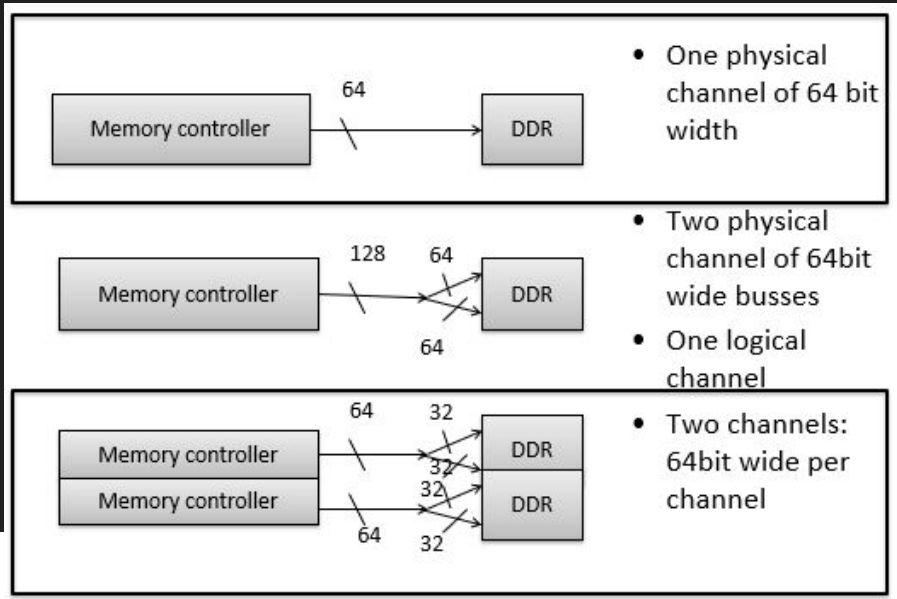
DRAM Configuration

- Data transferred through channels
- Organized by Bank, Chips, Rank, DIMM
  - DRAM row (also called a DRAM page)
  - Can access banks independently
- Understand how to map addresses to components to interleave types of addresses (for example randomize banks accesses to avoid conflicts)
- Rows also need to refresh, and are unavailable during the refresh
  - Can do distributed refreshes to reduce latency
- Understand where latencies come from and how changing aspects of organization affects the system


Rank 0
<0:63>
Chip 0
Chip 1
Chip 7
<0:7>
<8:15>
<56:63>
Data <0:63>


Chip 0
<0:7>
8 banks
Bank 0
<0:7>
<0:7>
<0:7>
<0:7>





- **Row interleaving**
  - Consecutive rows of memory in consecutive banks

Row (14 bits)	Bank (3 bits)	Column (11 bits)	Byte in bus (3 bits)

- **Cache block interleaving**
  - Consecutive cache block addresses in consecutive banks
  - 64 bytes cache blocks

Row (14 bits)	High Column	Bank (3 bits)	Low Col.	Byte in bus (3 bits)
	8 bits		3 bits	

