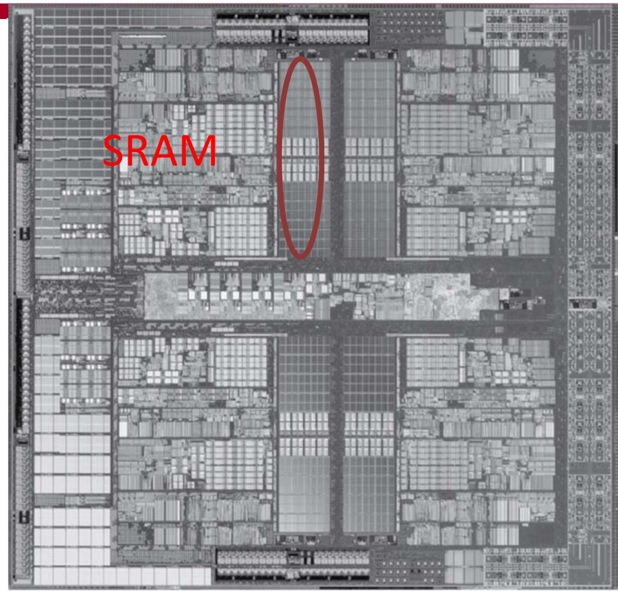
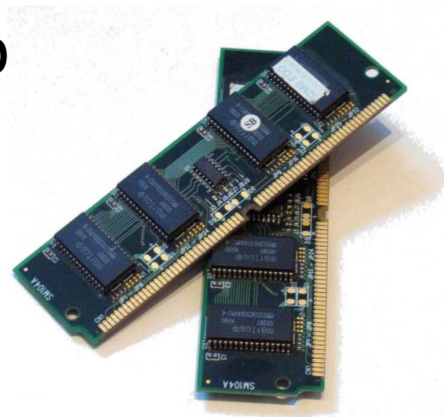
Chapter 5

Large and Fast: Exploiting Memory Hierarchy

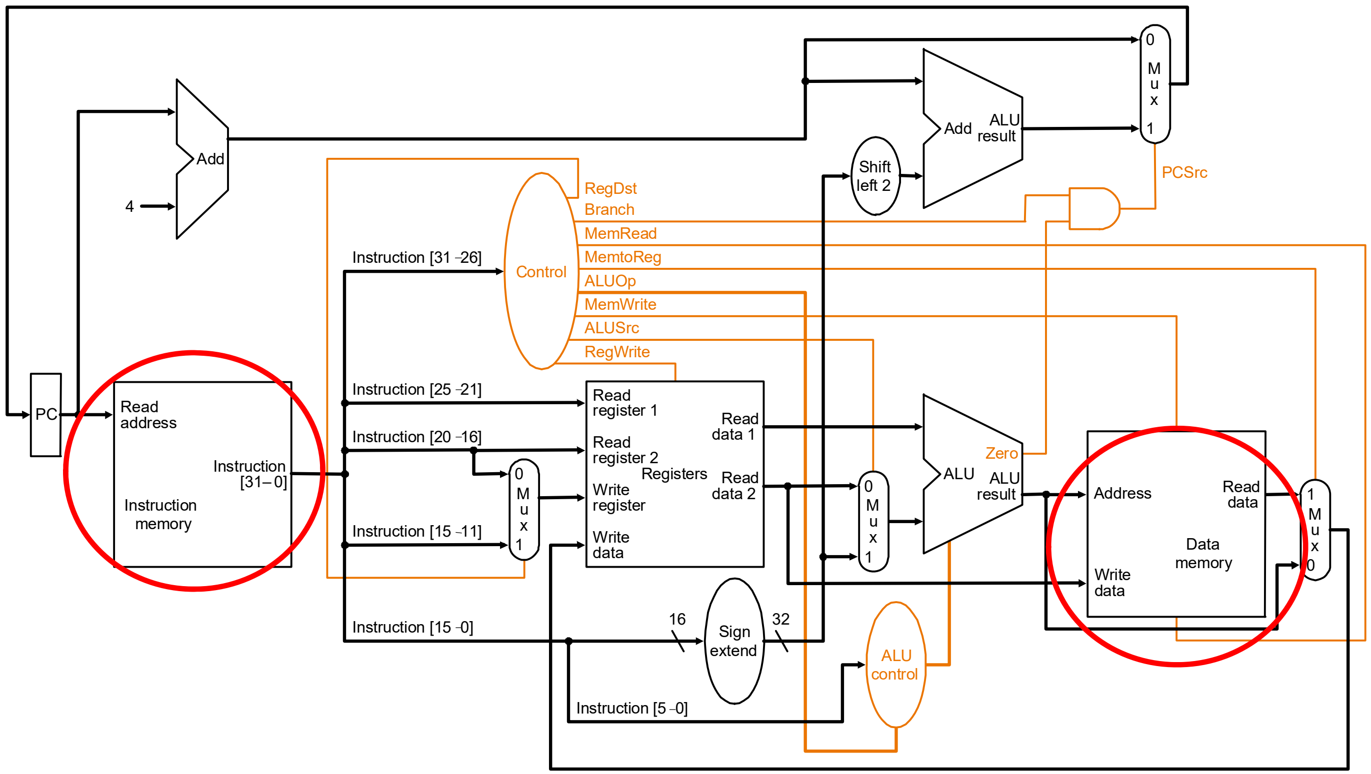
Memory Technology



- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $20 – $75 per GB
- Magnetic disk
  - 5ms – 20ms, $0.20 – $2 per GB
- Ideal memory (fast and cheap
  - Access time of SRAM
  - Capacity and cost/GB of disk

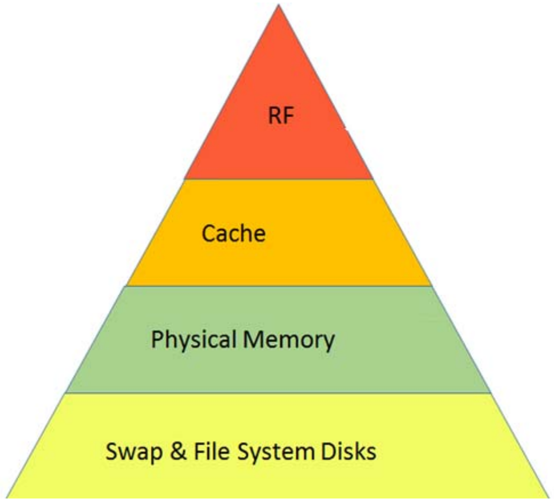


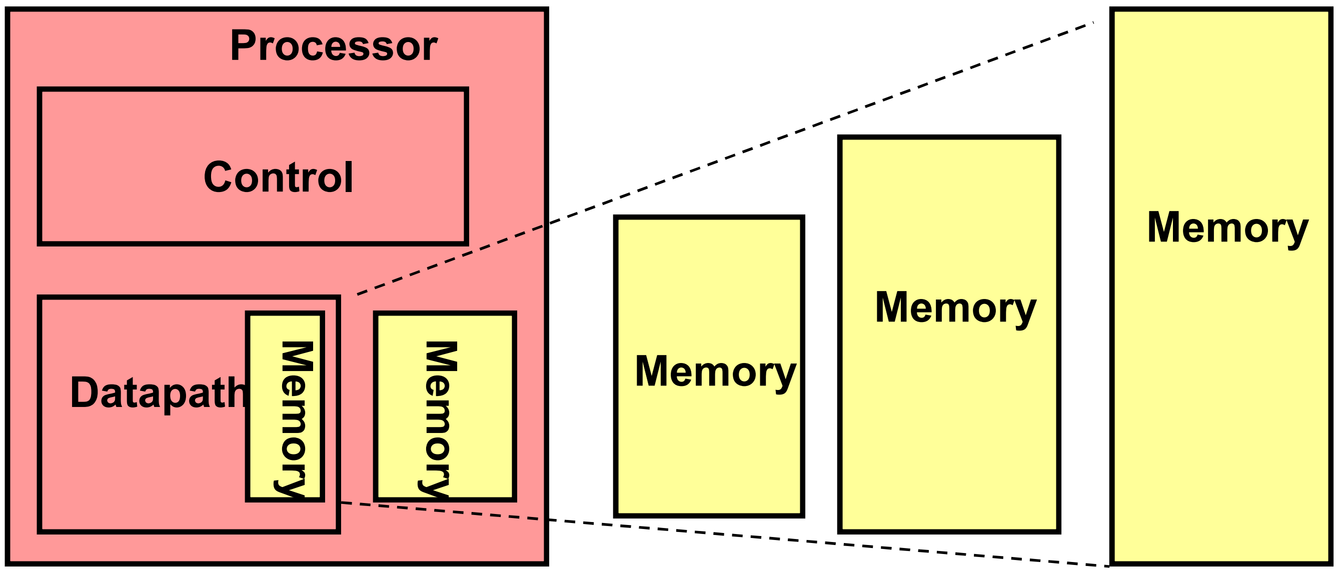
Recap: Single-Cycle MIPS



Two memory are used: Instruction and data memory

Memory Hierarchy: Concept





Speed: Fastest → Slowest

Size: Smallest → Biggest

Cost: Highest → Lowest

Why Hierarchy Works?

