Lecture 25
Translation Lookaside Buffer

School of Computer Science and Engineering
Soongsil University

5. Large and Fast: Exploiting Memory Hierarchy

5.1 Introduction

5.2 Memory Technologies

5.3 The Basics of Caches

5.4 Measuring and Improving Cache Performance

5.5 Dependable Memory Hierarchy

5.6 Virtual Machines

5.7 Virtual Memory

5.8 A Common Framework for Memory Hierarchies

5.9 Using a Finite-State Machine to Control a Simple Cache

5.10 Parallelism and Memory Hierarchies: Cache Coherence

5.11 Parallelism and Memory Hierarchy: RAID

5.12 Advanced Material: Implementing Cache Controllers

5.13 Real Stuff: The ARM Cortex-A8 and Intel Core i7 Memory Hierarchies

Making Address Translation Fast: the TLB

- **Double memory accesses in virtual memory**
  1. to obtain the physical address (i.e. to read page table)
  2. to get the data
- **TLB (translation-lookaside buffer)**
  - Using locality of reference to the page table
  - A cache that keeps track of recently used address mappings to try to avoid an access to the page table
- **Typical parameters for a TLB**

TLB size	16–512 entries
Block size	1–2 PTEs (typically 4–8 bytes each)
Hit time	0.5–1 clock cycle
Miss penalty	10–100 clock cycles
Miss rate	0.01%–1%

