Computer Organization AND Design
The Hardware/Software Interface

Large and Fast: Exploiting Memory Hierarchy

Dr. Feng Li
fli@sdu.edu.cn
https://funglee.github.io

Outline

- 5.1 Introduction
- 5.2 The Basics of Caches
- 5.3 Measuring and Improving Cache Performance
- 5.4 Virtual Memory
- 5.5 A Common Framework for Memory Hierarchies
- 5.6 Virtual Machines
- 5.7 Using a Finite-State Machine to Control a Simple Cache
- 5.8 Parallelism and Memory Hierarchies: Cache Coherence
- 5.9 Advanced Material: implementing Cache Controllers
- 5.10 Real Stuff: the AMD Opteron X4 (Barcelona) and intel Nehalem Memory Hierarchies
- 5.11 Fallacies and Pitfalls

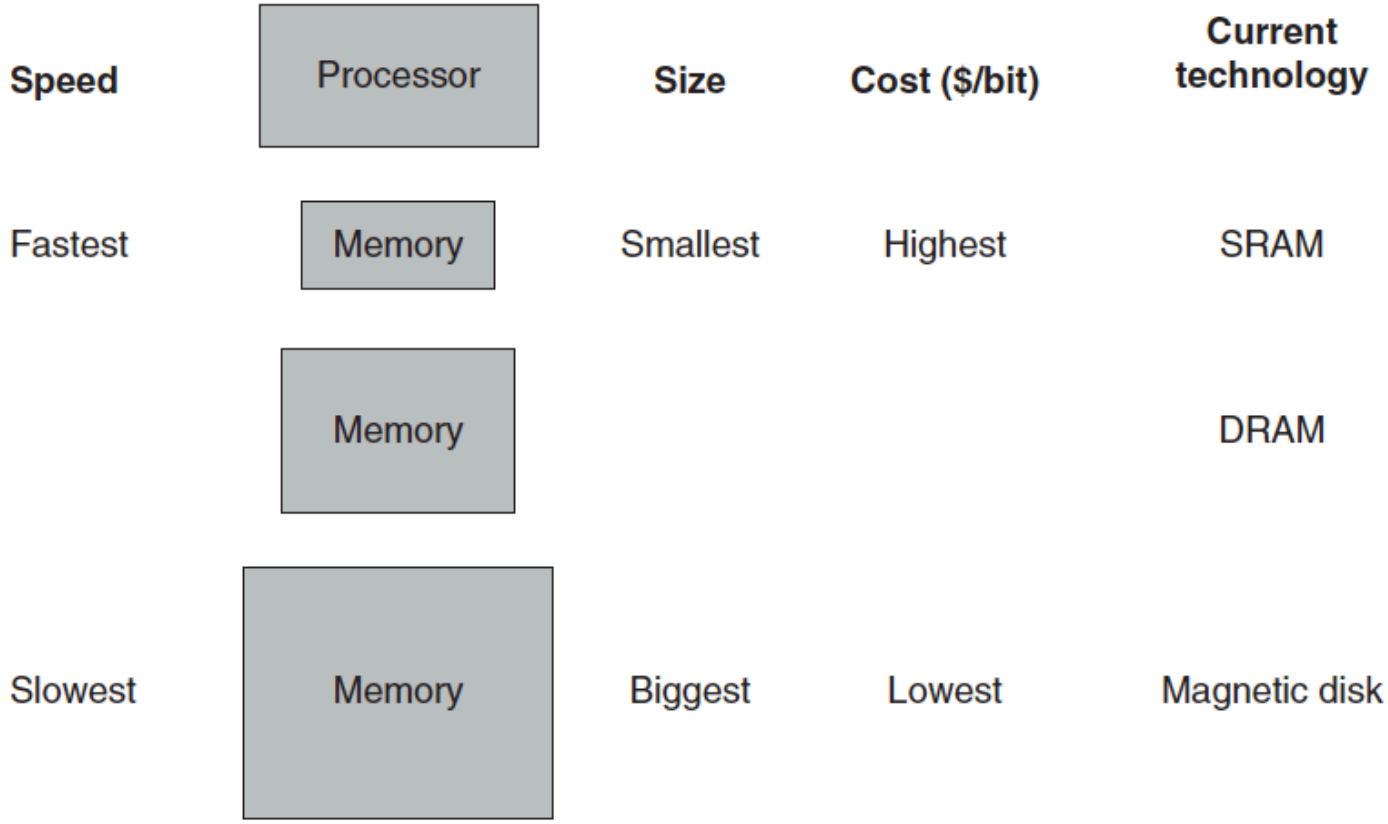
Introduction

- Locality---- two important concepts
  - **Temporal locality** (locality in time): If an item is referenced, it will tend to be referenced again soon.
  - **Spatial locality** (locality in space): If an item is referenced, items whose addresses are close by will tend to be referenced soon.

Memory hierarchy

- A structure that uses multiple level of memories; as the distance from the processor increases, the size of the memories and the access time both increase
- Different levels of memory have different speeds and sizes
- The faster memories are more expensive per bit than slower memories and thus are smaller

The basic structure of a memory hierarchy



By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the lowest level of the hierarchy, but can be accessed as if it were all built from the fastest memory. Flash memory has replaced disks in many embedded devices, and may lead to a new level in the storage hierarchy for desktop and server computers

- Block (or line): The minimum unit of information that can be either present or not present in cache

Memory Basics

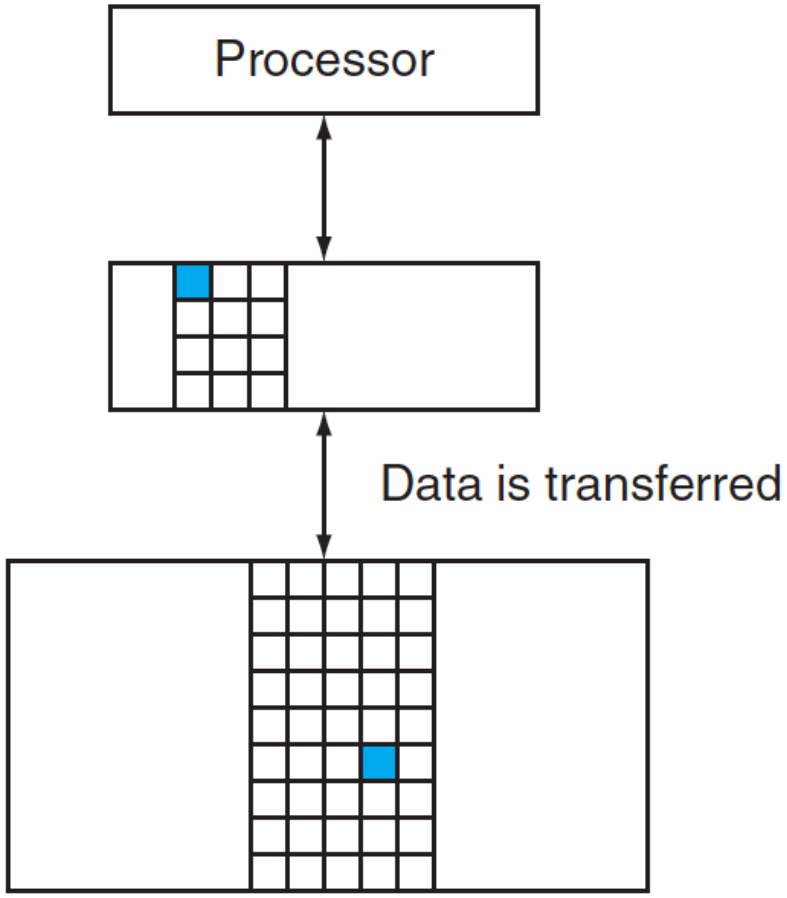
- RAM: Random Access Memory
  - Historically defined as memory array with individual bit access
  - Refers to memory with both read and write capabilities
- ROM: Read Only Memory
  - No capabilities for “online” memory write operations
  - Write typically requires high voltage or erasing by UV light
- Volatility of memory
  - Volatile memory loses data over time or when there is no power (e.g., RAM)
  - Non-volatile memory stores data even when power is removed (e.g., ROM)

- Static vs Dynamic Memory
  - Static: Hold data as long as power is applied (e.g., SRAM, or Static Random Access Memory)
  - Dynamic: Will lose data unless refreshed periodically (e.g., DRAM, Dynamic Random Access Memory)
- DRAM to SRAM Comparison
  - DRAM is smaller and less expensive per bit
  - SRAM is faster
  - DRAM requires more peripheral circuitry

- ROM: Read Only Memory
  - ➢ No capacities for “online” memory write operations
  - ➢ Data programmed
    - ✓ During fabrication: ROM
    - ✓ With high voltage: PROM
    - ✓ By control logical: PLA
- PROM: Programmable Read Only Memory
  - ➢ Programmable by special program tools/modes
  - ➢ Read only memory during normal use
  - ➢ Non-volatile
  - ➢ Erase operation
    - ✓ EPROM: Erasable PROM uses UV light to reset all bits
    - ✓ EEPROM: Electrically-erasable PROM, erase with control voltage

- DRAM
  - very high
  - must be periodically refreshed and thus is slower than SRAM
  - Volatile: no good for program (long term) storage
- SRAM
  - fastest type of memory
  - low density, more expensive
- EEPROM
  - slow/complex to write and thus is not good for fast cache
  - non-volatile; best choice for program memory
- ROM
  - hardware coded data; rarely used except for bootup code
- Register (flip flop)
  - functionally similar to SRAM but less dense (and thus expensive)
  - reserved for data manipulation applications

- Every pair of levels in the memory hierarchy can be thought of as having an upper and lower level
- Within each level, the unit of information that is present or not is called a block or a line
- We usually transfer an entire block when we copy something between levels



Performance of CPU, DRAM and disk

		1980	1990	2000	2010	2010:1980
CPU	Name	8080	386	Pentium II	Core i7	/
	Clock rate(MHz)	1	20	600	2,500	**2,500**
	Cycle time(ns)	1,000	50	1.6	0.4	**2,500**
	Cores	1	1	1	4	**4**
	Effective Cycle time(ns)	1,000	50	1.6	0.1	**10,000**
SRAM	$/MB	19,200	320	100	60	**320**
	access time(ns)	300	35	3	1.5	**200**
DRAM	$/MB	8,000	100	1	0.06	**130,000**
	access time(ns)	375	100	60	40	**9**
	typical size(MB)	0.064	4	64	8,000	**125,000**
Disk	$/MB	500	8	0.01	0.0003	**1,600,000**
	access time(ms)	87	28	8	3	**29**
	typical size(MB)	1	160	20,000	1,500,000	**1,500,000**

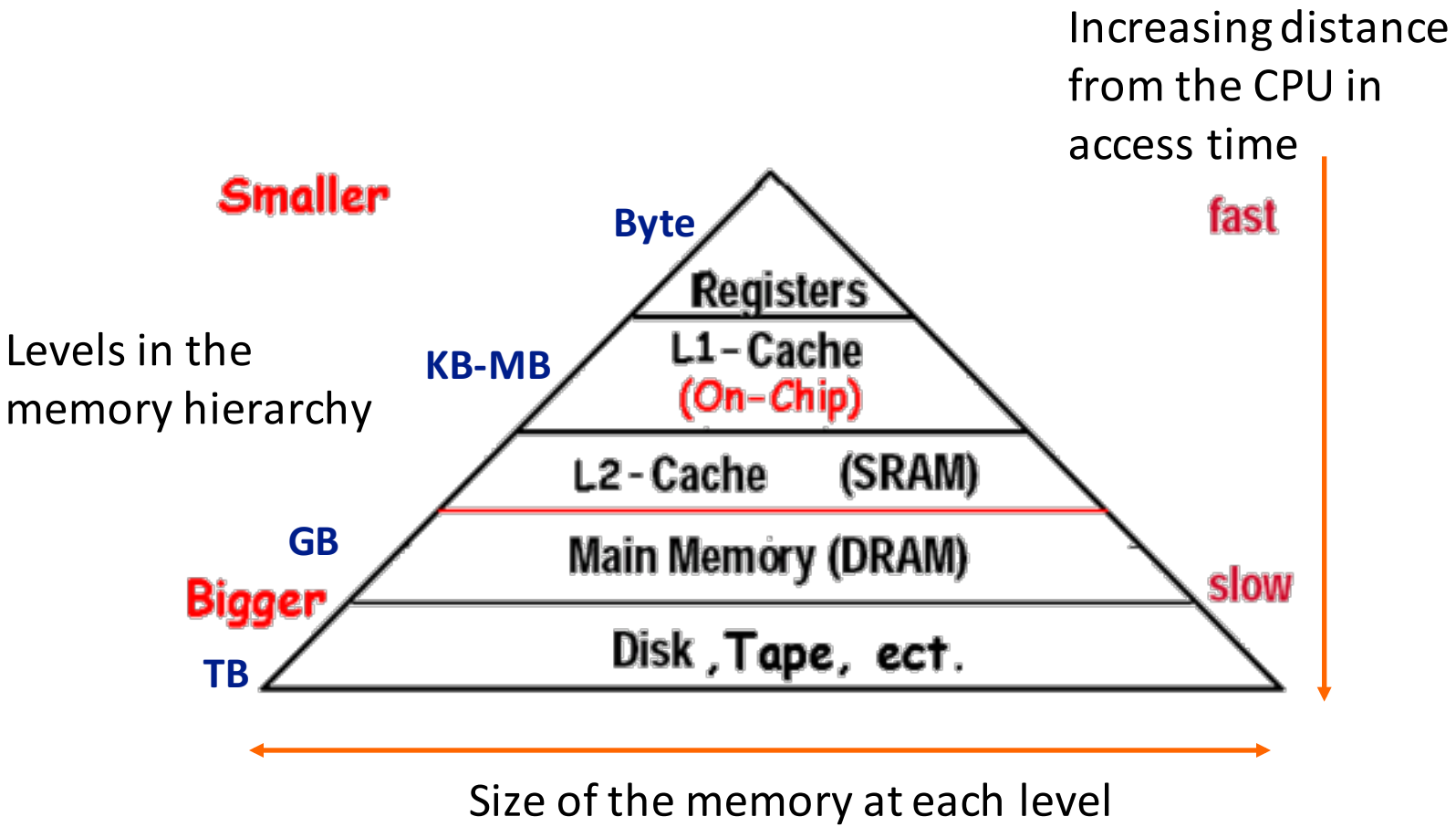
- Main memory is implemented from DRAM
- Caches use SRAM
- Magnetic disk
- Flash memory (EEPROM) is used instead of disks in many embedded devices

Memory technology	Typical access time	$ per GB in 2008
SRAM	0.5–2.5 ns	$2000–$5000
DRAM	50–70 ns	$20–$75
Magnetic disk	5,000,000–20,000,000 ns	$0.20–$2

Some important items

- **Hit**: The CPU accesses the upper level and finds the required data in some block.
- **Miss**: The CPU accesses the upper level and fails, but does not find the required data in any block.
- **Hit rate** (or hit ratio) is the fraction of memory accesses found in the upper level, which is usually used as a measurement of the performance of the memory hierarchy
- **Miss rate** (1 – hit rate) is the fraction of memory accesses not found in the upper level
- **Hit time**: The time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or a miss.
- **Miss penalty**: The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor

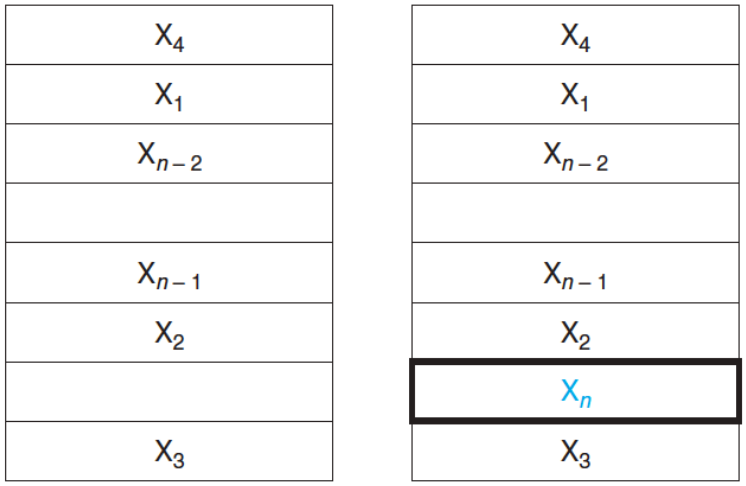
Exploiting Memory Hierarchy



The Basics of Caches

- Cache represents the level of the memory hierarchy between the processor and main memory; it is also used to refer to any storage managed to take advantage of locality of access

- A simple case: the processor requests each one word and the blocks also consist of a single word



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

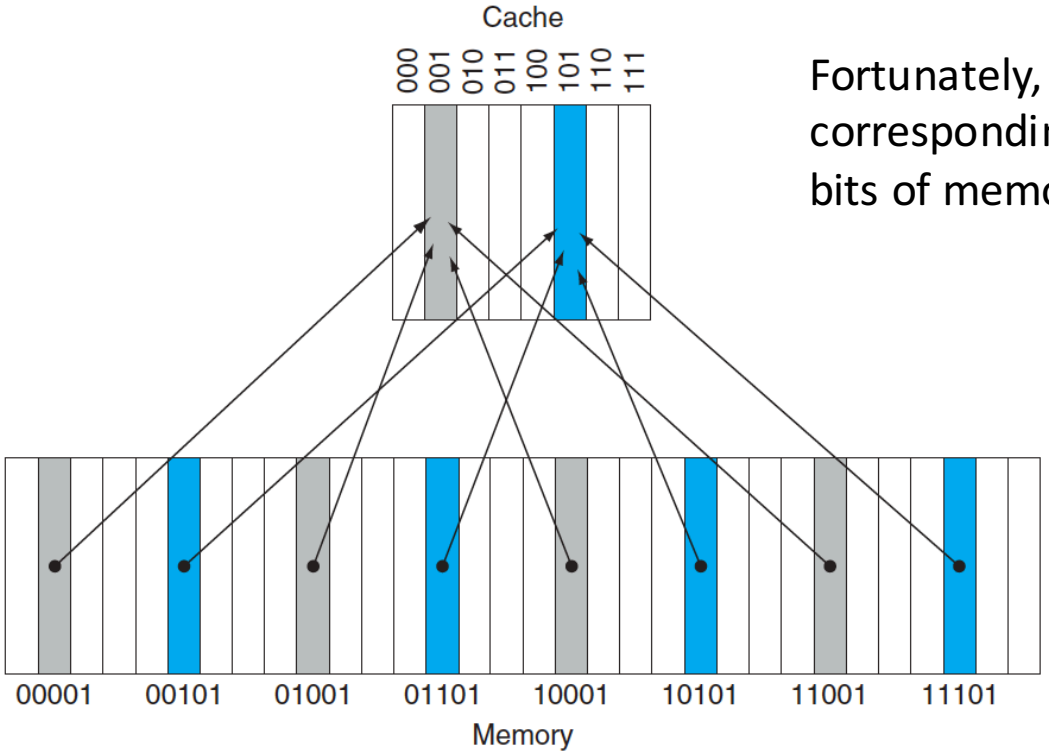
- The figure shows the cache just before and just after a reference to a word X_n that is not initially in the cache
- This reference causes a miss that forces the cache to fetch X_n from memory and insert it into the cache

- Two questions:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?

