Appendix C

Memory Hierarchy

Slides: W. Gross, V. Hayward, T. Arbel

1

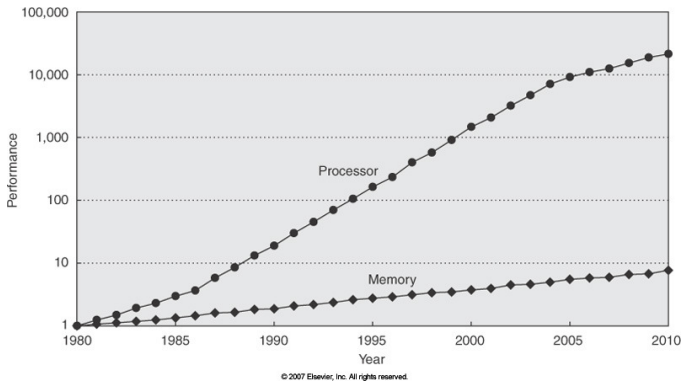
"Ideally one would desire an idefinitely large memory capacity such that any particular…word would be immediately available…We are…forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

A. W. Burks, H. H. Goldstine, and J. von Neumann, 1946

2

Introduction and Motivation

- Goal: unlimited amounts of fast memory.
- Figure 5.2



3

Principle of Locality

- Why does it work?
  - Principle of locality – nonuniform access
  - Smaller memories are faster
- Problem: fast memory is expensive !
- Solution: hierarchy of memory organized into different levels, each progressively larger, but slower

4

Memory Hierarchy

- Requirements: (apparently contradictory)
  - Provide virtually unlimited storage
  - Allow the CPU to work at register speed
- Analogy with personal belongings:
  - wear what you need for a day
  - suitcase for a week
  - house or apartment as permanent storage

Memory Hierarchy

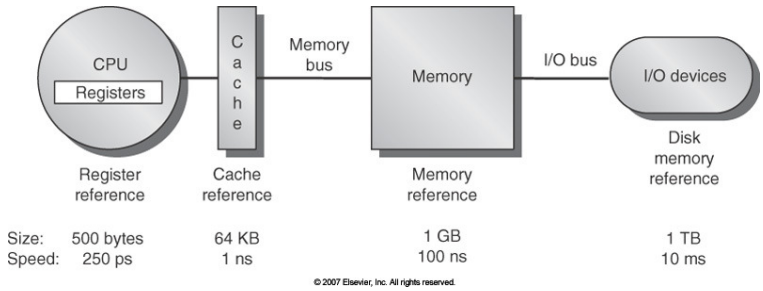


Figure 5.1

Memory Hierarchy in 2007

Level	1 registers	2 cache	3 main memory	4 disk storage
Typical size	< 1 KB	< 16 MB	< 512 GB	> 1 TB
Technology	Multi-port custom CMOS memory	On-chip/off-chipd CMOS SRAM	CMOS DRAM	Magnetic disk
Access time (ns)	0.25 – 0.5	0.5 - 25	50-250	5,000,000
Bandwidth (Mb/s)	50,000-500,000	5000 - 20,000	2500 - 10,000	50 - 500
Managed by	compiler	hardware	OS	OS/people
Next level	cache	main memory	disk	CD/tape

7

ABCs of Caches

- Cache is a buffer between CPU registers and main memory
  - Divided into blocks to take advantage of temporal and spatial locality in programs
  - Similarly, virtual memory divides a large address space in pages stored in main memory
- Principle: keep in higher level storage a copy of a subset of the next lower level.
  - E.g. registers hold a copy of subset of blocks
  - Blocks are a copy of a subset of the main memory, etc…

8

ABCs…

- Every time an item is requested
  - Hit: if found in next lower level
  - Miss: if not found
- E.g.
  - load or fetch found in cache => cache hit
- E.g.
  - If cache loads a block and it is not found in main memory => page fault
  - Page is retrieved from the disk
- If a hit …deliver data item
- If a miss…fetch the correct block from the lower level of the hierarchy
  - CPU must stall

9

CPU Performance with Caches

$$\begin{aligned} \text{MemoryStallCycles} &= \text{NumberOfMisses} \times \text{MissPenalty} \\ &= \text{IC} \times \frac{\text{Misses}}{\text{Instructions}} \times \text{MissPenalty} \\ &= \text{IC} \times \frac{\text{MemoryAccesses}}{\text{Instruction}} \times \text{MissRate} \times \text{MissPenalty} \end{aligned}$$

10

Miss Rate

- The measurement problem becomes finding the miss rate
  - the average number of misses per access
- Address trace
  - A large sequential collection of addresses accessed by a benchmark and fed to a cache simulator

11

Reads / Writes

If we refine the previous formula to distinguish reads and writes, we have:

$$\begin{aligned}\text{MemoryStallCycles} &= \text{IC} \times \frac{\text{MemoryAccesses}}{\text{Instruction}} \times \text{MissRate} \times \text{MissPenalty} \\ &= \text{IC} \times \text{ReadsPerInstruction} \times \text{ReadMissRate} \times \text{ReadMissPenalty} \\ &+ \text{IC} \times \text{WritesPerInstruction} \times \text{WritesMissRate} \times \text{WritesMissPenalty}\end{aligned}$$

12

Example.

Assume that a machine has a CPI of 1.0 when all memory accesses are hits in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 25 clock cycles, and the miss rate is 2%, how much faster would the machine be if all accesses were cache hits?

$$\text{CPUTime} = (\text{CPUClockCycles} + \text{MemoryStallCycles}) \times \text{CCTime}$$

In the perfect case:

$$\text{CPUTime}_{\text{perfect}} = (\text{IC} \times \text{CPI} + 0.0) \times \text{CCTime} = \text{IC} \times \text{CCTime}$$

With the real cache:

$$\text{MemoryStallCycles} = \text{IC} \times \frac{\text{MissRate} \times \text{MemoryAccesses}}{\text{Instruction}} \times \text{MissPenalty}$$

$$= \text{IC} \times 0.02 \times (1.0 + 0.5) \times 25 = \text{IC} \times 0.75$$

$$\text{CPUTime}_{\text{real}} = (\text{IC} + 0.75 \times \text{IC}) \times \text{CCTime} = \text{IC} \times 1.75 \times \text{CCTime}$$

The machine with no cache misses is 1.75 faster.

