CS152
Computer Architecture and Engineering
Lecture 22

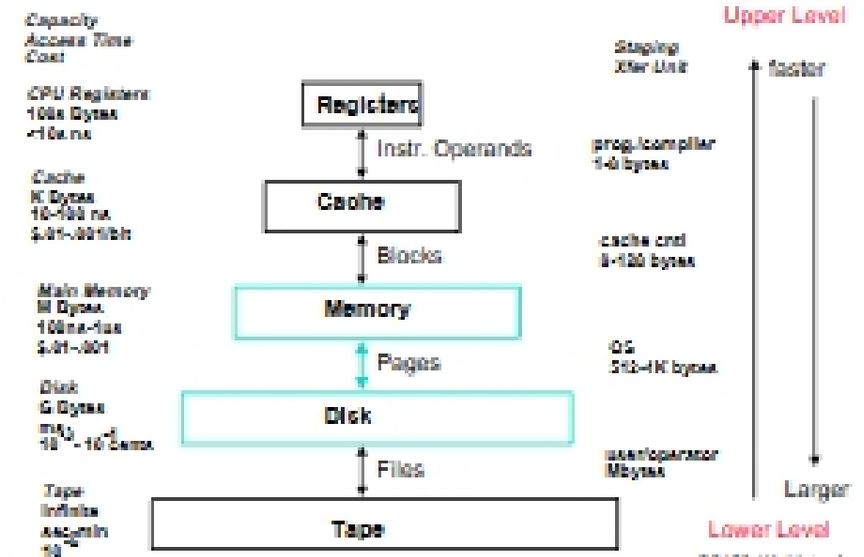
Buses and I/O
#1

April 26, 1999

John Kubiatowicz (http.cs.berkeley.edu/~kubitron)

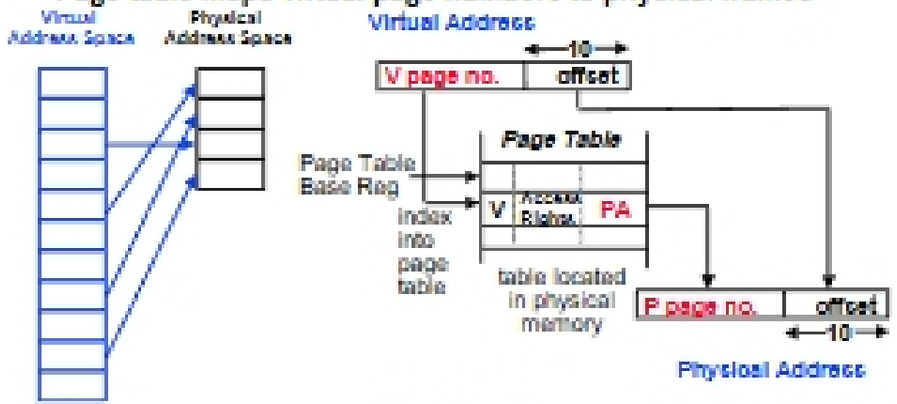
lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

Recap: Levels of the Memory Hierarchy



Recap: What is virtual memory?

- Virtual memory => treat memory as a cache for the disk
- Terminology: blocks in this cache are called "Pages"
- Typical size of a page: 1K — 8K
- Page table maps virtual page numbers to physical frames

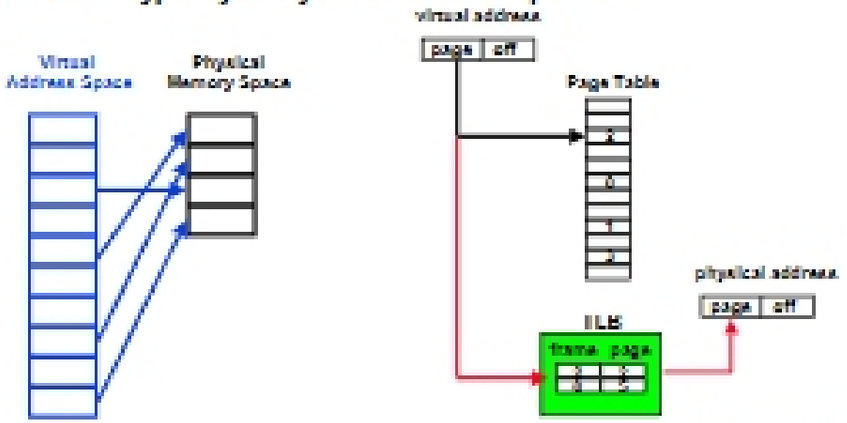


Recap: Three Advantages of Virtual Memory

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    - (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs => Far more "viruses" under Microsoft Windows
- Sharing:
  - Can map same physical page to multiple users ("Shared memory")

Recap: Making address translation practical: TLB

- Translation Look-aside Buffer (TLB) is a cache of recent translations
- Speeds up translation process "most of the time"
- TLB is typically a fully-associative lookup-table

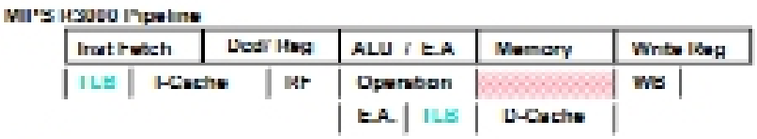


Recap: TLB organization: include protection

Virtual Address	Physical Address	Dirty	Ref	Valid	Access	ASID
0xFA00	0x0003	Y	N	Y	R/W	34
0x0040	0x0010	N	Y	Y	R	0
0x0041	0x0011	N	Y	Y	R	0

- TLB usually organized as fully-associative cache
  - Lookup is by Virtual Address
  - Returns Physical Address + other info
- Dirty => Page modified (Y/N)?
  Ref => Page touched (Y/N)?
  Valid => TLB entry valid (Y/N)?
  Access => Read? Write?
  ASID => Which User?