Direct Mapped Cache

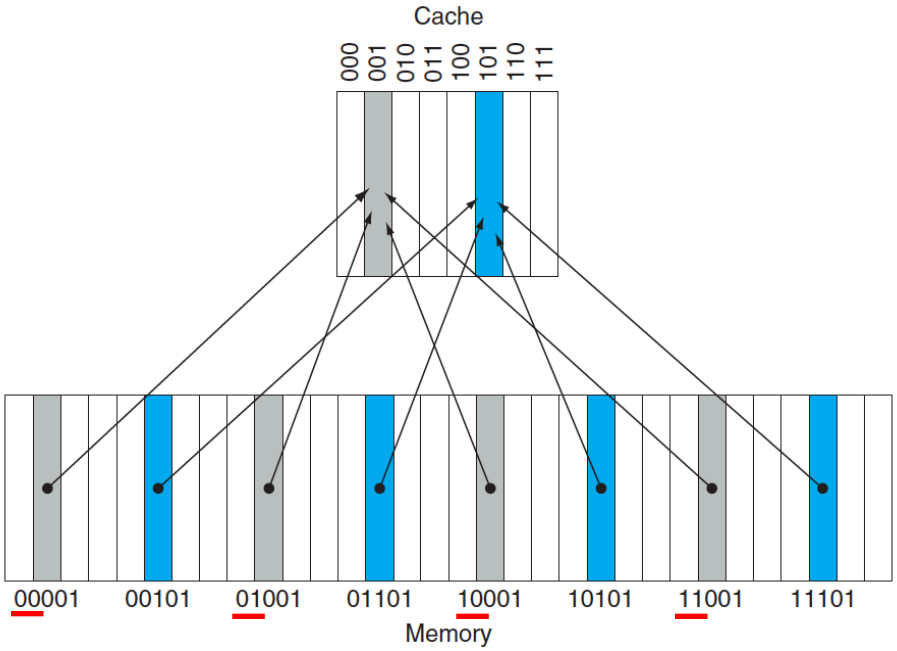
- Direct-mapped cache: A cache structure in which each memory location is mapped to exactly one location in the cache
- Direct-mapping algorithm

  (Block address) modulo (The number of blocks in the cache)

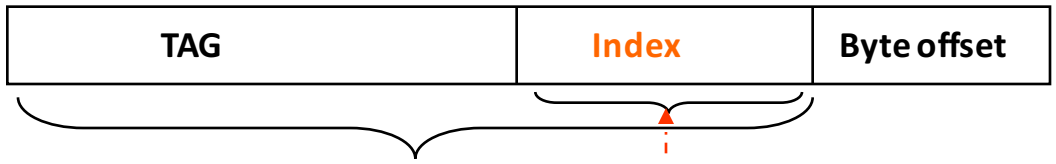


Fortunately, while the cache has 2^n blocks, the corresponding index is equal to the lowest n bits of memory block address

- Unfortunately, each cache location may contain the contents of a number of different memory locations, how do we know the data contained in the cache is exactly the one we request
- Adding a tag to the cache
  - The tag carries the address information according to which we can identify whether a word in the cache corresponds to the requested word



- How to recognize if a cache block has valid information?
- Adding a valid bit



valid bit

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

a. The initial state of the cache after power-on

Accessing a Cache

- A sequence of nine memory references to an empty eight-block cache

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.6f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Access sequence

- 10110,11010,10110,11010

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

b. After handling a miss of address(10110)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

c. After handling a miss of address(11010)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

d. After handling a hit of address(10110)

Index	V	Tag	Data
000	N		
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

e. After handling a hit of address(11010)

Access sequence-2

- 10110,11010,10110,11010,10000,00011,10000,10010

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	N		
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

f. After handling a miss of address(10000)

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

g. After handling a miss of address(00011)

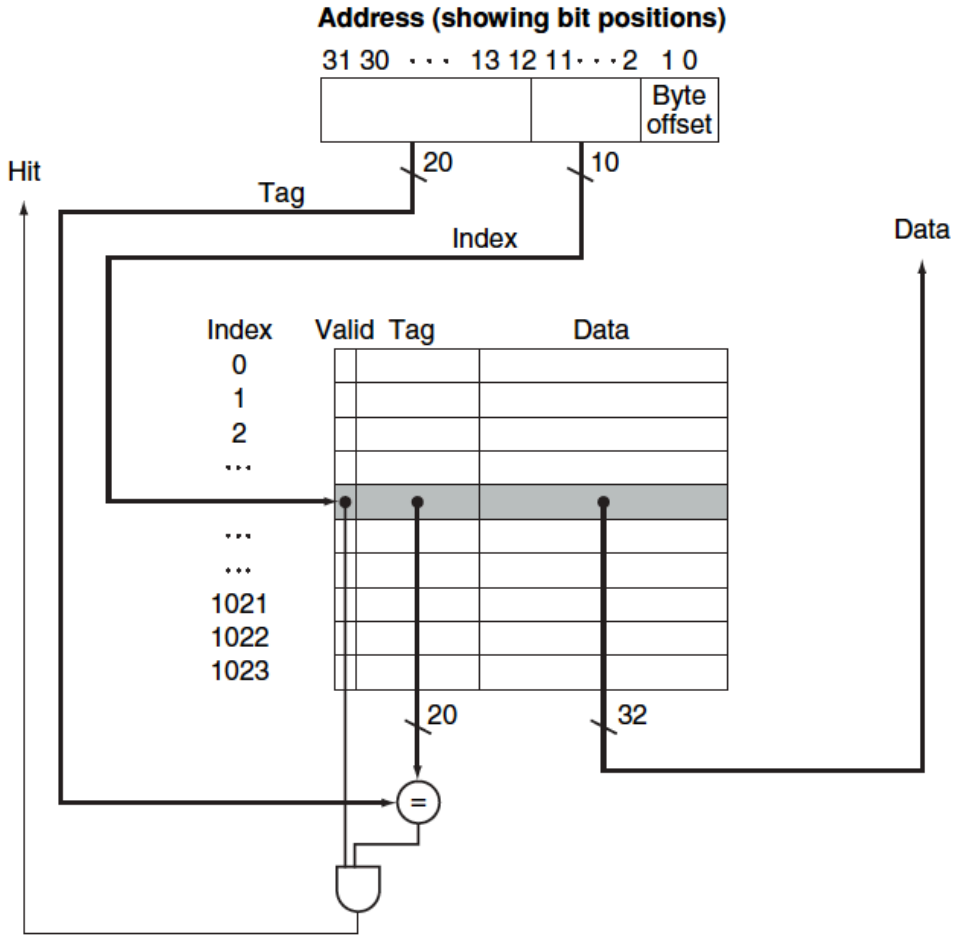
Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11)_2$	Memory(11010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

h. After handling a hit of address(10000)

Index	V	Tag	Data
000	Y	$(10)_2$	Memory(10000)
001	N		
010	Y	$(11) \rightarrow (10)_2$	Memory(10010)
011	Y	$(00)_2$	Memory(00011)
100	N		
101	N		
110	Y	$(10)_2$	Memory(10110)
111	N		

i. After handling a miss of address(10010)

- A referenced address = tag + index
  - Tag field is used to compare with the value of the tag field of the cache
  - A cache index is used to select the block



- What if a block consists of multiple wrods?
  - 32-bit addresses
  - A direct-mapped cache
  - The cache size is 2^n blocks, so n bits are used for the index
  - The block size is 2^m words (2^{m+2} bytes), so m bits are used for the word within the block, and 2 bits are used for the byte part of the address
  - The size of the tag field is $32 - (n + m + 2)$
  - The total number of bits in a direct-mapped cache is

$$2^n \times (block\ size + tag\ size + valid\ field\ size)$$
$$= 2^n \times (2^m \times 32 + (32 - n - m - 2) + 1)$$
$$= 2^n \times (2^m \times 32 + 31 - n - m)$$

Bits in Cache

Example

- How many total bits are required for a direct-mapped cache 16KiB of data and 4-word blocks, assuming a 32-bit address?

Answer

- One block=4 words = 16 bytes = 128 bits
- Cache blocks = 16KiB ÷ 16 bytes = 2^{10} blocks
- The size of index field = 10 bits
- 2 bits are used to search for words within a block
- 2 bits are used to search for bytes within a word
- Tag bits = 32-10–2-2 =18 bits
- Total Cache size = $2^{10} \times (128+18+1) = 2^{10} \times 147$ bits
- It is about 1.15 times as many as needed just for the data

Mapping an Address to Multiword Cache Block

Example

- Consider a cache with 64 blocks and a block size of 16 bytes.
- What block number does byte address 1200 map to?

Answer

- The block is indicated by

  (Block address) modulo (Number of cache blocks)

- The address of the block is

  (Byte address)/(Bytes per block)

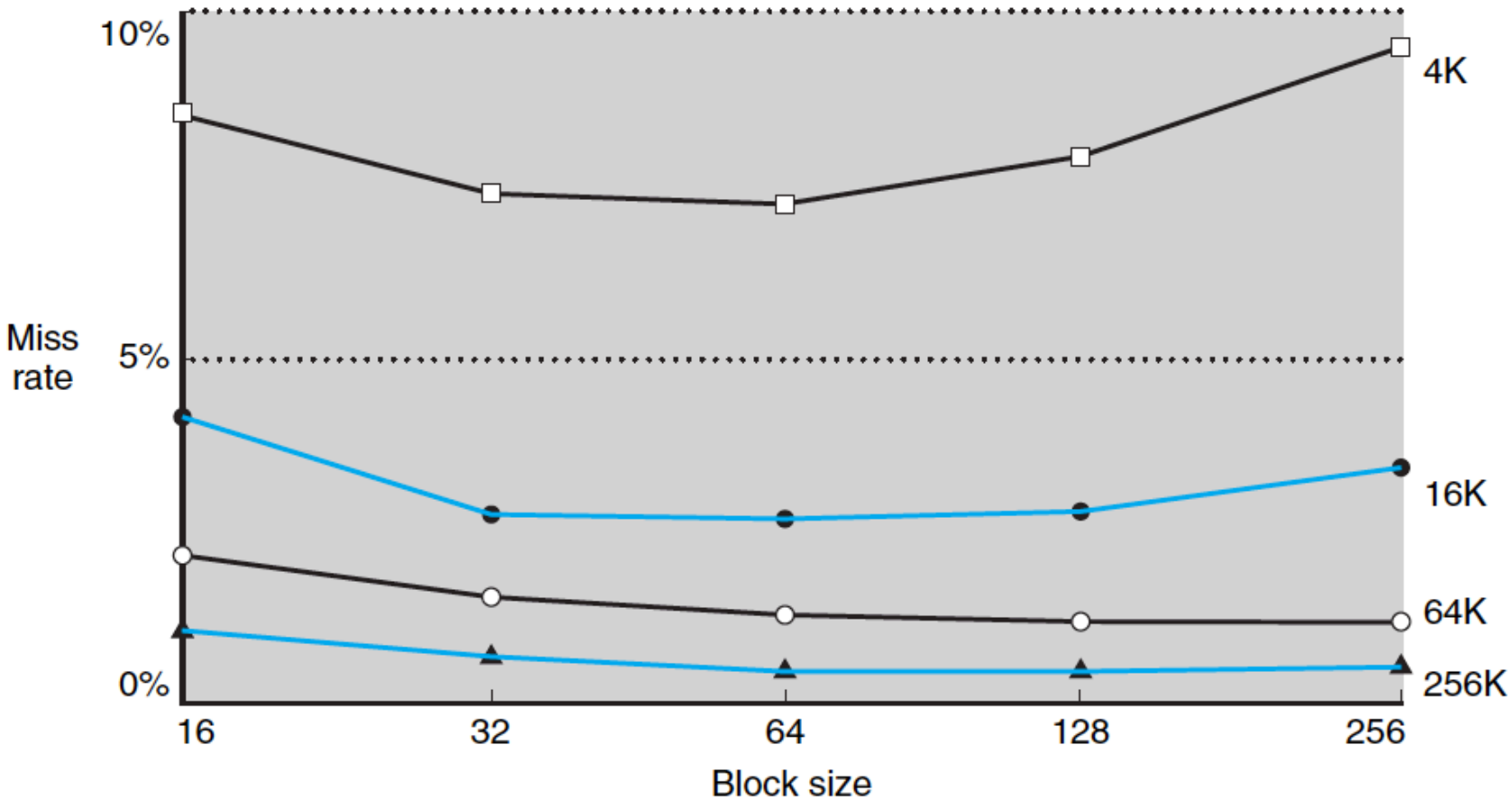
- Note: The block address is the block containing all addresses between

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block} \quad \text{and} \quad \left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block} + (\text{Bytes per block} - 1)$$

- Therefore, the block number is

$$\left\lfloor \frac{Byte\ address}{Bytes\ per\ block} \right\rfloor modulo\ 64 = \left\lfloor \frac{1200}{16} \right\rfloor modulo\ 64 = 75\ modulo\ 64 = 11$$

Miss Rate VS. Block Size



- Increase block size (reasonably) could decrease miss rate
- The miss rate may go up eventually if the block size becomes a significant fraction of the cache size, because the number of block that can be contained in the cache will become small, and there will be a great amount of competition for these blocks
- The excessively large block size may lead to heavier miss penalty, since we have to transfer more data from main memory to cache.

Handling Cache Miss

- Cache miss: A request for data from the cache that cannot be filled because the data is not present in the cache
- Cache miss is handled in collaboration with the processor control unit and with a separate controller that initiates the memory access and refills the cache.
- Misses—two kinds of misses
  - Instruction cache miss
  - Data cache miss

Main steps taken on an instruction cache miss

- Stall the CPU, fetch block from memory, deliver to cache, restart CPU read
  1. Send the original PC value (current PC-4) to the memory.
  2. Instruct main memory to perform a read and wait for the memory to complete its access.
  3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
  4. Restart the instruction execution at the first step, which will refetch the instruction again, this time finding it in the cache.

Handling Writes

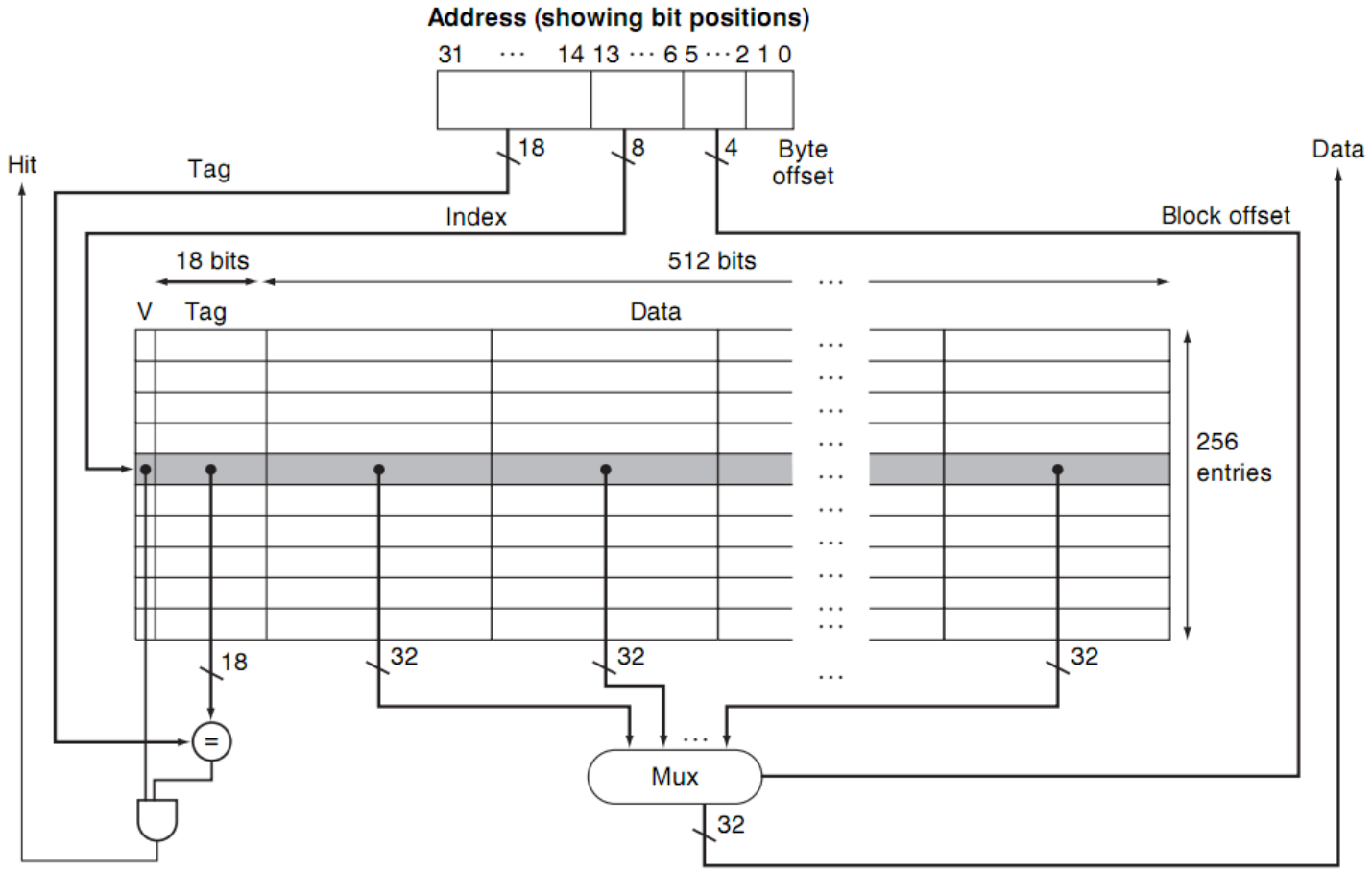
- **Inconsistence**: If we are performing a store instruction, we write the data into only data cache (without changing main memory); thereafter, memory would have a different value from that in the cache.
- **Solution I**: Write-through
  - A scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data is always consistent between the two
  - When write miss occurs, the block containing the targeted word are first fetched from the main memory
  - **Shortcoming**: Writing to main memory usually takes a long time

- **Advanced write-through**
  - The data are written into both cache and write buffer, such that the instructions can continue to be executed
  - The write buffer can hold the data temporally until the data is written into the main memory
  - After that, the corresponding entry of the write buffer can be freed.
  - If the write buffer is full, the instruction should be stalled until the write buffer is available

- **Solution II: write-back**
  - When a write occurs, the new value is written only to the block of the cache
  - The modified block is written back to the low-level of the hierarchy when it is replaced

An Example Cache: The Intrinsity FastMATH Processor

- 12-stage pipeline
- Separated instruction and data caches
- Each cache is 16 KiB (or 4096 words), with 16-word blocks



Read Request

1. Send the address to the appropriate cache. The address comes either from the PC (for an instruction) or from the ALU (for data).
2. If the cache signals hit, the requested word is available on the data lines. Since there are 16 words in the desired block, we need to select the right one. A block index field is used to control the multiplexor (shown at the bottom of the figure), which selects the requested word from the 16 words in the indexed block.
3. If the cache signals miss, we send the address to the main memory. When the memory returns with the data, we write it into the cache and then read it to fulfill the request.