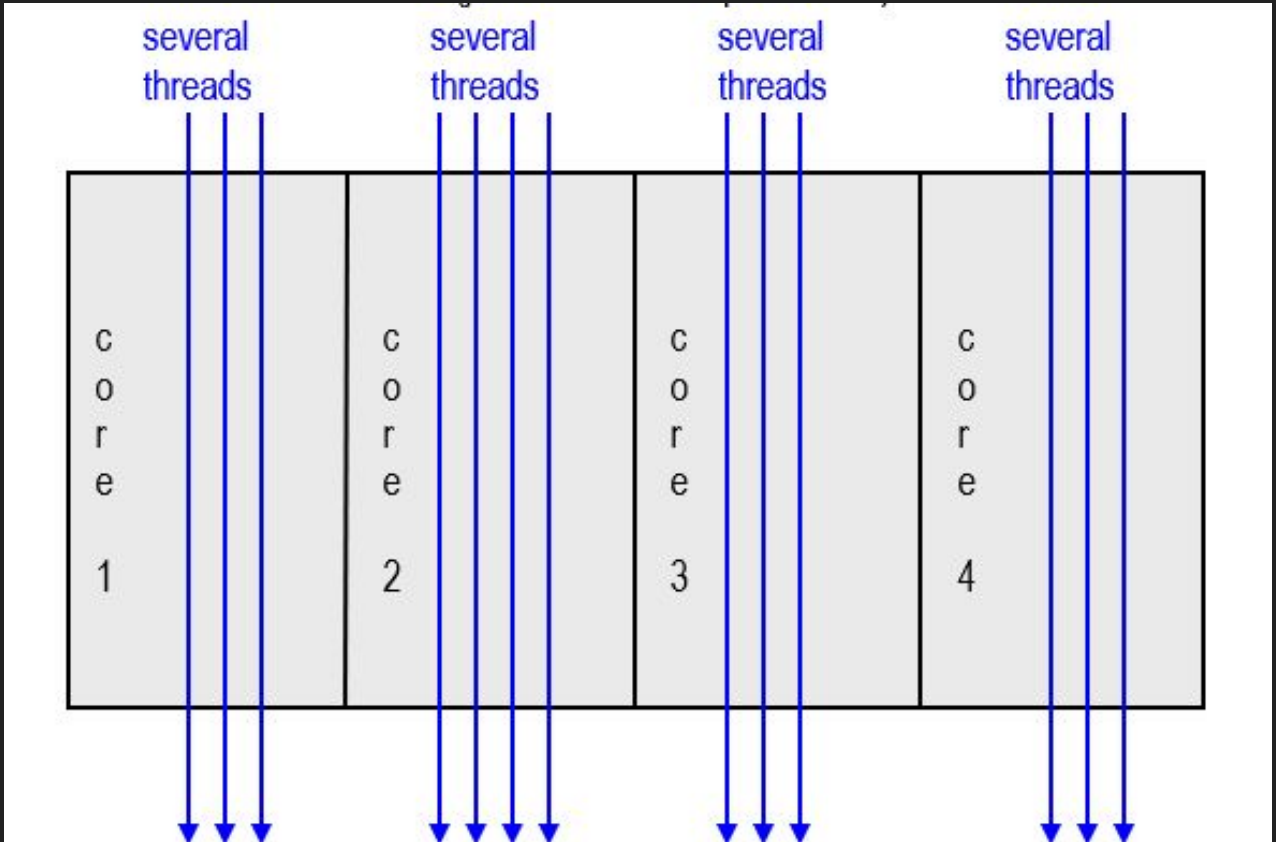
Multicore

- Challenging to make single core architectures better
  - Difficult to extract more ILP (more power hungry, slower, and more complicated circuits)
  - Power, Thermal, and speed-of-light limitations, difficult to design and verify deeper pipelines
- Take advantage of more thread-level parallelism
  - MIMD (multiple instruction multiple data)
- Two possible memory organizations: shared or distributed memory
- Simultaneous Multithreading (SMT): If your pipeline has multiple unused functional units, you can take advantage of these by running multiple threads on the same core at once through different functional units
  - Also called hyperthreading
  - **Threads must use different functional units to make full use of SMT**
  - Know trade-offs between same numbers of SMT threads vs Cores
  - Understand limitations of SMT (when is it useless?)
- Know Amdahl's Law and how we can apply it to multicore questions