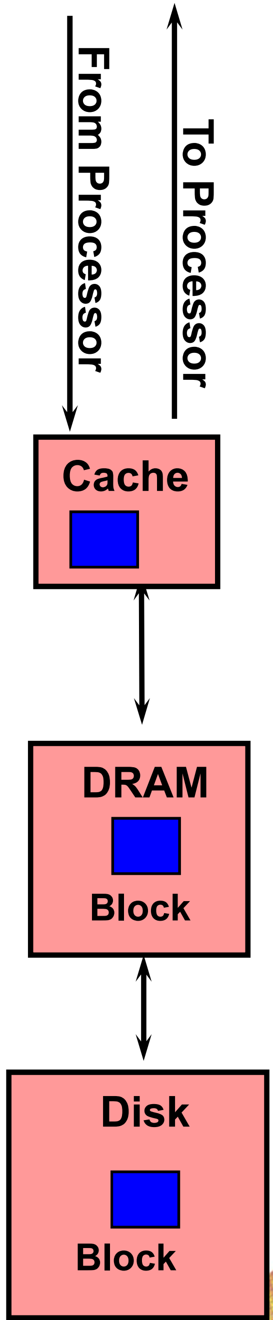
- Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time
  - 90/10 rule or 80/20 rule: 10% (20%) of code executes 90% (80%) of time
- Types of locality:
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon
    - E.g., some counter variable in a loop
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon
    - E.g., nearby array data A[i] and A[i+1]

```
for (i=0; i<1000; i++) {
  A[i+1]=A[i]+1
}
```

- i is referenced frequently
- A[i+1], A[i] are nearby

Taking Advantage of Locality

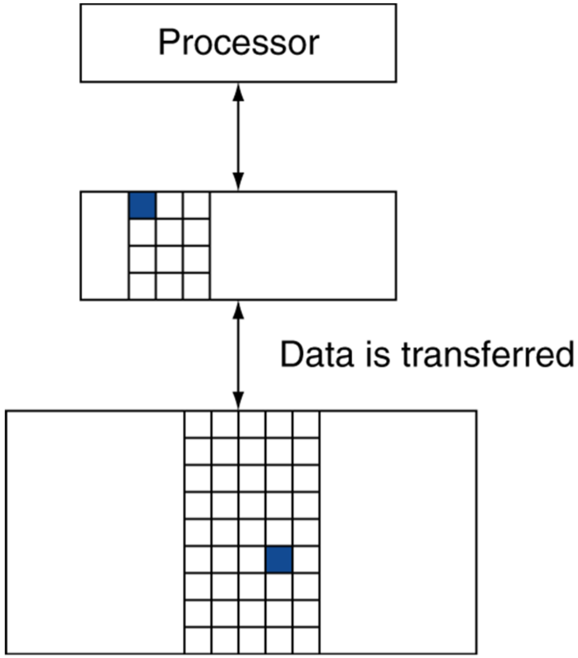
- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU



Memory Hierarchy Levels

- Block (aka line): unit of copying, may be multiple words

- Hit: data is present in upper level
  - Access satisfied by upper level
- Miss: data is absent in upper level
  - data needs to be retrieved from a block in the lower level (Block Y)
  - Then accessed data are supplied from upper level



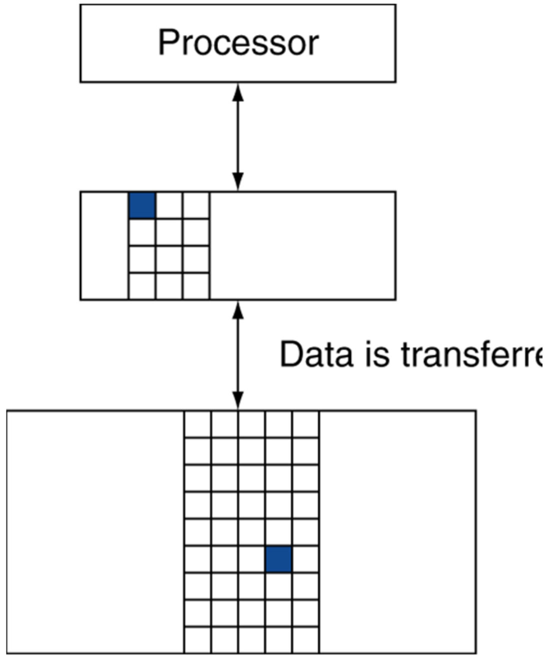
Memory Hierarchy Levels: Terminology

- Hit rate (ratio): fraction of memory access found in the upper level

  Hit Rate= hits / accesses

- Hit time: time to access the upper level
- Miss rate = 1 - (Hit rate)
- Miss penalty:
  - Time to place (replace) a block (with a block) in the upper level + time to deliver the block to the processor

  =Time to determine hit/miss + Mem access time



Hit Time << Miss Penalty ⇨ It is important to reduce miss rate

Exercise

- Suppose 20 misses per 1000 access. The total number of accesses is 10,000, and miss penalty is 100 cycles

1. What is the miss rate?

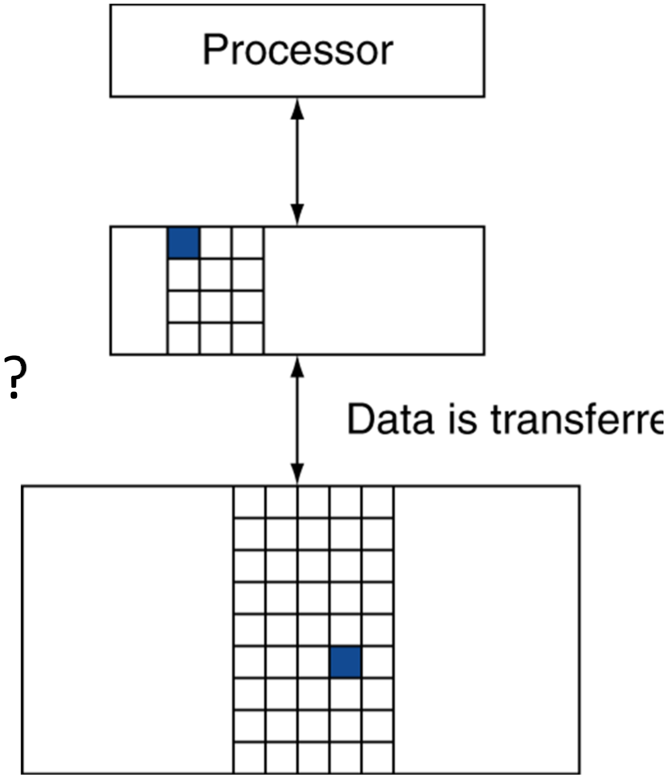
   20/1000=2%

2. What are the extra cycles caused by miss?

   10000*2%*100=20000

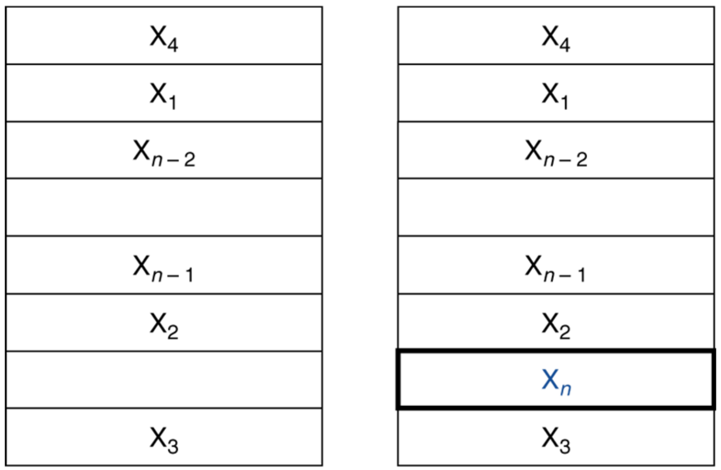
3. What are the total cycles for the 10,000 access? Assume 1 cycle for hit access