Writes

- Both write-through and write-back, leaving it up to the operating system to decide which strategy to use for an application.
- It has a one-entry write buffer.

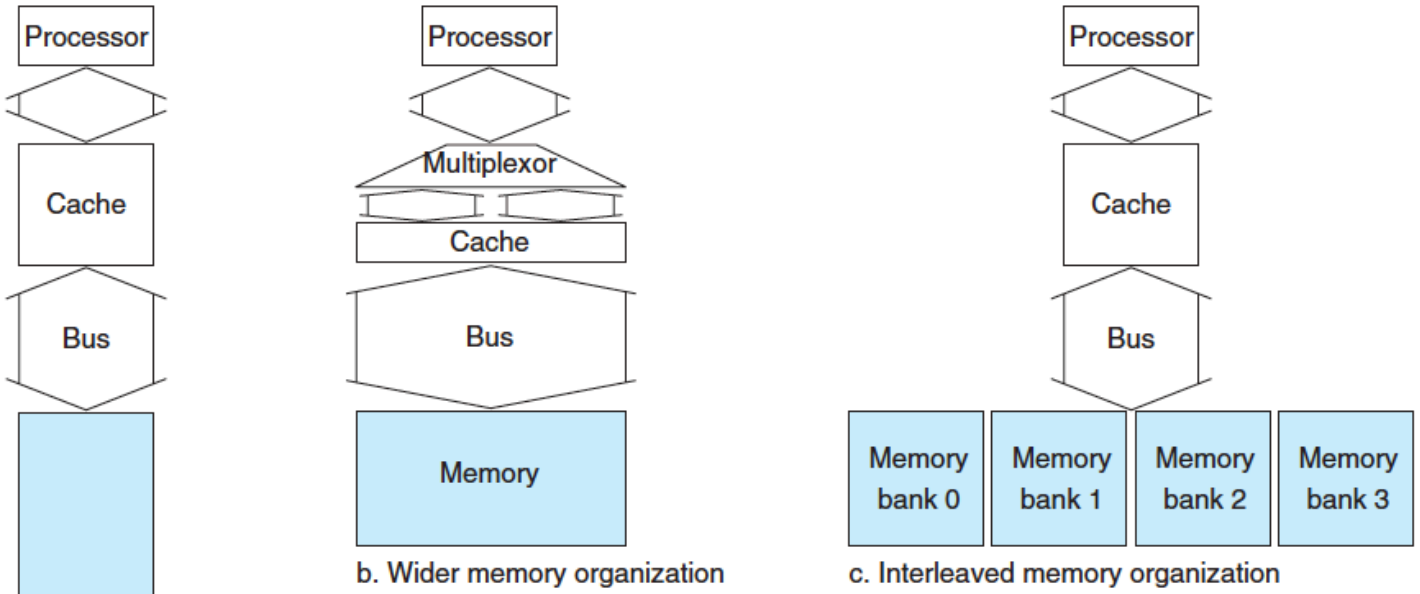
- Approximate instruction and data miss rates for the Intrinsity FastMATH processor for SPEC2000 benchmarks

Instruction miss rate	Data miss rate	Effective combined miss rate
0.4%	11.4%	3.2%

FIGURE 5.10 Approximate instruction and data miss rates for the Intrinsity FastMATH processor for SPEC CPU2000 benchmarks. The combined miss rate is the effective miss rate seen for the combination of the 16 KB instruction cache and 16 KB data cache. It is obtained by weighting the instruction and data individual miss rates by the frequency of instruction and data references.

- Combined Cache vs Split Caches
  - Total cache size: 32 KiB
  - Split cache effective miss rate: 3.24%
  - Combined cache miss rate: 3.18%
  - Nevertheless, almost all processors today use split instruction and data caches to increase cache bandwidth to match what modern pipelines expect. The advantage of increasing cache bandwidth can easily overcomes the disadvantage of a slightly increased miss rate

Designing the Memory System to Support Cache

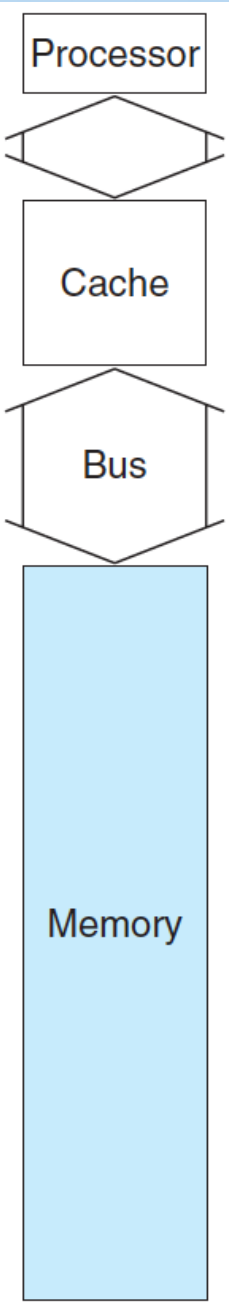


a. One-word-wide memory organization

b. Wider memory organization

c. Interleaved memory organization

a. Memory is one word wide, and all accesses are made sequentially
b. Increasing the bandwidth to memory by widening the memory and the buses between the processor and memory, which allows parallel access to multiple words of the block
c. Increasing the bandwidth by widening the memory but not the interconnection bus



Assume

1 clock cycles to send the address

15 memory bus clock cycles for each DRAM access initiated

1 bus clock cycles to send a word of data

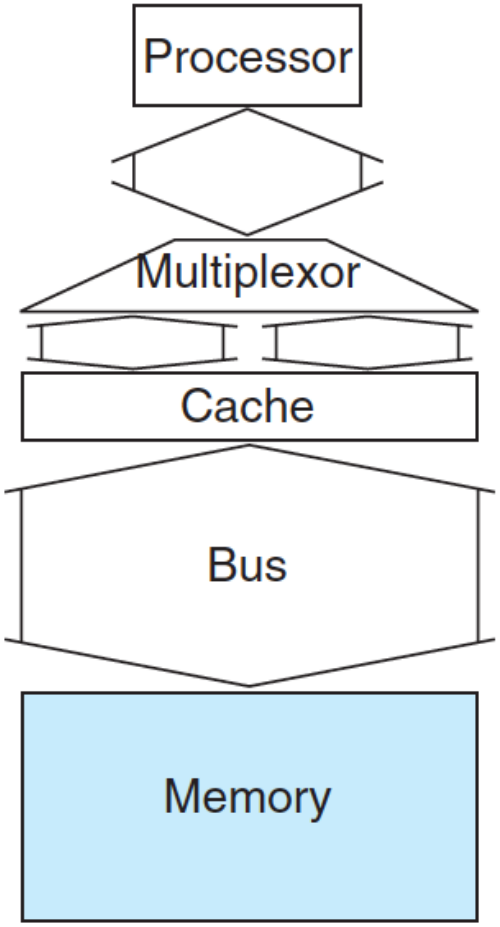
Each block consists of 4 words

Every word is 4 bytes

The miss penalty (The time to transfer one block is):

1+4×(1+15)＝65 CLKs

Bandwidth : $4\times4 \div 65 \approx 1/4$



- With a main memory width of 2 words (64bits)

  The miss penalty: $1+2\times(15+1)=33$ CLKs

  Bandwidth: $4\times4/33 \approx 0.48$

Only two times that needed to transfer one word.

- With a main memory width of 4 words(128bits)

  The miss penalty: $1+1\times(15+1)=17$ CLKs

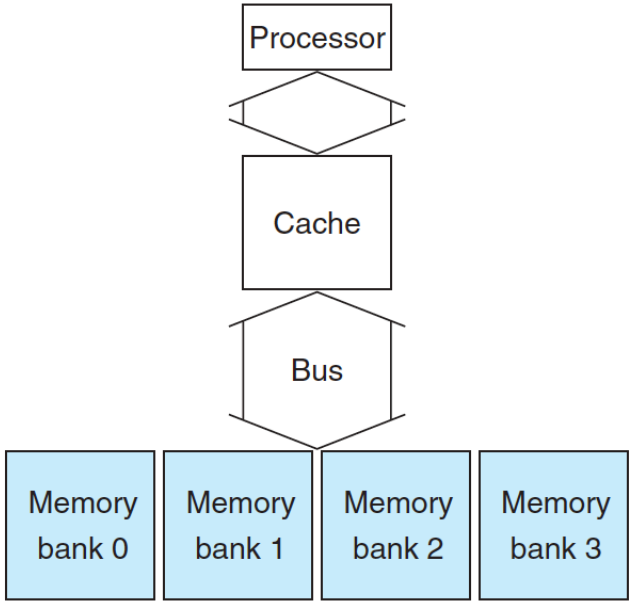
  Bandwidth: $4\times4 \div 17 \approx 0.98$

Equal to time to transfer one word.

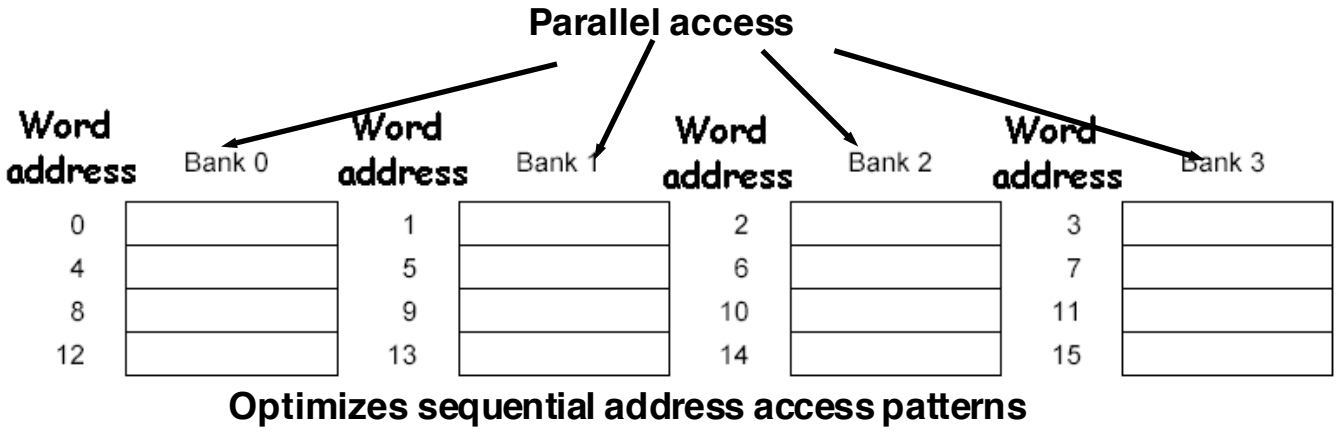
- With 4 banks Interleaved Memory

  The miss penalty: 1+15 +(4 × 1)＝20

  Bandwidth: $4 \times 4 \div 20 = 0.8$



Four-way interleaved memory



Measuring and Improving Cache Performance

- Improving cache performance
  - Reducing the miss rate by decreasing the probability that two different memory blocks contends for the same cache location
  - Reducing the miss penalty by introducing an additional level to the hierarchy (so-called *multilevel caching*)

- CPU time involves execution clock cycles and memory-stall clock cycles

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

- We assume the memory-stall clock cycles mainly stem from cache misses

$$\text{Memory-stall clock cycles} = \text{Read-stall cycles} + \text{Write-stall cycles}$$

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

$$\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} \right) + \text{Write buffer stalls}$$

- In write-through cache, the read and write misses have the same penalties

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

where we assume that there is no write buffer stall

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Calculating Cache Performance

- Assume:

Instruction cache miss rate	2%
Data cache miss rate	4%
CPI without any memory stalls	2
Miss penalty	100 cycles for all misses
The frequency of all loads and stores	36%

- Question: How faster a processor would run with a perfect cache?
- Answer: Instruction count is denoted by I

Instruction miss cycles = $I \times 2\% \times 100 = 2.00I$

Data miss cycles = $I \times 36\% \times 4\% \times 100 = 1.44I$

Total memory-stall cycles = $2.00\ I + 1.44\ I = 3.44I$

CPI with stall = CPI with perfect cache + total memory-stalls

= 2 + 3.44 = 5.44

- The ratio of the CPU execution times is

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{\text{I} \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{\text{I} \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}} = \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2}$$

- The performance with the perfect cache is better by 5.44/2=2.72

- What if the processor is made faster, but the memory system is not?
  - Reducing the CPI from 2 to 1 without changing the clock rate

$$\frac{CPU\ time\ with\ stalls}{CPU\ time\ with\ perfect\ cache} = \frac{1 + 3.44}{1} = 4.44$$

  - The amount of execution time spent on memory stalls would have risen from $\frac{3.44}{5.44} = 63\%$ to $\frac{3.44}{4.44} = 77\%$

- The above example does not take into account hit time
- To capture the fact that the time to access data for both hit and miss affects performance, we use average memory access time (AMAT) as a metric to examine cache design

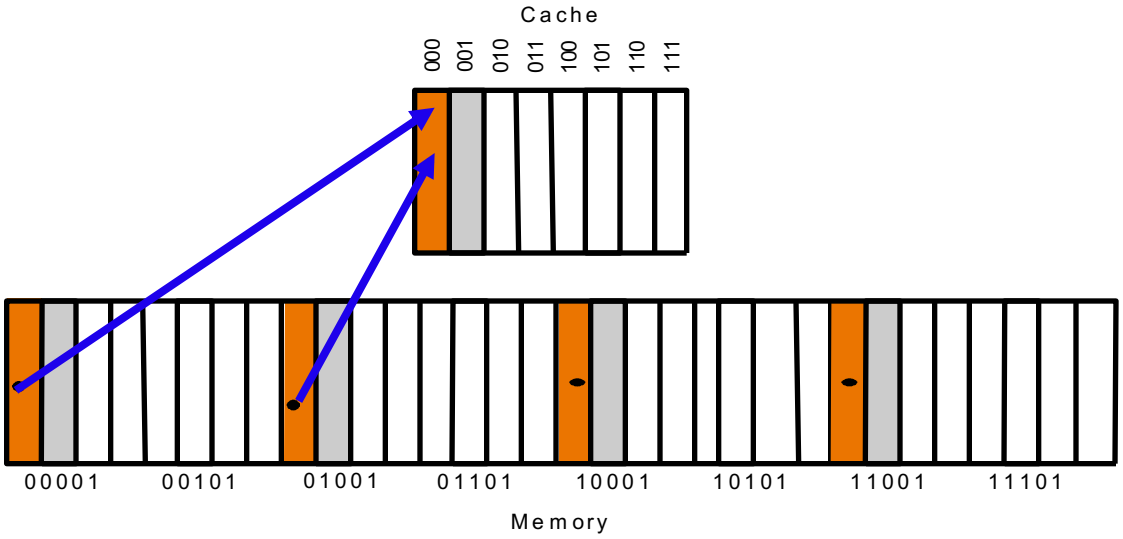
$$AMAT = Time\ for\ a\ hit + Miss\ rate \times Miss\ penalty$$

- Find the AMAT for a processor with a 1 ns clock cycle time, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls.

$$\begin{aligned} AMAT &= Time\ for\ a\ hit + Miss\ rate \times Miss\ panalty \\ &= 1 + 0.05 \times 20 \\ &= 20\ ns \end{aligned}$$

The disadvantage of a direct-mapped cache

- If the CPU requires the following memory units sequentially: word 0,word 8 and word 0. Word 0 and word 8 both are mapped to cache block 0, so the third access will be a miss.
- But obviously, if one memory block can be placed in any cache block , the miss can be avoided. So, there is possibility that the miss rate can be improved.