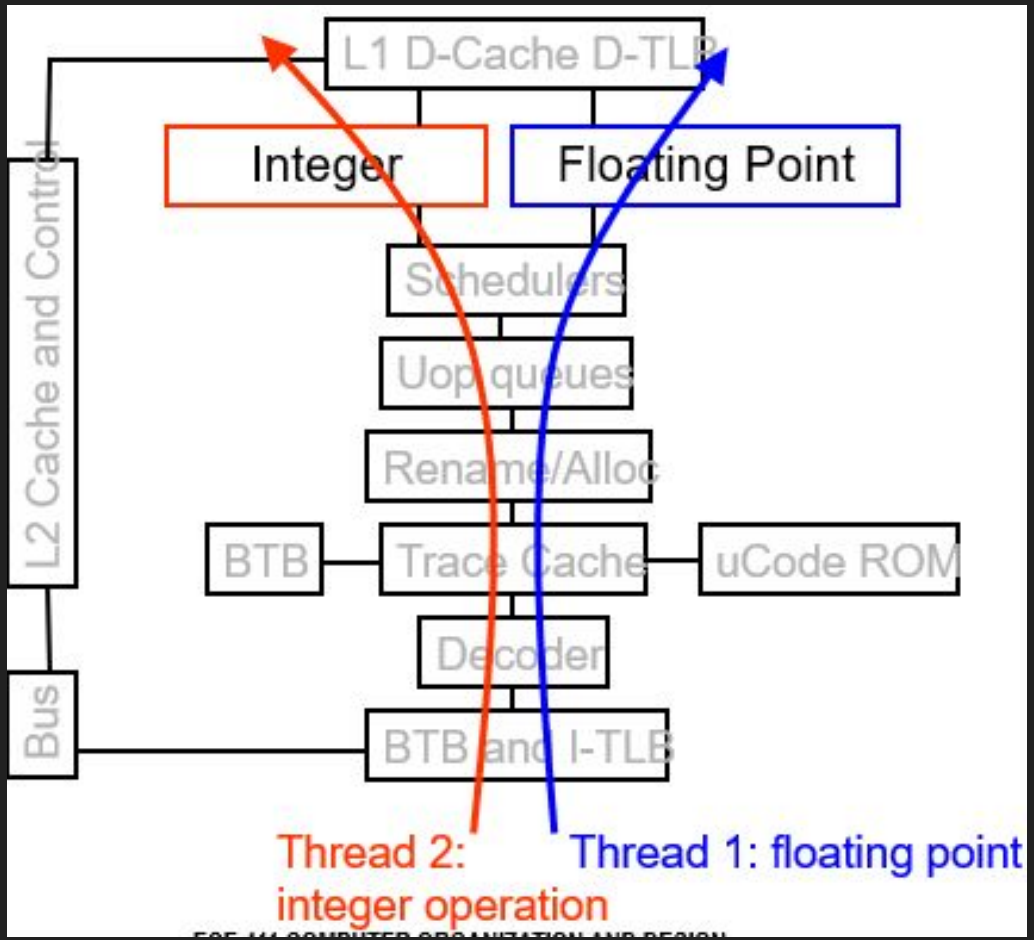
Amdahl’s Law (Review)

$$Speedup = \frac{1}{(1-p) + p/N}$$

Multicore


several threads
several threads
several threads
several threads
core 1
core 2
core 3
core 4

SMT


L1 D-Cache D-TLB
Integer
Floating Point
Schedulers
Uop queues
Rename/Alloc
BTB
Trace Cache
uCode ROM
Decoder
BTB and I-TLB
L2 Cache and Control
Bus
Thread 2:
integer operation
Thread 1: floating point

Cache Coherence: Definition

- A memory system is coherent if
  - A read R from address X on processor P1 returns the value written by the most recent write W to X on P1 if no other processor has written to X between W and R.
  - If P1 writes to X and P2 reads X after a sufficient time, and there are no other writes to X in between, P2's read returns the value written by P1's write.
  - Writes to the same location are serialized: two writes to location X are seen in the same order by all processors.
- What this boils down to is that individual program order is preserved, all writes are eventually seen, and causal ordering is preserved

