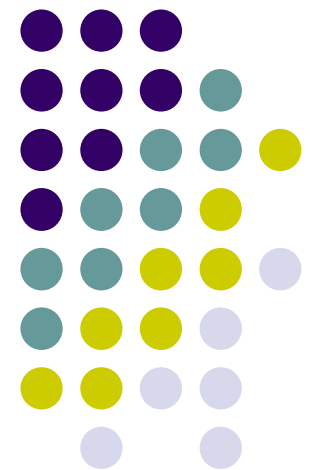


Computer Architecture

Assoc. Prof. Nguyễn Trí Thành, PhD
UNIVERSITY OF ENGINEERING AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS
ntthanh@vnu.edu.vn



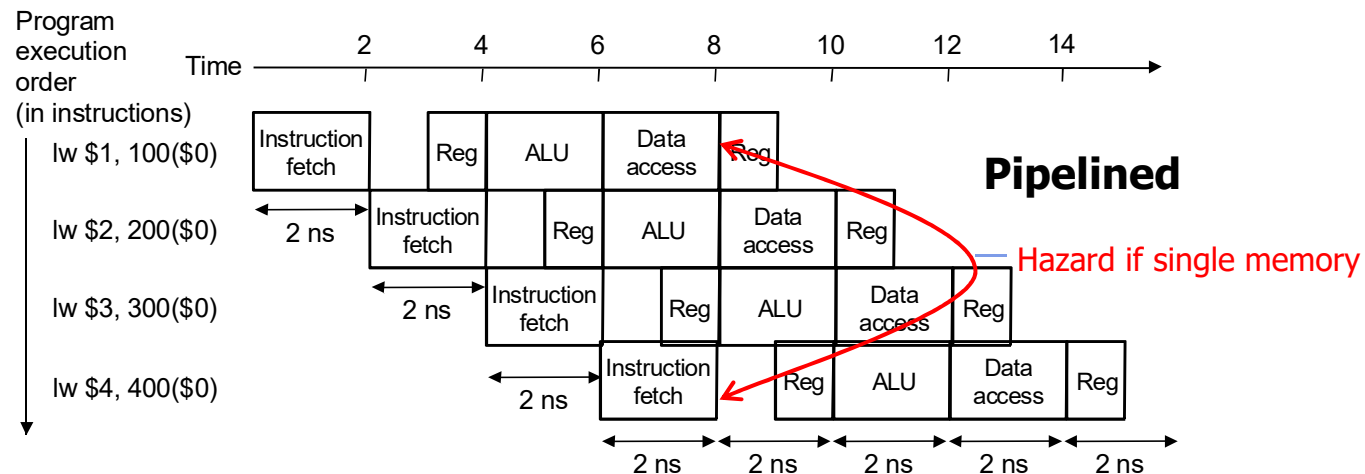


Cache

Structural Hazards - Reminded



- *Structural hazard*: inadequate hardware to simultaneously support all instructions in the pipeline in the same clock cycle
- E.g., suppose *single – not separate* – instruction and data memory in pipeline below with *one read port*
 - then a structural hazard between first and fourth `lw` instructions



- *MIPS was designed to be pipelined*: structural hazards are easy to avoid!

LENOVO B470 TECHNICAL SPECIFICATIONS

Processor	2 nd Generation Intel® Core™ i5-2410M Processor with Intel Turbo Boost Technology* 2.0 (2.30GHz, 3MB Cache) 2 nd Generation Intel® Core™ i3-2310M Processor (2.10GHz, 3MB Cache)
Operating System	Genuine Windows® 7 Professional Genuine Windows® 7 Home Premium Genuine Windows® 7 Home Basic
Display/Resolution	14.0" high-definition (1366*768), 16:9 widescreen
Video Graphics	Up to NVIDIA® GeForce® 410M 1GB graphics
Memory	Up to 8GB DDR3 1066/1333MHz memory
Hard Disk Drive	250GB/320GB/500GB/640GB/750GB/1TB 5400rpm or 320GB/500GB/750GB/1TB 7200rpm HDD
Optical Drive	Integrated DVD reader/writer or Blu-ray Disc™1 drive
Sound	2.0 speakers (2*1.5W)
Integrated Communications	802.11bg/bgn WiFi, 10/100M LAN, Bluetooth® (optional)

LENOVO

Processor / Chipset

	CPU	Intel 2nd Gen Core i5 2 i5-2410M / 2.3 GHz	
Processor	Max Turbo Speed	2.9 GHz	
Operating System	Number of Cores	Dual-Core	
Display/Resolution	Cache	L3 cache - 3.0 MB	
Video Graphics	64-bit Computing	Yes	
Memory	Features	Intel Turbo Boost Technology 2.0, Hyper-Threading Technology, Integrated memory controller	
Hard Disk Drive			750 GB HDD
Optical Drive			
Sound			
Integrated Communications			
	Memory		
	RAM	4.0 GB (2 x 2 GB)	
	Max RAM Supported	8.0 GB	
	Technology	DDR3 SDRAM	
	Speed	1333.0 MHz / PC3-10600	
	Slots Qty	2	

5

LENOVO Processor / Chipset

LENOVO		CPU	Intel 2nd Gen Core i5 2 i5-2410M / 2.3 GHz	
Processor		Max Turbo Speed	2.9 GHz	
Op	Now Here are some Tech Specs for my fellow nerds:			
Dis	CPU: i7 2670			
Vic	4 core 8 Thread			
Me	2.2ghz 3.1ghz turbo			
Ha	32nm 45watt TDP			
Op	5 GT/s x22 multiplier			
So	Socket G2 (rPGA988B)			
Int	Caches:			
Co	L1 data: 128kb			
	L1 inst: 128kb			
	L2: 1024kb			
	L3: 6mb			
	Bus speed: 99.8mhz			
	max ram supported		8.0 GB	
	Technology		DDR3 SDRAM	
	Speed		1333.0 MHz / PC3-10600	
	Slots Qty		2	