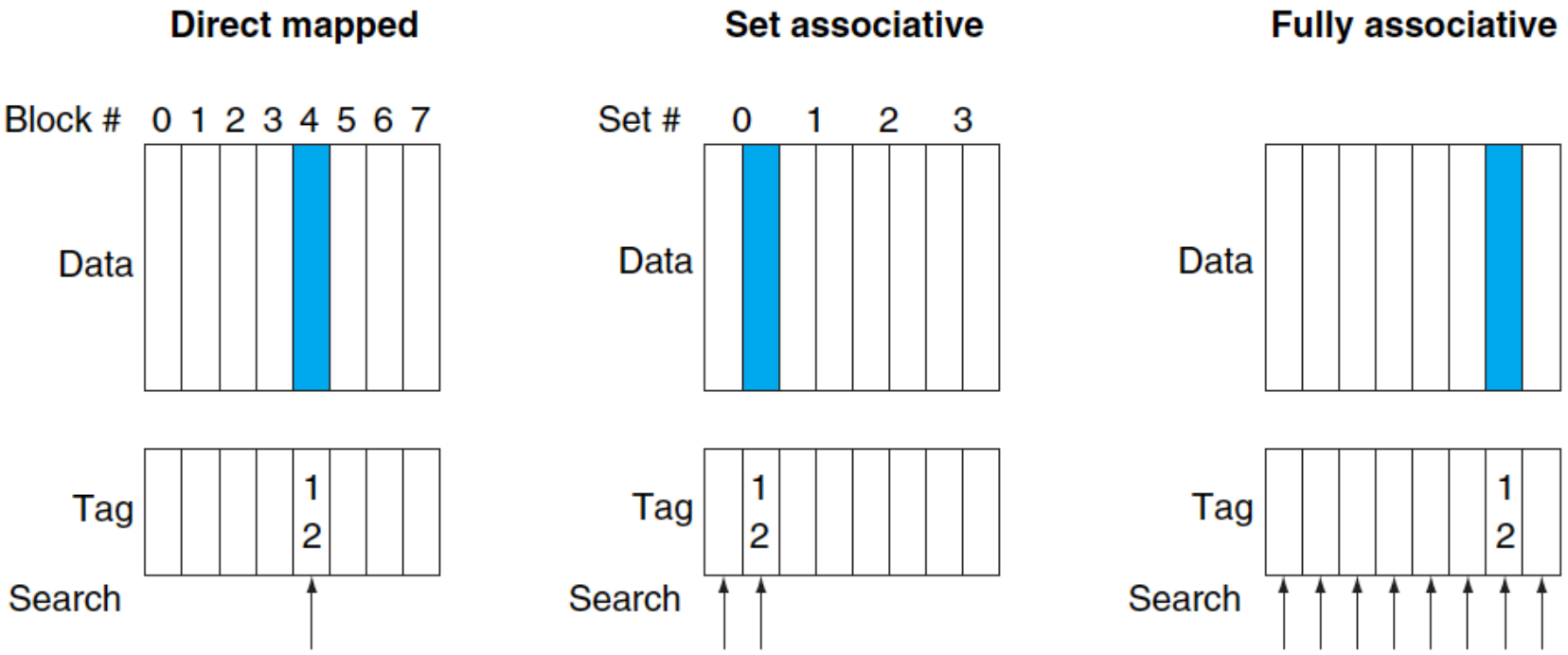


The basics of a set-associative cache

- Fully associative cache: A cache structure in which a block can be placed in any location in the cache
- Set-associative cache: A cache that has a fixed number of locations (at least two) where each block can be placed
  - A set-associative cache with n locations for a block is called an n-way set-associative cache
  - A block is directly mapped into a set, and then all the blocks in the set are searched for a match
  - The mapping algorithm: The set containing a memory block is

    (Block number) modulo (Number of sets in the cache)

- Memory block whose address is 12 in a cache with 8 blocks for different mapped



- An eight-block cache configured as variety-way

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

Misses and Associativity in Caches

Assume:

There are three small caches, each consisting of four one-word blocks: one cache is direct-mapped, the second is two-way set associative, and the third is fully associative.

Question:

Given the following sequence of block addresses: 0,8,0,6,8, find the number of misses for each cache organization.

- Direct-mapped cache: Five misses for the five accesses

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

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

- The set-associative cache has two sets with two elements per set
- The least recently used entry is replaced when a miss occurs
- Four misses for five accesses

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

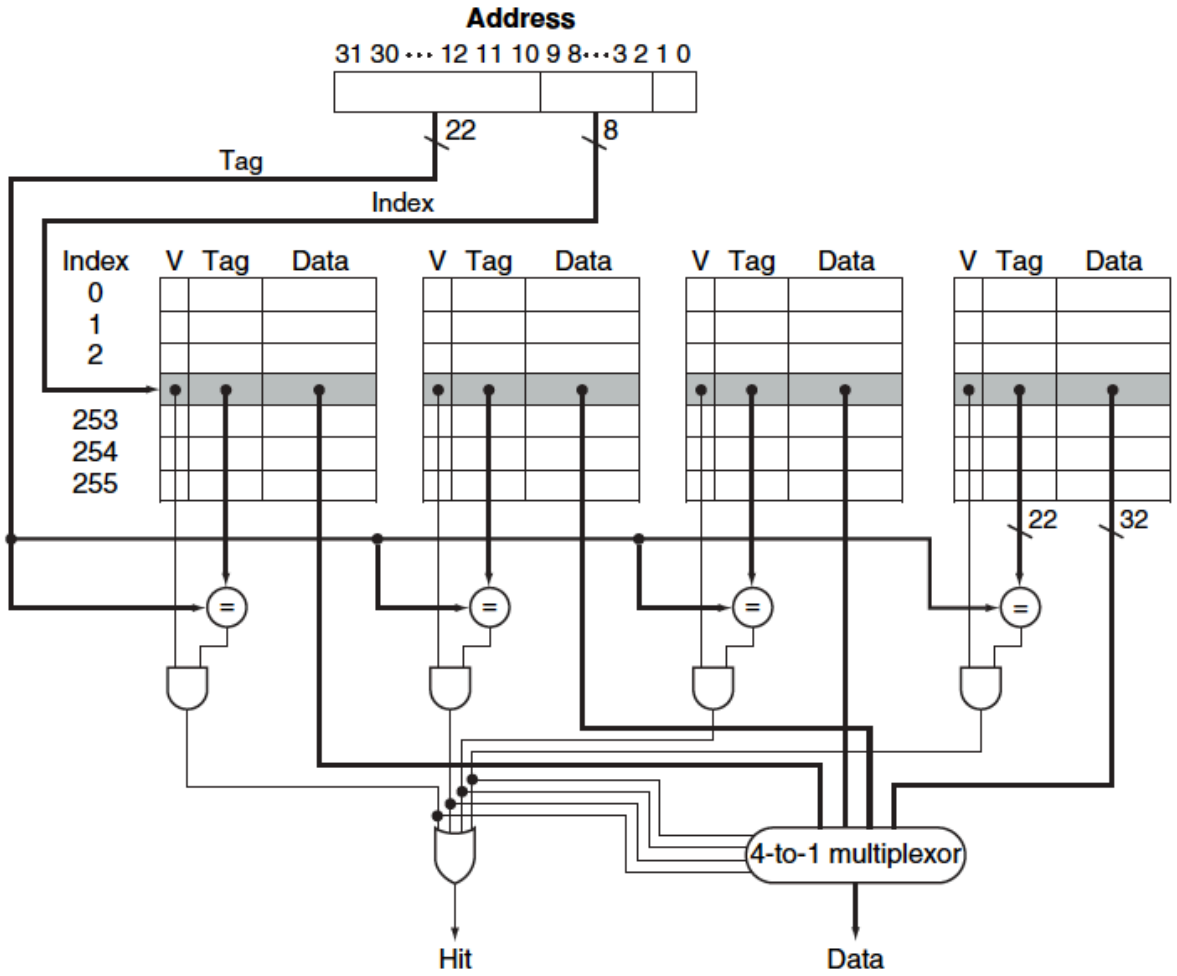
- Fully associative cache where a block can be placed in any available location
- Only three misses for five access

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

Locating a block in a set-associative cache

- The index is used to select the set
- The tag is used to choose the block by comparison with the blocks in the selected set
- The block offset is the address of the desired data within the block

- The implementation of a four-way set-associative cache requires four comparators and a 4-to-1 multiplexor.



Size of tags versus set associativity

Assume

Cache size is 4K blocks

Block size is 4 words

Physical address is 32bits

Question

Find the total number of sets and total number of tag bits for variety associatively

Answer

Offset size (Byte) = 16= 2^4 — 4 bits for byte address

Number of memory block = $2^{32} \div 16 = 2^{28}$ — 28 bits for memory block address

Number of cache block = 2^{12} — 12 bits for cache block address

For direct-mapped

Bits of index = 12 bits

bits of Tag = (32-12-4) ×4K=16×4K=64 Kbits

For two-way associative

Number of cache set = $2^{12} \div 2 = 2^{11}$

Bits of index = 12-1=11 bits

Bits of Tag = (28-11) ×2×2K=17×2×2K=68 Kbits

For four-way associative

Number of cache set = $2^{12} \div 4 = 2^{10}$

Bits of index = 12-2=10 bits

Bits of Tag = (28-10) ×4×1K=18×4×1K=72 Kbits

For fully associative

Number of cache set = $2^{12} \div 2^{12} = 2^0$

Bits of index = 12-12=0 bits

Bits of Tag = (28-0) ×4K×1=112 Kbits

	Direct	2-way	4-way	Fully
Index(bit)	12	11	10	0
Tag(bit)	16	17	18	28

Choosing which block to replace

- In a direct-mapped cache, the requested block can go in exactly one position, and the block occupying that position must be replaced.
- In a fully associative cache, all blocks are candidates for replacement.
- In a set-associative cache, we must choose among the blocks in the selected set.
- The most commonly used scheme is least recently used (LRU), which we used in the previous example. In an LRU scheme, the block replaced is the one that has been unused for the longest time.
- For a two-way set associative cache, the LRU can be implemented easily. We could keep a single bit in each set. We set the bit whenever a specific block in the set is referenced, and reset the bit whenever another block is referenced.
- As associativity increases, implementing LRU gets harder.

Reducing the Miss Penalty Using Multilevel Caches

- Most microprocessors support an additional level of caching.
- This second-level cache is usually on the same chip and is accessed whenever a miss occurs in the primary cache. If the second-level cache contains the desired data, the miss penalty for the first-level cache will be essentially the access time of the second-level cache, which will be much less than the access time of main memory.
- If neither the primary nor the secondary cache contains the data, a main memory access is required, and a larger miss penalty is incurred.

Performance of multilevel caches

- Example:
  - CPI of 1.0 on a 4GHz machine with a 2% miss rate, 100ns DRAM access
  - Adding 2nd level cache with 5ns access time decreases miss rate to 0.5%
- Miss penalty to main memory is

$$\frac{100ns}{0.25\frac{ns}{clock\ cycle}} = 400\ clock\ cycles$$

- The effective CPI with one level of caching

Total CPI = Base CPI + Memory-stall cycles per instruction

- For the processor with one level of caching

Total CPI = 1.0 + 2%×400 = 9

- The CPI with Two level of cache with 0.5% miss rate for main memory

$$\text{Total CPI} = 1.0 + \text{Primary stalls per instruction} + \text{Secondary stalls per instruction}$$

$$= 1 + 2\% \times 20 + 0.5\% \times 400$$

$$= 1.0 + 0.4 + 2.0 = 3.4$$

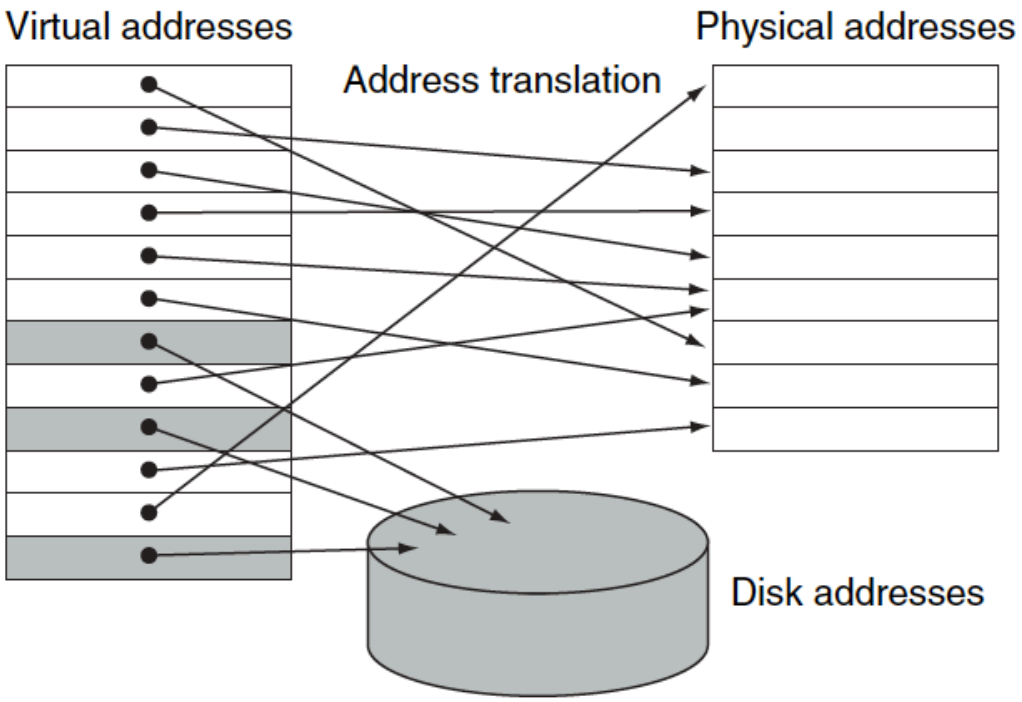
- The processor with secondary cache is faster by 9/3.4=2.6

- Different cache levels have different functionalities
  - The primary cache to focus on minimizing hit time to yield a shorter clock cycle or fewer pipeline stages,
  - The secondary cache to focus on miss rate to reduce the penalty of long memory access times.
- In comparison to a single-level cache, the primary cache of a multilevel cache is often smaller. Furthermore, the primary cache may use a smaller block size, to go with the smaller cache size and also to reduce the miss penalty.
- In comparison, the secondary cache will be much larger than in a single-level cache, since the access time of the secondary cache is less critical. With a larger total size, the secondary cache may use a larger block size than appropriate with a single-level cache. It often uses higher associativity than the primary cache given the focus of reducing miss rates.

Virtual Memory

- Virtual memory: A technique that uses main memory as a "cache" for secondary storage (e.g., that implemented by magnetic disks)
  - Allow efficient and safe sharing of memory among multiple programs
  - Remove the programming burdens of a small, limited amount of main memory
- A program is compiled with its own address space, while virtual memory implements the translation of a program's address space to physical addresses, which enforces protection of a program's address space from other programs.
- Relying on virtual memory, a single user program can exceed the size of primary memory
  - Automatically manage the two levels of the memory hierarchy represented by main memory (sometimes called physical memory to distinguish it from virtual memory) and secondary storage.

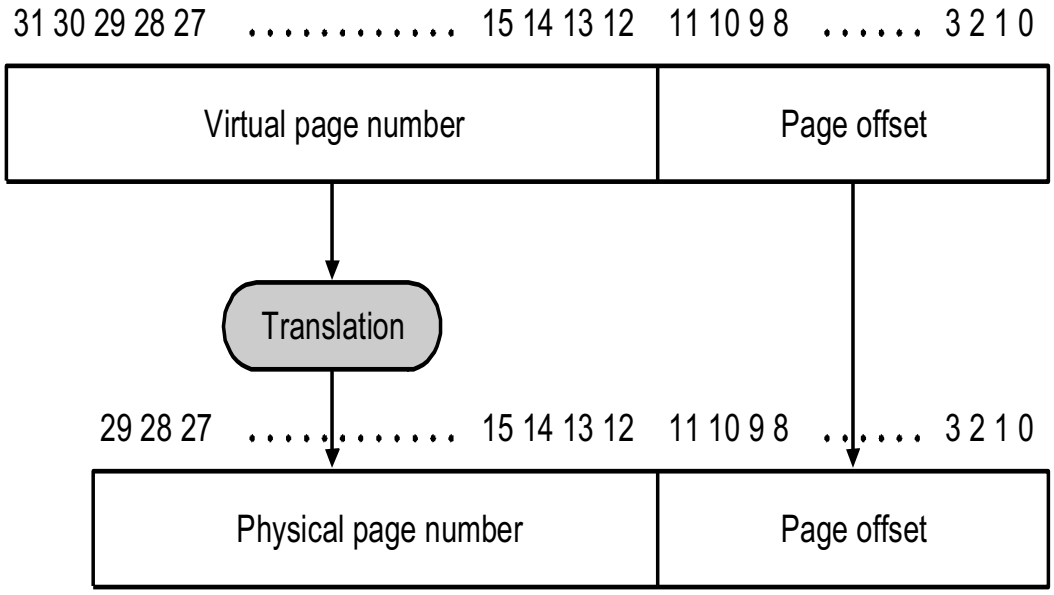
- A virtual memory block is called a page, and a virtual memory miss is called a page fault.
- With virtual memory, the processor produces a virtual address, which is translated by a combination of hardware and software to a physical address, which in turn can be used to access main memory.
- Virtual memory also simplifies loading the program for execution by providing relocation



- The virtually addressed memory with pages mapped to main memory.
- This process is called address mapping or address translation.