   10000+10000*2%*100=30000

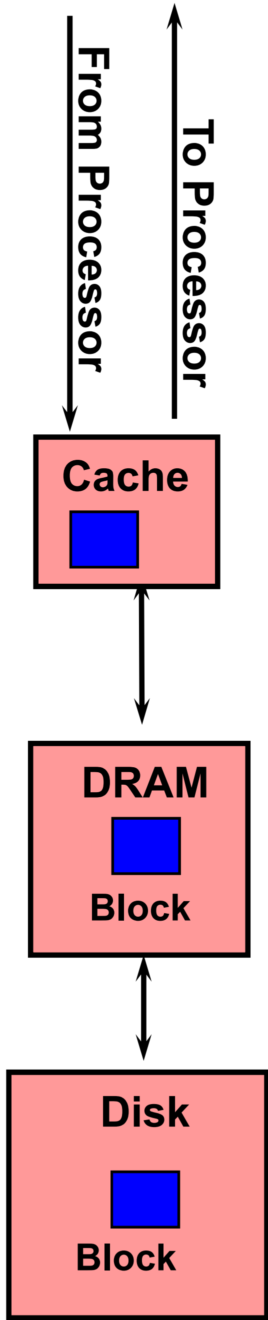


Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$, how do we know if the data is present? Where do we look?
  - First technique: directed mapped



a. Before the reference to X_n b. After the reference to X_n

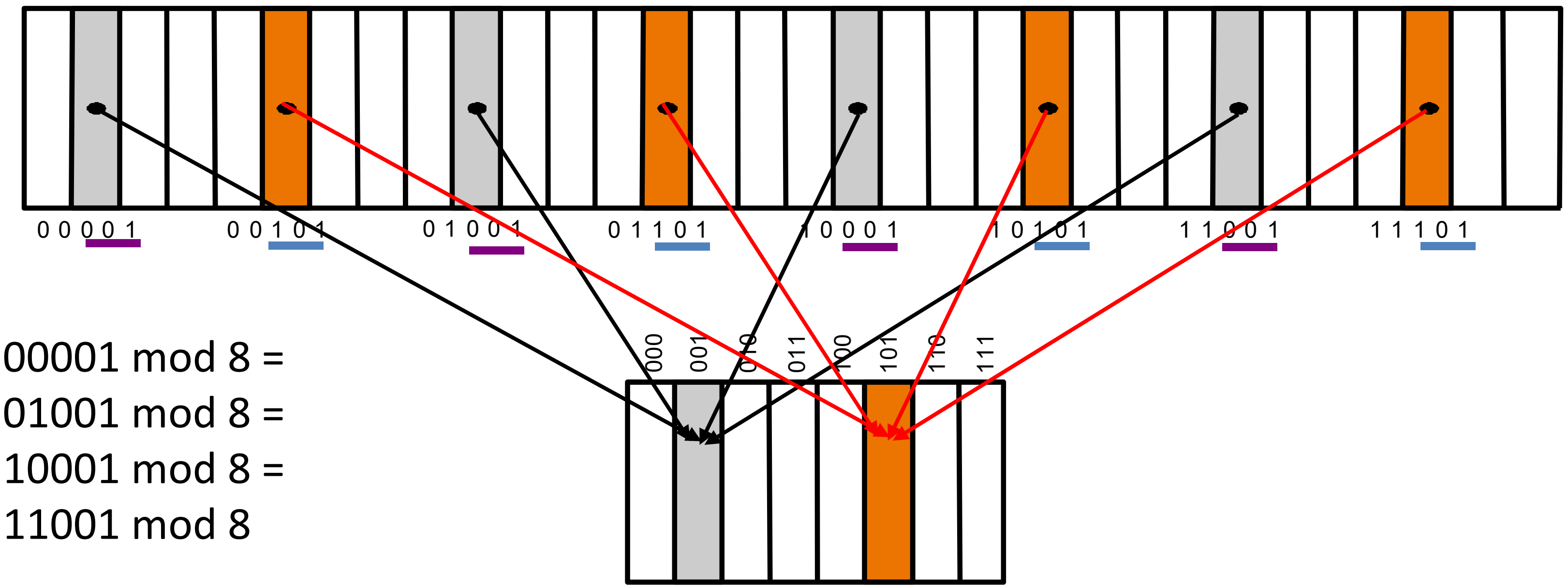


Basics of Cache – Direct-mapped cache

tag	index	Block Offset	
tag	index	W.O.	B.O.

Block address (tag + index) — 2-bit (B.O.)

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)
- 32-block main memory mapped into a 8-block cache block address are 00000, 00001, 00010…,



00001 mod 8 =
01001 mod 8 =
10001 mod 8 =
11001 mod 8

Cache Example

- Cache 8-blocks, direct mapped
- Accessed Block Address are: 10110, 11010, 10110, 11010, 10000, 00011, 10000, 10010, 10000
- Initial state

Valid bit: 1=present, 0 = not present

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

Cache Example (access 10110_2)

	Decimal addr	Binary addr	Hit/miss	Assigned cache block (Index)
(1)	22	10110	Miss	110

(1) Address referred 10110 (*miss*):

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 (access 11010_2)

	Decimal addr	Binary addr	Hit/miss	Assigned cache block (Index)
(1)	22	10110	Miss	110
(2)	26	11010	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

	Decimal addr	Binary addr	Hit/miss	Cache block
(3)	22	10 110	Hit	110
(4)	26	11 010	Hit	010

Nothing is changed in cache

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

	Decimal addr	Binary addr	Hit/miss	Cache block
(5)	16	10 000	Miss	000
(6)	3	00 011	Miss	011
(7)	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

	Decimal addr	Binary addr	Hit/miss	Cache block
(8)	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		

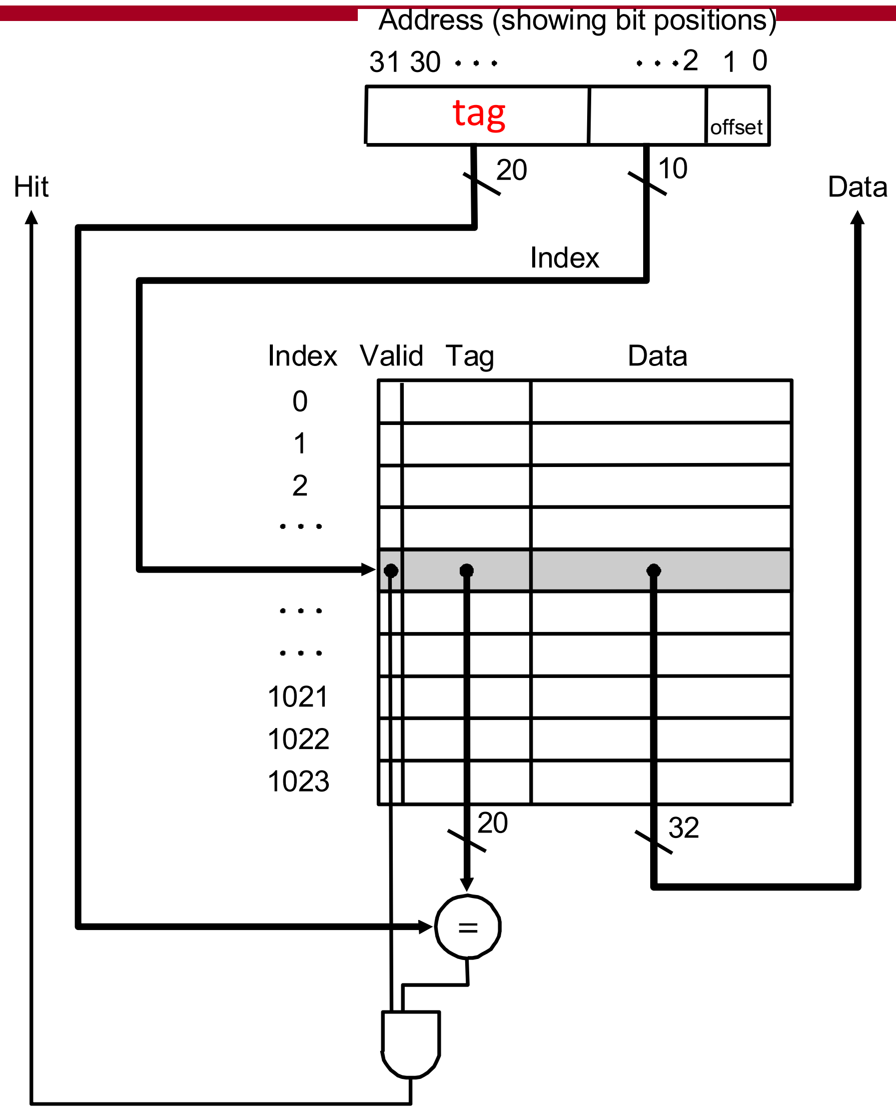
Direct-Mapped Cache Organization

How do we know where a block is stored in a cache?

- Use Tag and valid bit
- Example, cache has 1024 blocks, each block is 1-word (4bytes)
  - Offset : 2 bits (2^2=4)
  - Cache index: lower 10 bits (2^{10}=1024)
  - Cache tag: upper 20 bits (32-2-10)
  - Valid bit (when start up, valid is 0)

tag	index	offset	
tag	index	W.O.	B.O.

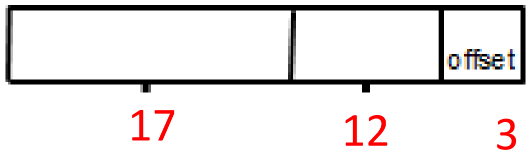
$\underbrace{\text{tag} \quad \text{index}}_{\text{Block address}}$



Example

Address has 32 bits. A direct-mapped cache has 4096 blocks, each block is 2-word.