6

Processor / Chipset

Architecture / Microarchitecture

Processor		Microarchitecture	Sandy Bridge
Processor	Processor core ?		Sandy Bridge
	Core stepping ?		D2 (Q1HK, SR00B)
	Manufacturing process		0.032 micron High-K metal gate process
	Die size		216mm ²
Op	Now H	Data width	64 bit
CPU: i		The number of cores	4
Dis	4 core	The number of threads	8
2.2ghz		Floating Point Unit	Integrated
32nm		Level 1 cache size ?	4 x 32 KB instruction caches 4 x 32 KB data caches
Me	5 GT/s	Level 2 cache size ?	4 x 256 KB
Ha	Socket	Level 3 cache size	8 MB shared cache
Op	Caches	Cache latency	4 (L1 cache) 11 (L2 cache) 25 (L3 cache)
So	L1 dat	Physical memory	32 GB
Int	L1 inst	Multiprocessing	Uniprocessor
Co	L2: 10	Features	<ul style="list-style-type: none"> ◦ MMX instructions ◦ SSE / Streaming SIMD Extensions ◦ SSE2 / Streaming SIMD Extensions 2 ◦ SSE3 / Streaming SIMD Extensions 3 ◦ SSSE3 / Supplemental Streaming SIMD Extensions 3 ◦ SSE4 / SSE4.1 + SSE4.2 / Streaming SIMD Extensions 4 ? ◦ AES / Advanced Encryption Standard instructions ◦ AVX / Advanced Vector Extensions ◦ EM64T / Extended Memory 64 technology / Intel 64 ? ◦ NX / XD / Execute disable bit ? ◦ HT / Hyper-Threading technology ? ◦ TBT 2.0 / Turbo Boost technology 2.0 ? ◦ VT-x / Virtualization technology ? ◦ VT-d / Virtualization for directed I/O
	L3: 6n		
	Bus sp		

Processor / Chipset

Architecture / Microarchitecture

Processor

Op

Dis

Vis

Me

Ha

Op

So

Int

Co

Now H

CPU: i

4 core

2.2ghz

32nm

5 GT/s

Socket

Caches

L1 dat

L1 inst

L2: 10

L3: 6n

Bus sp

Microarchitecture Sandy Bridge
 Processor core ? [Sandy Bridge](#)
 Core stepping ? D2 (01HK SR00B)
 Manufactur CPU-Z

Die size

Data width

The numb

The numb

Floating Po

Level 1 ca

Level 2 ca

Level 3 ca

Cache late

Physical m


Multiproces

Features

CPU-Z

CPU Mainboard Memory SPD Graphics Bench About

Processor

Name	Intel Core i7 6700			
Code Name	Skylake	Max TDP	65.0 W	
Package	Socket 1151 LGA			
Technology	14 nm	Core VID	0.911 V	
Specification	Intel® Core™ i7-6700 CPU @ 3.40GHz			
Family	6	Model	E	Stepping
Ext. Family	6	Ext. Model	5E	Revision
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX			

Clocks (Core #0)

Core Speed	2694.06 MHz
Multiplier	x 27.0 (8 - 40)
Bus Speed	99.78 MHz
Rated FSB	

Cache

L1 Data	4 x 32 KBytes	8-way
L1 Inst.	4 x 32 KBytes	8-way
Level 2	4 x 256 KBytes	4-way
Level 3	8 MBytes	16-way