13

Four Memory Hierarchy Questions

- **Caches: Main memory is divided into blocks each consisting of several data elements (e.g. bytes)**

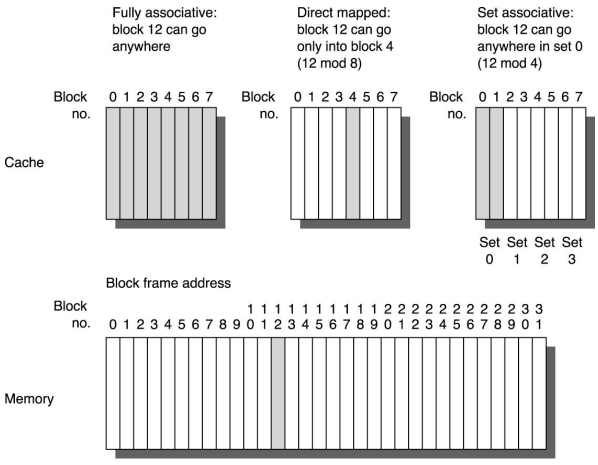
1. **Where can a block be placed in the upper level?**
   - **Block placement**
2. **How is a block found if it is in the upper level?**
   - **Block identification**
3. **Which block should be replaced on a miss?**
   - **Block replacement**
4. **What happens on a write?**
   - **Write strategy**

14

Q1: Block Placement

- Fully associative
  - A block can appear anywhere in the cache
- Direct mapped cache
  - Each block can only appear in one place in the cache
    - » *block address* mod *# blocks in cache*
- Set associative
  - A block can be placed in a restriced set of places in the cache
  - First, map a block onto a set
  - The block can be placed anywhere in that set
  - Set chosen by bit selection
    - » *Block address* mod *# sets in cache*
  - n blocks in a set => n-way set associative

Block Placement



Associative Caches

- In general, m blocks in cache, n blocks in a set, s sets in cache

  m = s * n

- n-way set associative => n > 1 and s > 1
- fully associative => n = m and s = 1 (m-way s.a.)
- direct mapped => n = 1 and s = m (1-way s.a.)

- Mapping: set number is called "index"
- *index* = *block #* mod *s*

17

Real Caches

- Most processor caches today are
  - Direct mapped, or
  - 2-way set associative, or
  - 4-way set associative

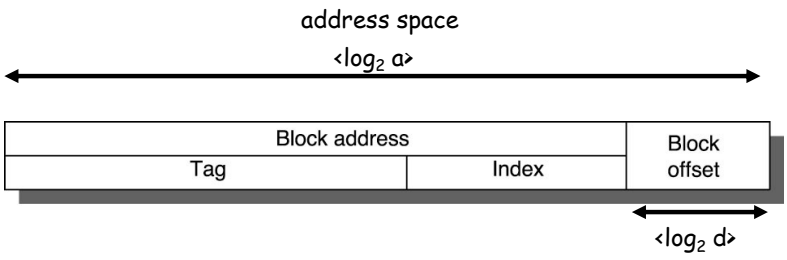
18

Q2: Block Identification

- How is a block found if it is in the cache ?
- Why is this an issue?
  - Many blocks in main memory map to one or few blocks in cache
- A complete address in the address space of *a* addresses can be divided into fields
  - Each field has a significance in the hierarchy
- Each block contains d data items (bytes, words…). So a field in the address called the "block offset" (lower-order bits) indicates which of the d data items in a particular block we want to access
- We then also need a field "block address" to indicate which block number in memory we will access (higher-order bits).

19

Address Fields



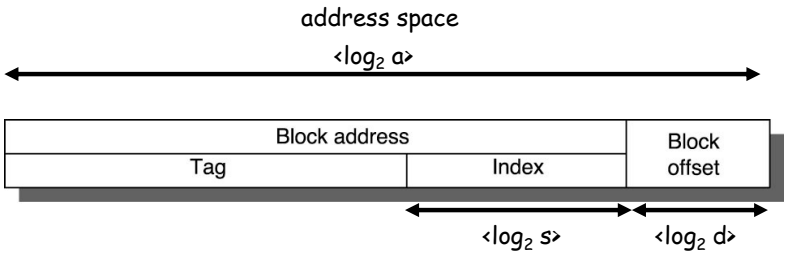
20

Search

- Now that we know which block to look for, we need to identify which one of the *s* sets the block could be in (*s* is called the "index")
- Now, the block could be anywhere in this set, so we need to do a search within this set to identify the particular block
  - Search for the "tag"

21

Address Fields

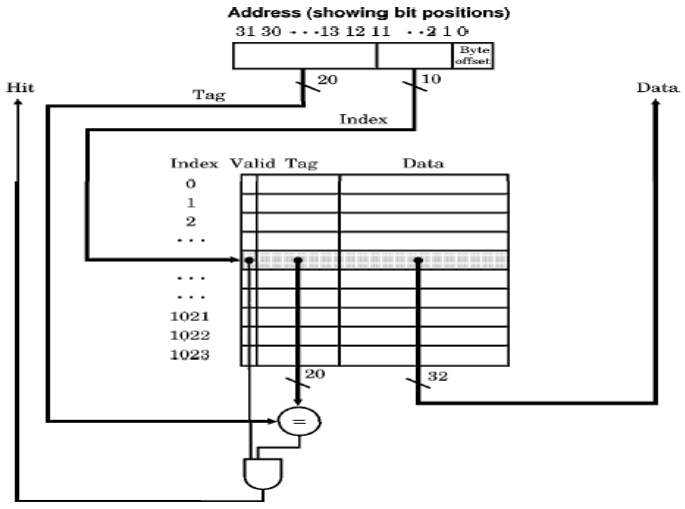


22

Searching…

- Direct mapped
  - No search
  - Index directly provides block number in cache
- n-way set associative
  - smaller index and a larger tag as n gets larger
- fully associative
  - no index
- Tag check search is done in parallel for speed – if the tag is found => a hit !
- How do we know if the cache block has been loaded with valid data ?
  - Add a "valid bit" to every block in the cache

Direct Mapped Cache



Q3: Block Replaement

- Which block should be replaced on a miss?
- Direct mapped
  - no choice !
- Fully associative or set associative
  - Many blocks to choose to replace