Pages: virtual memory blocks

- The physical page number constitutes the upper portion of the physical address, while the page offset, which is not changed, constitutes the lower portion.
- The number of bits in the page offset field determines the page size.
- The number of virtual pages are usually much larger than the one of physical pages

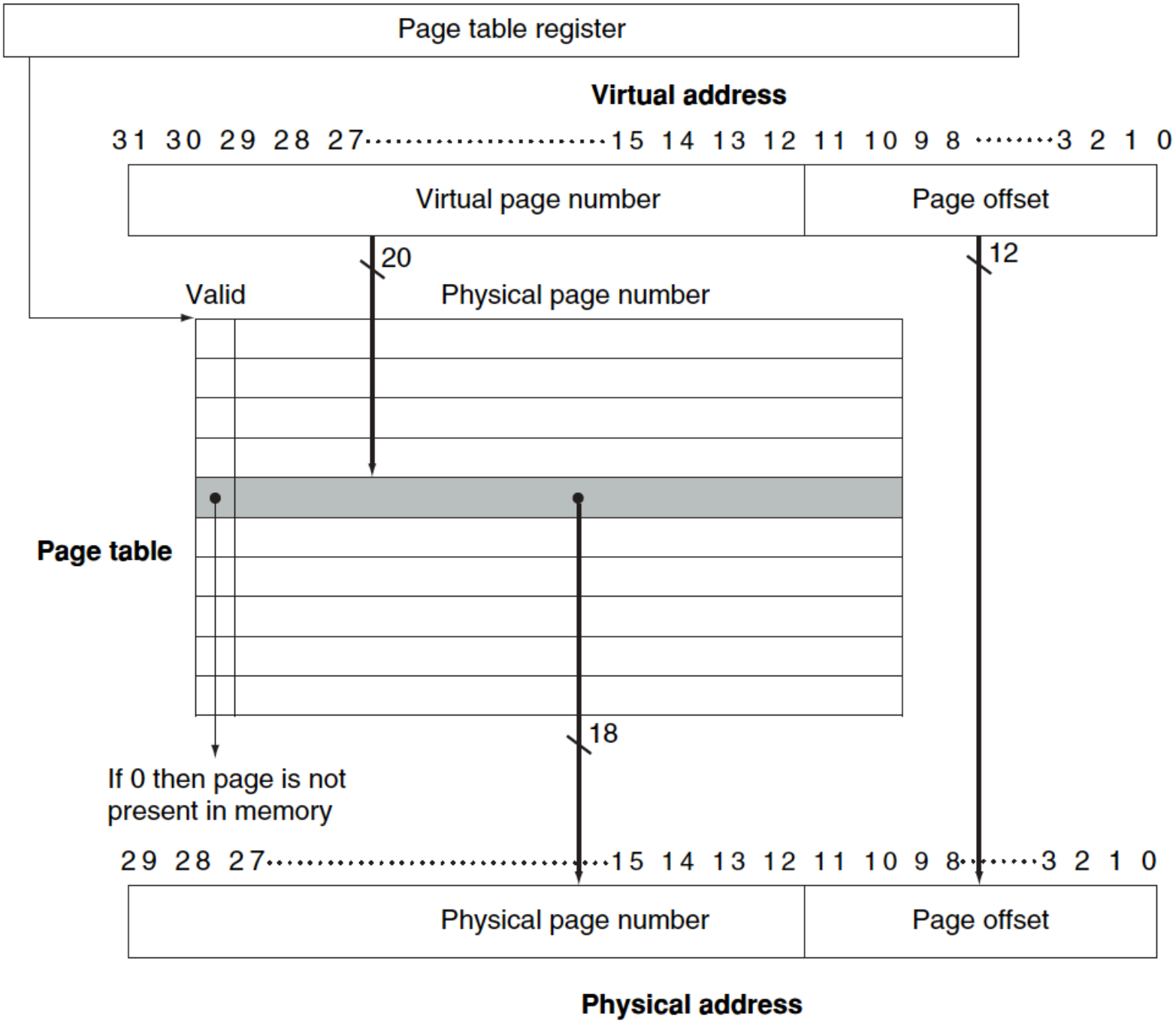


- Page fault: An event that occurs when an accessed page is not present in main memory
- A page fault may take millions of clock cycles to process
  - Pages should be large enough to try to amortize the high access time (e.g., 32 KB or 64 KB for today's desktops and servers)
  - Organizations that reduce the page fault rate are attractive (fully associative placement of pages in memory is usually employed.
  - Page faults are handled in software, since it is of light-weight overhead and has sufficient flexibility for efficient page replacement
  - Write-through will not work for virtual memory, since writes take too long. Instead, virtual memory systems use write-back.

Placing a page and finding it again

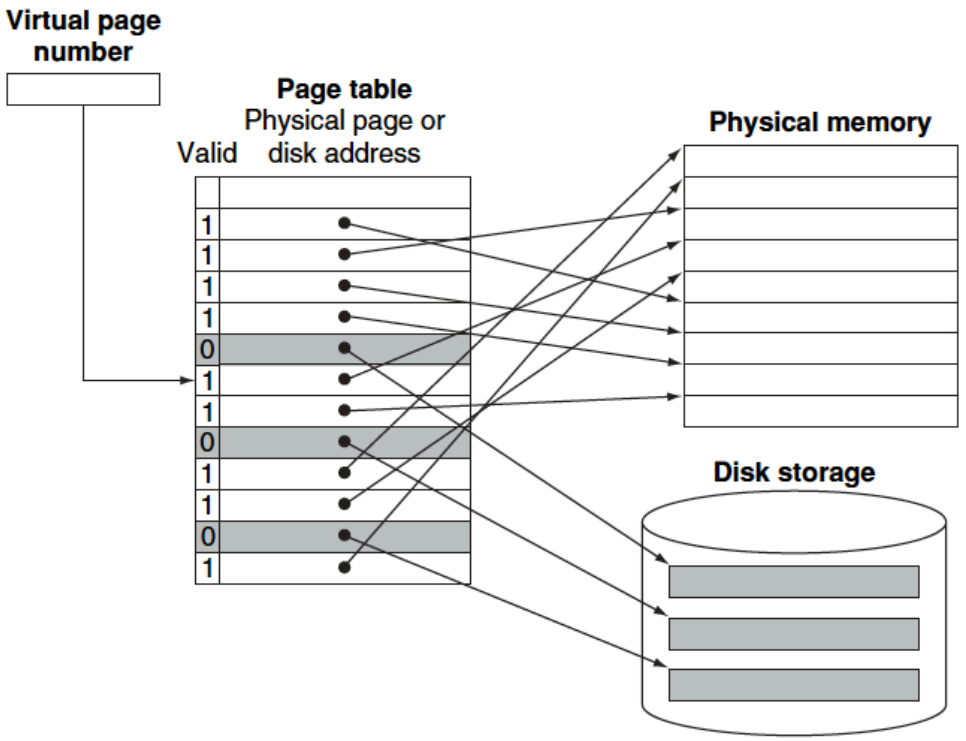
- A virtual page can be mapped to any physical page; therefore, the OS is allowed to design sophisticated algorithms and complex data structures for efficient page replacement
- The main difficult of fully associative replacement is in locating an entry, as it may be anywhere in the upper memory hierarchy
- To resolve the above problem, we adopt page table

- Page table: The table containing the virtual to physical address translations in a virtual memory system.
- Page table is stored in memory, and its start is indicated in page table register
- Each program has its own page table
- Page table is typically indexed by the virtual page number; each entry in the table contains the physical page number for that virtual page if the page is currently in memory.


Page table register
Virtual address
31 30 29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
If 0 then page is not present in memory
29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Physical page number
Page offset
Physical address

Page faults

- In a virtual memory system, we must keep track of the location on disk of each page in virtual address space
- When the OS creates a process, it usually creates a space on disk for all the pages of a process.
- When a page fault occurs, the OS will be given control through exception mechanism.
- The OS will find the page in the disk by the page table.
- Next, the OS will bring the requested page into main memory. If all the pages in main memory are in use, the OS will use LRU strategy to choose a page to replace



- Implementing a completely accurate LRU scheme is too expensive
- Most operating systems approximate LRU by keeping track of which pages have and which pages have not been recently used.
- To help the operating system estimate the LRU pages, some computers provide a reference bit or use bit, which is set whenever a page is accessed.
- The operating system periodically clears the reference bits and later records them so it can determine which pages were touched during a particular time period.
- With this usage information, the operating system can select a page that is among the least recently referenced (detected by having its reference bit off).

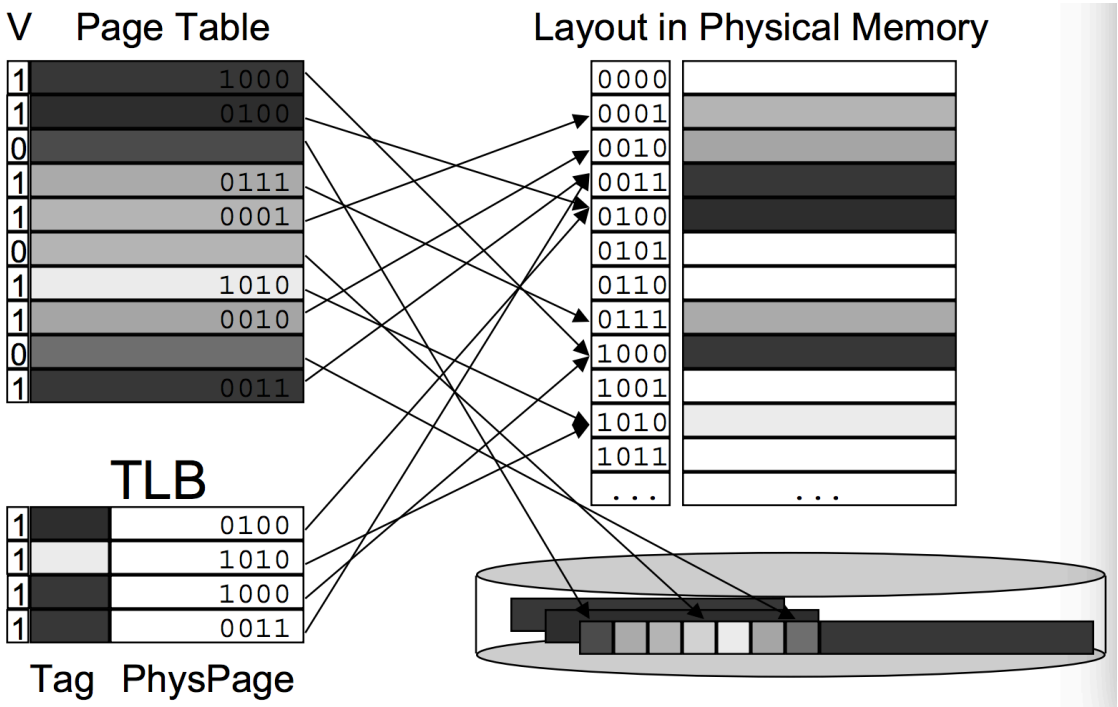
What about writes?

- Because disk accesses are too slow, virtual memory systems can not use write-through strategy.
- Instead, they must use write-back strategy. To do so, the machines need add a dirty bit to the entry of page table.
- The dirty bit is set when a page is first written. If the dirty bit of a page is set, the page must be written back to disk before being replaced.

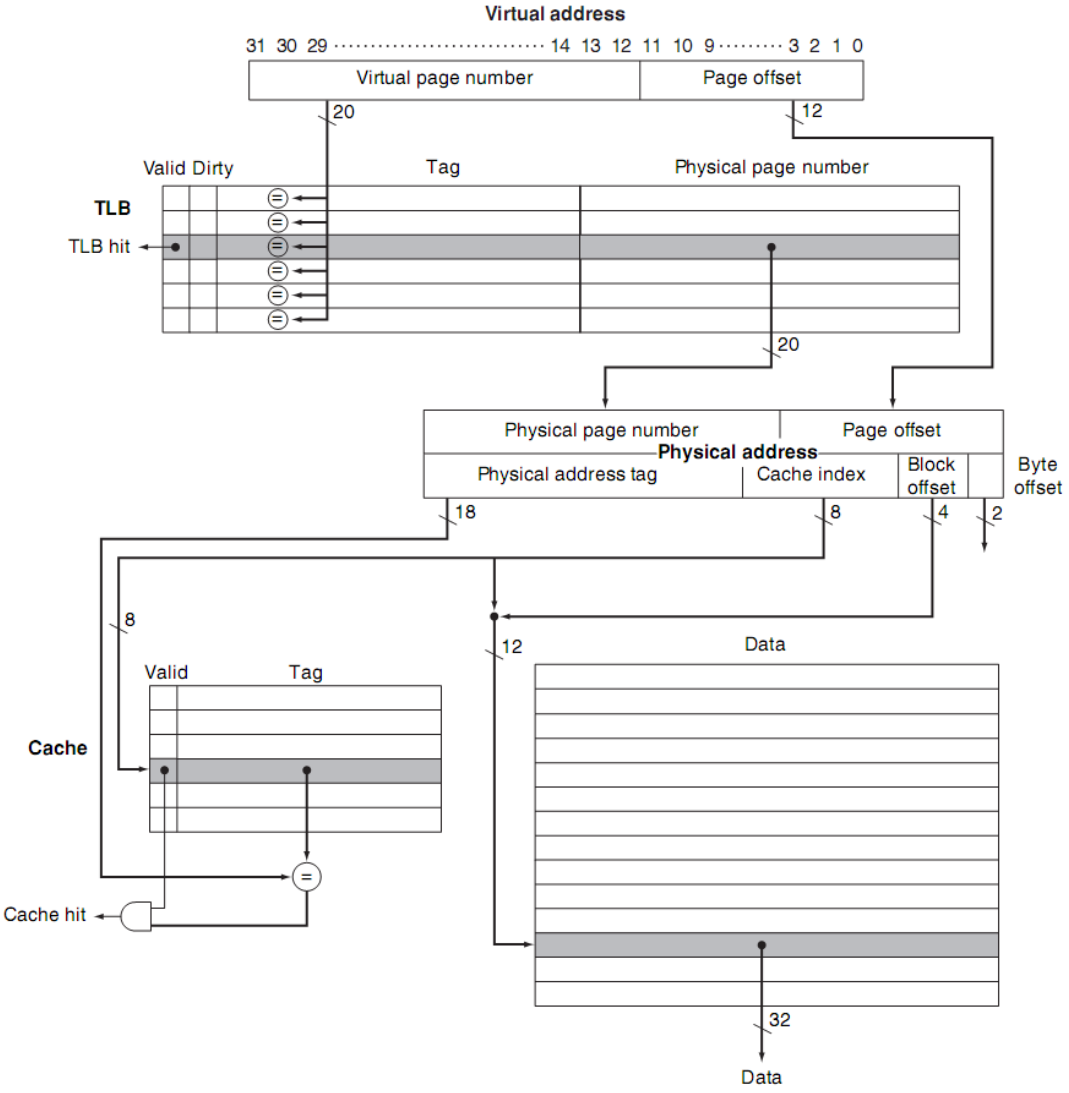
Making Address Translation Fast----TLB

- Virtual Memory would not be very effective if every memory address had to be translated by looking up the associated physical page in memory. The solution is to cache the recent translations in a Translation Lookaside Buffer (TLB)

- Translation-Lookaside Buffer (TLB): A cache that keeps track of recently used address mappings to try to avoid an access to the page table.

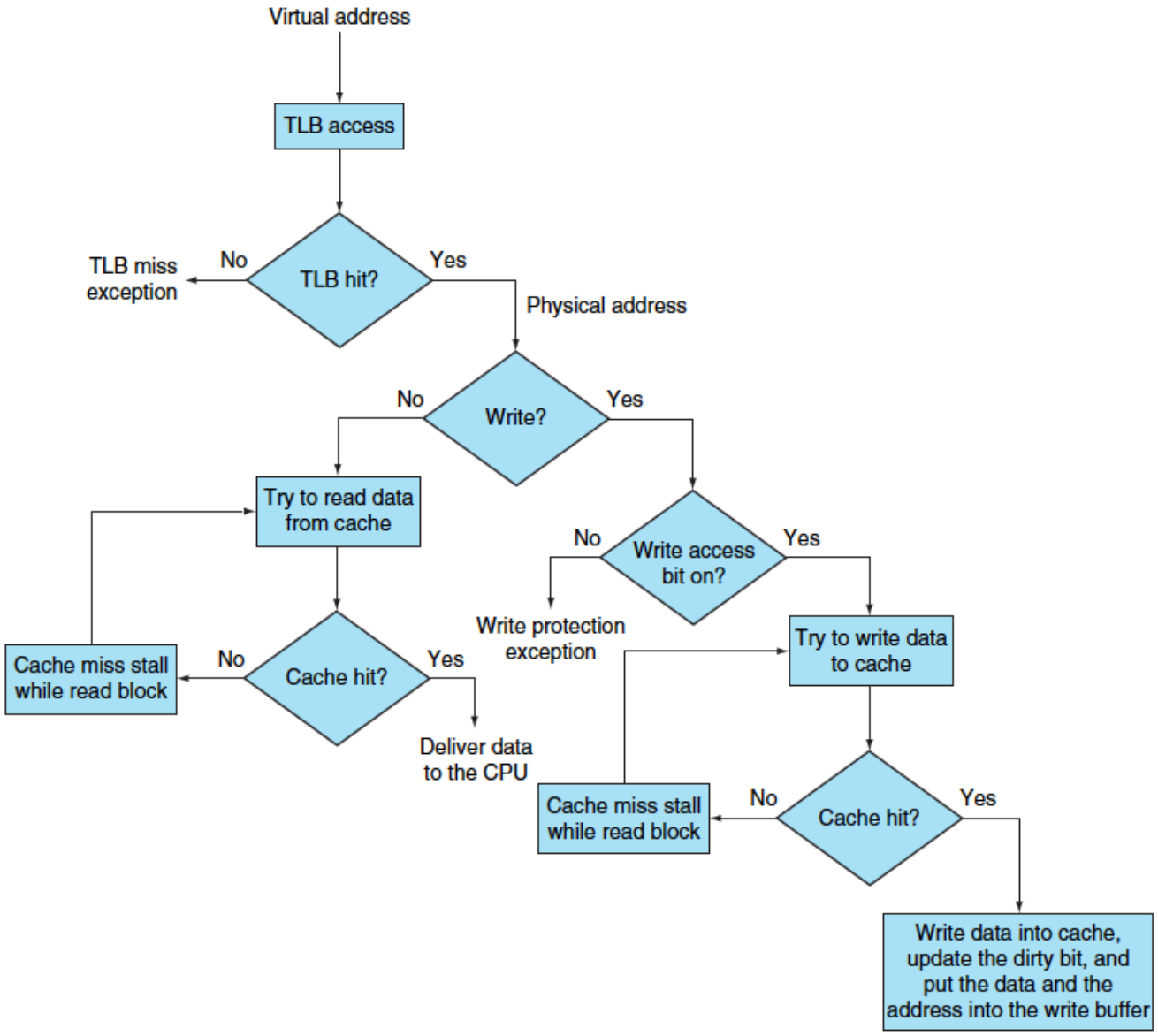


- The TLB contains a subset of the virtual-to-physical page mappings that are in the page table
- Because the TLB is a cache ,it must have a tag field. If there is no matching entry in the TLB for a page, the page table must be examined.
- The page table either supplies a physical page number for the page (which can then be used to build a TLB entry) or indicates that the page resides on disk, in which case a page fault occurs.
- Since the page table has an entry for every virtual page, no tag field is needed; in other words, unlike a TLB, a page table is not a cache.



- Some typical values for a TLB
  - TLB size: 16–512 entries
  - Block size: 1–2 page table entries (typically 4–8 bytes each)
  - Hit time: 0.5–1 clock cycle
  - Miss penalty: 10–100 clock cycles
  - Miss rate: 0.01%–1%

TLBs and caches



Integrating caches, BLTs and memory

- Our virtual memory and cache systems work together as a hierarchy, so that data cannot be in the cache unless it is present in main memory.
- The operating system helps maintain this hierarchy by flushing the contents of any page from the cache when it decides to migrate that page to disk.
- At the same time, the OS modifies the page tables and TLB, so that an attempt to access any data on the migrated page will generate a page fault.

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

Implementing Protection with Virtual Memory

- To enable the operating system to implement protection in the virtual memory system, the hardware must provide at least the three basic capabilities:
  - Indicating the current process is a user process or an OS process (or a supervisor process, kernel process, an executive process)
  - A portion of the processor state that users can read but not write
  - Processor can go from user mode to supervisor mode and vice versa.
- Preventing a process from reading the data of another process
- Placing the page tables in the protected address space of the OS, such that i) one process cannot modify its own page table; ii) OS can modify the page tables
- OS is supposed to help processes share information in a limited way

Context switch or process switch

- A changing of the internal state of the processor to allow a different process to use the processor that includes saving the state needed to return to the currently executing process

Handling TLB Misses and Page Faults

- A TLB miss can indicate one of two possibilities:
  - The page is present in memory, and we need only create the missing TLB entry.
  - The page is not present in memory, and we need to transfer control to the operating system to deal with a page fault.
- When we process the TLB miss, we will look for a page table entry to bring into the TLB. If the matched page table entry has a valid bit that is turned off, then the corresponding page is not in memory ,and we have a page fault, rather than just a TLB miss.
- If the valid bit is on, we can simply retrieve the desired entry.

- MIPS traditionally handles a TLB miss in software
  - Bring in the page table entry from memory
  - Re-execute the instruction that causes the TLB miss, and get a hit
  - If the page table entry indicates the page is not in memory, it results in a page fault exception
- A TLB miss or page fault exception must be asserted by the end of the same clock cycle that the memory access occurs
- When an exception first occurs, the processor sets a bit that disables all other exceptions and then save enough state for recovering

Register	CP0 register number	Description
EPC	14	Where to restart after exception
Cause	13	Cause of exception
BadVAddr	8	Address that caused exception
Index	0	Location in TLB to be read or written
Random	1	Pseudorandom location in TLB
EntryLo	2	Physical page address and flags
EntryHi	10	Virtual page address
Context	4	Page table address and page number

- Once the operating system knows the virtual address that caused the page fault, it must complete three steps:
  1. Look up the page table entry using the virtual address and find the location of the referenced page on disk.
  2. Choose a physical page to replace; if the chosen page is dirty, it must be written out to disk before we can bring a new virtual page into this physical page.
  3. Start a read to bring the referenced page from disk into the chosen physical page.

- Page fault exceptions for data accesses are difficult to implement properly in a processor because of a combination of three characteristics:
  1. They occur in the middle of instructions, unlike instruction page faults.
  2. The instruction cannot be completed before handling the exception.
  3. After handling the exception, the instruction must be restarted as if nothing had occurred.
- Restartable instruction
  - An instruction that can resume execution after an exception is resolved without the exception's affecting the result of the instruction.

How does MIPS handle TLB miss?

- The page number of the reference is saved in register BadVAddr and an exception is generated
- The exception invokes the operating system, which handles the miss in software
  - Control is transferred to address $8000\ 0000_{hex}$ (the location of TLB handler)
  - Find the physical address for the missing page and update the TLB

```
TLBmiss:
  mfc0  $k1,Context   # copy address of PTE into temp $k1
  lw    $k1, 0($k1)   # put PTE into temp $k1
  mtc0  $k1,EntryLo   # put PTE into special register EntryLo
  tlbwr               # put EntryLo into TLB entry at Random
  eret                # return from TLB miss exception
```

- The TLB miss handler does not check to see of the page table entry is valid

A Common Framework for Memory Hierarchies

- The key quantitative design parameters that characterize the major elements of memory hierarchy in a computer.

Feature	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Total size in blocks	250–2000	15,000–50,000	16,000–250,000	40–1024
Total size in kilobytes	16–64	500–4000	1,000,000–1,000,000,000	0.25–16
Block size in bytes	16–64	64–128	4000–64,000	4–32
Miss penalty in clocks	10–25	100–1000	10,000,000–100,000,000	10–1000
Miss rates (global for L2)	2%–5%	0.1%–2%	0.00001%–0.0001%	0.01%–2%

Deep concept in Cache

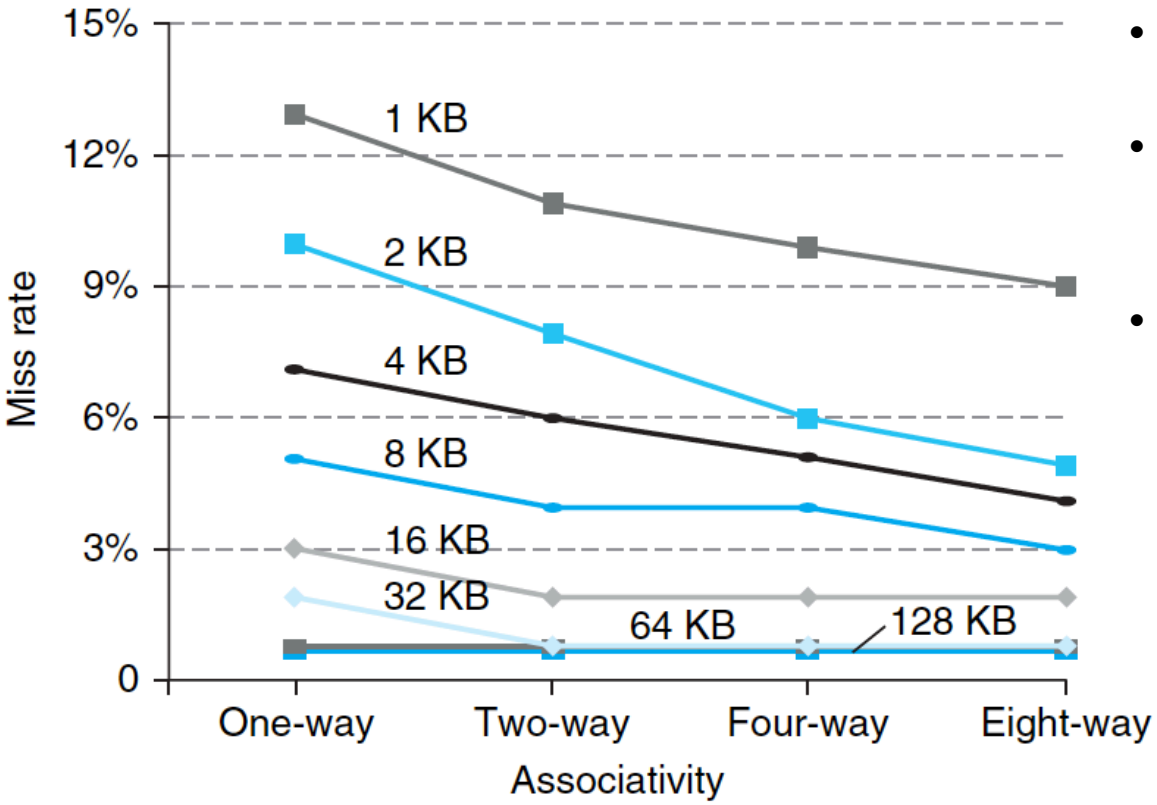
Caching is a general concept used in processors, operating systems, file systems, and applications.

Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level?
  - Block placement: Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level?
  - Block identification: Tag/Block
- Q3: Which block should be replaced on a miss?
  - Block replacement: Random, LRU,FIFO
- Q4: What happens on a write?
  - Write strategy: Write Back or Write Through (with Write Buffer)

Q1: Block Placement

Scheme name	Number of sets	Blocks per set
Direct mapped	Number of blocks in cache	1
Set associative	$\frac{\text{Number of blocks in the cache}}{\text{Associativity}}$	Associativity (typically 2–16)
Fully associative	1	Number of blocks in the cache



- Increasing associativity can decrease miss rate
- As cache sizes grow, the relative improvement from associativity increases only slightly
- The potential disadvantages of associativity are increased cost and slower access time

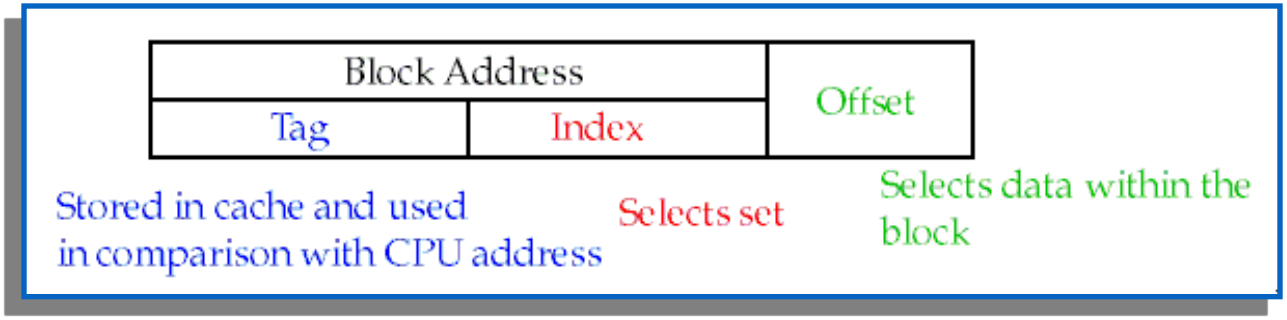
Q2: Block Identification

- Every block has an address tag that stores the main memory address of the data stored in the block.
- When checking the cache, the processor will compare the requested memory address to the cache tag -- if the two are equal, then there is a cache hit and the data is present in the cache
- Often, each cache block also has a valid bit that tells if the contents of the cache block are valid

Associativity	Location method	Comparisons required
Direct mapped	Index	1
Set associative	Index the set, search among elements	Degree of associativity
Full	Search all cache entries	Size of the cache
	Separate lookup table	0

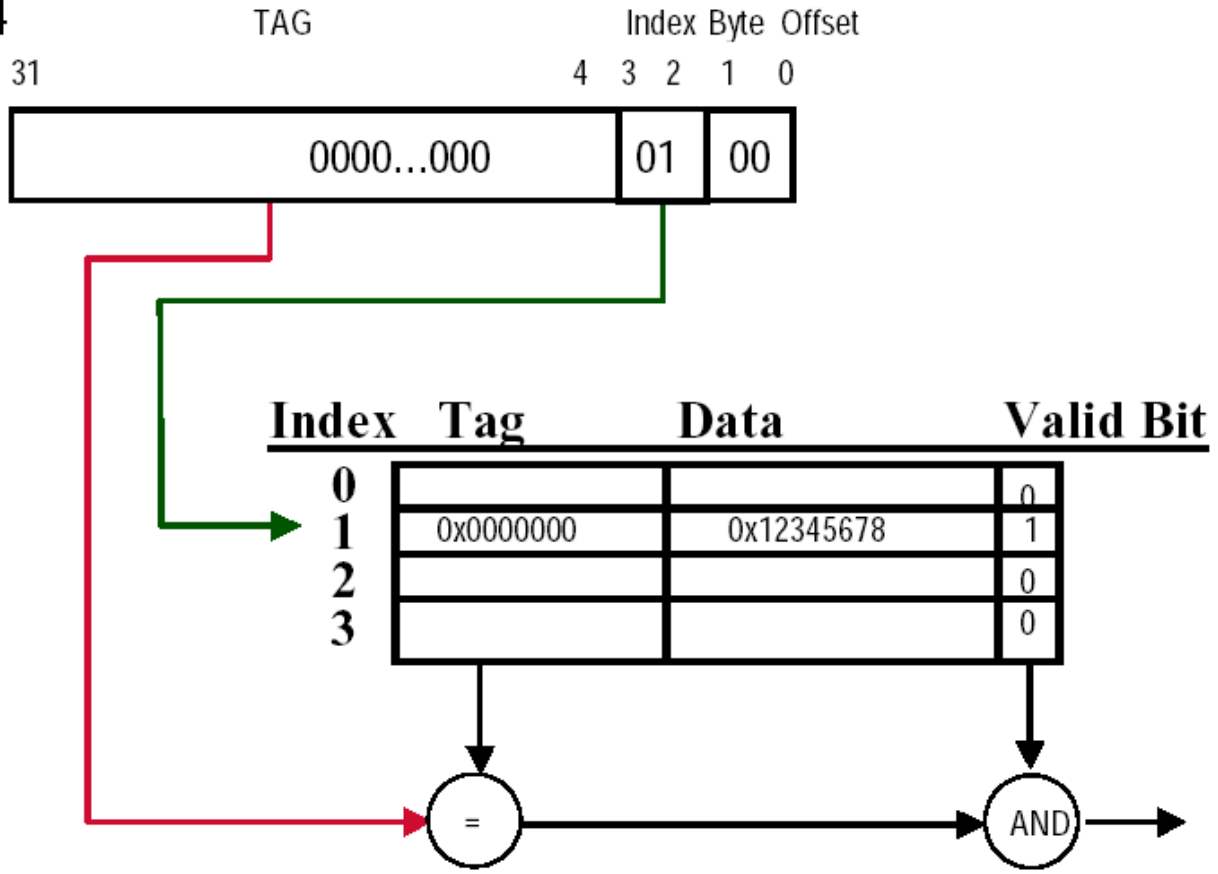
The Format of the Physical Address

- The Index field selects
  - The set, in case of a set-associative cache
  - The block, in case of a direct-mapped cache
  - Has as many bits as log2(# of sets) for set-associative caches, or log2(# of blocks) for direct-mapped caches
- The Byte Offset field selects
  - The byte within the block
  - Has as many bits as log2(size of block)
- The Tag is used to find the matching block within a set or in the cache
  - Has as many bits as
    Address_size – Index_size – Byte_Offset_Size



Direct-mapped Cache Example (1-word Blocks)

LOAD R1, 0x04

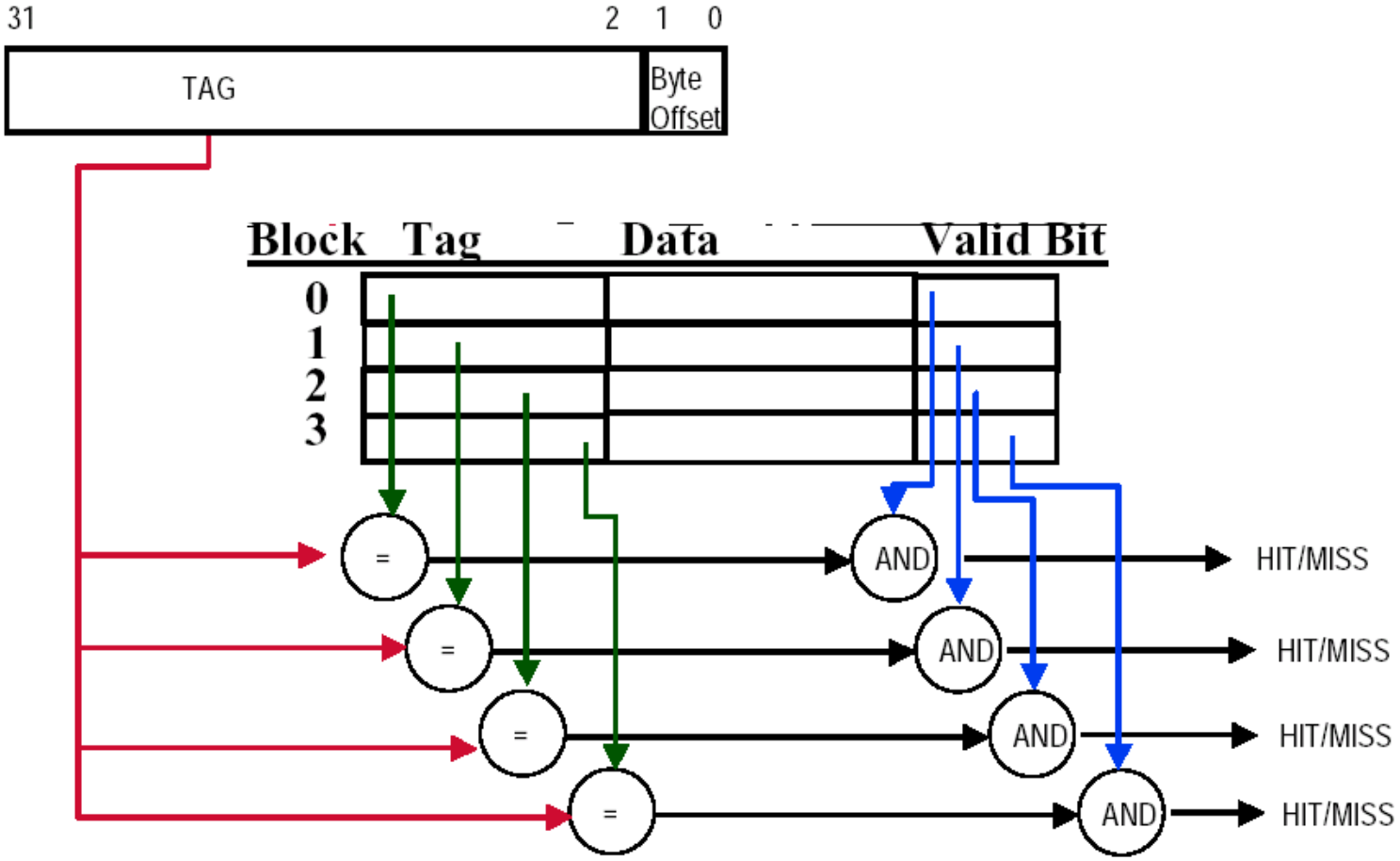


MEMORY

Address	Data
0x00	0x00000000
0x04	0x12345678
0x08	0x87654321
0x0C	0x11111111
0x10	0x22222222
0x14	0x33333333
0x18	0x44444444
0x1C	0x55555555
0x20	0x10101010

Fully-Associative Cache example (1-word Blocks)

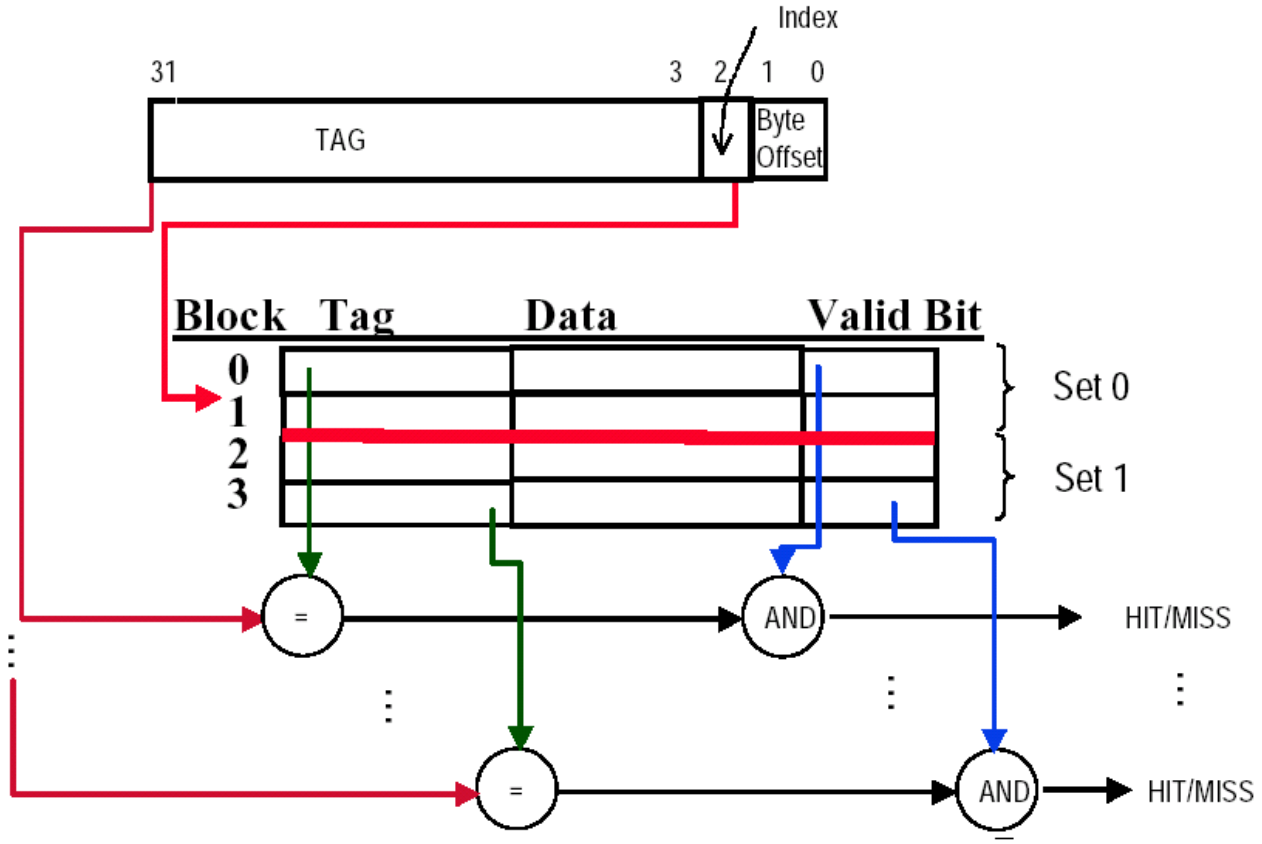
- Assume cache has 4 blocks



- The choice of full associativity for page placement and the extra table is motivated by these facts:
  1. Full associativity is beneficial, since misses are very expensive.
  2. Full associativity allows software to use sophisticated replacement schemes that are designed to reduce the miss rate.
  3. The full map can be easily indexed with no extra hardware and no searching required.

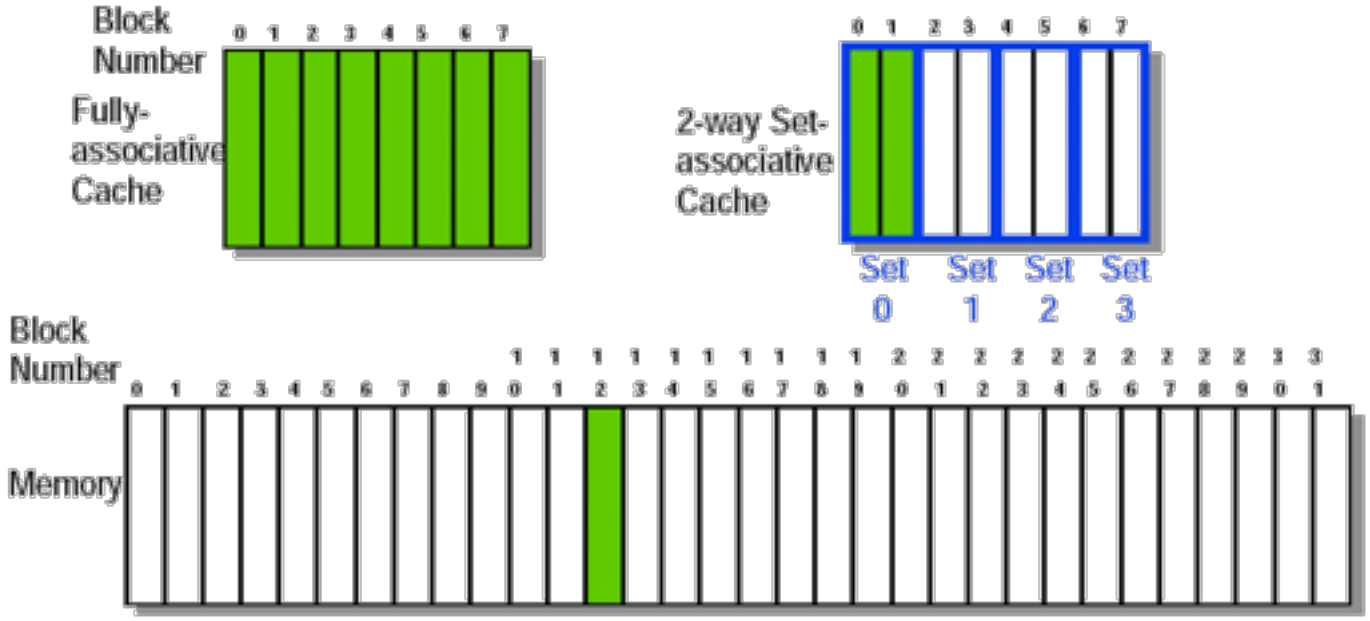
2-Way Set-Associative Cache

- Assume cache has 4 blocks and each block is 1 word
- 2 blocks per set, hence 2 sets per cache



Q3: Block Replacement

- In a direct-mapped cache, there is only one block that can be replaced
- In set-associative and fully-associative caches, there are N blocks (where *N* is the degree of associativity)



Strategy of block Replacement

- Several different replacement policies can be used
  - Random replacement - randomly pick any block
    - Easy to implement in hardware, just requires a random number generator
    - Spreads allocation uniformly across cache
    - May evict a block that is about to be accessed
  - Least-recently used (LRU) - pick the block in the set which was least recently accessed
    - Assumed more recently accessed blocks more likely to be referenced again
    - This requires extra bits in the cache to keep track of accesses.
  - First in,first out(FIFO)-Choose a block from the set which was first came into the cache

Q4: Write Strategy

- When data is written into the cache (on a store), is the data also written to main memory?
  - **If the data is written to memory, the cache is called a** *write-through cache*
    - Can always discard cached data - most up-to-date data is in memory
    - Cache control bit: only a *valid* bit
    - memory (or other processors) always have latest data
  - **If the data is NOT written to memory, the cache is called a** *write-back cache*
    - Can't just discard cached data - may have to write it back to memory
    - Cache control bits: both *valid* and *dirty* bits
    - much lower bandwidth, since data often overwritten multiple times
- Write-through pro: Read misses don't result in writes, memory hierarchy is consistent and it is simple to implement.
- Write back pro: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

- The key advantages of write-back
  - Individual words can be written by the processor at the rate that the cache rather than the memory, can accept them
  - Multiple writes within a block require only one write to the lower level in the hierarchy
  - When blocks are written back, the system can make effective use of a high-bandwidth transfer, since the entire block is written

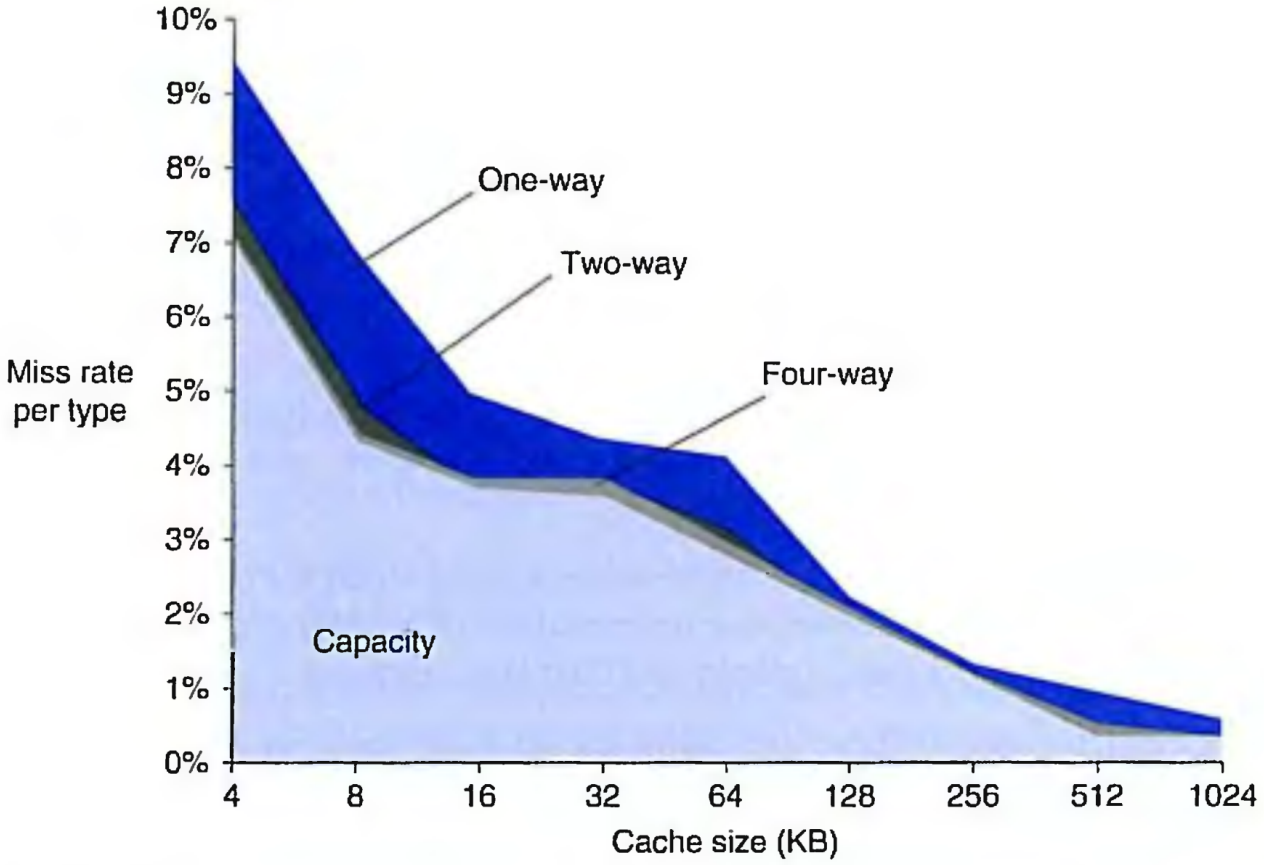
- The key advantages of write-through
  - Misses are simpler and cheaper because they never require a block to be written back the lower level
  - Write-through is easier to implement than write-back, although to be practical, a write-through cache will still need to use a write buffer

The Three Cs: An Intuitive Model for Understanding the Behavior of Memory Hierarchies

- Compulsory misses:
  - These are cache misses caused by the first access to a block that has never been in the cache. These are also called cold-start misses.
- Capacity misses
  - These are cache misses caused when the cache cannot contain all the blocks needed during execution of a program. Capacity misses occur when blocks are replaced and then later retrieved.
- Conflict misses
  - These are cache misses that occur in set-associative or direct-mapped caches when multiple blocks compete for the same set. Conflict misses are those misses in a direct-mapped or set-associative cache that are eliminated in a fully associative cache of the same size. These cache misses are also called collision misses.

Design change	Effect on miss rate	Possible negative performance effect
Increase cache size	Decreases capacity misses	May increase access time
Increase associativity	Decreases miss rate due to conflict misses	May increase access time
Increase block size	Decreases miss rate for a wide range of block sizes due to spatial locality	Increases miss penalty. Very large block could increase miss rate

The miss rate can be broken into three sources of misses.



Using a finite-state machine to control a simple cache

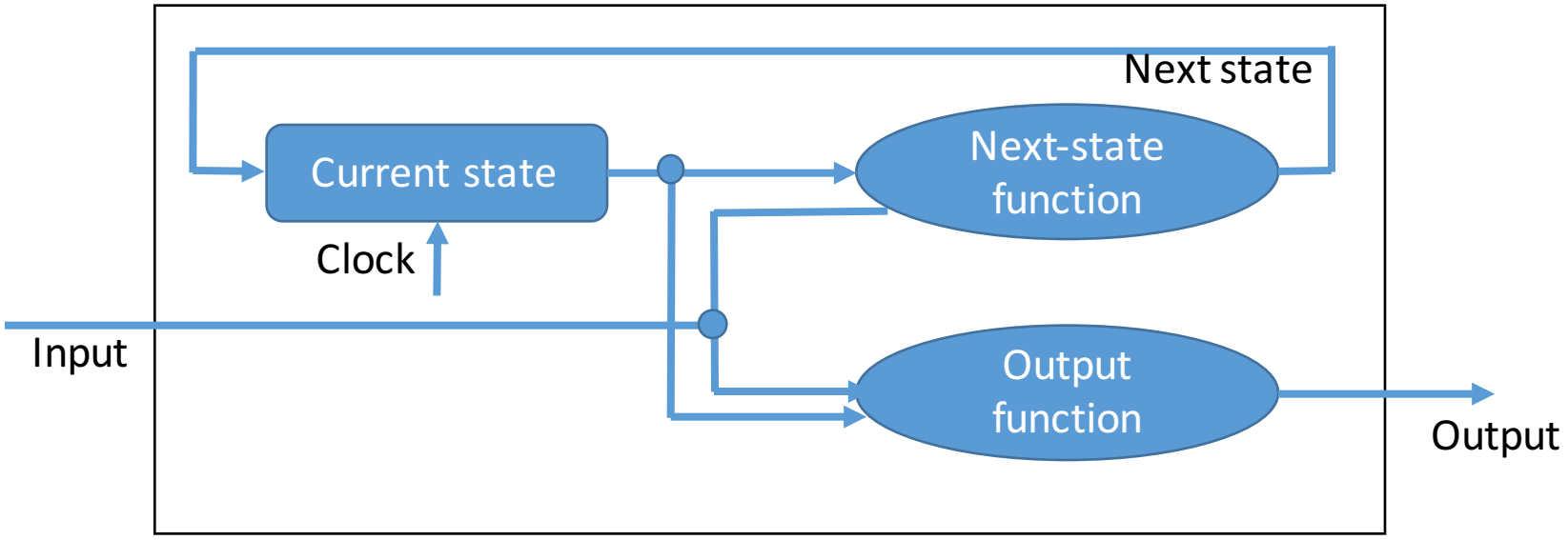
- A simple cache model
  - Direct-mapped cache
  - Write-back
  - Block size is 4 words
  - Cache size is 16 KiB (so it holds 1024 blocks)
  - 32-byte address
  - The cache includes a valid bit and dirty bit per block
- Cache address
  - Cache index is 10 bits
  - Block offset is 4 bits
  - Tag size is 18 bits

- The signals between the processor and the cache
  - 1-bit Read or Write signal
  - 1-bit Valid signal, saying whether there is a cache operation or not
  - 32-bit address
  - 32-bit data from processor to cache
  - 32-bit data from cache to processor
  - 1-bit Ready signal, saying the cache operation is complete

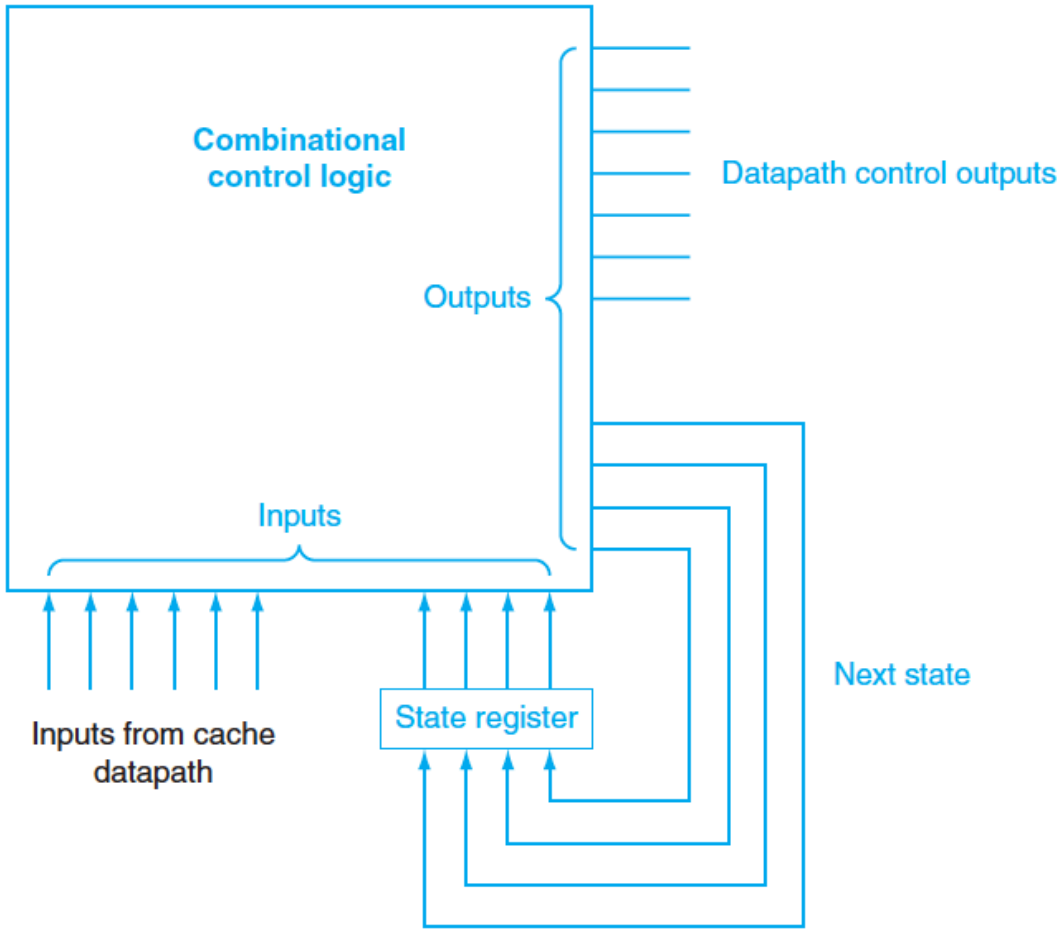
- The signals between the memory and the cache
  - 1-bit Read or Write signal
  - 1-bit Valid signal, saying whether there is a memory operation or not
  - 32-bit address
  - 128-bit data from processor to cache
  - 128-bit data from cache to processor
  - 1-bit Ready signal, saying the memory operation is complete