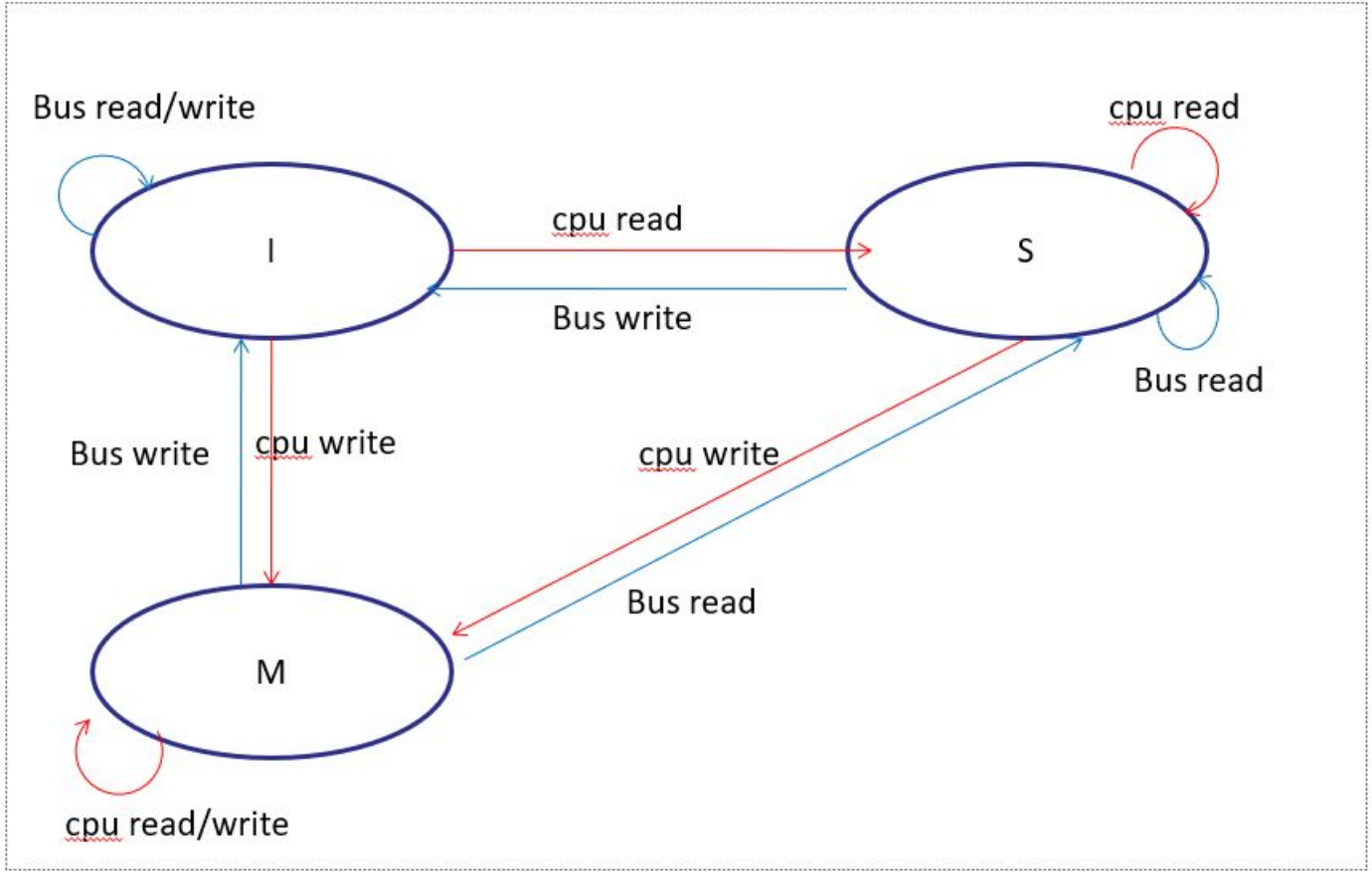
Cache Coherence: How?

- Snooping: Write-update vs. Write-invalidate
  - When you write to a block, should you update everyone else that has it or should you invalidate their blocks instead
  - Know the trade-offs between the two
- Know MSI and MESI protocols, the difference, and why one would be better
  - State of block B in cache C can be
    - Invalid: B is not cached or invalid in C
    - Modified: B is dirty in C
    - Shared: B is clean in C (In MESI and MOESI also that others have a clean copy)
    - Exclusive (MESI and MOESI only): B is clean in C, C has only copy
    - Owned (MOESI only): B is most up-to-date in C, C is responsible for sourcing B
  - MESI saves on bandwidth since on write no need to invalidate other copies
- Given a protocol know why it’s good or bad in what situations