Selection

Socket #1

Cores

4

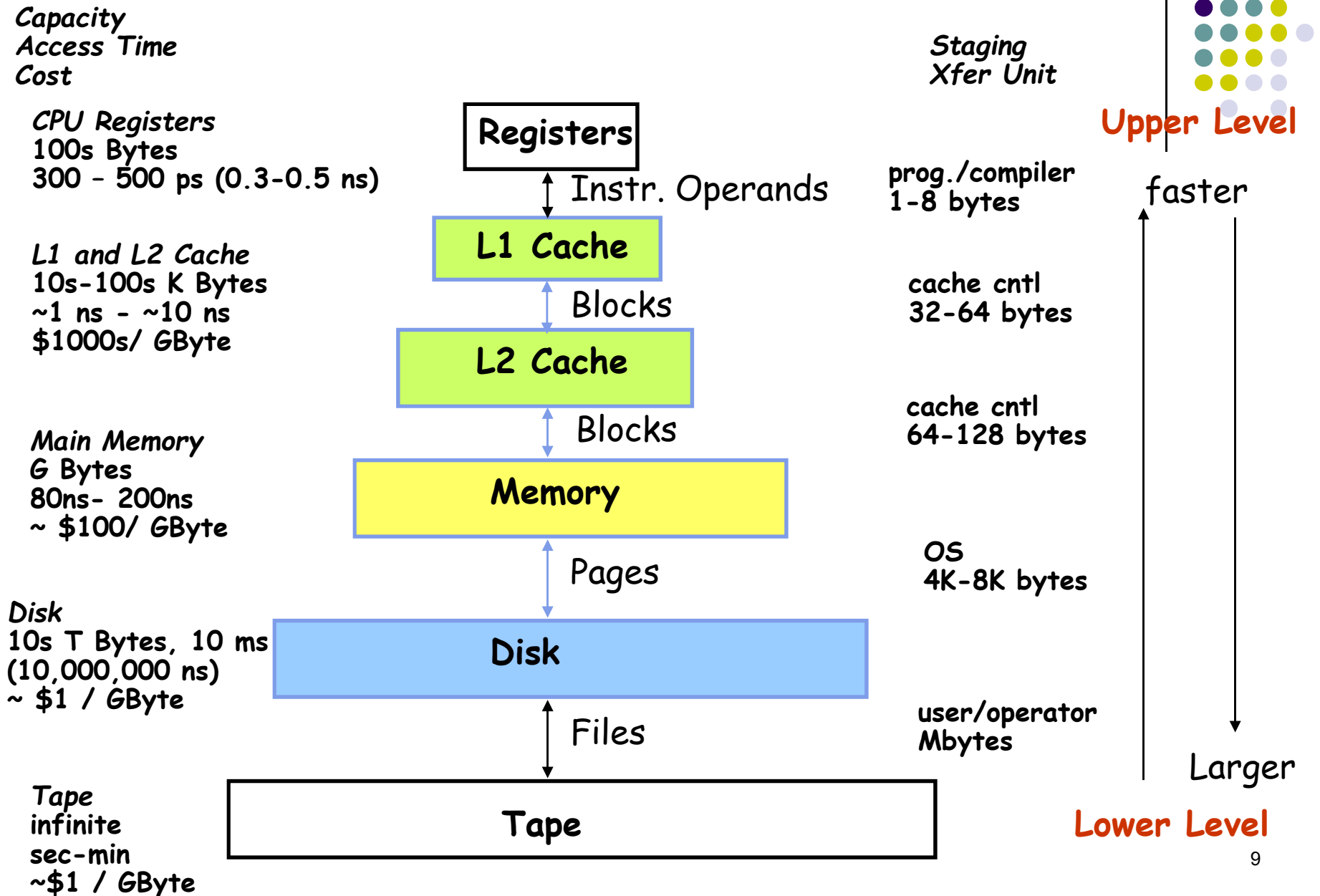
Threads

8

CPU-Z Ver. 1.97.0.x64 Tools Validate Close

- NX / XD / Execute Disable Bit ?
- HT / Hyper-Threading technology ?
- TBT 2.0 / Turbo Boost technology 2.0 ?
- VT-x / Virtualization technology ?
- VT-d / Virtualization for directed I/O

Memory Hierarchy Levels



Control structure



- *Loop (temporal locality)*

```
cout << "Adding the numbers\n";  
for (i = 0; i < v1.size(); i++) {  
    v3[i] = v1[i] + v2[i];  
}
```

- *Sequential (spatial locality)*

```
    lw $t2, 0($t3)  
    lw $t3, 4($t3)  
    beq $t2, $t3, Label #assume not equal  
    add $t5, $t2, $t3  
    sw $t5, 8($t3)  
Label:    j 32
```

Principle of Locality



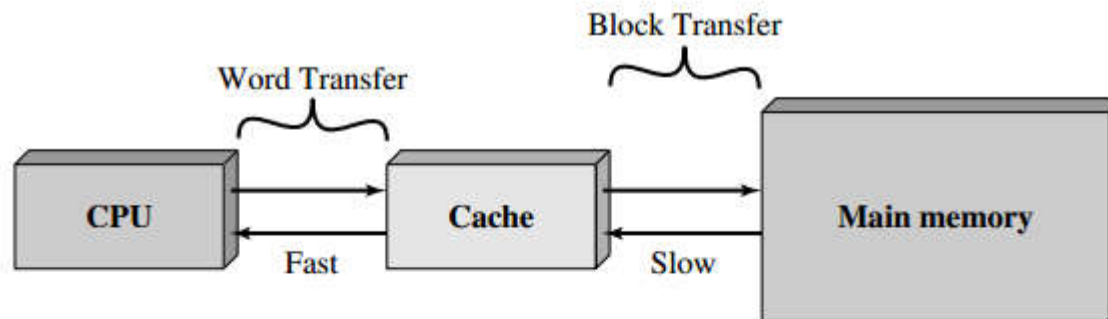
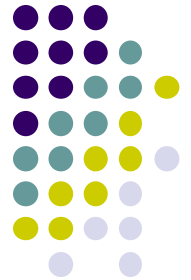
- Programs access a small proportion of their address space at any time
- **Temporal locality**
 - Items accessed recently are likely to be accessed again soon
 - e.g., instructions in a loop, induction variables
- **Spatial locality**
 - Items near those accessed recently are likely to be accessed soon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

Taking Advantage of Locality

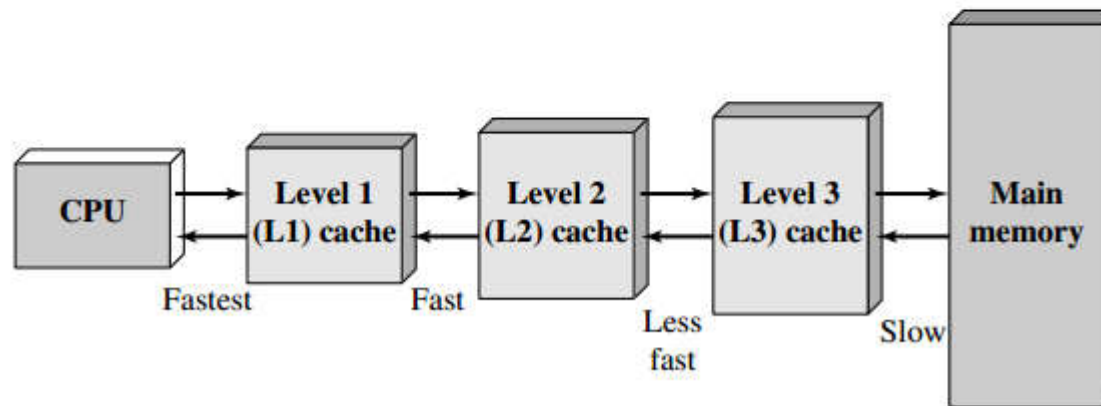


- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
 - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
 - Cache memory attached to CPU