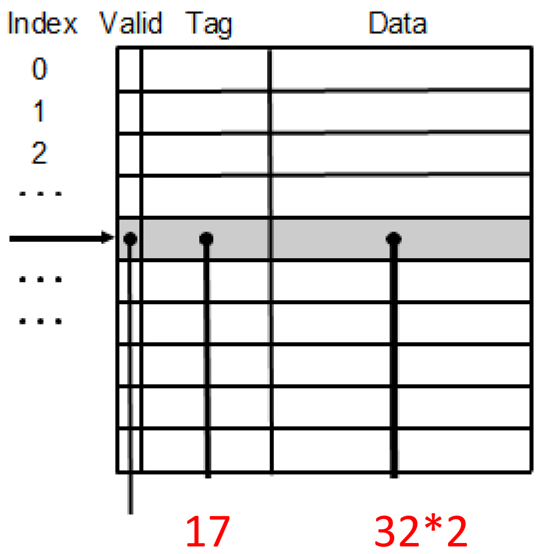
1) What is the width for the offset?



2 words = 8 bytes = 2^3 Width=3

2) What is the bit width for cache index?

2^{12}=4096 Width= 12



3) What is the bit length for tag?

Width=32-3-12= 17

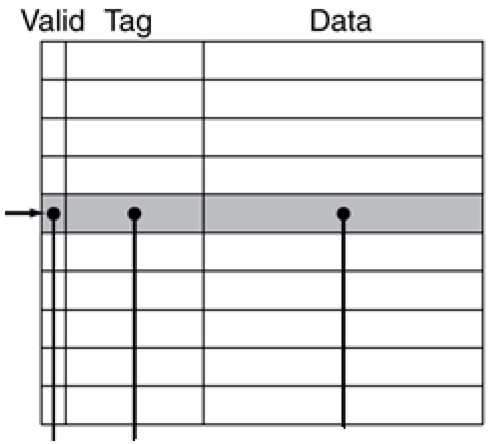
4) Cache entry size

(1+32*2+17) bits

5)Total cache size (in bits)

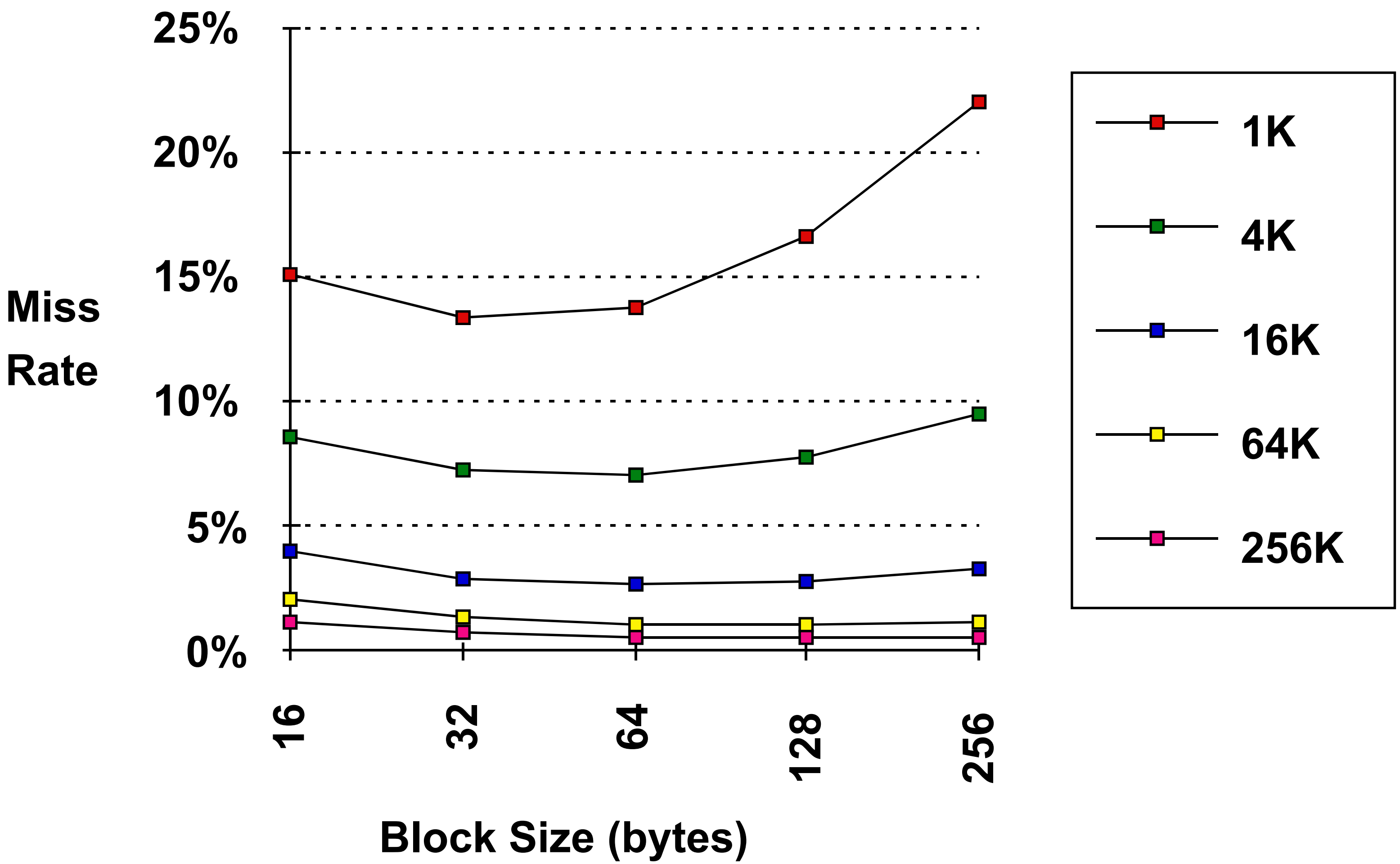
(1+32*2+17)x 4096

Block Size Considerations



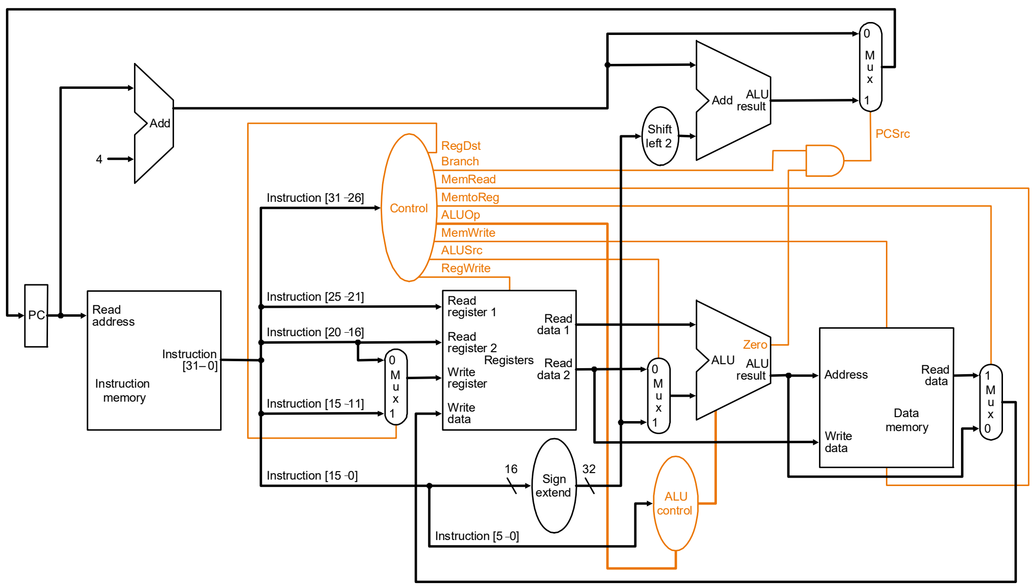
- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ less number of blocks
    - More address compete for a block
      ⇒ increased miss rate
  - Larger blocks ⇒ more data are not used in a block ⇒ pollution
  - Larger blocks ⇒ Larger miss penalty
    - Take more time to move a block
    - Can override benefit of reduced miss rate

Miss Rate vs. Block Size



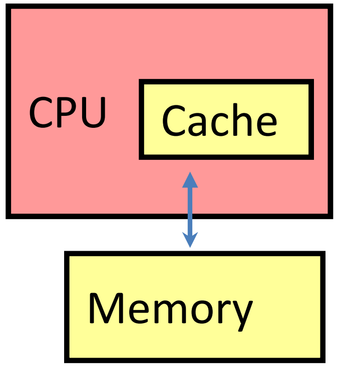
Cache Misses

- 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



Handling Writes

- On data-write hit, if only block in cache is updated
  - cache and memory would be inconsistent
- Two policies: Write Through and Write back
- Write through: update both cache and memory
  - makes writes take longer because needs to wait for MEM operation