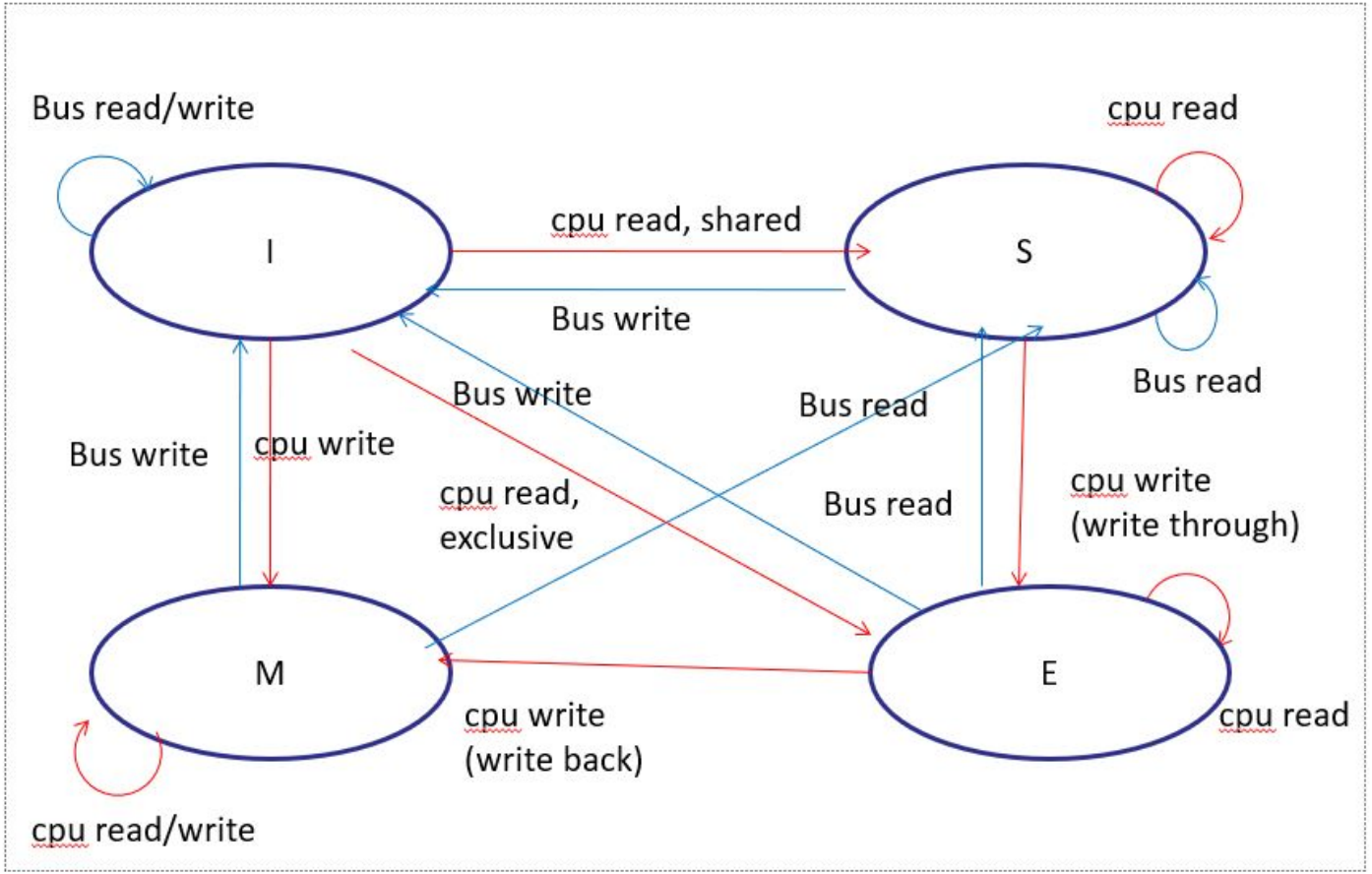
Cache Coherence Contd.

- Directory-Based Coherence: use local directory to track who has what block in what state (typically in distributed shared memory)
- False sharing: Have to invalidate line because processors use different data in same block
  - One way to mitigate: smaller block size
- Know the effects of needing to maintain coherence on performance
  - e.g. Given two threads that use the same blocks often where would you schedule them? If a thread has a high memory contention what would you schedule it with? etc.

MSI

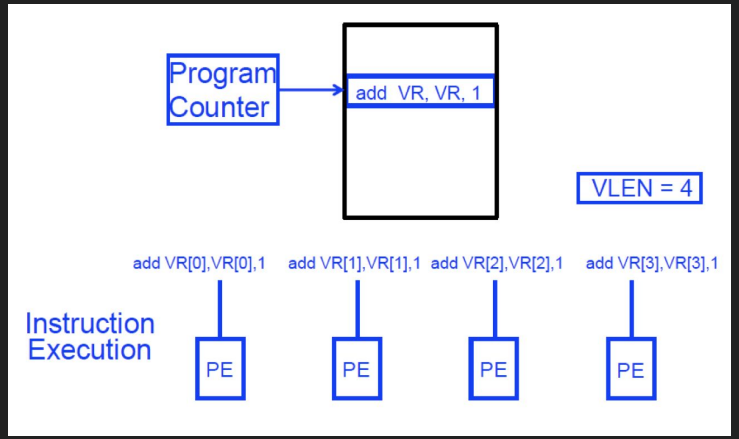

Bus read/write
cpu read
I
cpu read
S
Bus write
Bus read
Bus write
cpu write
cpu write
Bus read
M
cpu read/write

MESI (Illinois Protocol)



SIMD + GPU

- Single Instruction, Multiple Data
  - Vector processor, GPUs, shaders, etc.
- Why SIMD?
  - Increased parallelism, simple design, energy efficiency, small area
- Overheads?
  - Packing and unpacking data
  - Alignment overhead



Questions?