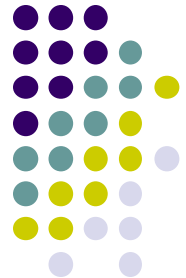
Cache position



(a) Single cache

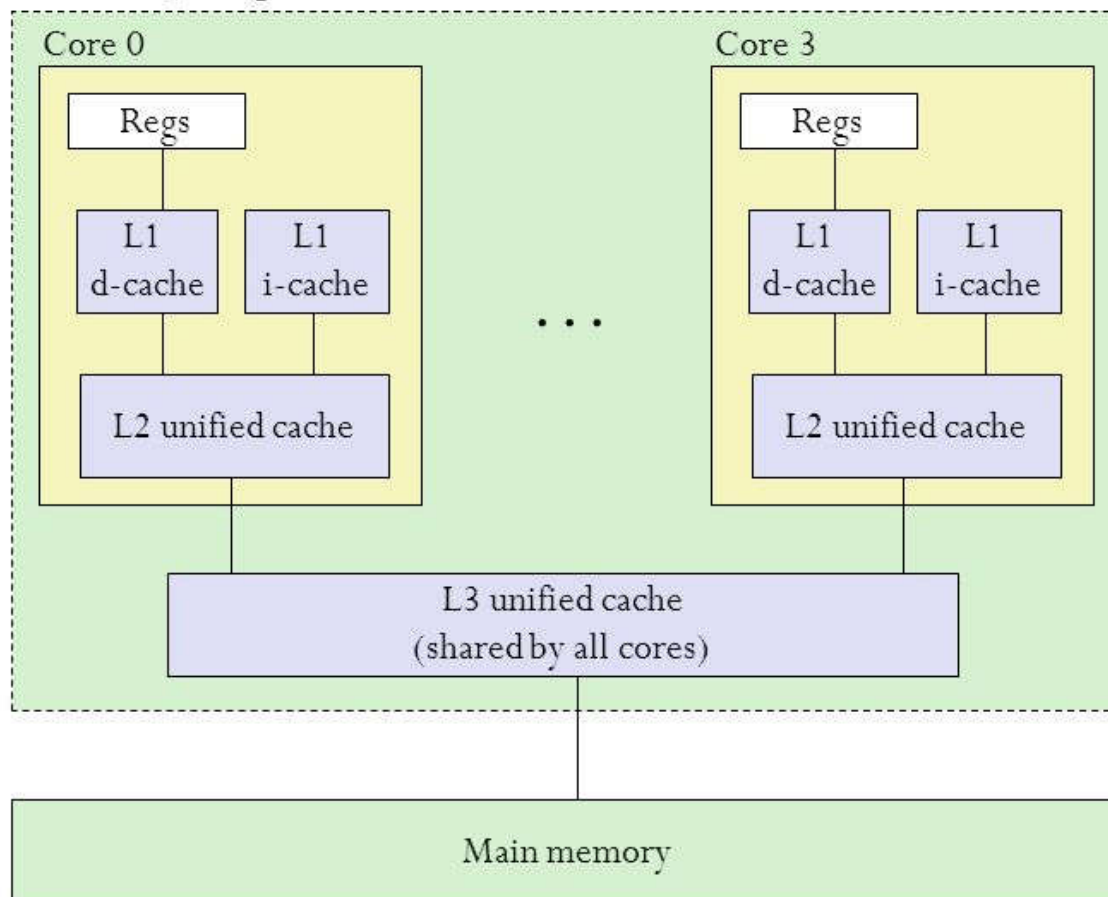


(b) Three-level cache organization



Intel Core i7 Cache Hierarchy

Processor package



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

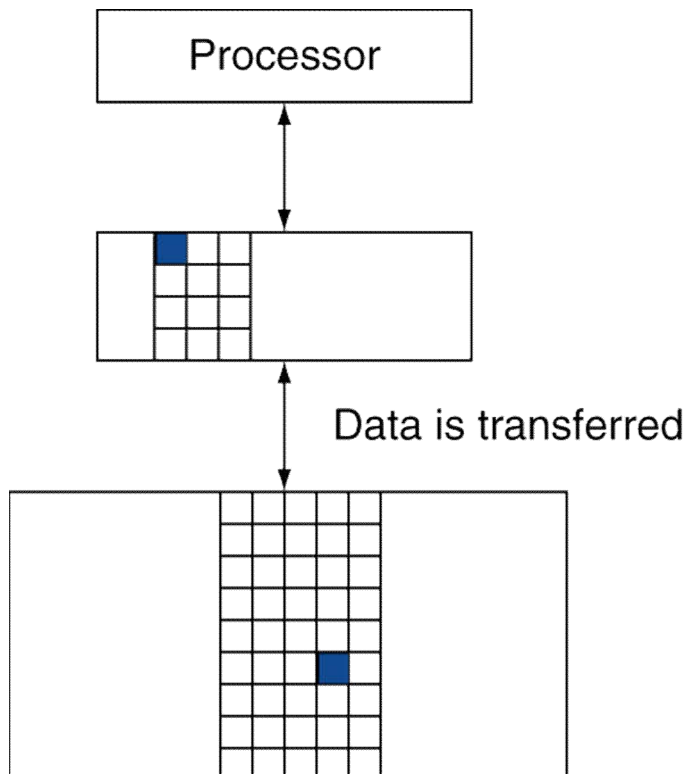
L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for
all caches.

Memory Hierarchy Levels



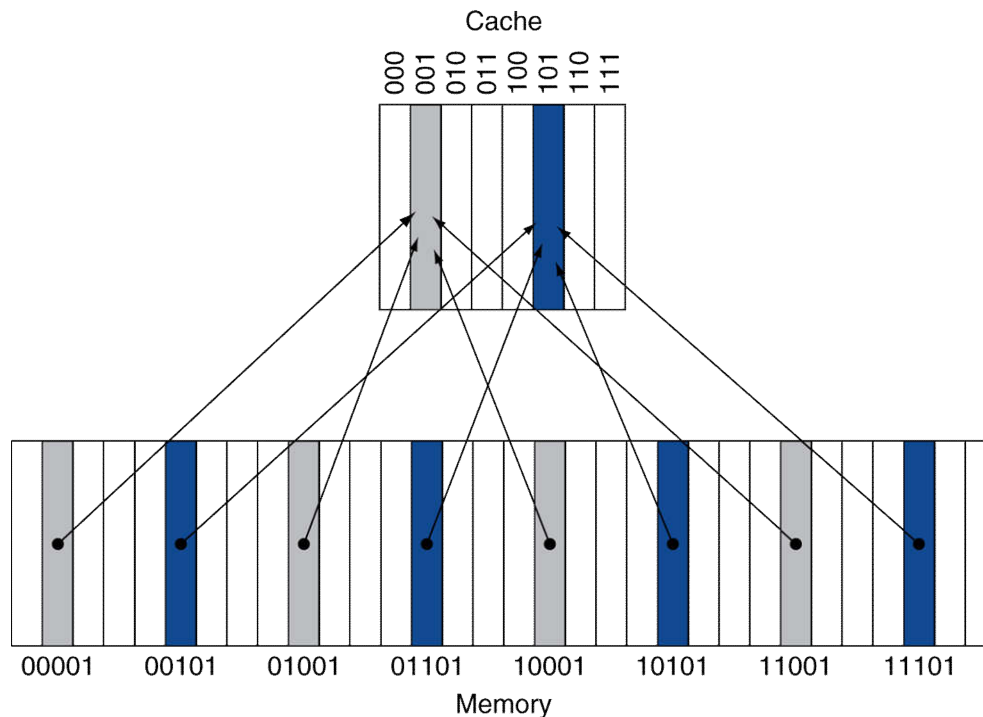
- Block (aka line): unit of copying
 - May be multiple words
- If accessed data is present in upper level
 - Hit: access satisfied by upper level
 - Hit ratio: hits/accesses
- If accessed data is absent
 - Miss: block copied from lower level
 - Time taken: miss penalty
 - Miss ratio: misses/accesses
 $= 1 - \text{hit ratio}$
 - Then accessed data supplied from upper level