Address Translation with TLB

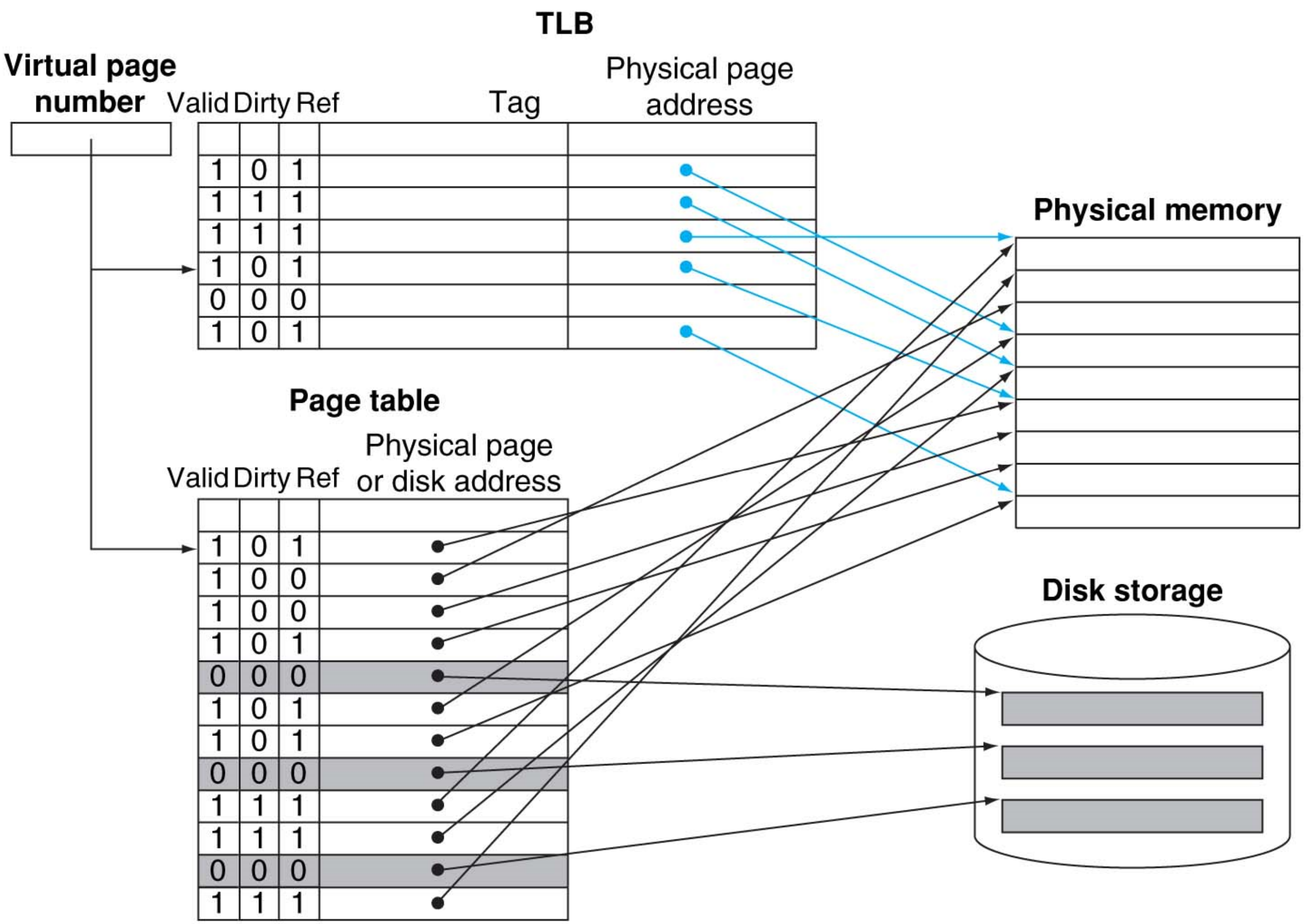


Figure 5.30

The Intrinsity FastMATH TLB

- 16 entries, 64 bits/entry
- Fully associative mapping
- Entry
  = 20-bit tag
  + 20-bit physical page number
  + valid bit
  + dirty bit
  + other bookkeeping bits

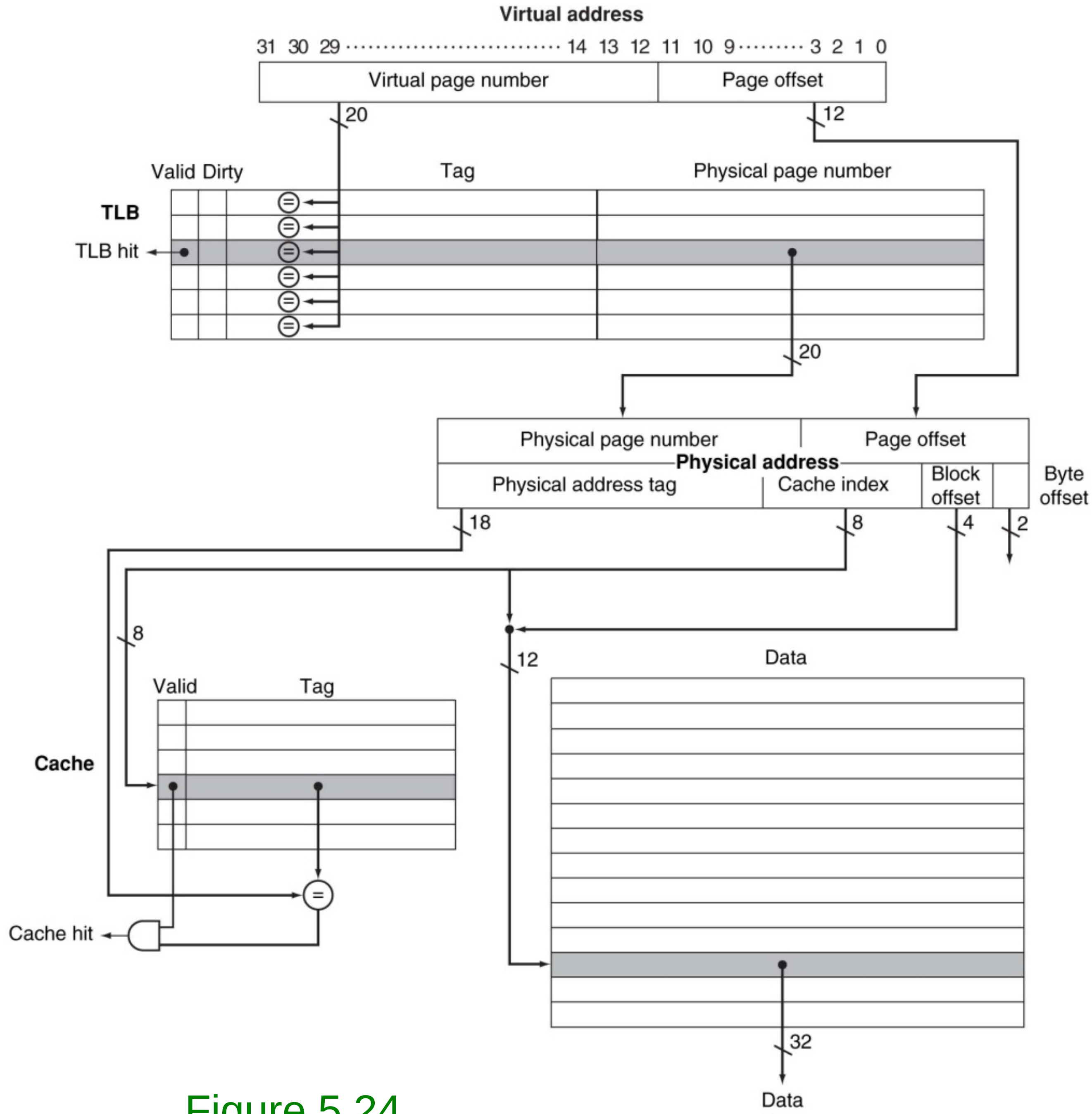
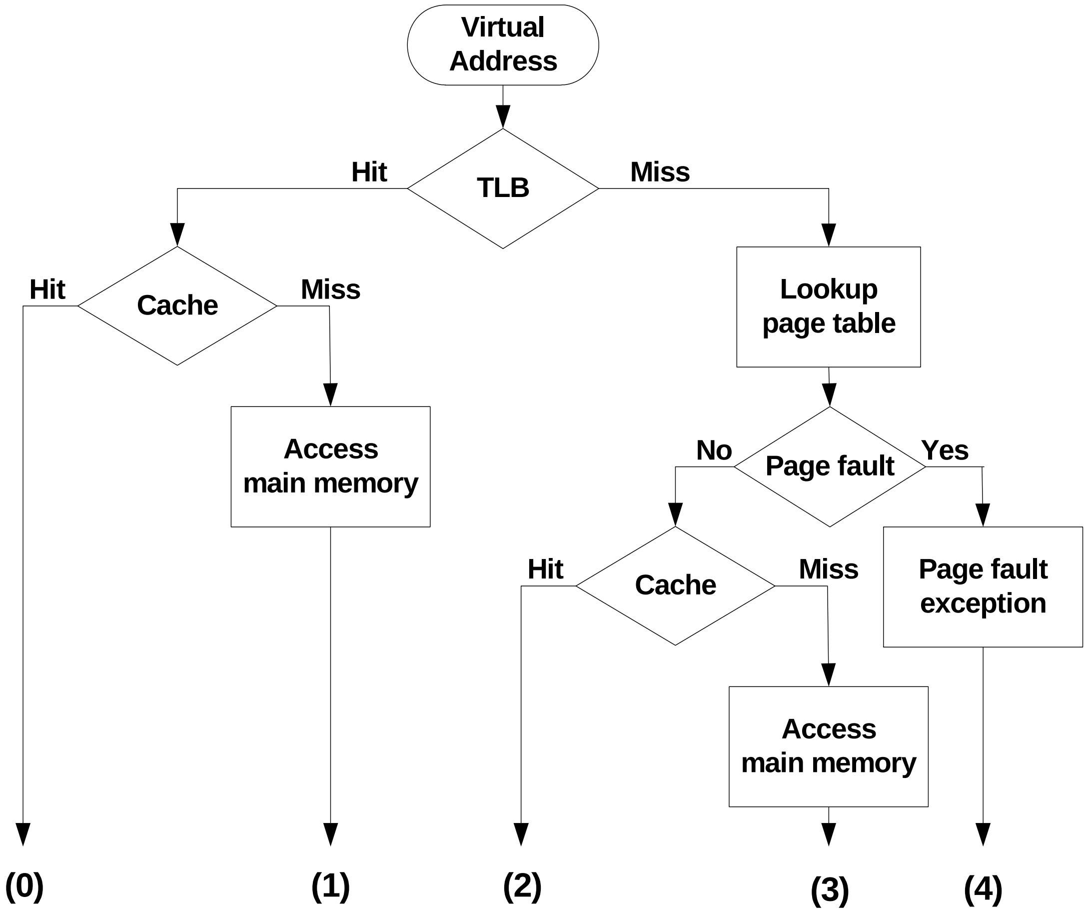


Figure 5.24

Integrating Virtual Memory, TLBs, and Caches



	cache	TLB	virtual memory	
(0)	hit	hit	hit	Best case
(1)	miss	hit	hit	Possible, no page table access
(2)	hit	miss	hit	TLB miss
(3)	miss	miss	hit	TLB miss and cache miss
(4)	miss	miss	miss	Page fault
	miss	hit	miss	Impossible
	hit	hit	miss	Impossible
	hit	miss	miss	Impossible

Figure 5.32

ARM Cortex-A8 and Intel Core i7 920

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

Figure 5.43

Elaboration

- **Physically indexed and physically tagged cache**
  - Cache hit time = TLB access time + cache access time
- **Virtually indexed and virtually tagged cache**

  (= Virtually addressed cache = Virtual cache)
  - Address translation hardware is unused during normal cache access.
  - Aliasing (or synonyms) problem
- **Virtually indexed and physically tagged cache**
  - Cache index $\subset$ page offset
  - Parallel accesses of cache and TLB

Implementing Protection with Virtual Memory

- **Protection and page table**
  - Mapping each process' virtual address space to disjoint physical pages
    - cannot access other process' data
  - Preventing user process from table modification
  - Page table: in the protected address space of OS
- **Write protection bit**
  - Included in each page table entry
  - Checked on every memory access

Hardware/Software Interface

- **Hardware's 3 basic capabilities for the OS to implement protection**
  - (1) 2 modes
    - user process
    - OS process (= kernel, supervisor or executive process)
  - (2) A CPU state that a user process can read but not write
  - (3) A mechanisms whereby the CPU can go from user mode to supervisor mode, and vice versa
    - system call exception (`syscall` instruction)
    - **ERET** (return from exception) instruction

Elaboration

- **Context switch or process switch from P1 to P2**
  - It must ensure that P2 cannot get access to the page tables of P1.
  - Without TLB, it suffices to change the page table register.
  - With TLB, TLB entries of P1 must be cleaned.
- **Process identifier (PID) or task identifier**
  - ID of currently running process
  - Concatenated to the tag portion of the TLB
  - Address Space ID (ASID)
    - MIPS, Intrinsity FastMATH, ARM, IA-64, SPARC (ASI)
- **Similar problems in a virtual cache**
  - Various solution such as PID

Handling TLB Misses and Page Faults

- **TLB miss or page fault**
  - Handled with exception mechanism
- **TLB misses**
  1) resident page → create the missing TLB entry
  2) non-resident page → page fault exception
- **If resident page,**
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- **If non-resident page, (page fault)**
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

Handling Page Faults

Find out the virtual address that caused the page fault.

1. Look up the page table to find out the location on disk.
2. Choose a victim. If it is dirty, write back to disk.
3. Read the page from disk.
   - Processor executes another process.
4. OS restores the state of the process.
5. Execute **ERET** (return from exception) instruction.
   - Restore PC.
   - Reset processor from kernel to user mode.
6. Reexecute the instruction that faulted.
   - Access the requested page.

Summary

- **Techniques for reducing page fault rate**
  (1) large page → exploiting spatial locality
  (2) fully associative mapping
  (3) LRU and a reference bit
- **Techniques for reducing disk writes**
  - write-back and dirty-bit
- **Memory protection**
  - restricting page table access
- **TLB**
  - reducing address translation time

5.8 A Common Framework for Memory Hierarchies

- Key design parameters for 4 major memory hierarchies

Features	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Size (blocks)	250-2,000	2,500-50,000	16,000-250,000	40-1024
Size (KiB)	16-64	125-2,000	1,000,000-1,000,000,000	0.25-16
Block size (B)	16-64	64-128	4,000-64,000	4-32
Miss penalty (clocks)	10-25	100-1000	10,000,000-100,000,000	10-1000
Miss rates	2-5%	0.1-2%	10^{-5}-10^{-4}%	0.01-2%

Figure 5.35

The Big Picture

Question 1 : Where can a block be placed?
Answer 1 : direct mapped, set associative, fully associative

Question 2 : How is a block found?
Answer 2 : indexing, limited search, full search,
separate lookup table

Question 3 : What block is replaced on a miss?
Answer 3 : LRU, random

Question 4 : How are writes handled?
Answer 4 : write through, write back

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

- **Classifications of misses**
  1. Compulsory misses (= cold start misses)
     - The first access to a block
  2. Capacity misses
     - Cache cannot contain all the blocks needed by the process
     - When blocks are replaced and then later retrieved
  3. Conflict misses (= collision misses)
     - When too many blocks map to a set

Miss Rates and 3Cs

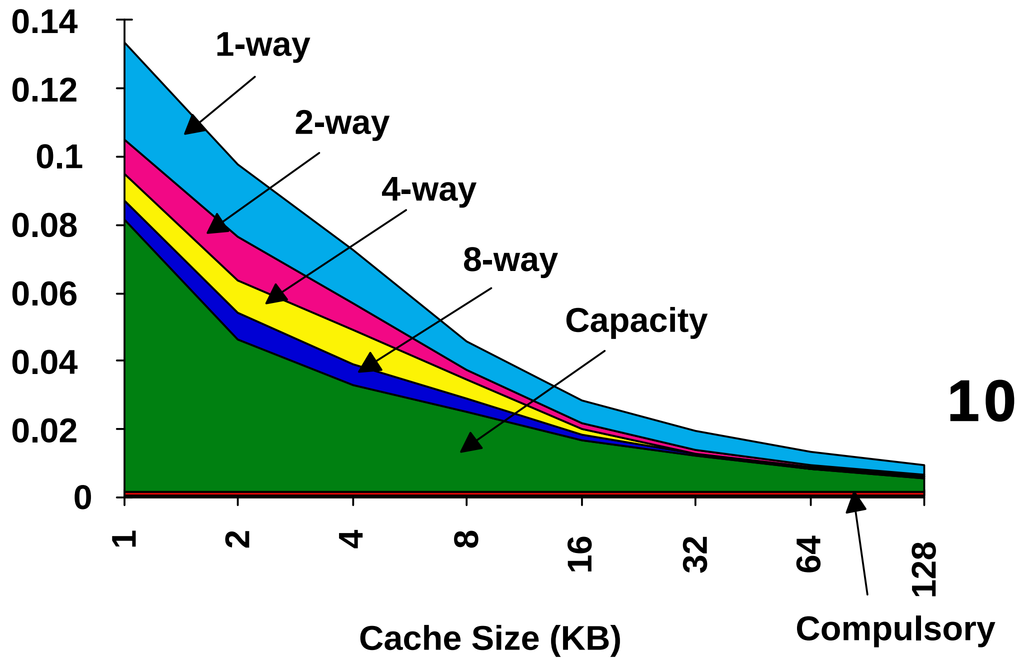
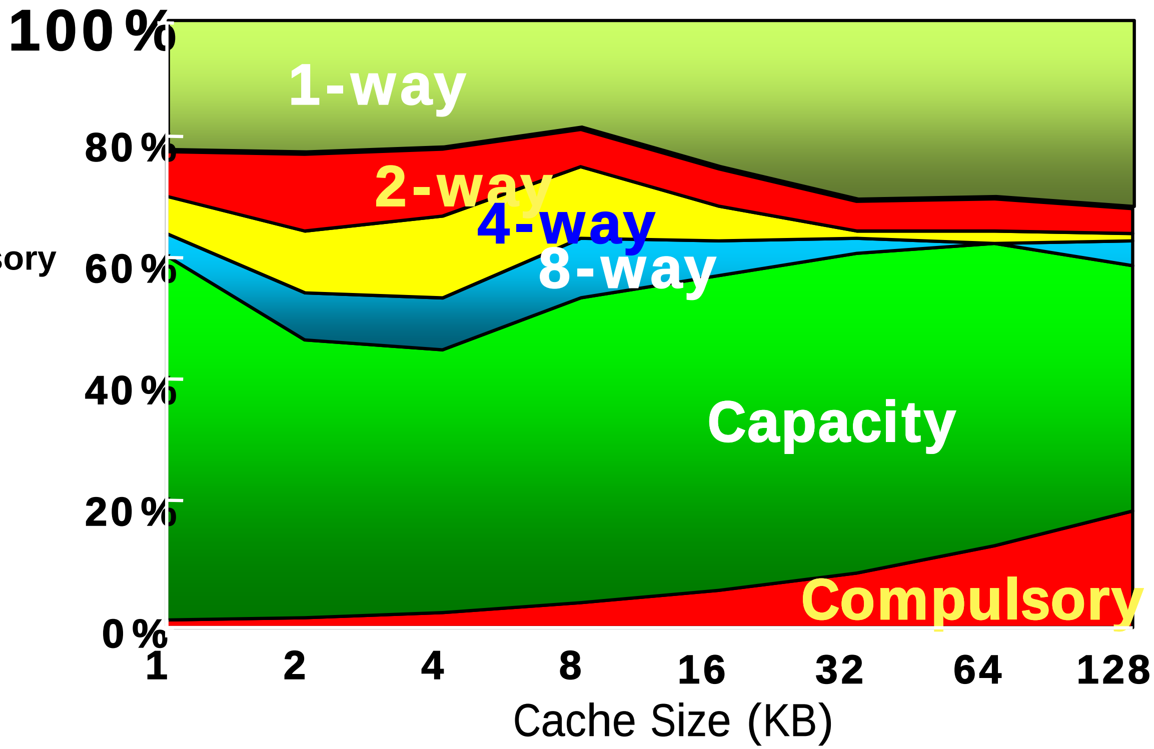


Figure 5.37



Reducing Misses

1. Conflict misses

- fully associative placement
- with increased access time

2. Capacity misses

- large cache
- with increased access time

3. Compulsory misses

- increased block size
- with increased miss penalty

5.10 Parallelism and Memory Hierarchy: Cache Coherence

- **Cache coherence problem**

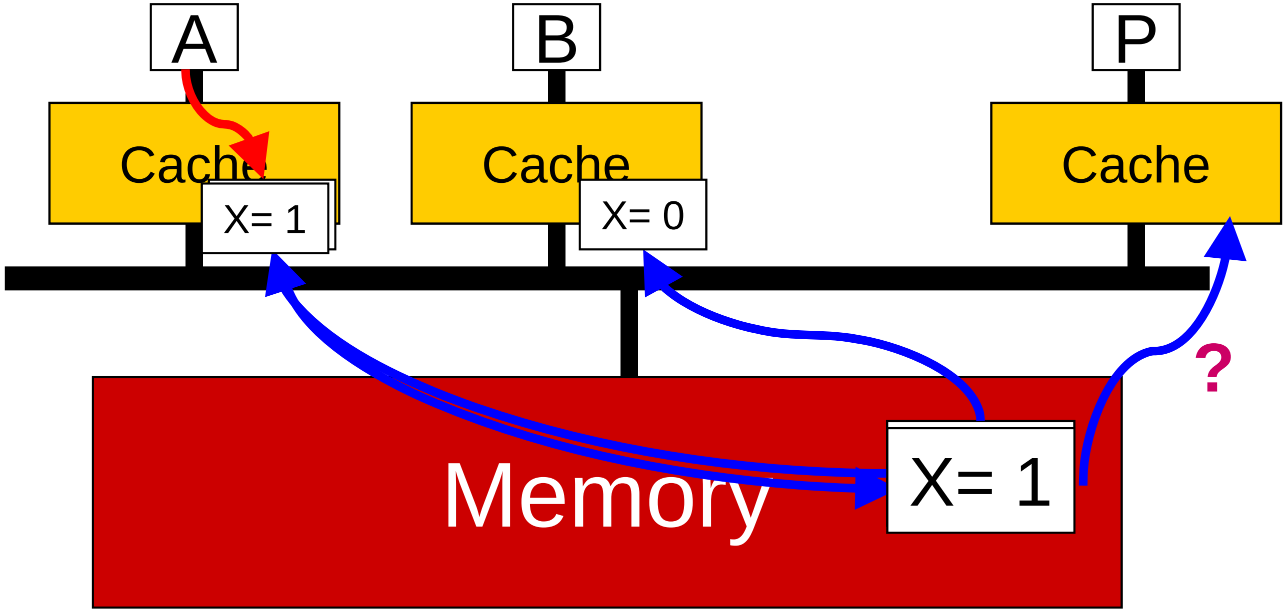


Figure 5.35

Time	**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

Cache Coherence Protocols

- **Operations performed by caches in multiprocessors to ensure coherence**
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access

1. **Snooping protocols**
   - Each cache monitors bus reads/writes
2. **Directory-based protocols**
   - Caches and memory record sharing status of blocks in a directory

Supplement

5.6 Virtual Machines

- Software implementation of a machine that executes programs like a physical machine (cf) Wikipedia
  1. System virtual machine (= hardware virtual machine)
  2. Process virtual machines (= application virtual machine)
- **(Operating) System Virtual Machine**
  - Presents the illusion that the users have an entire computer to themselves, including a copy of the operating system
  - Multiple OSes all share the hardware resources
- **Virtual machine monitor (VMM) (= hypervisor)**
  - The software that supports VMs
  - Maps virtual resources to physical resources
  - Guest code runs on native machine in user mode
    - Traps to VMM on privileged instructions and access to protected resources

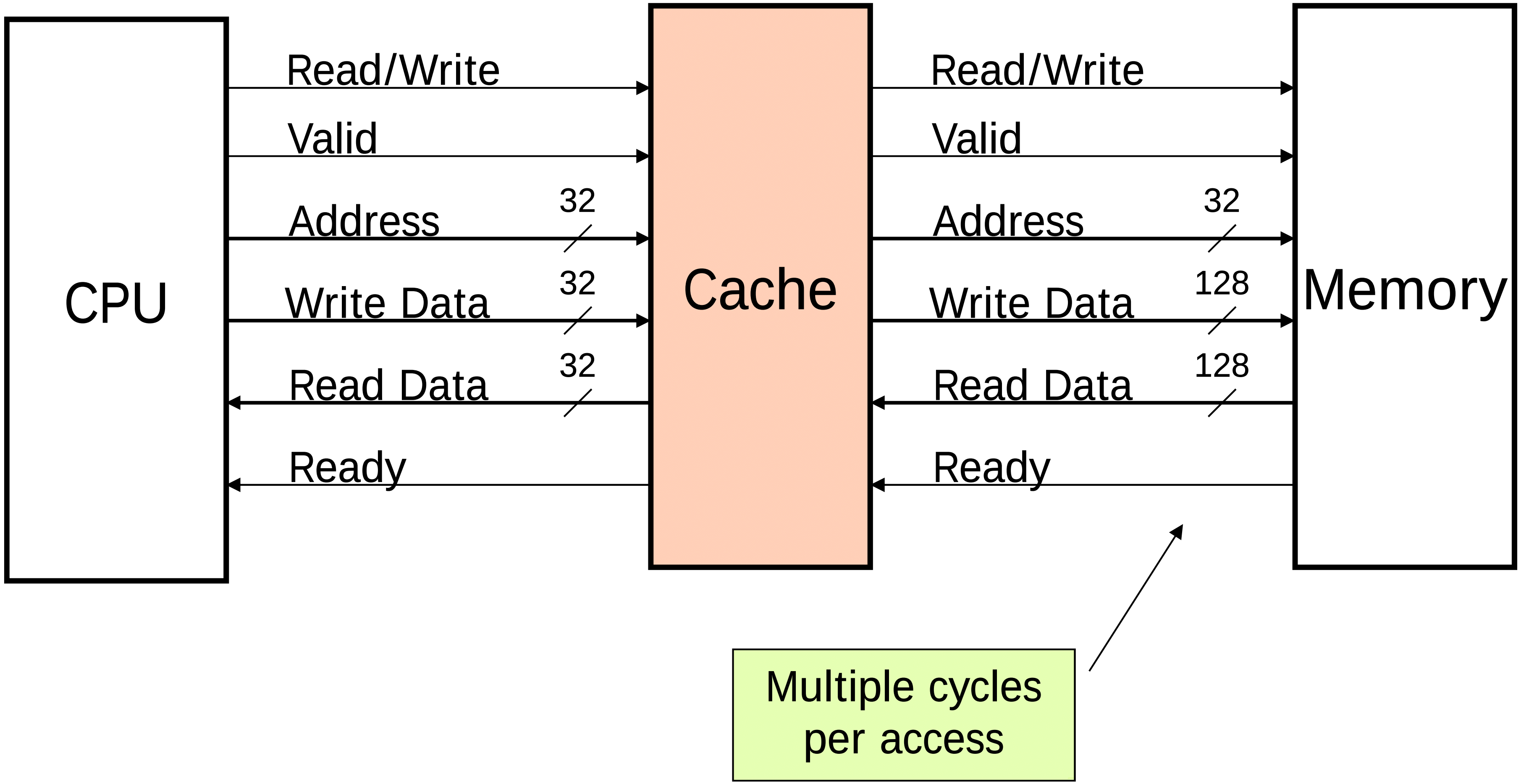
5.9 Using a Finite-State Machine to Control a Simple Cache

A Simple Cache

- Direct-mapped, write-back, write allocate
- Block size: 4 words (16 bytes)
- Cache size: 16 KB (1024 blocks)
- 32-bit byte addresses
- Valid bit and dirty bit per block
- Address

31 10	9 4	3 0
Tag	Index	Offset
18 bits	10 bits	4 bits

Signals between the Processor to the Cache



Finite-State Machines

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
- Next state
  = f_n (current state, current inputs)
- Control output signals
  = f_o (current state)

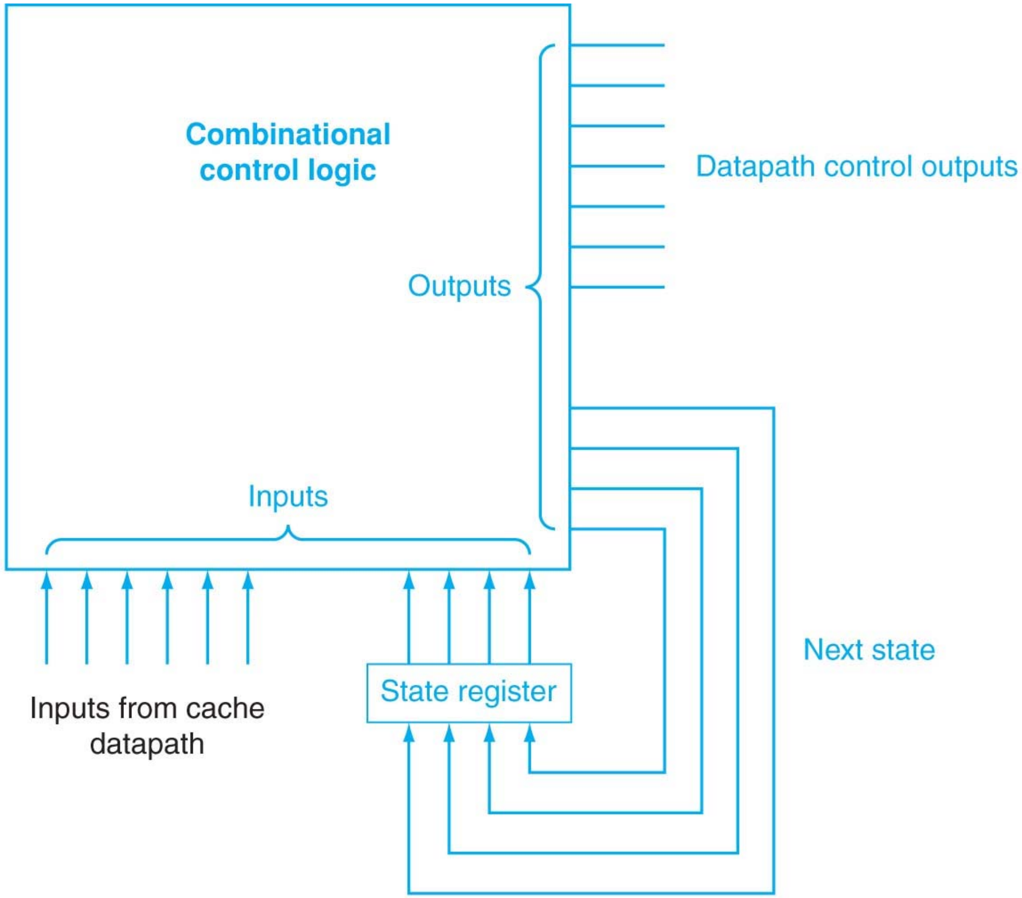


Figure 5.39

FSM for a Simple Cache Controller

- **Idle state**
  - Waiting for a valid read or write request from the processor
- **Compare Tag state**
  - Testing if hit or miss
  - If hit, set Cache Ready after read or write => Idle state
  - If miss, updates the cache tag
    - If dirty => Write-Back state, else => Allocate state
- **Write-Back state**
  - Writing the 128-bit block to memory
  - Waiting for the Ready signal from memory => Allocate state
- **Allocate state**
  - Fetching new block is from memory

Cache Controller FSM

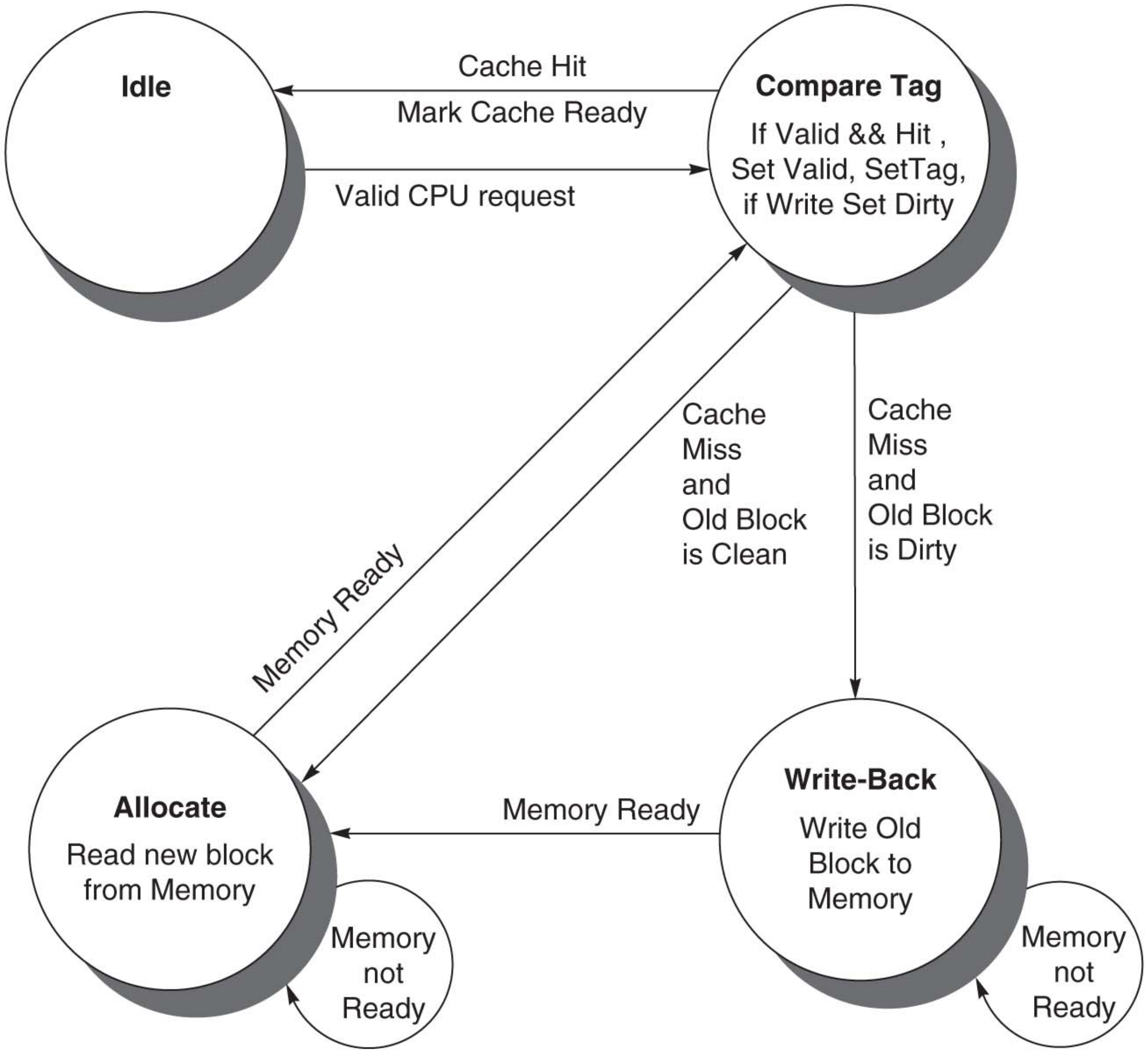


Figure 5.40