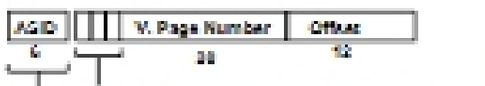
Recap: MIPS R3000 pipelining of TLB



TLB

64 entry, on-chip, fully associative, software TLB fault handler

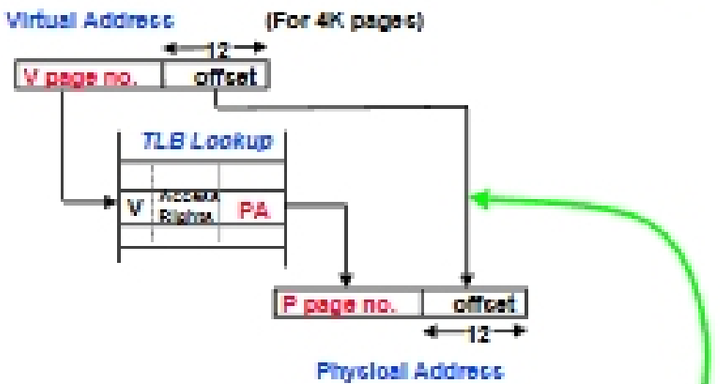
Virtual Address Space



0xx User segment (caching based on PT/TLB entry)
100 Kernel physical space, cached
101 Kernel physical space, uncached
11x Kernel virtual space

Allows context switching among
64 user processes without TLB flush

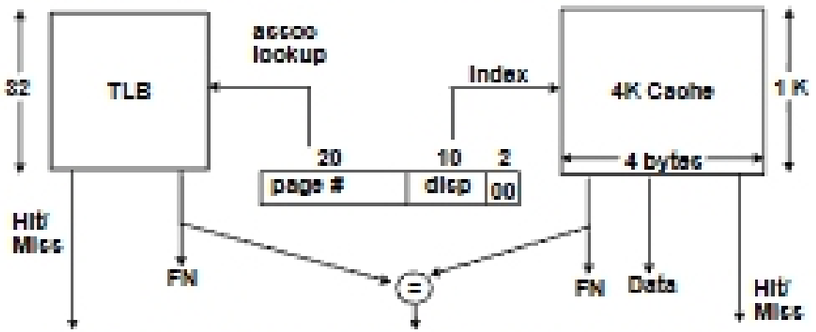
Reducing Translation Time I: Overlapped Access



- Machines with TLBs overlap TLB lookup with cache access.
  - Works because lower bits of result (offset) available early

Overlapped TLB & Cache Access

° If we do this in parallel, we have to be careful, however:



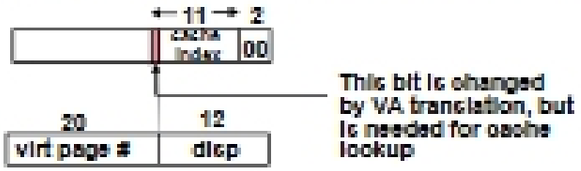
° With this technique, size of cache can be up to same size as pages.

⇒ What if we want a larger cache???

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

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

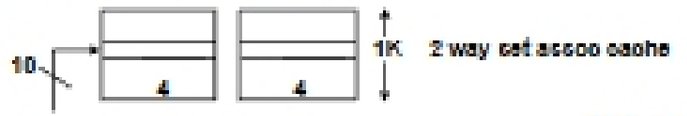


° Solutions:

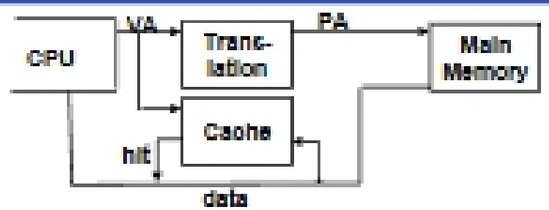
⇒ Go to 8K byte page sizes;

⇒ Go to 2 way set associative cache; or

⇒ SW guarantee VA[13]=PA[13]



Reduced Translation Time II: Virtually Addressed Cache



° Only require address translation on cache miss!

- Very fast as result (as fast as cache lookup)
- No restrictions on cache organization

° *Synonym problem*: two different virtual addresses map to same physical address ⇒ two cache entries holding data for the same physical address!

° *Solutions:*

- Provide associative lookup on physical tags during cache miss to enforce a single copy in the cache (potentially expensive)
- Make operating system enforce one copy per cache set by selecting virtual⇒physical mappings carefully. This only works for direct mapped caches.

° Virtually Addressed caches currently out of favor because of synonym complexities

Survey

° R4000

- 32 bit virtual, 36 bit physical
- variable page size (4KB to 16 MB)
- 48 entries mapping page pairs (128 bit)

° MPC601 (32 bit implementation of 64 bit PowerPC arch)

- 52 bit virtual, 32 bit physical, 16 segment registers
- 4KB page, 256MB segment
- 4 entry instruction TLB
- 256 entry, 2-way TLB (and variable sized block xlate)
- overlapped lookup into 8-way 32KB L1 cache
- hardware table search through hashed page tables

° Alpha 21064

- arch is 64 bit virtual, implementation subset: 43, 47,51,55 bit
- 8,16,32, or 64KB pages (3 level page table)
- 12 entry ITLB, 32 entry DTLB
- 43 bit virtual, 28 bit physical octword address