Direct Mapped Cache



- Location determined by address
- Direct mapped: only one choice
 - (referenced address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Cache Memory

- Cache memory
 - The level of the memory hierarchy closest to the CPU
- Given accesses X_1, \dots, X_{n-1}, X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

Tags and Valid Bits



- How do we know which particular block is stored in a cache location?
 - Store block address as well as the data
 - Actually, only need the high-order bits
 - Called the tag
- What if there is no data in a location?
 - Valid bit: 1 = present, 0 = not present
 - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state



Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

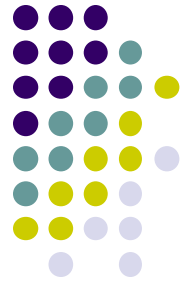
Cache Example (cont'd)



Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (cont'd)



Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (cont'd)



Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

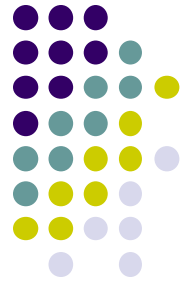
Cache Example (cont'd)



Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (cont'd)



Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Mapping from memory to cache



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	...
Block offset				Block offset				Block offset				Block offset				...
0				1				2				3				...

Program is divided into equal chunks of the same size as the cache block

$\text{BlockIndex} = x / \text{block_size}$

$\text{BlockOffset} = x \bmod \text{block_size}$

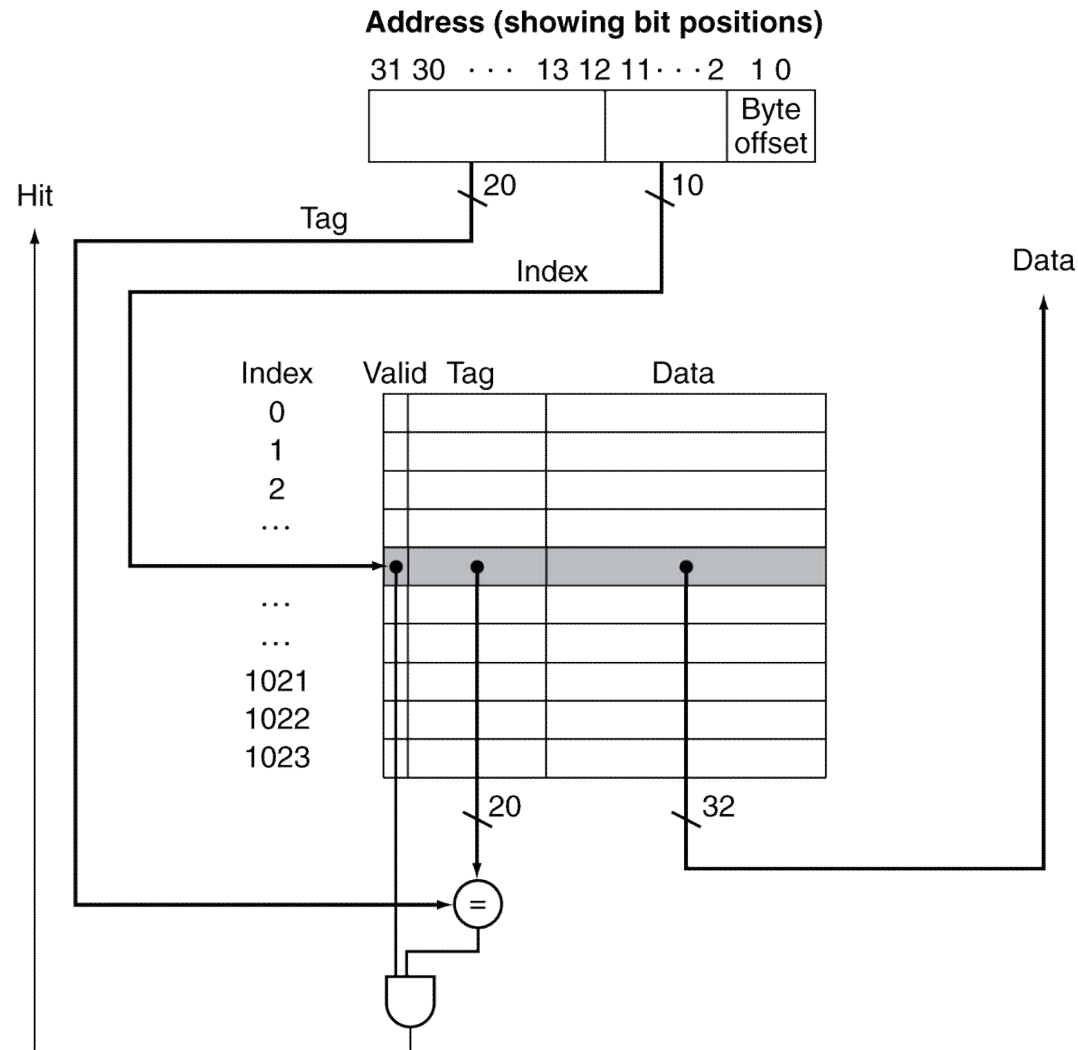
For example: $x=14$

- $\text{BlockIndex} = 14 / 4 = 3$
- $\text{BlockOffset} = 14 \bmod 4 = 2$

Map: use BlockIndex for calculating the Block in the cache =>

$\text{index} = \text{BlockIndex} \bmod \text{\#Cache block}$

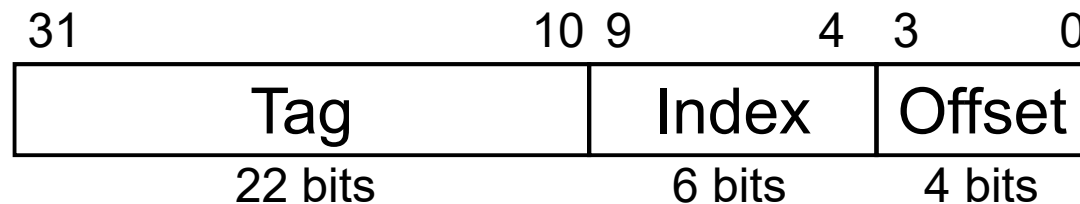
Address Subdivision





Example: Larger Block Size