Finite-state machines (FSM)

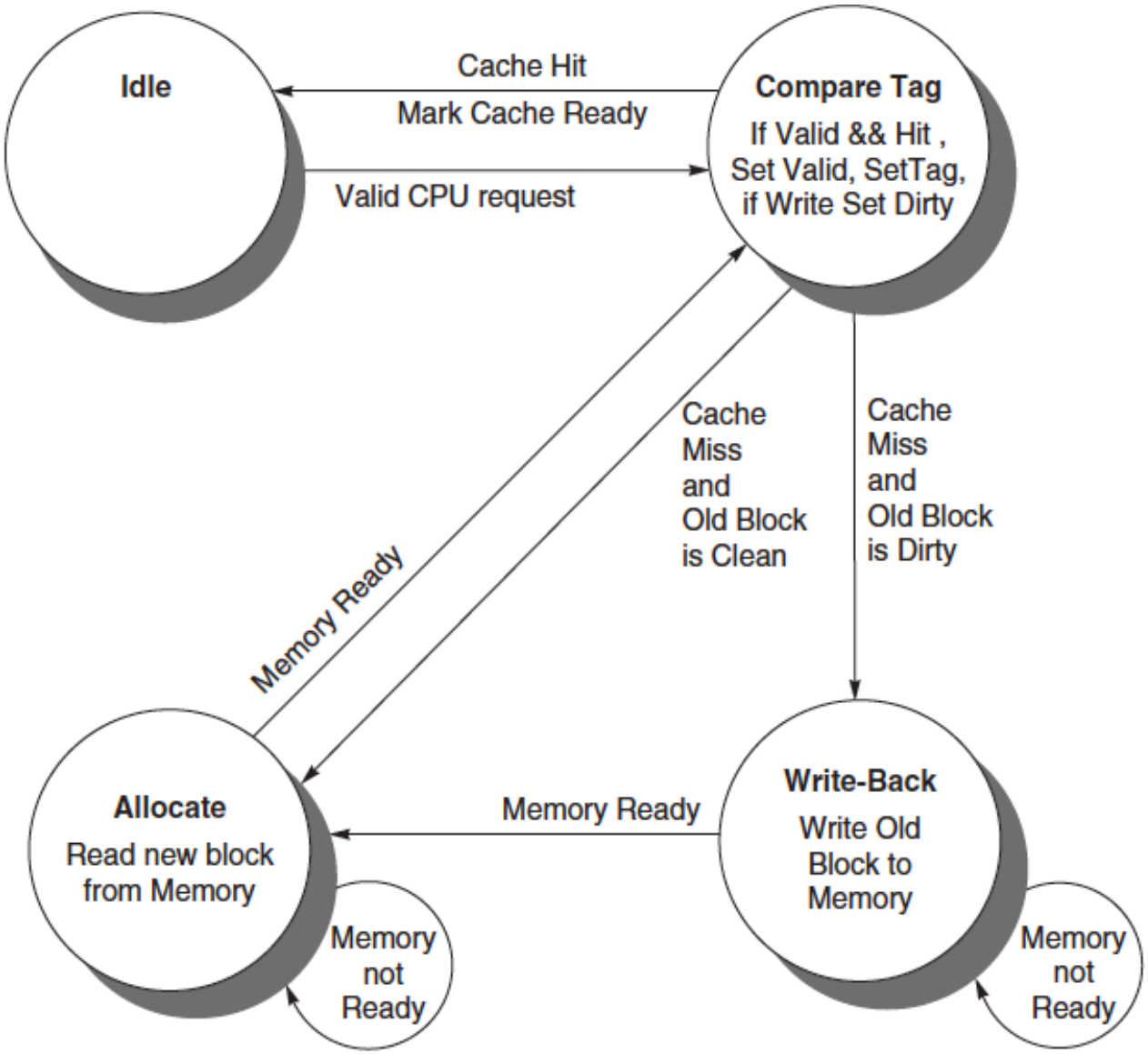
- A finite-state machine consists of a set of states and directions on how to change the states
- The directions are defined by a next-state function, which maps the current state and the inputs to a new state
- Each state specifies a set of asserted outputs
- The implementation of a finite-state machine usually assumes that all output that are not explicitly asserted are deasserted

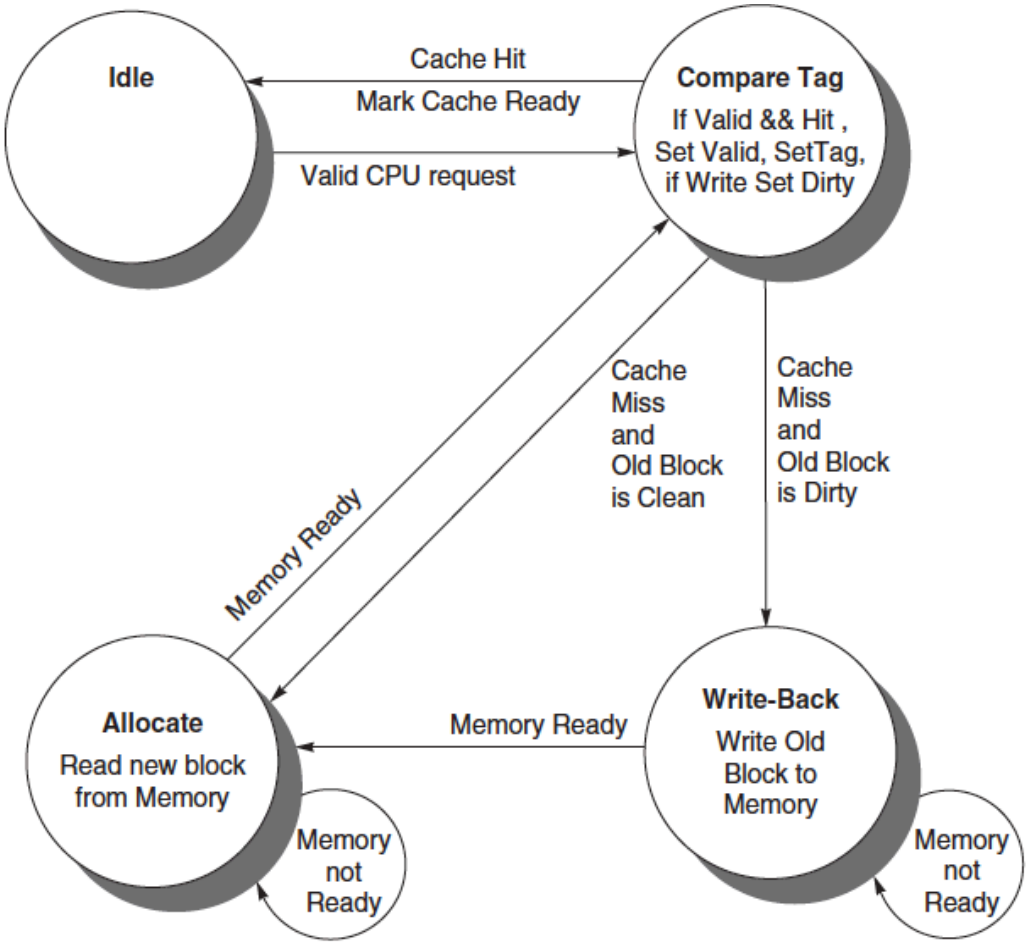


- A finite-state machine is implemented with
  - a temporary register (that holds the current state)
  - a block of combinational logic (that determines both the data-path signals to be asserted and the next state)

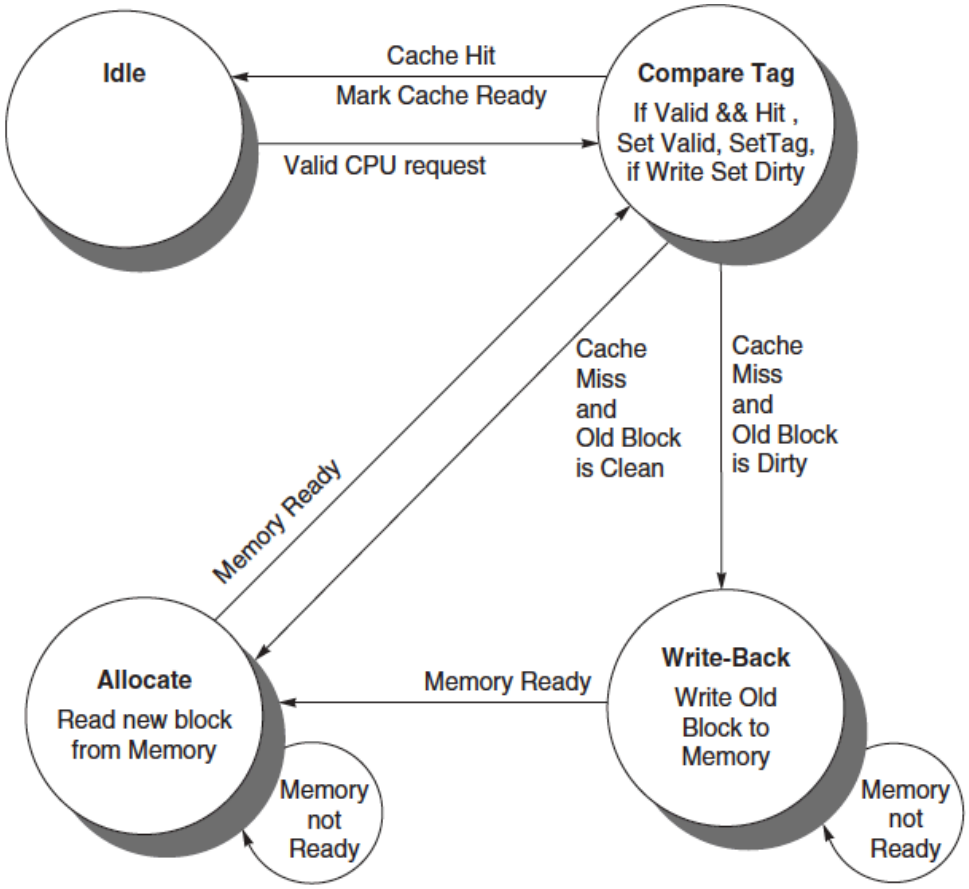


Four states of the simple controller

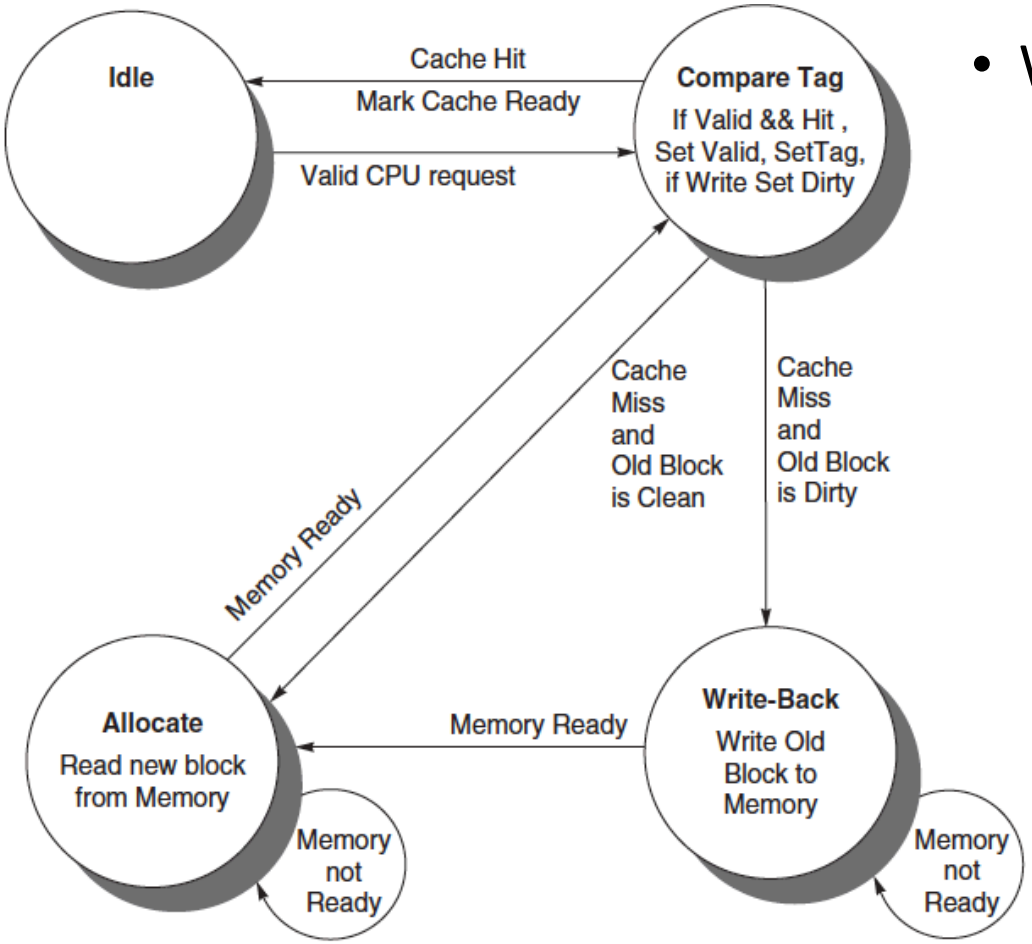




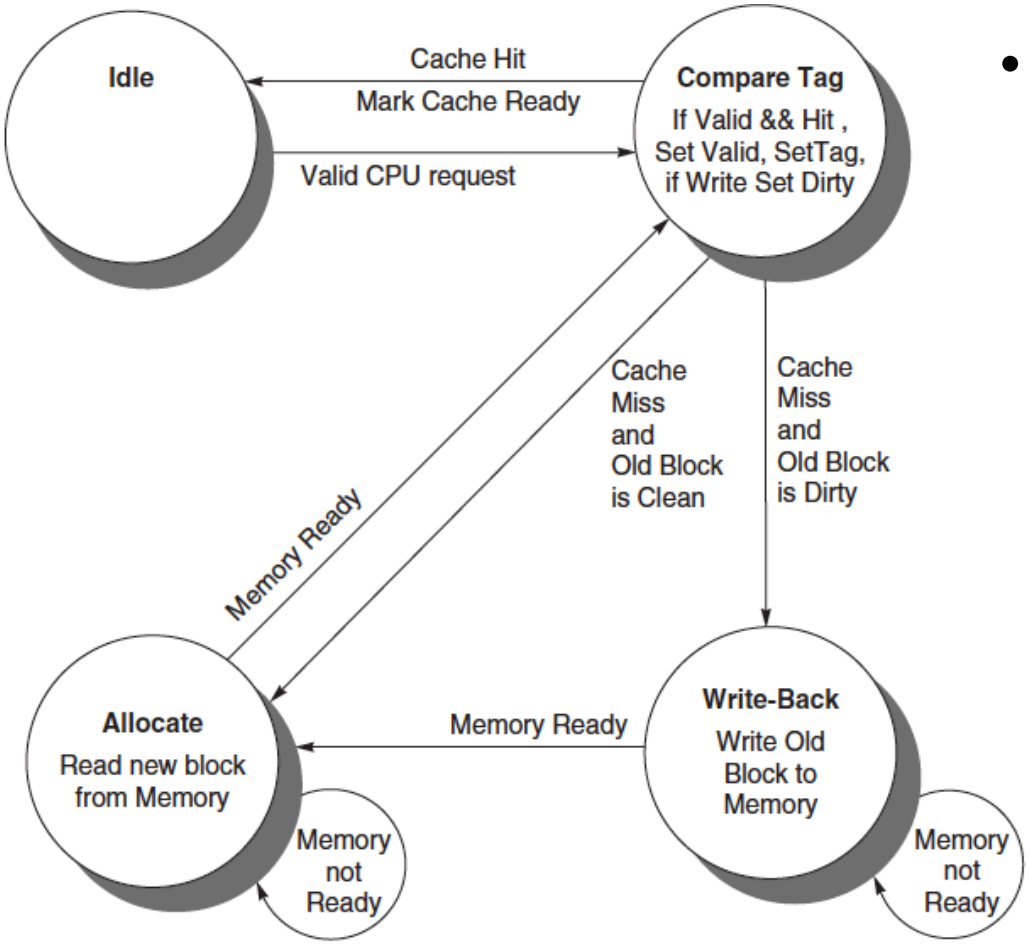
- Idle: waiting for a valid read or write request from the processor



- Compare Tag
  - Check if the request read or write is a hit or a miss
  - If a hit, either the data is read from the selected word or written to the selected word
  - Set the Cache Ready signal
  - If it is a write, set the dirty bit
  - A write hit also sets the valid bit and the tag field in memory
  - If it is a hit and the block is valid, the FSM returns to the idle state
  - A miss first updates the cache tag and then goes either to the Write-Back state (if the block at this location has dirty bit value of 1), or to the Allocation state (if it is 0)



- Write-Back
  - Writing the 128-bit block to memory
  - Staying in this state and waiting for the Ready signal from memory
  - When the memory write is completed, the FSM goes into Allocate state



- Allocate
  - Fetching a new block from memory
  - Going into the state of Compare Tag when the memory read is completed.

Parallelism and Memory Hierarchy: Cache Coherence

- A multicore multiprocessor means multiple processors on a single chip
- These processors share the same physical address space
- Each processor interacts with memory through their individual caches

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

- A memory system is coherent if
  1. A read by a processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P.
  2. A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.
  3. Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors.

Basic Schemes for Enforcing Coherence

- Migration
  - A data item can be moved to a local cache and used there in a transparent fashion. Migration reduces both the latency to access a shared data item that is allocated remotely and the bandwidth demand on the shared memory.
- Replication
  - When shared data are being simultaneously read, the caches make a copy of the data item in the local cache. Replication reduces both latency of access and contention for a read shared data item.

Cache coherence protocol

- Snooping protocol
  - Every cache that has a copy of the data from a block of physical memory also has a copy of the sharing status of the block, but no centralized state is kept. The caches are all accessible via some broadcast medium (a bus or network), and all cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access.
  - One method of enforcing coherence is to ensure that a processor has exclusive access to a data item before it writes that item.
  - This style of protocol is called a write invalidate protocol because it invalidates copies in other caches on a write.
  - Exclusive access ensures that no other readable or writable copies of an item exist when the write occurs: all other cached copies of the item are invalidated.

Thanks !