

CPS104
Computer Organization and Programming
Lecture 16: Virtual Memory

Robert Wagner

cps 104 VM.1 ©RW Fall 2000

Outline of Today's Lecture

- Virtual Memory.
 - Paged virtual memory.
 - Virtual to Physical translation: The page table.
 - Fragmentation.
 - Page replacement policies.
 - Reducing the Virtual to Physical address translation time.
 - The TLB
 - Parallel access to the TLB and Cache
 - Memory Protection
 - Putting it all together: The SPARC-20 memory system

cps 104 VM.2 ©RW Fall 2000

Virtual Memory

cps 104 VM.3 ©RW Fall 2000

Memory System Management Problems

- Different Programs have different memory requirements.
 - How to manage program placement?
- Different machines have different amount of memory.
 - How to run the same program on many different machines?
- At any given time each machine runs a different set of programs.
 - How to fit the program mix into memory? Reclaiming unused memory? Moving code around?
- The amount of memory consumed by each program is dynamic (changes over time)
 - How to effect changes in memory location: add or subtract space?
- Program bugs can cause a program to generate reads and writes outside the program address space.
 - How to protect one program from another?

Virtual Memory

Provides *illusion* of very large memory
– Sum of the memory of many jobs greater than physical memory
– Address space of each job larger than physical memory

Allows available (fast and expensive) physical memory to be well utilized.

Simplifies memory management: code and data movement, protection, ... (*main reason today*)

Exploits memory hierarchy to keep average access time low.

Involves at least two storage levels: *main* and *secondary*

Virtual Address – address used by the programmer

Virtual Address Space – collection of such addresses

Memory Address – address in physical memory
also known as “physical address” or “real address”

Paged Virtual Memory: Main Idea

- Divide memory (virtual and physical) into fixed size blocks (Pages, Frames).
 - Pages in Virtual space.
 - Frames in Physical space.
- Make page size a power of 2: (page size = 2^k)
- All pages in the virtual address space are contiguous.
- Pages can be mapped into physical Frames in any order.
- Some of the pages are in main memory (DRAM), some of the pages are on secondary memory (disk).
- All programs are written using Virtual Memory Address Space.
- The hardware does on-the-fly translation between virtual and physical address spaces.
- Use a Page Table to translate between Virtual and Physical addresses

Basic Issues in Virtual Memory System Design

- size of information blocks (pages) that are transferred from secondary to main storage (M)
- block of information brought into M, and M is full, then some region of M must be released to make room for the new block --> *replacement policy*
- which region of M is to hold the new block --> *placement policy*
- missing item fetched from secondary memory only on the occurrence of a fault --> *demand load policy*

The diagram shows a flow from 'reg' to 'cache', then to 'mem' (containing a 'frame'), and finally to 'disk' (containing 'pages').

Paging Organization

virtual and physical address space partitioned into blocks of equal size

pages

frames

cps 104 VM.8 ©RW Fall 2000

Virtual and Physical Memories

The diagram shows 'Virtual Memory' with pages Page-0, Page-1, Page-2, Page-3, ..., Page N-2, Page N-1, Page N. 'Physical Memory' has frames Frame-0, Frame-1, Frame-2, Frame-3, Frame-4, Frame-5. A 'Disk' contains Page -2. Arrows show Page-0 mapping to Frame-1, Page-1 to Frame-0, Page-2 to Frame-3, Page-3 to Frame-4, and Page N to Page -2 on the disk.

cps 104 VM.9 ©RW Fall 2000

Virtual to Physical Address translation

Page size: 4K

The diagram shows a 'Virtual Address' with bits 31 to 0. Bits 31 to 11 are the 'Virtual Page Number' and bits 11 to 0 are the 'Page offset'. The 'Virtual Page Number' is used to look up the 'Physical Frame Number' in the 'Page Table'. The 'Physical Address' is formed by concatenating the 'Physical Frame Number' (bits 29 to 11) and the 'Page offset' (bits 11 to 0).

cps 104 VM.10 ©RW Fall 2000

Address Map

$V = \{0, 1, \dots, n-1\}$ virtual address space $n > m$
 $M = \{0, 1, \dots, m-1\}$ physical address space

MAP: $V \rightarrow M \cup \{\emptyset\}$ address mapping function
 $MAP(a) = a'$ if data at virtual address a is present in physical address a' and a' in M
 $= \emptyset$ if data at virtual address a is not present in M

cps 104 VM.11 ©RW Fall 2000

The page table

cps 104 VM.12 ©RW Fall 2000

Address Mapping Algorithm

If $V = 1$
 then page is in main memory at frame address stored in table
 else address located page in secondary memory

Access Control
 R = Read-only, R/W = read/write, X = execute only

If kind of access not compatible with specified access rights,
 then *protection_violation_fault*

If valid bit not set then *page fault*

Protection Fault: access control violation; causes trap to hardware,
 or software fault handler

Page Fault: page not resident in physical memory, also causes a trap;
 usually accompanied by a *context switch*: current process
 suspended while page is fetched from secondary storage

e.g., VAX 11/780
 each process sees a 4 gigabyte (2^{32} bytes) virtual address space
 1/2 for user regions, 1/2 for a system wide name space shared
 by all processes.

page size is 512 bytes (too small)

cps 104 VM.13 ©RW Fall 2000

Choosing a Page Size

What if page is too small?

- Too many misses
- BIG page tables

What if page is too big?

- Fragmentation
 - * don't use all of the page, but can't use that DRAM for other pages
 - * want to minimize fragmentation (get good utilization of physical memory)
- Smaller page tables
- Trend is toward larger pages
 - * increasing gap between CPU/DRAM/DISK

cps 104 VM.17

©RW Fall 2000

Optimal Page Size

Choose page size that minimizes fragmentation (partial use of pages)

large page size => internal fragmentation more severe

BUT decreases the # of pages / name space => smaller page tables

In general, the trend is towards larger page sizes because

- Memories get larger as the price of RAM drops.
- The gap between processor speed and disk speed grow wider.
- Programmers demand larger virtual address spaces (larger program)
- Smaller page table.

Most machines at 4K-8K byte pages today, with page sizes likely to increase

cps 104 VM.18

©RW Fall 2000

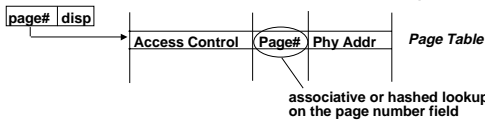
Fragmentation

Page Table Fragmentation occurs when page tables become very large because of large virtual address spaces; linearly mapped page tables could take up sizable chunk of memory

EX: VAX Architecture (late 1970s)	21	9
	XX Page Number	Disp
NOTE: this implies that page table could require up to 2^{21} entries, each on the order of 4 bytes long (8 M Bytes)	00 P0 region of user process	
	01 P1 region of user process	
	10 system name space	

Alternatives to linear page table:

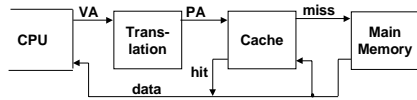
- (1) Hardware associative mapping: requires one entry per page frame ($O(M)$) rather than per virtual page ($O(N)$)
- (2) "software" approach based on a hash table (inverted page table)



cps 104 VM.19

©RW Fall 2000

Virtual Address and a Cache



It takes an extra memory access to translate VA to PA

This makes cache access very expensive, and this is the "innermost loop" that you want to go as fast as possible

cps 104 VM.20

©RW Fall 2000

Translation Lookaside Buffer (TLB)

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB*

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

TLB access time comparable to cache access time
(much less than main memory access time)

Typical TLB is 64-256 entries fully associative cache with random replacement

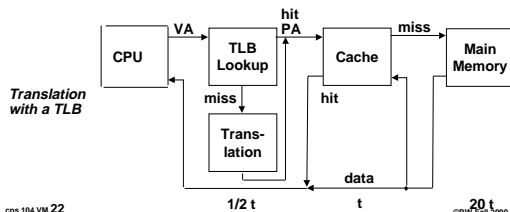
cps 104 VM.21

©RW Fall 2000

Translation Look-Aside Buffers

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Many mid-range machines use small n-way set associative organizations.



cps 104 VM.22

©RW Fall 2000

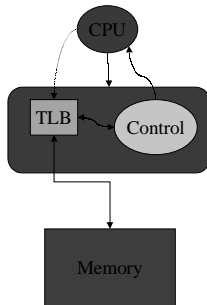
TLB Design

- Must be fast, not increase critical path
- Must achieve high hit ratio
- Generally small highly associative (64-128 entries FA cache)
- Mapping change
 - page added/removed from physical memory
 - processor must invalidate the TLB entry (special instructions)
- Page Table Entry (PTE) is a per-process entity
 - Multiple processes with same virtual addresses
 - Context Switches?
- Flush TLB
- Add Address Space ID (ASID), or Process ID (PID)
 - part of processor state, must be set on context switch

cps 104 VM.23

©RW Fall 2000

Hardware Managed TLBs

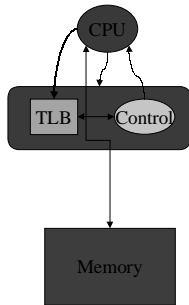


- Hardware Handles TLB miss
- Dictates page table organization
- Complicated state machine to "walk page table"
- Exception only if access violation

cps 104 VM.24

©RW Fall 2000

Software Managed TLBs



- Software Handles TLB miss
 - OS reads translations from Page Table and puts them in TLB
 - special instructions
- Flexible page table organization
- Simple Hardware to detect Hit or Miss
- Exception if TLB miss or access violation

cps 104 VM.25

©RW Fall 2000

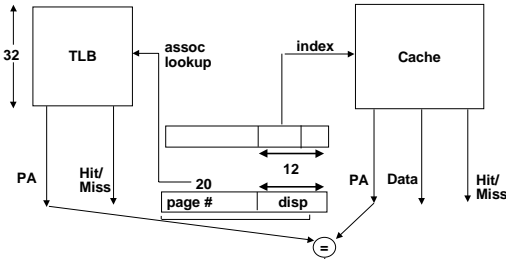
Reducing Translation Time

Machines with TLBs go one step further to reduce # cycles/cache access

They overlap the cache access with the TLB access

Works because high order bits of the VA are used to look in the TLB while low order bits are used as index into cache

Overlapped Cache & TLB Access



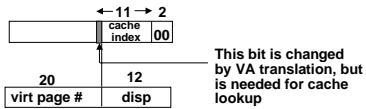
IF cache hit AND (cache tag = PA) then deliver data to CPU
 ELSE IF [cache miss OR (cache tag != PA)] and TLB hit THEN
 access memory with the PA from the TLB
 ELSE do standard VA translation

Problems With Overlapped TLB Access

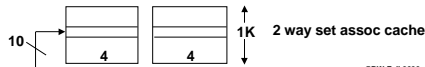
Overlapped access only works as long as the address bits used to index into the cache *do not change* as the result of VA translation

This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



Solutions:
 go to 8K byte page sizes;
 go to 2 way set associative cache; or
 SW guarantee VA[13]=PA[13]



More on Selecting a Page Size

- Reasons for larger page size
 - * Page table size is inversely proportional to the page size.
 - * faster cache hit time when cache size \leq page size; this allows use of virtual address as cache index (p 596) . Also, bigger page \Rightarrow bigger cache.
 - * Transferring larger pages to or from secondary storage, is more efficient (Higher bandwidth)
 - * The number of TLB entries is restricted by clock cycle time, so a larger page size reduces TLB misses.
- Reasons for a smaller page size
 - * don't waste storage; data must be contiguous within page.
 - * quicker process start for small processes(?)
- Hybrid solution: multiple page sizes:
Alpha, UltraSPARC: 8KB, 64KB, 512 KB, 4 MB pages

cps 104 VM.29

©RW Fall 2000

Memory Protection

- Paging Virtual memory provides protection by:
 - * Each process (user or OS) has different virtual memory space.
 - * The OS maintain the page tables for all processes.
 - * A reference outside the process's allocated space causes an exception that lets the OS decide what to do.
 - * Memory sharing between processes is done via different Virtual spaces but common physical frames.

cps 104 VM.30

©RW Fall 2000

Putting it together: The SparcStation 20:

- The SparcStation 20 has the following memory system.
- Caches: Two level-1 caches: I-cache and D-cache

Parameter	Instruction cache	Data cache
Organization	20Kbyte 5-way SA	16KB 4-way SA
Page size	4K bytes	4K bytes
Line size	8 bytes	4 bytes
Replacement	Pseudo LRU	Pseudo LRU

- TLB: 64 entry Fully Associative TLB, Random replacement
- External Level-2 Cache: 1M-byte, Direct Map, 128 byte blocks, 32-byte sub-blocks.

cps 104 VM.31

©RW Fall 2000