- 64 blocks, 16 bytes/block, 32 bit address regs
 - To what block number does address 1200 map?
- Offset=?
- Index=?
- Tag=?



Cache usage protocol



- Identify the index, tag, offset of the reference
- Check the valid bit
 - If v=valid, check the tag
 - If two tags are equal => return the data from the offset (cache hit)
 - Else raise a cache miss => Load the data into the WHOLE block, update tag and v=valid, return the data
 - If v=invalid, raise a cache miss
 - Load the data into the WHOLE block, update tag and v=valid, return the data

Example



- Suppose a cache 16KB, block size=32 bytes, address reg=32 bit. Identify the bits for tag, index and offset (ignore tag and valid bit).
- Given the below cache with direct map
 - Cache with 64 blocks, block size=32 bytes, address reg=16 bits
 - Suppose the cache is currently empty
 - Show hit/miss for the reference sequence of MIPS instructions: 184, 188, 192, 196, 200, 204, 208, 212, 216, 192, 196, 200, 204.

Associative Caches

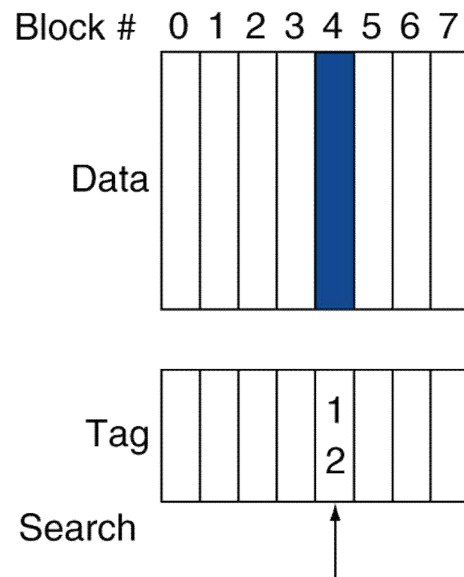


- Fully associative
 - Allow a given data to go in **any cache** block
 - Requires all blocks to be searched at once
 - Comparator per block (expensive)
- *n*-way set associative
 - Each set contains *n* blocks
 - Apply direct map for set, associative for blocks in a set
 - (Block number) modulo (#Sets in cache)
 - Search all blocks in a given set at once
 - *n* comparators (less expensive)

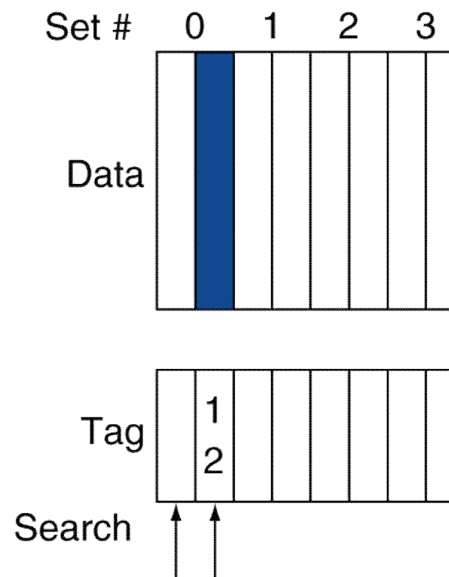
Associative Cache Example



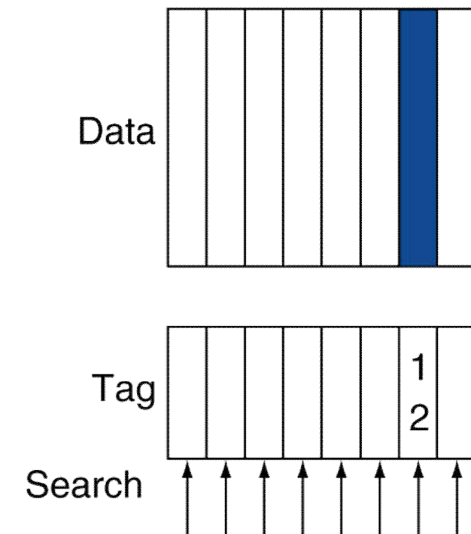
Direct mapped



Set associative



Fully associative



Spectrum of Associativity

- For a cache with 8 entries



**One-way set associative
(direct mapped)**

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Associativity Example



- Compare 4-block caches
 - Direct mapped, 2-way set associative, fully associative
 - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