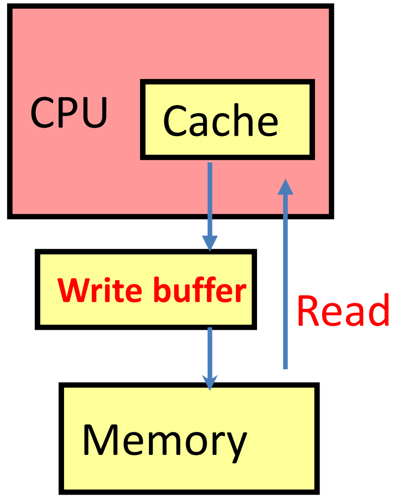


e.g. if base CPI = 1, write to memory takes 100 cycles (miss penalty), miss rate is 10%, find new CPI

New CPI= 1+10%x 100 = 11

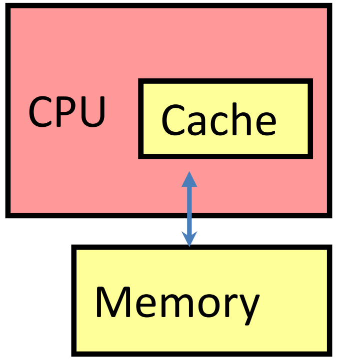
Handling Writes – Write through

- Write through: update both cache and memory
- Improve the performance of write through using write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

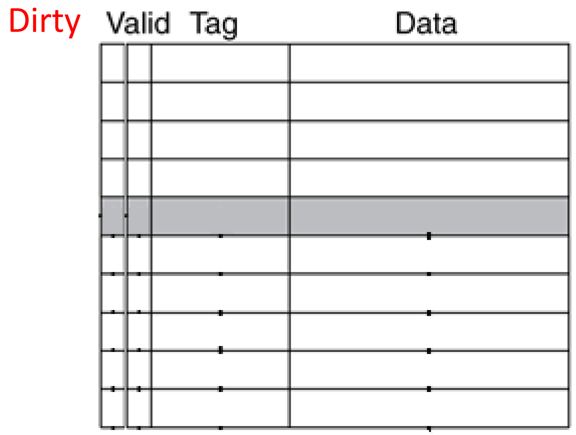


Handling Writes -- Write-Back

- On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty (need a dirty bit)
  - When Dirty is 1, mean it is already modified (cache and memory data are different)

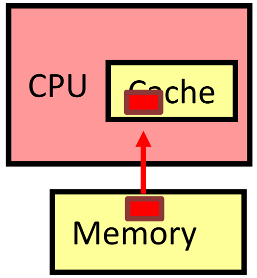


- When a dirty block is replaced
  - Write it back to memory
  - Can use a write back buffer to allow replacing block to be read first

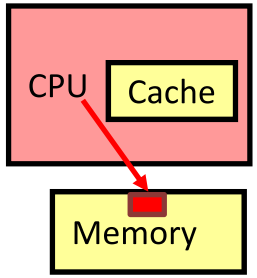


Handling write miss: Two options

- What should happen on a write miss (write a block that is not in cache)?
  - Write allocate and No-write allocate
- Write allocate
  - Allocate block in cache (both cache and memory has the block), and write cache and memory
  - Write miss is like read miss
  - Simple



- No-write allocate
  - Do not allocate block in cache. Write block in memory only
  - Block stay in memory until it is read (Only reads are cached)
  - More cache space for read (reduce miss rate)
  - Useful for initialization (initialize data before use it)



Write back or through vs. write allocate or no-write allocate

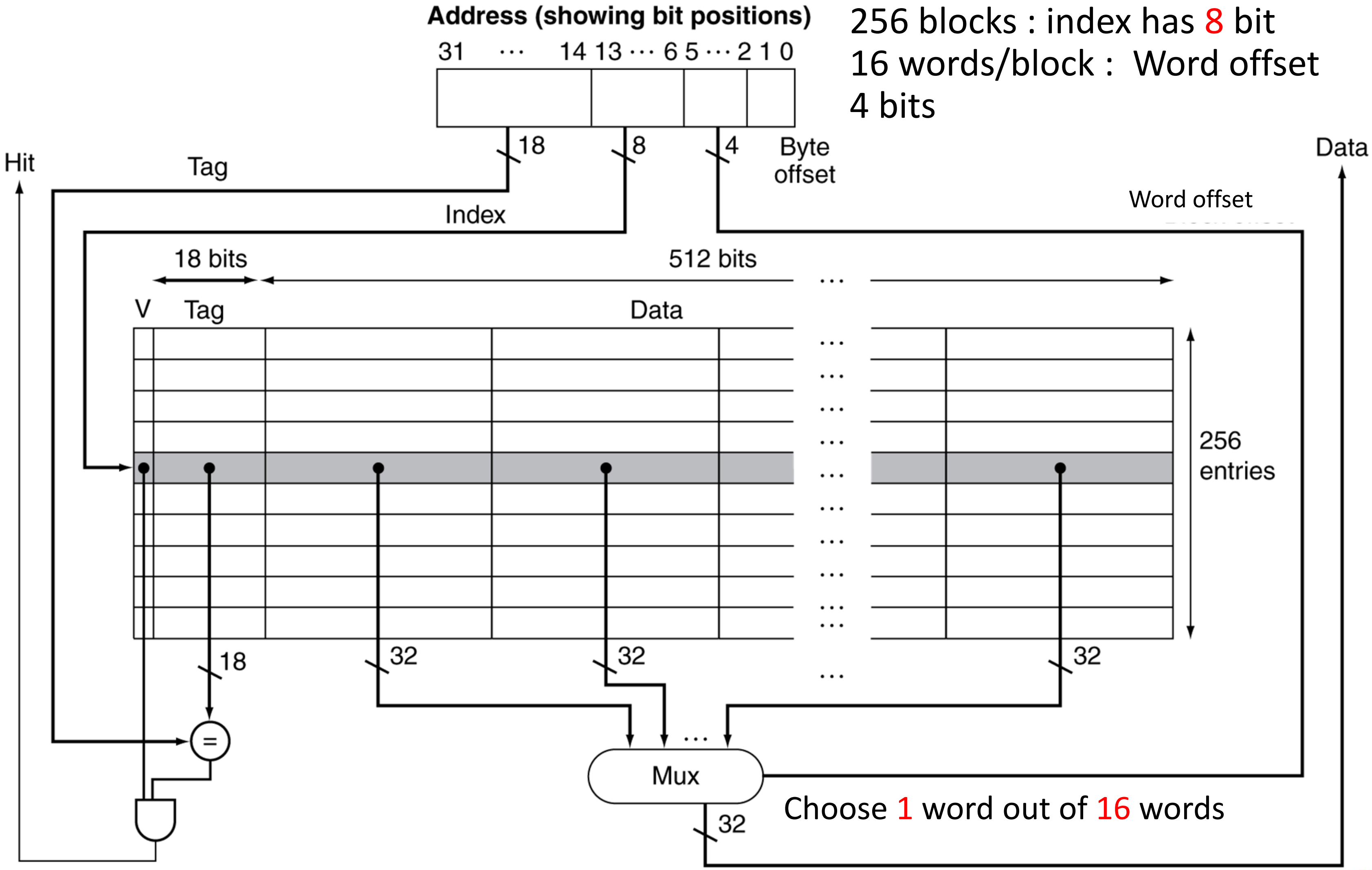
- Both write-through and write-back policies can use either of these write-miss policies, but usually they are paired in this way

	Write back	**Write through**
Write allocate	Subsequent writes (or even reads) to the same location, which is now cached.	
No write allocate		Subsequent writes still need to be written to the lower mem. => no advantage But cache space is saved for read

Example: Intrinsity FastMATH

- Embedded MIPS processor (See next slides)
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH

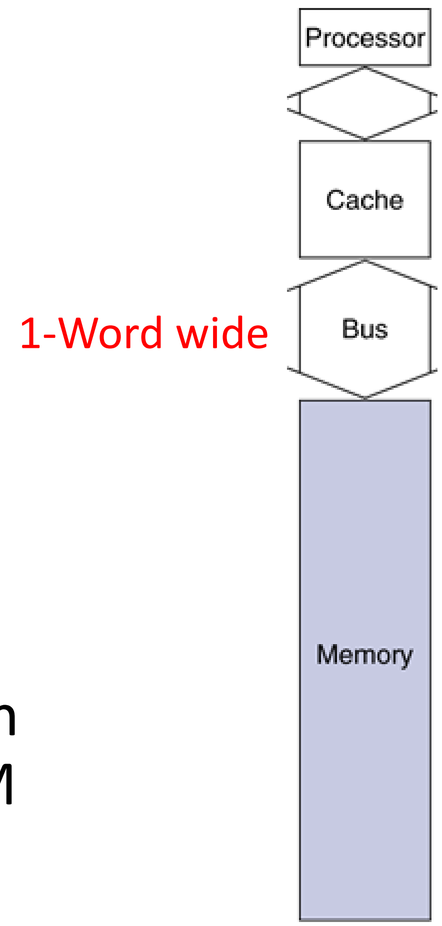


256 blocks : index has 8 bit
16 words/block : Word offset 4 bits

Choose 1 word out of 16 words

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
- e.g. cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access(read)
  - 1 bus cycle per data transfer



E.g. What is the miss penalty and bandwidth when transferring a 4-word block on 1-word-wide DRAM

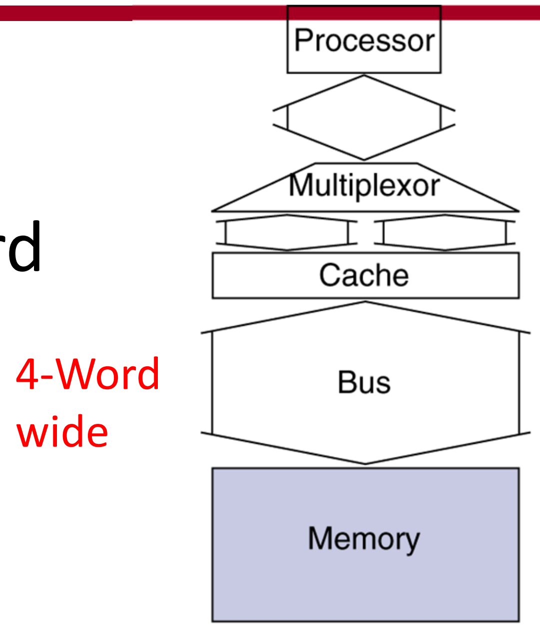
Miss penalty = $1 + 4\times15 + 4\times1 = 65$ bus cycles

Note: address transfer is 1 cycle because of sequential access

Bandwidth = 16 bytes / 65 cycles = 0.25 Bytes/cycle

Increasing Memory Bandwidth

- Method1: Wider memory and bus width
  - Access 4-word and transfer 4-word at a one time



b. Wider memory organization

E.g. What is the miss penalty and Bandwidth when transferring a 4-word block on 4-word-wide DRAM?

- Miss penalty = 1 + 15 + 1 = 17 bus cycles
- Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle

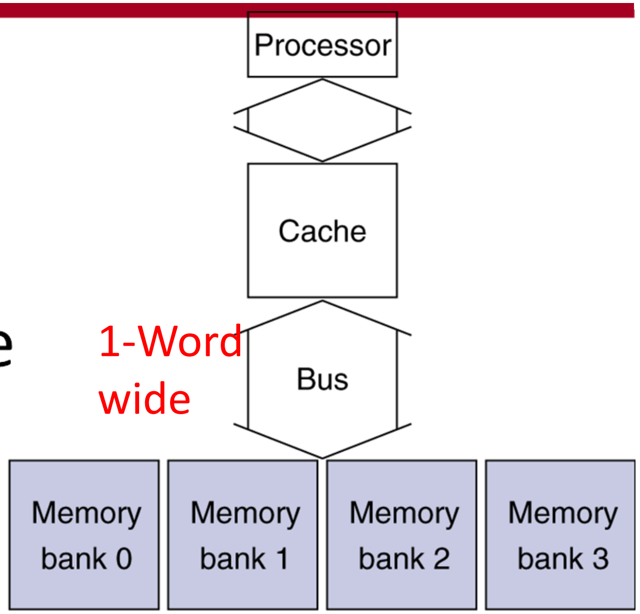
Increase bandwidth, but memory and bus cost is high

Increasing Memory Bandwidth

- Method 2: Interleaved memory organize
  - Bus width is the same
  - Sending address to 4 banks at the time

  => Four memory bank can be accessed at the same time

  => overlap the access time



c. Interleaved memory organization

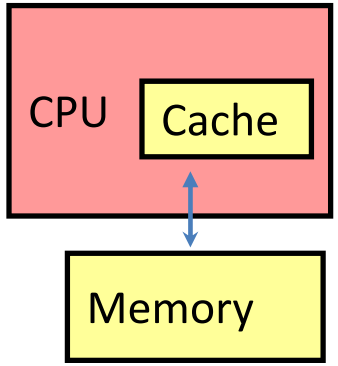
E.g. What is the miss penalty and Bandwidth when transferring a 4-word block on an interleaved DRAM?

Miss penalty = $1 + 15 + 4 \times 1 = 20$ bus cycles

Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses



$$\text{CPU Time} = (\text{CPU Clock cycle} + \text{Memory stall cycle}) \times \text{Cycle Time}$$

$$\text{Memory stall cycles} = (\text{Read\_stall cycle} + \text{Write\_stall cycle})$$

where

$$\text{Read stall cycle} = \text{Reads} \times \text{Read Miss Rate} \times \text{Read Miss Penalty}$$

$$\text{Write stall cycle} = \text{Writes} \times \text{Writes Miss Rate} \times \text{Write Miss Penalty} + \text{Write buffer stalls}$$

Measuring Cache Performance-2

In write-through cache, Read Miss Penalty and Write Miss Penalty are the same, and write buffer stalls can be ignored

$$\text{Memory stall cycles} = (\text{Read\_stall cycle} + \text{Write\_stall cycle})$$

$$\text{Memory stall cycles} = \text{Memory Accesses} \times \text{Miss Rate} \times \text{Miss Penalty}$$

$$\text{Memory stall per Instr.} = \text{Memory Access Per Instr.} \times \text{Miss Rate} \times \text{Miss Penalty}$$

$$\text{Memory stall cycles} = \text{Instrutions} \times \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss Penalty}$$

$$\text{Memory stall cycles Per Instr.} = \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss Penalty}$$

Measuring Cache Performance -CPI

$$\text{Memory stall per Instr.} = \text{Memory Access Per Instrrction} \times \text{Miss Rate} \times \text{Miss Penalty}$$

$$\text{Memory stall cycles Per Instr.} = \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss Penalty}$$

$$\text{CPI with stall} = \text{Base CPI} + \text{Memory stall cycle per Instr.}$$

$$\text{CPI with stall} = \text{Base CPI} + \text{Memory Access Per Instrutions} \times \text{Miss Rate} \times \text{Miss Penalty}$$

$$\text{CPI with stall} = \text{Base CPI} + \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss Penalty}$$

Cache Performance-Split cache

- Given a CPU with base CPI=2
  - I-cache miss rate = 2%, D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Load & stores are 36% of instructions

1. Find new CPI
2. Compare performance of ideal and actual CPU performance

$$\text{CPI with stall} = \text{Base CPI} + \text{Memory stall cycle per Instr.}$$

$$\text{Memory stall per Instr.} = \text{Memory Access Per Instr.} \times \text{Miss Rate} \times \text{Miss Penalty}$$

Memory stall cycle per instruction

- I-cache: 1x 0.02 x 100 =2
- D-cache: $0.36 \times 0.04 \times 100 = 1.44$

New CPI = 2 + 2 + 1.44 = 5.44

New CPI/Old CPI = 5.44/2 =2.72

New CPU is 2.72X slower than Ideal CPU

Cache Performance Example- Unified Cache

- Given
  - Unified Cache miss rate = 2%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions

Find new CPI

$$\text{CPI with stall} = \text{Base CPI} + \text{Memory stall cycle per Instr.}$$
$$\text{Memory stall per Instr.} = \text{Memory Access Per Instr.} \times \text{Miss Rate} \times \text{Miss Penalty}$$

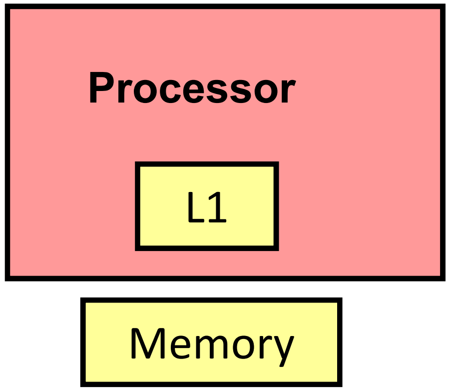
Memory Accesses Per Instruction= 1+ 0.36= 1.36

New CPI = 2 + 1.36*2%*100= 2+2.72 =4.72

Cache Performance Example-2

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Misses per instruction = 2%
  - Main memory access time = 100ns

Find New CPI with stalls



Miss penalty = 100ns/0.25ns = 400 cycles

CPI with stall = $1 + 0.02 \times 400 = 9$

$$\text{CPI with stall} = \text{Base CPI} + \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss Penalty}$$

Average Memory Access Time (AMAT)

- Hit time is also important for performance
- Use AMAT to estimate average time to access data for both hit and miss
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%, find AMAT

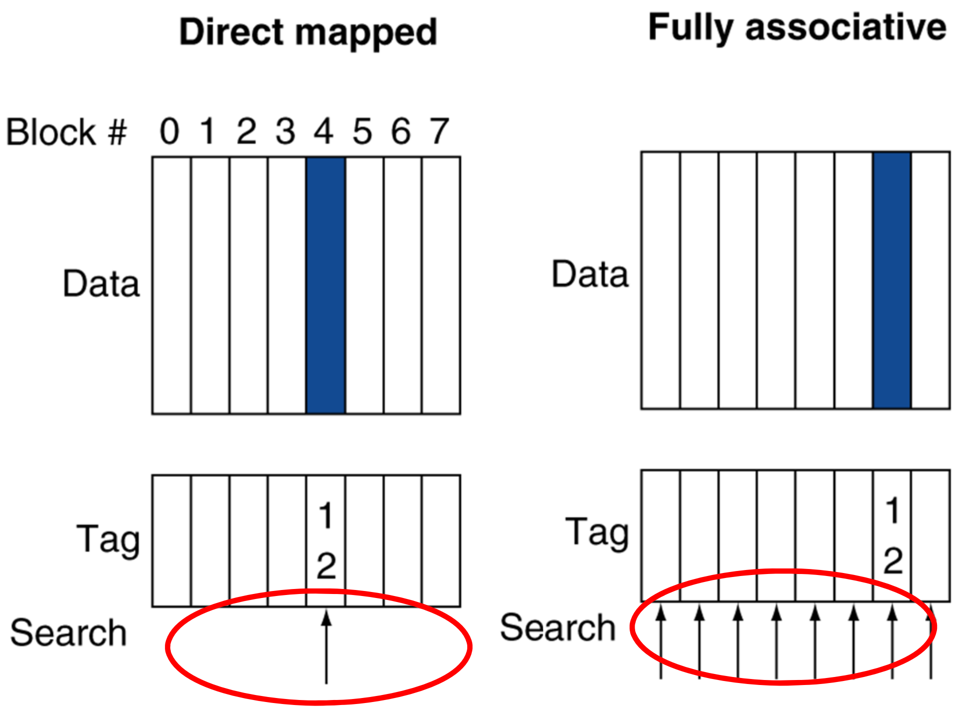
$$AMAT = 1 + 0.05 \times 20 = 2ns$$

AMAT is 2 cycles per instruction

Different Cache Architectures

- Direct mapped (have discussed)
- Fully associative: each memory block can locate anywhere in cache
  - Allow a given block to go in any cache entry
  - All cache entries are searched (in parallel) to locate block
  - Comparator per entry (expensive)

Location of a memory block with address 12 in a cache with 8 entries



Can be placed in any block, every block is searched

Different Cache Architectures

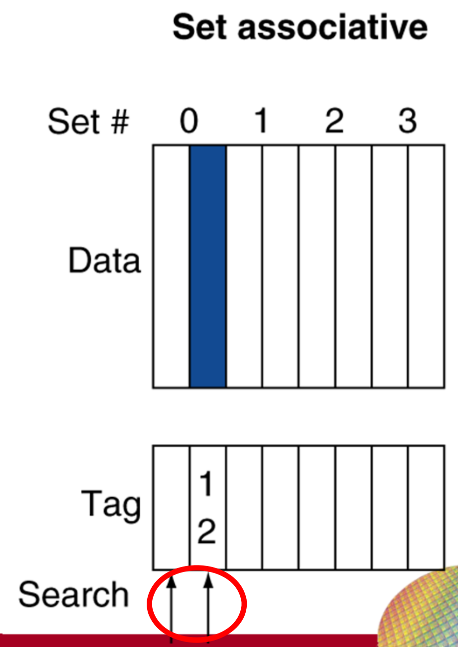
- N-way Set associative:
  - Each memory block can place in a set of cache locations (any location in the set , each set contain N entries)
  - cache set number = (Block number) modulo (number of Sets in cache)
  - all cache entries in the set are searched (in parallel) to locate block
    - N comparators (less expensive)

E.g. What is the location of a memory block with address 12 in a 2-way cache with 8 entries?

Total #entry=8, each set has 2 blocks => 4 Sets

Set address = 12 mod 4 = 0

block number 12 is put into set 0



Different Associativity

- For a cache with 8 entries, four different cache organization

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

One way set-associative (a set has one block, directed mapped)

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

2-way set-associative (a set has two blocks

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

4-way set-associative (a set has 4 blocks)

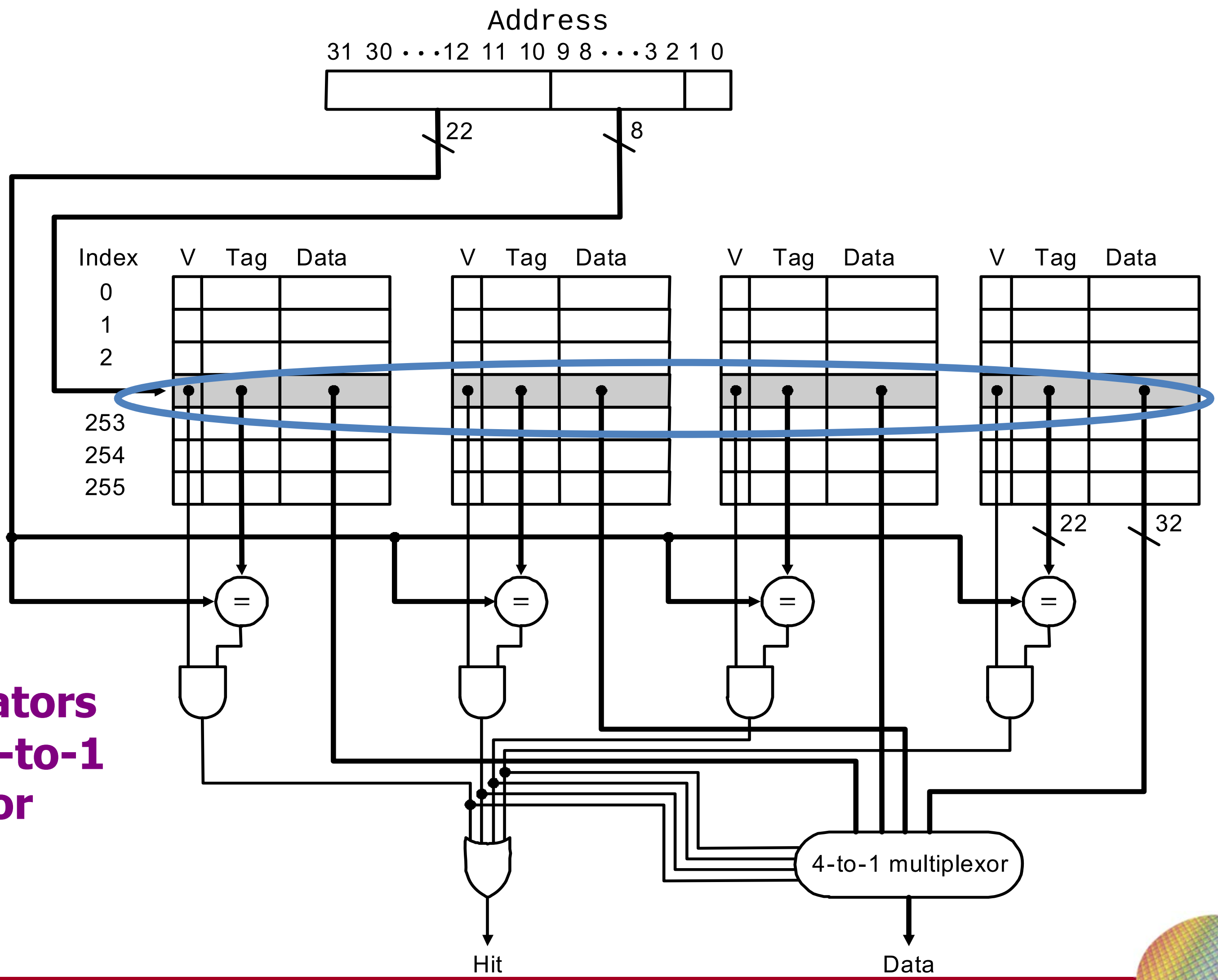
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

8-way set-associative (a set has 8 blocks), only one set, all blocks are search in a cache access

