

CSE 141 – Computer Architecture Spring 2005

Lectures 17 Virtual Memory

Pramod V. Argade
May 25, 2005

Announcements

- **Office Hour**
 - Monday, June 6th: 6:30 - 8 PM, AP&M 5218
 - Instead of regular Monday office hour 5 - 6 PM
- **Reading Assignment**
 - Chapter 7: The Basics of Caches: Sections 7.4 , 7.5
- **Recommended problems for Virtual Memory**
 - 7.39, 7.40
- **Next Lecture**
 - Course review. **Problems to be solved in class will be posted.**
- **Final**
 - When:** Friday, June 10th, 7 - 10 PM
 - Where:** SOLIS 107
 - No calculators!
- **Conflict with final? See me or send email**

Course Schedule

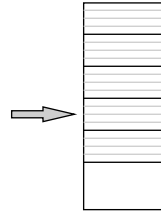
Lecture #	Date	Day	Lecture Topic	Quiz Topic	Homework Due
1	3/28	Monday	Introduction, Ch. 1	-	-
2	3/30	Wednesday	Performance, Ch. 4	-	-
3	4/4	Monday	ISA, Ch. 2	Performance	#1
4	4/6	Wednesday	Arithmetic, Ch. 3	-	-
5	4/11	Monday	Arithmetic, Ch. 3	ISA	#2
6	4/13	Wednesday	Single cycle CPU, Ch. 5	-	-
7	4/18	Monday	Single cycle CPU, Ch. 5	Arithmetic	#3
8	4/20	Wednesday	Multi-cycle CPU, Ch. 5	-	-
9	4/25	Monday	Multi-cycle CPU, Ch. 5	Single Cycle CPU	#4
10	4/27	Wednesday	Review for the Midterm	-	-
	5/2	Monday	Mid-term Exam Center 101	-	-
11	5/4	Wednesday	Exceptions, Ch. 5 and Pipelining, Ch. 6	-	-
12	5/9	Monday	Pipelining, Ch. 6	-	-
13	5/11	Wednesday	Data and control hazards, Ch. 6	-	-
14	5/16	Monday	Data and control hazards, Ch. 6	Pipeline Hazards	#5
15	5/18	Wednesday	Memory & cache design, Ch. 7	-	-
16	5/23	Monday	Memory & cache design, Ch. 7	-	-
17	5/25	Wednesday	Virtual Memory & cache design, Ch. 7	Cache	#6
No Class	5/30	Monday	Memorial Day Holiday	-	-
18	6/1	Wednesday	Course Review	-	-
	6/10	Friday	7 - 10 PM, SOLIS 107: Final Exam	-	-

The Three Cs

- Compulsory misses: **First time access**
 - Caused by the first access to a block that has never been in the cache
 - Also called cold-start misses
 - Can be reduced by increasing the block size
- Capacity misses: **Cache is not large enough**
 - Caused when cache cannot contain all the blocks needed
 - Occur because of blocks being replaced and later retrieved
 - Can be reduced by enlarging the cache
- Conflict misses: **Multiple addresses map to the same cache line**
 - Occur in direct mapped and set-associative caches
 - Multiple blocks compete for the same set
 - Can be eliminated by using fully associative cache

Which Block Should be Replaced on a Miss?

- Direct Mapped is Easy
- Set associative or fully associative:
 - Random (large associativities)
 - LRU (smaller associativities)
- Miss rates for the two schemes:



Associativity:	2-way		4-way		8-way	
	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

LRU is preferred scheme for a small size cache

Measuring Cache Performance

CPU time = (CPU execution clock cycles + Memory stall cycles)
 * Clock cycle time

Memory stall cycles = Read-stall cycles + Write-stall cycles

Read-stall cycles = (Reads per program) * Read miss rate *
 Read miss penalty

Write-stall cycles = (Writes per program) * Write miss rate *
 Write miss penalty

Cache Performance

- Ideal CPI for a processor (i.e. without memory stalls) is 1.4.
- 35% of the instructions are loads and stores.
- Miss penalty is 10 cycles
- How much do we improve the performance if the miss rate is reduced from 10% to 2%?

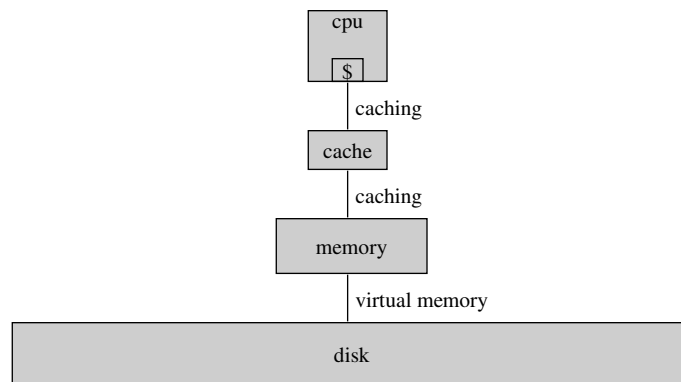
Summary

- The Principle of Locality:
 - Program likely to access a relatively small portion of the address space at any instant of time.
 - Temporal Locality: Locality in Time
 - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
 - Compulsory Misses: sad facts of life. Example: cold start misses.
 - Conflict Misses: increase cache size and/or associativity.
Nightmare Scenario: ping pong effect!
 - Capacity Misses: increase cache size
- Cache Design Space
 - total size, block size, associativity
 - replacement policy
 - write-hit policy (write-through, write-back)
- Caches give an illusion of a large, cheap memory with the access time of a fast, expensive memory

Virtual Memory

Virtual Memory

- Virtual memory is the name of the technique that allows us to view main memory as a cache of a larger memory space (on disk).
 - Allows efficient and safe sharing of memory among programs
 - Creates an illusion of providing unlimited memory to programs



Virtual Memory (VM)

- Each program is compiled to run in its own address space
- A single program may exceed the size of primary memory
- Multiple programs may dynamically share portions of memory
- Main memory need contain only the active portions of the program
- VM and caching have different historical roots
- VM is like caching, but uses different terminology

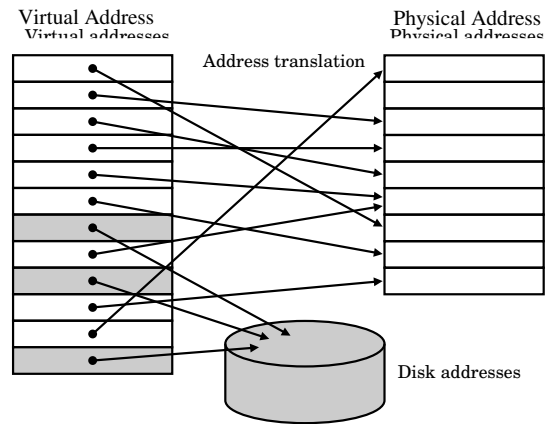
<u>Cache</u>	<u>VM</u>
block	page
cache miss	page fault
address	virtual address
index	physical address (sort of)

Advantages of Virtual Memory

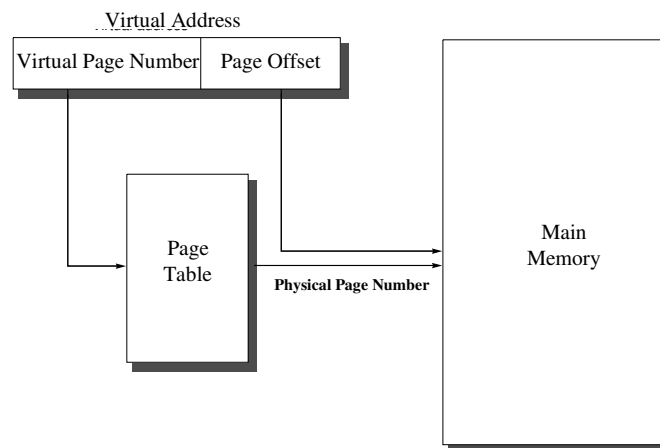
- Performance
 - Large amount of memory accessed efficiently
- Memory sharing among multiple programs
- Protection
 - Simultaneous (time-sharing) execution of multiple programs
 - Use of “kernel space” and “user space”
- Ease of programming/compilation
- Efficient use of memory

Memory Mapping/Address Translation

- Virtual to physical address mapping
- Page may be present or absent in main memory
- Page may be resident on the disk
- Two virtual pages may map to the same physical address



Mapping Virtual to Physical Address



Cache vs Virtual Memory Access

- Access time
time between when a read is requested and when the desired word arrives
- Transfer time
time it takes to transfer the whole request (ties up bandwidth)

Parameter	First-level Cache	Virtual memory
Block (Page) size	16 - 128 bytes	4096 - 65,536 bytes
Hit time	1 - 2 clock cycles	40 - 100 clock cycles
Miss Penalty	8 - 100 clock cycles	700,000 - 6,000,000 clock cycles
(Access time)	(6 - 60 clock cycles)	(500,000 - 4,000,000 clock cycles)
(Transfer time)	(2 - 40 clock cycles)	(200,000 - 2,000,000 clock cycles)
Miss Rate	0.5 - 10%	0.00001 - 0.001%
Data memory size	0.016 - 1 MB	16 - 8192 MB

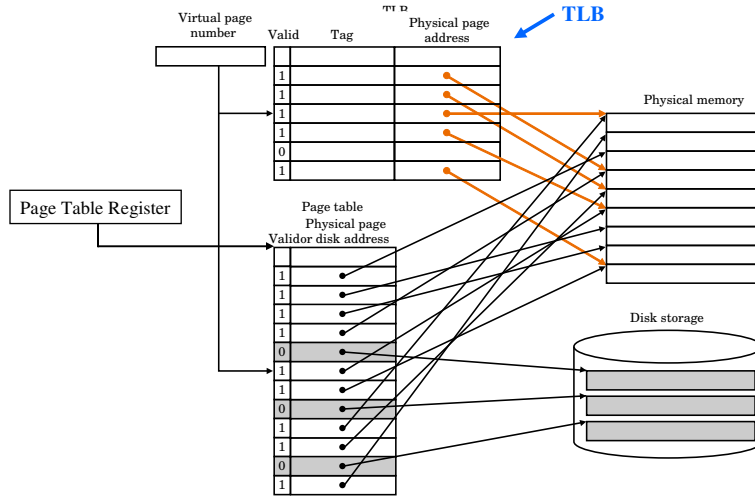
- VM has very high miss penalty
 - large pages (4 KB to MBs)
 - associative mapping of pages (typically fully associative)
 - software handling of misses (but not hits!!)
 - *write-through* not an option, only write-back

Translation Look-aside Buffer

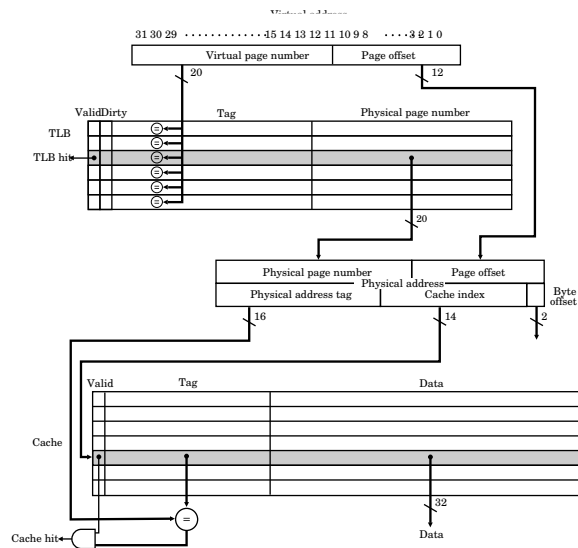
- Address translation could be expensive to perform for every memory access
 - page tables are stored in main memory
 - need to access page table before accessing data location
- Solution is to remember the last address translation so the mapping lookup can be skipped
 - use a translation buffer to hold the last N translations

TLB: Making Address Translation Fast

Translation Lookaside Buffer: A cache for address translations



TLB and Cache



Example: Page Table Size

- What is the total page table size if:
 - Virtual address is 32 bits
 - 8 Kbytes page size
 - Assume that each page table entry is 4 bytes
- Total Page Table Size:

Example: Address translation

- Using the page table shown, translate following 32-bit virtual addresses into physical addresses. Make entries in the TLB assuming LRU replacement.
 - The page size is 4 Kbytes
 - Addresses: 0x0000 3040, 0x0000 1040, 0x0000 2040

Virtual Physical

0x0000 3040 ⇒

0x0000 1040 ⇒

0x0000 2040 ⇒

Page Table Register	
Valid	Physical Page Number
1	0x0010 0
1	0x000D 2
0	0xdead 1
1	0x0023 9
...	...

TLB		
Valid	Tag	Physical Page Number
0		
0		
0		
0		

Memory Access Problems

- TLB Miss
 - Entry does not exist in the TLB
 - A mechanism must be provided to bring a page table entry into the TLB
- Page fault
 - The valid bit is not set for the page table entry
- Cache miss
 - Tag mismatch or valid bit not set
 - Cache line is brought in (depending on the policy for a write)

Processing a Page Fault

- If the valid bit for a virtual page is off, page fault occurs
- CPU generates an exception
- OS takes control
- OS finds the page in the next level of hierarchy (disk)
- OS decides where to place the requested page in memory
- OS copies the page from next level of hierarchy to memory
- OS sets valid bit in the page table entry for the virtual address
- OS returns from the exception
- Program re-executes the same instruction
- Page translation finds valid bit on for the virtual page
- Data access succeeds

What is a Process?

- Program state consists of:
 - Page tables, PC and the registers
- This state is referred to as a process
- Process is an instance of a program executing on a CPU

Implementing Protection with VM

- Protection is essential for:
 - Allowing to share a single main memory among multiple processes
 - Prevent one process from writing into the memory space of another
 - Prevent a user process from modifying its own page tables
 - Controlling raw access to peripheral devices
- Hardware capabilities needed for protection
 - Two operating modes: user mode and kernel mode of execution
 - A portion of the CPU state that a user process can read, but not write
 - This is the usr/kernel mode bit
 - A mechanism to switch between user mode and kernel mode
 - Accomplished by a system call

Additional bits in the Page Table

- User or Kernel bit
 - This bit restricts access to some pages to kernel only
- Write bit
 - This bit restricts read-only or read/write access to a page
- Referenced bit
 - OS periodically sets this bit to zero
 - It is set by CPU hardware when the page is referenced
 - Used by OS for replacing the page with other memory pages
- Dirty bit
 - If a process writes to a page, the dirty bit is set
 - It is used by OS to write the page to secondary storage before replacing it

Virtual Memory Key Points

- How does virtual memory provide:
 - illusion of large main memory?
 - sharing?
 - performance?
 - protection?
- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.
- Three things can go wrong on a memory access: cache miss, TLB miss, page fault.