Associativity Example



- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

- Fully associative

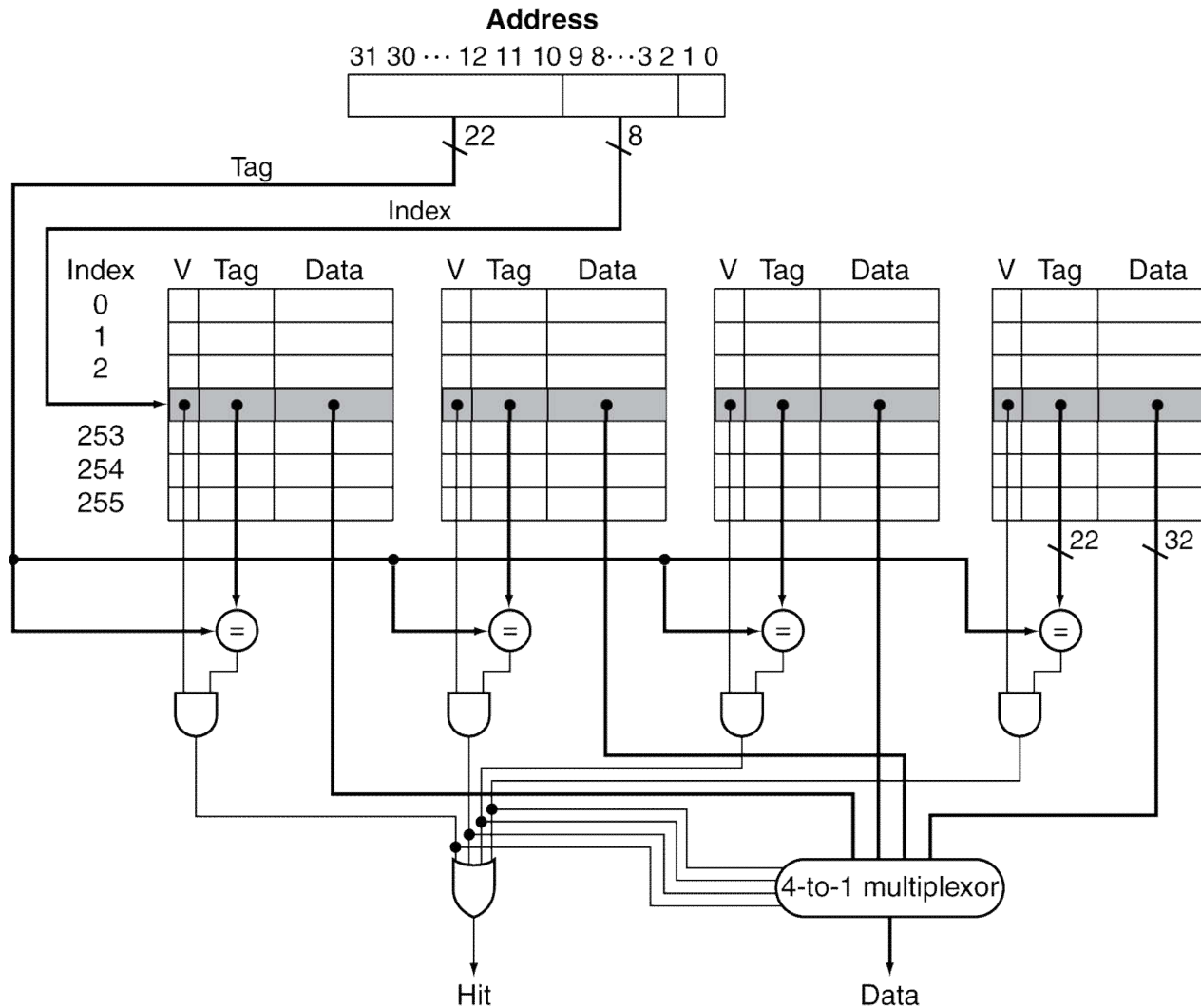
Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

How Much Associativity

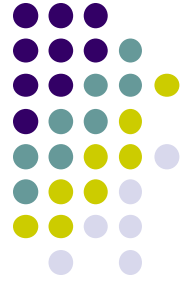


- Increased associativity decreases miss rate
 - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
 - 1-way: 10.3%
 - 2-way: 8.6%
 - 4-way: 8.3%
 - 8-way: 8.1%

Set Associative Cache Organization



Exercise



- Given a 4-way cache, size =128KB (for data), block size=32 bytes, the address register=32 bits.
 - Identify the components of the register.
 - Given a reference $x=1864$, identify the components



Replacement Policy

- Direct mapped: no choice
- Set associative
 - Prefer non-valid entry, if there is one
 - Otherwise, choose among entries in the set
- Least-recently used (LRU)
 - Choose the one unused for the longest time
 - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
 - Gives approximately the same performance as LRU for high associativity



Block Size Considerations

- Larger blocks should reduce miss rate
 - Due to spatial locality
- But in a fixed-sized cache
 - Larger blocks \Rightarrow fewer of them
 - More competition \Rightarrow increased miss rate
 - Larger blocks \Rightarrow pollution
- Larger miss penalty
 - Can override benefit of reduced miss rate
 - Early restart and critical-word-first can help

Cache Misses



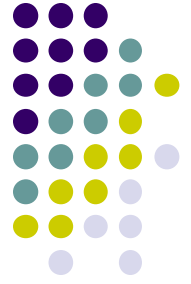
- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - **Fetch** block from next level of hierarchy
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access

Write-Through



- On data-write hit, could just update the block in cache
 - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
 - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
 - Effective CPI = $1 + 0.1 \times 100 = 11$
- Solution: write buffer
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only stalls on write if write buffer is already full

Write-Back



- Alternative: On data-write hit, just update the block in cache
 - Keep track of whether each block is dirty
- When a dirty block is replaced
 - Write it back to memory
 - Can use a write buffer to allow replacing block to be read first

Write Allocation



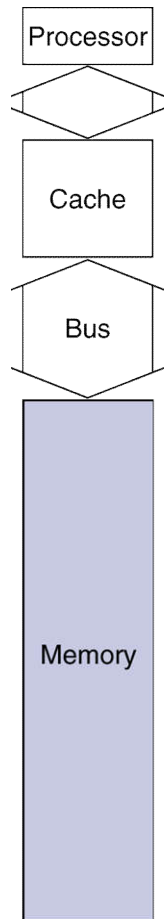
- What should happen on a write miss?
- Alternatives for write-through
 - Allocate on miss: fetch the block
 - Write around: don't fetch the block
 - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
 - Usually fetch the block