A 4-way set-associative Cache example

4-way set-associative cache size of cache is 1K blocks, each block is 1 word

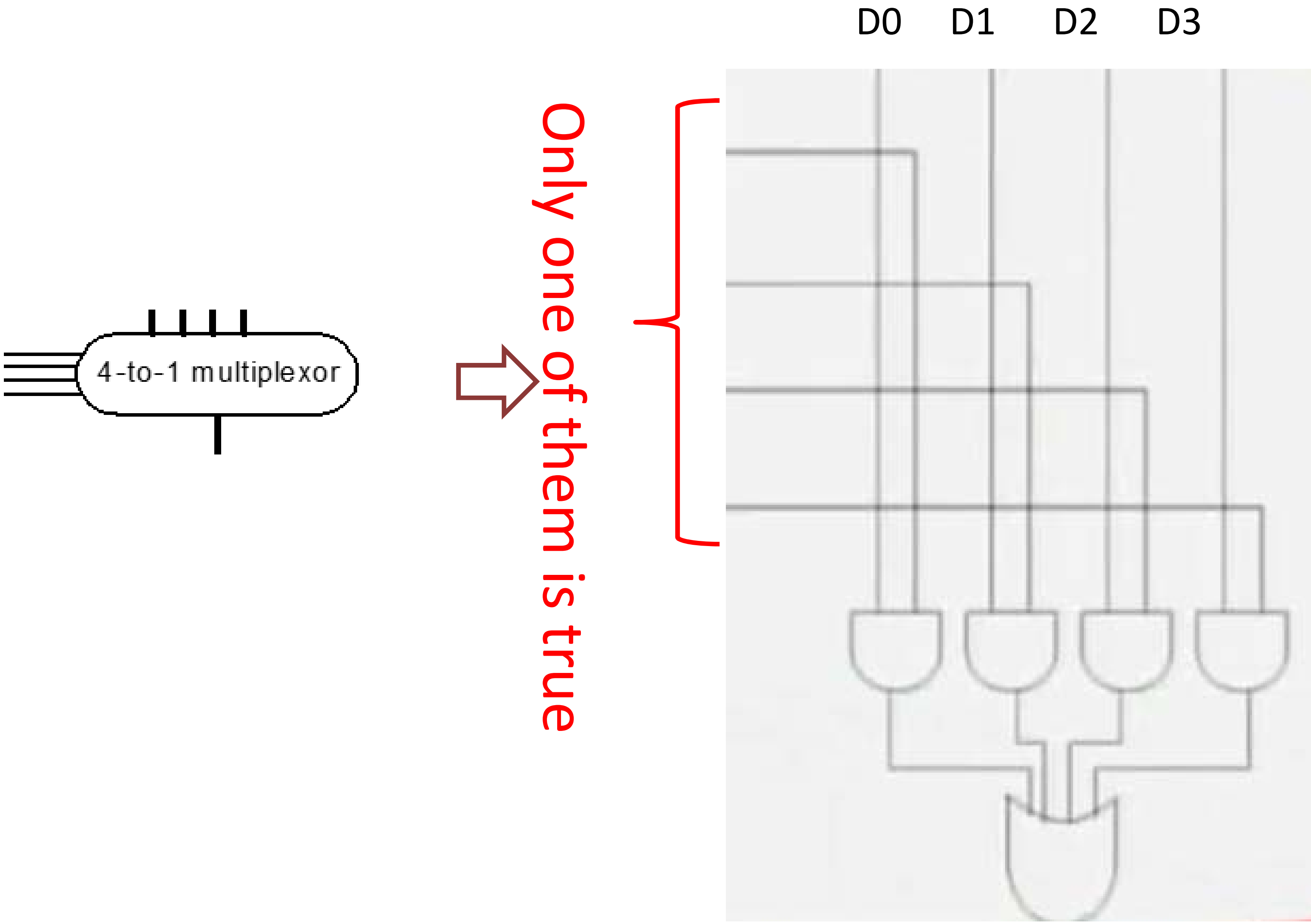
Block size = 4 byte
Block offset = 2 bits
Block = 1 K = 2^{10}
Set = 1024/4 =256
Set index width = 8



Set

4 comparators and one 4-to-1 multiplexor

4-to-1 multiplexer



Example Problem

Find the number of misses for a 4-block cache with (each block is 1-word) given the following sequence of word addresses of memory block accesses: 0, 8, 0, 6 and 8, for each of the following cache configurations

1. direct mapped
2. 2-way set associative (use LRU replacement policy)
3. fully associative

Associativity Example

- Direct mapped for four 1-word blocks
- Block addresses : 0, 8, 0, 6 ,8

Block address	Cache block
0	0(=0 mod 4)
6	2(=6 mod 4)
8	0(=8 mod 4)

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 (4 misses, 1 hit)

address	Set
0	0(=0 mod 2)
6	0(=6 mod 2)
8	0(=8 mod 2)

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]		

Use LRU (least recently used), so Mem[8] is replaced

- 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]	

(3 misses, 2 hit)

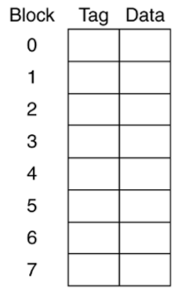
Why Set Associate Cache?

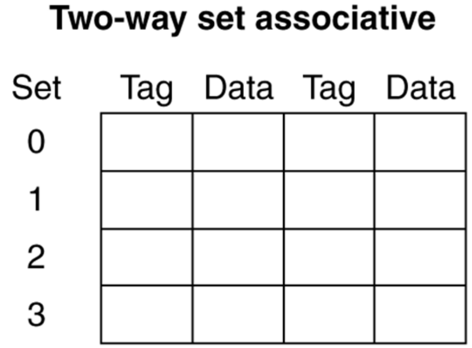
- Directed mapped may introduce too many conflict misses
  - Conflict miss: >=1 memory blocks contend a cache line
- Fully associative is too expensive
  - Minimize the number of conflict misses
- Set associative intends to come up with a balance

How Much Associativity

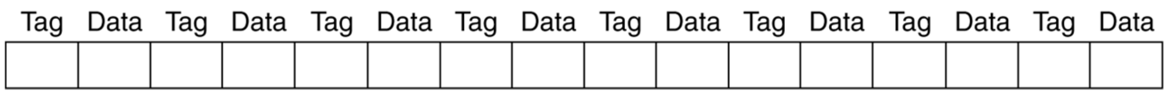
- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Replacement Policy





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



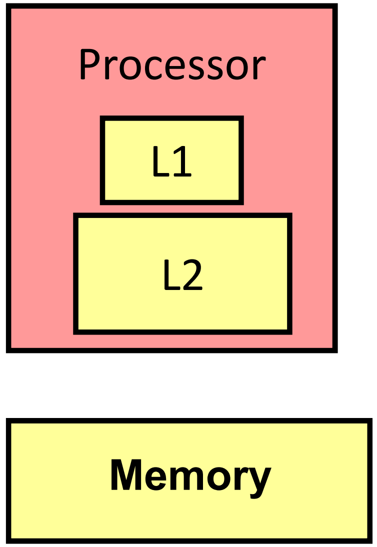
Block Replacement Policy: LRU

- LRU(Least Recently Used)
- Idea: replace block which has been accessed (read or write) least recently
- Pro: temporal locality
  - recent past use implies likely future use: in fact, this is a very effective policy
- Con:
  - with 2-way set associative, easy to keep track (one LRU bit);
  - with 4-way or greater, requires complicated hardware and much time to keep track of this

Multilevel Caches

- Use Multilevel Caches to reduce miss penalty
- Primary cache (Level 1) attached to CPU
  - Small, but fast
- Level 2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- L2 cache reduces the write on mem

  => reduce miss penalty
- Some high-end systems include L-3 cache



New CPI = Base CPI + L1 stalls per Instr. + L2 stalls per Instr.

Multilevel Cache Example

Assume CPU base CPI = 1, clock rate = 4GHz (0.25 ns)

Miss rate/instruction = 2%
Main memory access time = 100ns

With just L1 cache, find New CPI=?

Miss penalty = 100ns/0.25ns = 400 cycles

Effective CPI = 1 + 0.02 $\times$ 400 = 9

Now CPI with L2 Cache
L2 Access time = 5ns
Miss rate to main memory = 0.5%

L1 Miss Penalty = 5ns/0.25ns
= 20 cycles
L2 miss Miss Penalty = 400 cycles

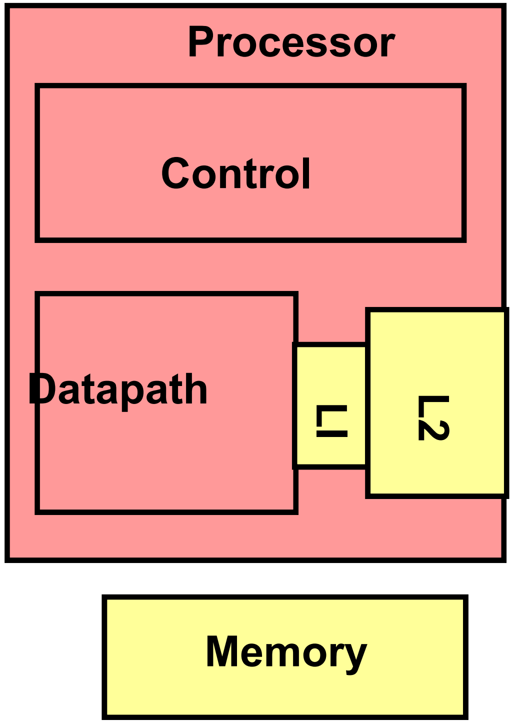
New CPI = Base CPI +
L1 stalls per Instr. + L2 stalls per Instr.

CPI = 1 + 0.02 $\times$ 20 + 0.005 $\times$ 400
= 3.4

Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L1 cache usually smaller than a single cache
  - L1 block size smaller than L-2 block size



Backup Slides