25

3 Replacement Strategies

1. Random
   - Simple
2. Least-recently used (LRU)
   - If recently used blocks are likely to be used again (locality) then a good candidate for disposal is the LRU block
   - Record accesses to blocks
3. First in, first out (FIFO)
   - Replace oldest block
   - Approximation to LRU

26

Q4: Write Strategy

- Most cache accesses are reads
  - Instruction accesses are reads
  - MIPS: 10% stores, 37% loads
  - Writes 7% of overall memory traffic
  - Data cache only: writes are 21%

27

Reads / Writes

- Make common case fast
  - Optimize caches for reads
  - Reads are the easiest to make fast
  - Read block at same time as tag check
  - Only drawback to reading a block immediately is power
  - Ahmdahl's law: don't neglect speed of writes
- Why writes so slow?
  - Writes cannot begin until after tag check
  - Writes can also be of variable width (as can reads, but there is no harm in reading more, except power)

28

2 Write Strategies

1. **Write through**
   - Write information to the block in the cache and to the block in lower-level memory
2. **Write back**
   - Write information only to block in the cache. Write the modified cache block to the main memory only when it is replaced

29

Write Back

- "Dirty bit" kept for each block
  - Indicates whether the block is "dirty" (modified while in the cache) or "clean" (not modified)
  - Reduces frequency of write back
- Advantages:
  - writes occur at speed of cache
  - Multiple writes to a block require only one write to main memory
  - Some writes never make it to main memory
    - » Memory bandwidth reduced
    - » Lower power

30

Write Through

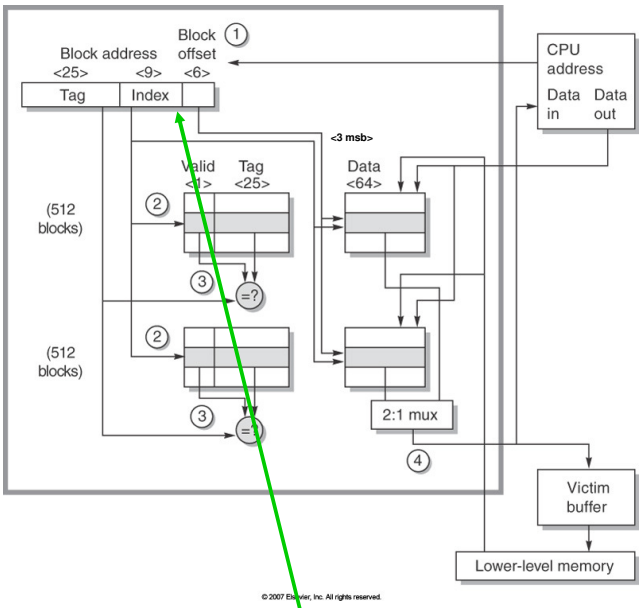
- Advantages:
  - Simple, easier to implement
- Cache is always clean
  - Read misses never result in writes to lower level
- Write stalls
  - CPU must wait for writes to complete
  - Use a write buffer

31

Write Misses

- Two strategies (used with both WB and WT)
- Write allocate
  - Make write misses act like read misses
  - Allocate the block in the cache then follow same steps as a write hit
- No-write allocate
  - A little weird- write misses do not write the cache
  - Block is modified only in lower-level memory
  - Blocks stay out of the cache until the program tries to read the blocks
- Normally:
  - Write back caches use write allocate (benefit from locality)
  - Write through caches use "no write allocate" (avoid redundant writes)

32



Example: AMD Opteron Data Cache

- 64 Kbytes
- 64-byte blocks
- 2-way set associative
- LRU
- Write back
- Write-allocate on a write miss

2^{index} = cache size / (block size * associativity) = # blocks / associativity

33

Unified vs. Split Caches

- Instruction and data streams are different
  - Instructions are fixed size, read-only, very local (except for branches)
- Two caches: one optimized for instructions and one for data
  - Drawback: Fixed capacity is split between the two caches
- Split cache:

$$\begin{aligned} \text{MemoryStallCycles} &= \text{NumberOfMisses} \times \text{MissPenalty} \\ &= \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{MissPenalty} \\ &= \text{IC} \times \left(\frac{\text{FetchMisses}}{\text{Instruction}} \times \text{FetchMissPenalty} \right. \\ &\quad \left. + \frac{\text{DataMisses}}{\text{Instruction}} \times \text{DataMissPenalty} \right) \end{aligned}$$

34

Cache Performance

- Miss rate is independent of the speed of the hardware
- Better measure of mem. Hierarchy performance is Average Memory Access Time

AMAT = HitTime + MissRate x MissPenalty

35

AMAT and CPU Performance

CPUTime = IC x CPI x CCtime

- Assumes perfect memory hierarchy performance (all accesses were hits)
- Pipeline: assumes all memory accesses complete within one clock cycle
  - Hit time < CCtime
- How to account for real memory hierarchy?

36

CPU Performance with Imperfect Caches

$$\text{CPUTime} = \text{IC} \times \left(\text{CPI}_{\text{base}} + \frac{\text{MemoryStalls}}{\text{Instruction}} \right) \times \text{CCTime}$$

$$= \text{IC} \times \left(\text{CPI}_{\text{base}} + \frac{\text{MemoryAccesses}}{\text{Instruction}} \times \text{MissRate} \times \text{MissPenalty} \right) \times \text{CCTime}$$

37

Example

- In-order execution processor (e.g Ultra SPARC III)
- Cache miss penalty = 200 cc
- Instructions: 1.0 cc (ignoring mem. stalls)
- Miss rate = 2%
- 1.5 mem ref/instructions on average
- Cache misses → 30 misses / 1000 instr.

- Q: what is the impact on performance when the cache is considered?

38

Example

$$\text{CPUTime} = \text{IC} \times \left(\text{CPI}_{\text{base}} + \frac{\text{MemoryStalls}}{\text{Instruction}} \right) \times \text{CCTime}$$

$$= \text{IC} \times \left(\text{CPI}_{\text{base}} + \frac{\text{MemoryAccesses}}{\text{Instruction}} \times \text{MissRate} \times \text{MissPenalty} \right) \times \text{CCTime}$$

$\text{CPUTime}_{\text{cache}}$ = IC x (1.0+(30/1000) x 200) x CC
= IC x 7.00 x CC

- or use miss rate

$\text{CPUTime}_{\text{cache}}$ = IC x (1.0+(1.5 x 2% x 200)) x CC
= IC x 7.00 x CC

Example

- Compare to machine with no memory hierarchy:
- CPI would be 1.0 + 200 X 1.5 = 301
- More than 40 times longer execution time than with a cache !

Cache Impact on Performance

- With low CPI (< 1) the relative impact of a cache miss is higher
- With faster clocks, a fixed memory delay yields more stall clock cycles

41

AMAT is not CPUtime !

- Example: 2 cache designs fitted to the same CPU CPI = 1.6 with perfect cache
  - Cycle time = 0.35 ns
  - 1.4 mem refs / instruction
- Both caches are 128 KB and have block sizes of 64 bytes, miss penalty = 65 ns, hit time = 1 cc

1. Direct mapped: mr = 2.1%
2. 2-way set assoc.: CC stetched by factor of 1.35, mr = 1.9%

42

$AMAT_{1\text{-way}} = 0.35 + (0.21 \times 65) = 1.72$ ns

$AMAT_{2\text{-way}} = 0.35 * 1.35 + (0.019 * 65) = 1.71$ ns

CPU time = IC * (CPI_{exec} + misses/inst * mp) * CC
= IC * ((CPI_{exec} * CC) + (mr * mem access/inst * mp * CC))

$CPU\ time_{1\text{-way}} = IC * (1.6 * 0.35 + (0.021 * 1.4 * 65)) = 2.47 * IC$

$CPU\ time_{2\text{-way}} = IC * (1.6 * 0.35 * 1.35 + (0.019 * 1.4 * 65)) = 2.49 * IC$

- **AMAT is lower for 2-way**
- **CPU time is lower for 1-way !**
- **Why?**
- **2-way stretches clock cycle for ALL instructions even though there are fewer misses!**
  - **build the 1-way in this case**

43

Out-of-Order Execution

- **Miss penalty does not mean the same thing**
  - **The machine does not totally stall on a cache miss**
  - **Other instructions allowed to proceed**
- **What is MP for O-O-E?**
  - **Full latency of the memory miss?**
  - **The nonoverlapped latency when the CPU must stall ?**
- **Define:**

$$\frac{\text{MemoryStallCycles}}{\text{Instruction}} = \frac{\text{Misses}}{\text{Instruction}} \times (\text{TotalMissLatency} - \text{OverlappedMissLatency})$$

44

Performance Summary

- **Performance equations (12 of them!)**

$$2^{index} = \frac{Cache\ size}{Block\ size \times Set\ associativity}$$

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

$$Memory\ stall\ cycles = Number\ of\ misses \times Miss\ penalty$$

$$Memory\ stall\ cycles = IC \times \frac{Misses}{Instruction} \times Miss\ penalty$$

$$\frac{Misses}{Instruction} = Miss\ rate \times \frac{Memory\ accesses}{Instruction}$$

$$Avarage\ memory\ access\ time - Ilit\ time + Miss\ rate \times Miss\ penalty$$

$$CPU\ execution\ time = IC \times \left(CPI_{execution} + \frac{Memory\ stall\ clock\ cycles}{Instruction}\right) \times Clock\ cycle\ time$$

$$CPU\ execution\ time - IC \times \left(CPI_{execution} + \frac{Misses}{Instruction} \times Miss\ penalty\right) \times Clock\ cycle\ time$$

45

Performance Summary

- **Performance equations (12 of them!)**

$$CPU\ execution\ time = IC \times \left(CPI_{execution} + Miss\ rate \times \frac{Memory\ accesses}{Instruction} \times Miss\ penalty\right) \times Clock\ cycle\ time$$

$$\frac{Memory\ stall\ cycles}{Instruction} = \frac{Misses}{Instruction} \times (Total\ miss\ latency - Overlapped\ miss\ latency)$$

$$AMAT = Hit\ time_{L1} + Miss\ rate_{L1} \times (Hit\ time_{L2} + Miss\ rate_{L2} + Miss\ penalty_{L2})$$

$$\frac{Memory\ stall\ cycles}{Instruction} = \frac{Misses_{L1}}{Instruction} \times Hit\ time_{L2} + \frac{Misses_{L2}}{Instruction} \times Miss\ penalty_{L2}$$

46

Cache Optimizations

- AMAT = HitTime + MissRate x MissPenalty
- Cache optimizations will be studied next focusing on reducing hit time, miss rate and miss penalty (analogy to CPU performance equation)
- Miss rate reduction: larger block size, larger cache size, higher associativity,
- Miss Penalty reduction: multilevel caches, critical word first, read priority over writes
- Hit time reduction: cache size and organization, avoiding address translation when indexing the cache

47

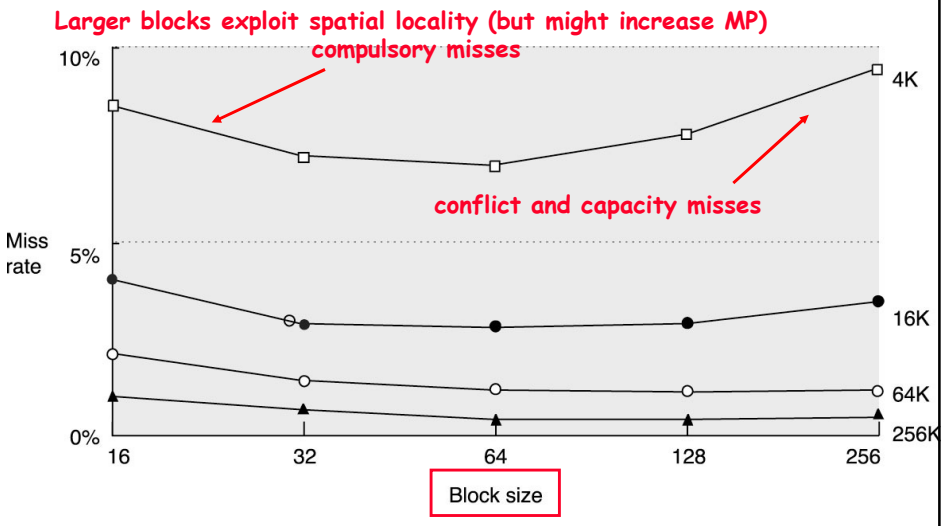
Miss Rate Reduction

- 3 kinds of misses

1. Compulsory – first access to a block cannot be in cache (cold start miss or first-reference miss)
2. Capacity – cache cannot contain all blocks needed during execution of a program because blocks are discarded and later retrieved
3. Conflict – for set-associative or direct mapped – if too many blocks map to a set (collision misses)
   - Hits in FA cache become misses in an n-way SA cache if there are more than n requests to a popular set

48

Optimization #1: Larger Block Sizes to Reduce Miss Rate



Optimization #2: Larger Caches to Reduce Miss Rate

- Cost tradeoff
- Higher hit times
  - Off-chip caches

50

Optimization #3: Higher Associativity to Reduce Miss Rate

- Rule of thumb: Direct mapped cache of size N has about the same miss rate as a 2-way set associative cache of size N/2
- Higher hit times (search)

51

Other techniques to reduce miss rate

- Way prediction: use extra bits to predict the next block in the set that will be accessed (similar to branch prediction)
- Pseudoassociativity – cheap form of associativity for direct mapped caches.
  - On a miss a second entry is checked, say by inverting index bits ('pseudo-set')
  - One fast hit and one slower one

52

Optimization #4: Reducing Miss Penalty – Multilevel Caches

- Reduce penalty by adding a cache to the cache !
- Penalty at a given level is determined by the AMAT of the next lower level
- E.g. penalty to replace a register is the load delay

$$AMAT_i = HitTime_i + MissRate_i \times MissPenalty_i$$
$$MissPenalty_i = AMAT_{i+1}$$

SO

$$AMAT_{L1} = HitTime_{L1} + MissRate_{L1} \times (HitTime_{L2} + MissRate_{L2} \times MissPenalty_{L2})$$
$$= \underbrace{HitTime_{L1} + MissRate_{L1} \times HitTime_{L2}}_{\text{New hit time}} + \underbrace{MissRate_{L1} \times MissRate_{L2}}_{\text{New miss rate}} \times MissPenalty_{L2}$$

53

Multilevel Caches

- For small miss rates:
- Small increase in hit time
- Huge reduction in miss penalty because misses that go all the way to memory (global misses) are rare (mr1 x mr2)
- L1 should be fast for a small hit time
- L1 usually on-chip to reduce interconnect delays
- L2 should be bigger (typically off-chip) and should have a more sophisticated organization to reduce miss rate

54

Multilevel Exclusion

- What if a design cannot afford a L2 cache that is much larger than L1
  - Wastes most of L2 with redundant copies of what is in L1?
- In this case use multilevel exclusion
  - L1 data is never found in L2 cache
  - L1 miss results in a swap instead of replacement
- AMD Opteron (2x64Kb L1 and 1 Mb L2)

55

Critical Word First and Early Restart

- Observation: CPU normally needs just one word of the block at a time
  - Impatience! Don't wait for whole block to be loaded before sending the requested word and restarting the CPU
- Critical word first – request the missed word from memory and send it CPU as soon as it arrives – CPU continues while rest of block fills in
- Early restart – fetch the words in normal order, but as soon as the requested word arrives, send it to the CPU and let it continue execution

56

Optimization #5: Priority of Read Misses Over Writes

- Serve reads before writes have been completed
- Write buffers for write-through caches cause RAW hazards
- E.g. direct-mapped, assume 512 and 1024 mapped to same block

```
SW R3, 512(R0)   ; R3 in write buffer
LW R1, 1024(R0) ; miss, replace block
LW R2, 512(R0)   ; miss
```

- If write buffer has not completed store then wrong value written to cache block and R2
- Soln: on read miss, check write buffer for conflicts and if memory system is available, let read miss continue

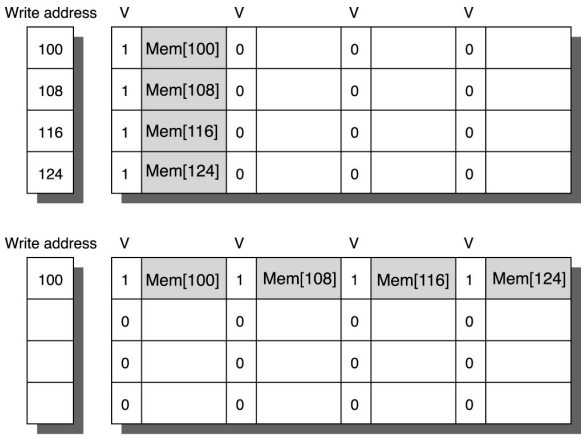
57

Merging Write Buffer

- When writing to the write buffer, check if address matches an existing write buffer entry
  - Combine data with that write – write merging
  - Multiword writes are faster than one word at a time

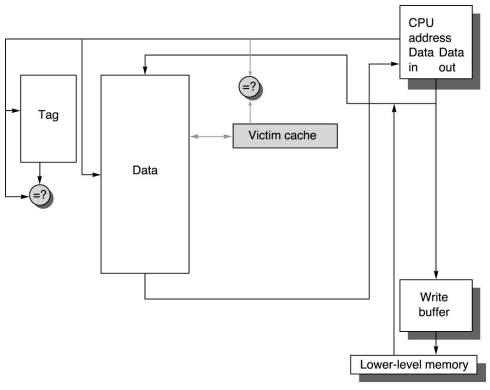
58

Merging Write Buffer



Victim Caches

- Remember what was discarded in case it is needed again
- Small fully associative cache between a cache and its refill path
- AMD Athlon – 8 entry victim cache



Optimization #6: Avoiding Address Translation during Indexing of the Cache to Reduce Hit Time

- We will come back to this after we study virtual memory…

61

Virtual Memory

- Original motivation: to increase the memory capacity of the computer beyond the size of the main memory
- The problem:
  - If a program became too large to fit into memory…
- The original solution (before VM):
  - The programmer was responsible for dividing the program up into mutually exclusive parts that would fit into main memory
  - The programmer was also responsible for making sure the correct part (overlay) was loaded in to the main memory at the proper time

62

Virtual Memory

- With "virtual memory", the disk is used as the lowest level in the memory hierarchy
- The address space is the range of memory addresses
  - usually much larger than the capacity of the main memory
  - E.g. 32-bit addresses => 2^{32} ~ 4 x 10^9 addresses (usually 4 Gigabytes capacity)

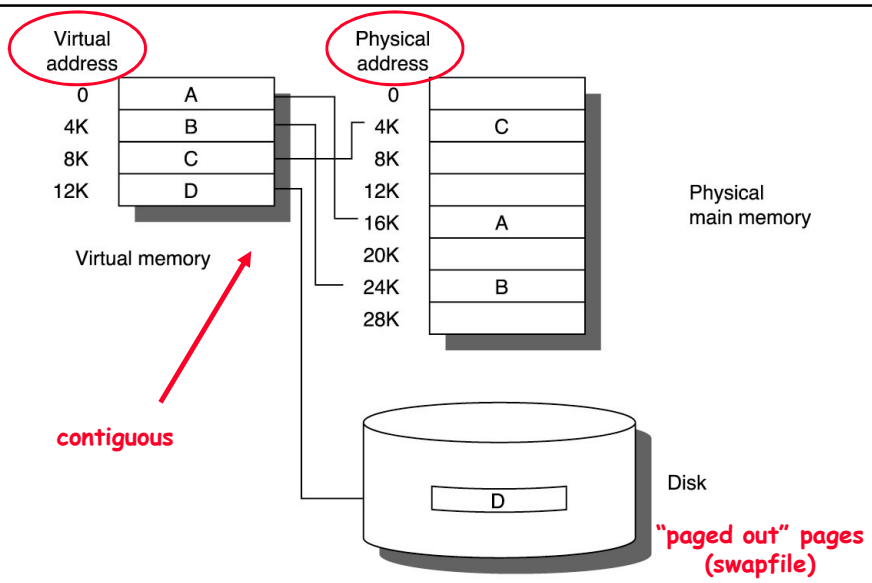
63

More Motivation for VM

- But there are other motivations for VM, which are just as important…

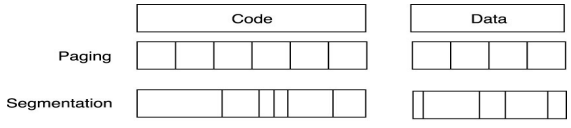
- Multitasking
  - Many processes (programs) are sharing the memory space
  - Each one thinks it has a contiguous chunk of memory – hide details from each process (# of processes,size of processes..)
  - Memory protection => don't let a process access another's memory
- Relocation
  - Allows a program to run anywhere in memory
  - maps the addresses generated by the compiler to the real address of the memory in the main memory or disk

64



Pages and Segments

- Chunks of memory are called pages or segments (instead of blocks in caches)
  - Pages are a fixed size (con: internal fragmentation)
  - Some machines use variable size pages called segments (hard to replace since you need to find contiguous, variable sized, free space)
- A reference to a page that is on the disk => page fault



4 Questions

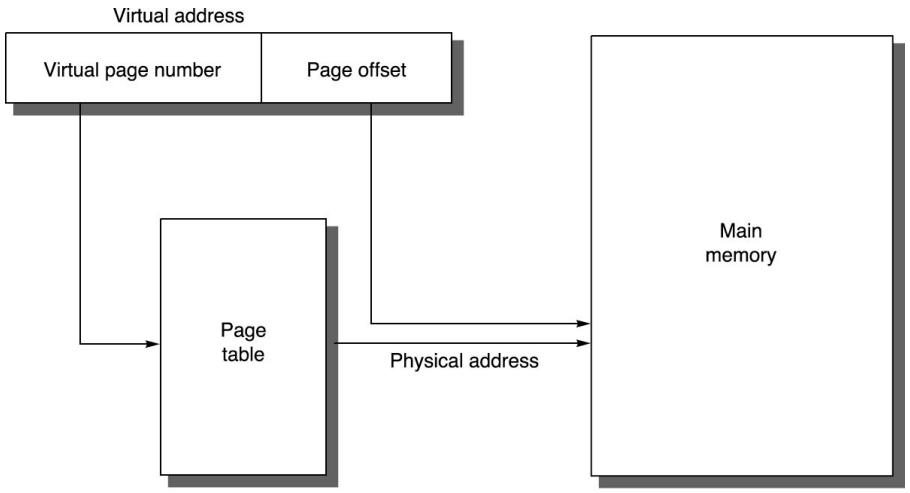
- **Block placement**
  - Miss penalty is huge ! (1,000,000 to 10,000,000 cycles)
  - Go for lower miss rate at expense of more complex algorithm (O/S)
  - VM uses a fully associative strategy (pages can be placed anywhere in main memory)
- **Block Replacement**
  - Minimize page faults !
  - LRU replacement (set "use bit" when a page is accessed. O/S keeps track of them)

67

4 Questions

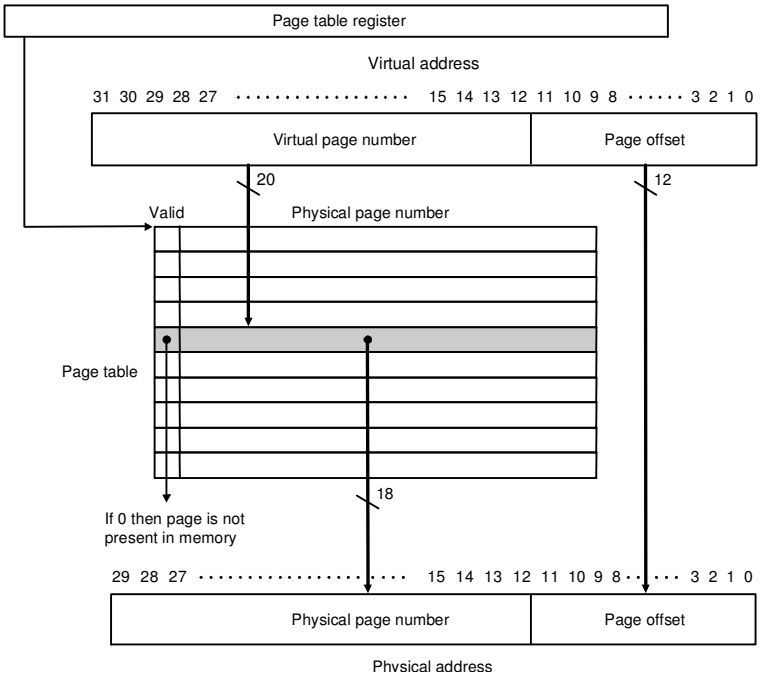
- **Write strategy**
  - Write back (avoid writing to disk whenever possible)
- **Block ID**
  - Need to translate (map) the virtual address on-the-fly to a physical address
  - Store the mapping in a page table (maintained by O/S)

68



Page Tables

- Index with the virtual page #
- The page table stores the corresponding physical address
- Each process gets a page table
- Page tables are stored in main memory (and can therefore be in the cache)
- Page tables can be large



71

Page Table Example

- 32-bit virtual address
- 4 KB pages
- 4 bytes per page table entry

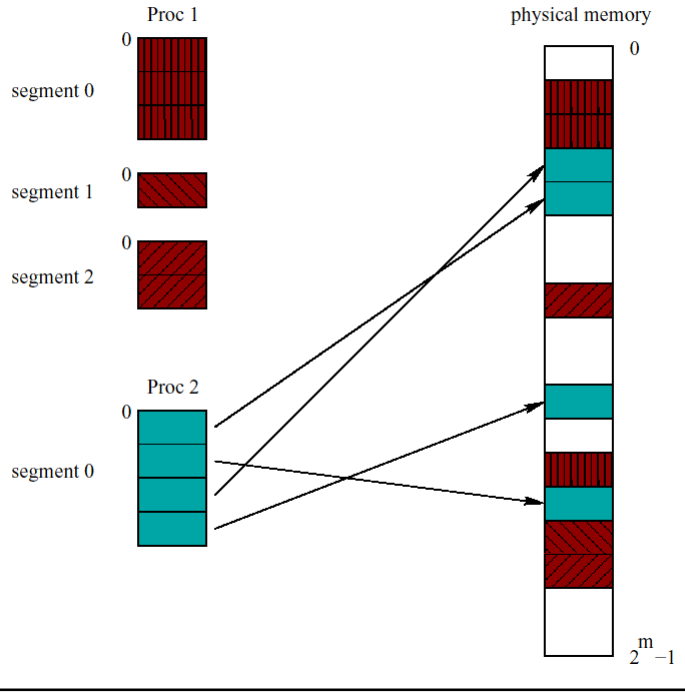
- The page table would be $2^{32}/2^{12} \times 2^2 = 2^{22}$ bytes (4 MB)

72

Combining Segmentation with Paging

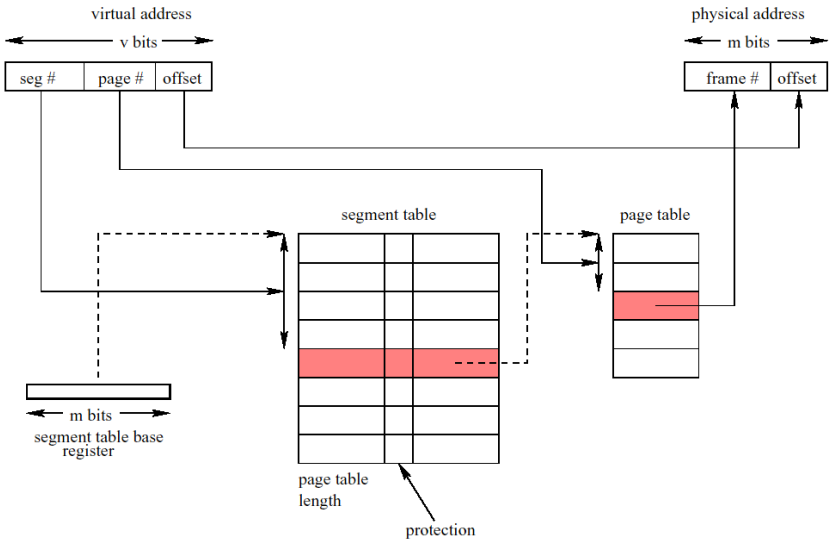
- Processes are often divided into several spaces for code, data, stack
- Segments can be used to dynamically adjust each process' memory usage
- Pure segmentation is not often used, but segments can be divided into multiples of pages

73



Courtesy: UW CS350

74



Courtesy: UW CS350

75

Fast Address Translation

- Paging means that every request results in two memory accesses
  - One to read the page table to get the physical address
  - One to get the data
- This is very costly. Especially wasteful since locality tells us that consecutive accesses are likely to be on the same page
- Why redo the address translation every time?
- Solution => cache the most recent translations !
  - notice how in a small number of architectural techniques keep coming up in this course…e.g. caching, using the past to predict the future, pipelining, parallelism…

76

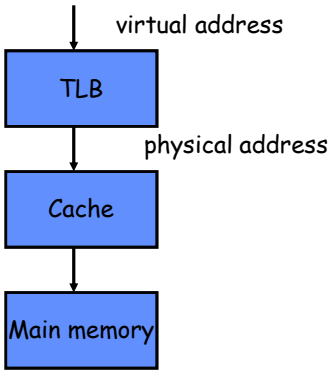
Translation Lookaside Buffers (TLB)

- The name of the special cache used to remember the most recent address translation is the translation lookaside buffer
- Just like a cache – tag is portion of virtual address and data is the physical page number (along with a field used for protection, valid bit, use bit, and dirty bit for the memory page)

77

TLB Placement

- The TLB is usually inserted between the CPU which supplies virtual addresses and the memory system which requires physical addresses

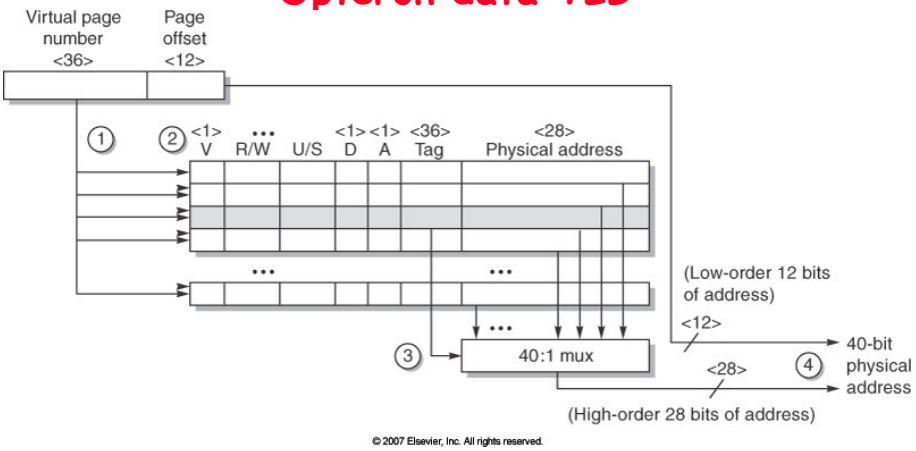


78

TLB Placement

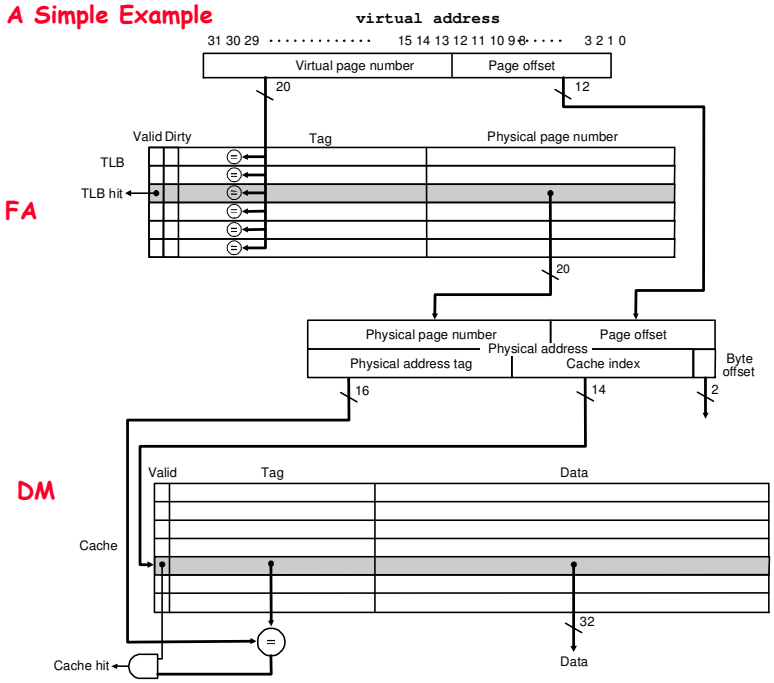
- This raises the question of whether caches should operate on physical or on virtual addresses
  - Caches can be virtually or physically tagged and indexed.

Opteron data TLB



- The Opteron data TLB is FA (in general, can use any strategy)

A Simple Example

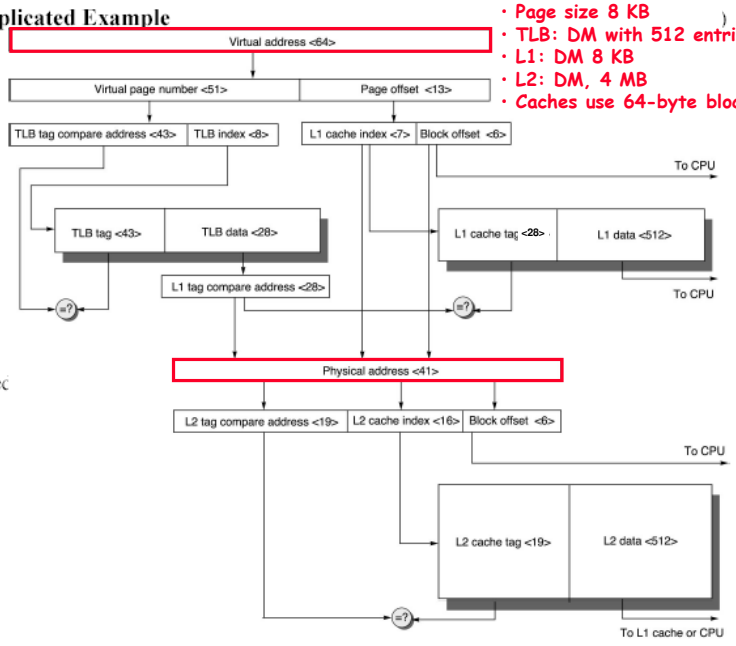


Optimization #6: Avoiding Address Translation during Indexing of the Cache to Reduce Hit Time

- Hits are much more common than misses (make the common case fast)
  - Use virtual addresses for caches
- So why doesn't everybody use virtual addresses for caches?
  - Protection for pages checked during address translation
- Cache must be flushed on every process switch
- Synonyms or aliases (two different virtual addresses for the same physical address)
- I/O uses physical addresses
- One soln: virtually indexed, physically tagged
  - Limit: DM cache can be no bigger than the page size

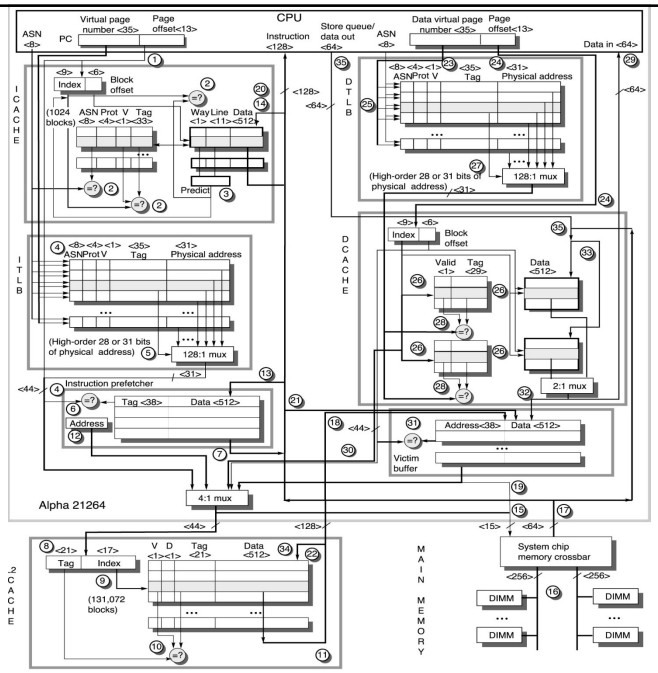
A More Complicated Example

- Page size 8 KB
- TLB: DM with 512 entries
- L1: DM 8 KB
- L2: DM, 4 MB
- Caches use 64-byte blocks



L1 is virtually indexed but physically tagged
L2 is physically indexed and tagged.
On a L1 miss, the physical address is used to fetch a block from L2.

83



Alpha 21264
Memory Hierarchy

84