Main Memory Supporting Caches

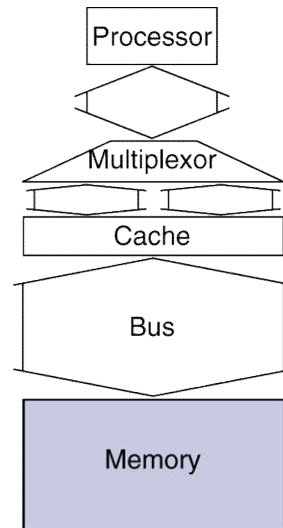


- Use DRAMs for main memory
 - Fixed width (e.g., 1 word)
 - Connected by fixed-width clocked bus
 - Bus clock is typically slower than CPU clock
- Example cache block read
 - 1 bus cycle for address transfer
 - 15 bus cycles per DRAM access
 - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
 - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 65 \text{ cycles} = 0.25 \text{ B/cycle}$

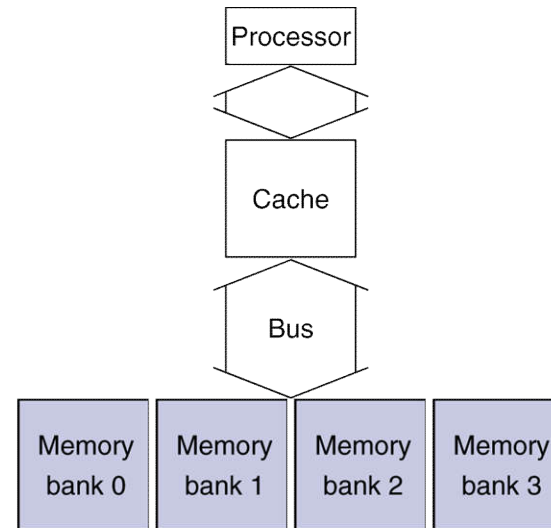
Increasing Memory Bandwidth



a. One-word-wide memory organization



b. Wider memory organization



c. Interleaved memory organization

- 4-word wide memory
 - Miss penalty = $1 + 15 + 1 = 17$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 17 \text{ cycles} = 0.94 \text{ B/cycle}$
- 4-bank interleaved memory
 - Miss penalty = $1 + 15 + 4 \times 1 = 20$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 20 \text{ cycles} = 0.8 \text{ B/cycle}$



Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
 - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Effective CPI
 - $ECPI = CPI(\text{ideal}) + \text{miss rate}_1 \times \text{Miss penalty}_1 + \dots$
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
 - $AMAT = 1 + 0.05 \times 20 = 2$
 - 2 cycles per instruction

Average Access Time



- If there is no cache
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 100%
 - $AMAT = 1 \times 20 = 20$
 - CPU with cache is $20/2=10$ times

Cache Performance Example



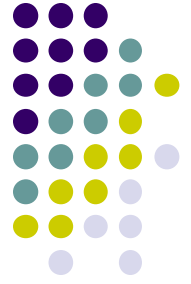
- Given
 - I-cache (instruction) miss rate = 2%
 - D-cache (data) miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Loads & stores are 36% of instructions
- Miss cycles per instruction
 - I-cache: $0.02 \times 100 = 2$
 - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = $2 + 2 + 1.44 = 5.44$
 - Ideal CPU is $5.44/2 = 2.72$ times faster

Cache Performance Example



- If there is no cache
 - Miss rate=100%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- Miss cycles per instruction
 - I-cache: $1 \times 100 = 100$
 - D-cache: $0.36 \times 1 \times 100 = 36$
- Effective CPI = $2 + 100 + 36 = 138$
 - CPU with cache is $138/5.44 = 25.4$ times faster

Multilevel Caches



- Primary cache attached to CPU
 - Small, but fast
- Level-2 cache services misses from primary cache
 - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache



Multilevel Cache Example

- Given
 - CPU base CPI = 1, clock rate = 4GHz
 - Miss rate/instruction = 2%
 - Main memory access time = 100ns
- With just primary cache
 - CPU clock cycle = $1/(4 \times 10^9) = \frac{1}{4} \text{ ns} = 0.25 \text{ ns}$
 - Miss penalty = $100 \text{ ns} / 0.25 \text{ ns} = 400 \text{ cycles}$
 - Effective CPI = $1 + 0.02 \times 400 = 9$



Multilevel Cache Example

- No cache
 - CPU base CPI = 1, clock rate = 4GHz
 - Miss rate/instruction = 100%
 - Main memory access time = 100ns
- Performance
 - CPU clock cycle = $1/(4 \times 10^9) = \frac{1}{4} \text{ ns} = 0.25 \text{ ns}$
 - Miss penalty = $100 \text{ ns} / 0.25 \text{ ns} = 400 \text{ cycles}$
 - Effective CPI = $1 + 1 \times 400 = 401$
 - CPU with cache is $401/9 = 44.6$ times faster

Example (cont.)

	CPU	
hit	CACHE L1	miss
		2%
hit	CACHE L2	miss
		0.50%
	RAM	



- Now add L-2 cache
 - Access time = 5ns
 - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
 - Penalty = $5\text{ns}/0.25\text{ns} = 20$ cycles
- Primary miss with L-2 miss
 - Extra penalty = 500 cycles
- $\text{ECPI} = 1 + 0.02 \times 20 + 0.005 \times 500 = 3.9$
- Improved performance = $9/3.9 = 2.3$ times

Example (cont.)



- No cache
- CPU with cache performance ratio is $401/3.9$
= 102.8 times faster

Performance Summary



- When CPU performance increased
 - Miss penalty becomes more significant
- Decreasing base CPI
 - Greater proportion of time spent on memory stalls
- Increasing clock rate
 - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance