

Lecture 19: Virtual Memory

COS 471a, COS 471b / ELE 375

Computer Architecture and Organization

Princeton University
Fall 2005

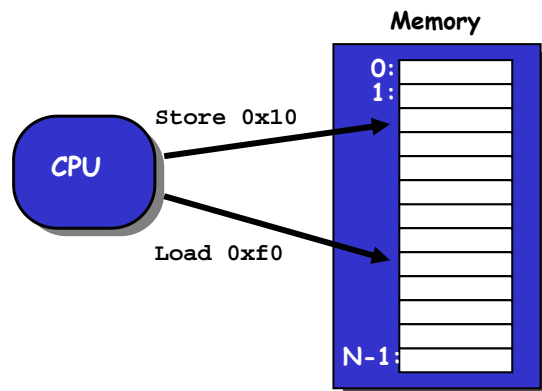
Prof. David August

1



A System with Physical Memory

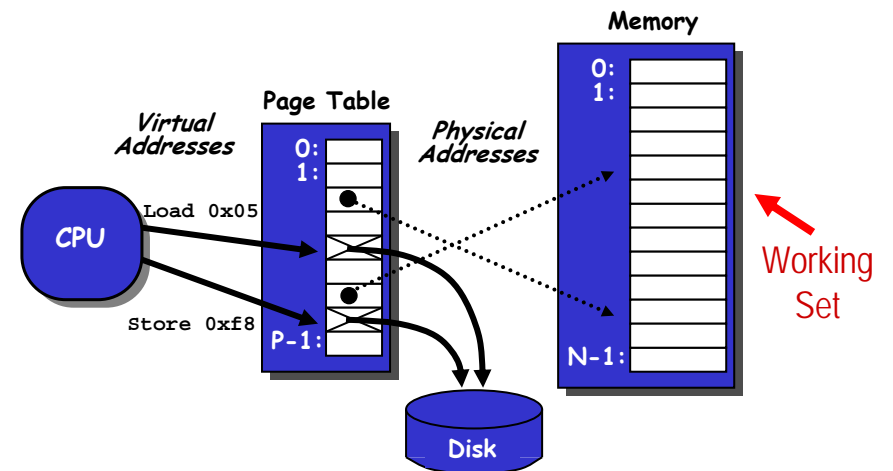
Examples: Most Cray machines, early PCs, nearly all current embedded systems, etc.



CPU's load or store addresses used directly to access memory.

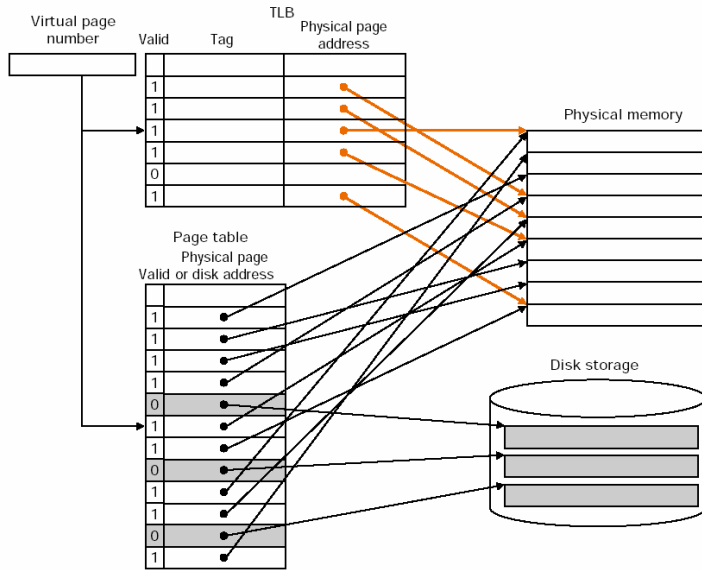
5

A System with Virtual Memory



6

Making Translation Faster: The TLB Translation Look-Aside Buffer



11

Virtual Memory Summary

Virtual memory provides

- Protection and sharing
- Illusion of large main memory
- Speed/Caching (when viewed from disk perspective)
- Virtual Memory requires twice as many memory accesses, so cache page table entries in the TLB.
- Three things can go wrong on a memory access
 - TLB miss
 - Page fault
 - Cache miss

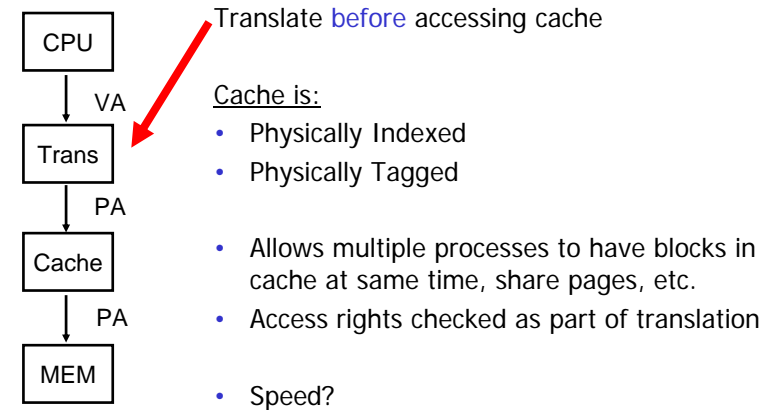
Caches and virtual memory?

12



Virtually Memory and Caches: 3 Options

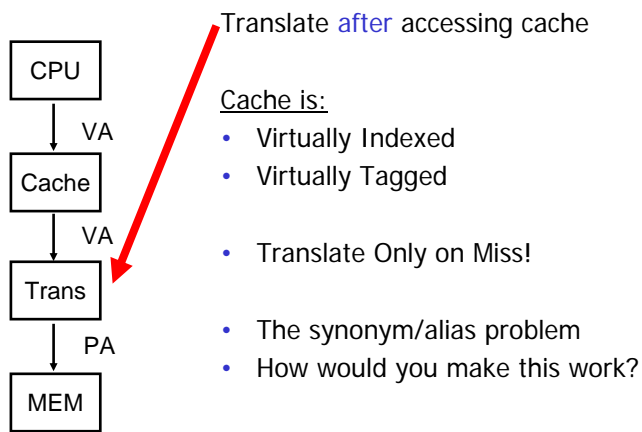
1. Physically Addressed Cache



14

Virtually Memory and Caches: 3 Options

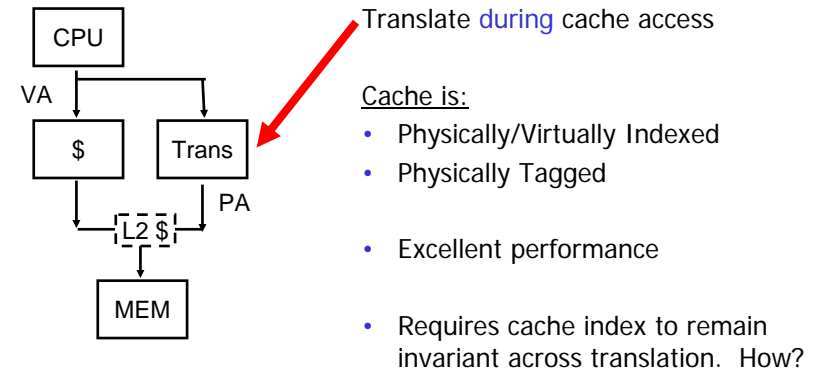
2. Virtually Addressed Cache



15

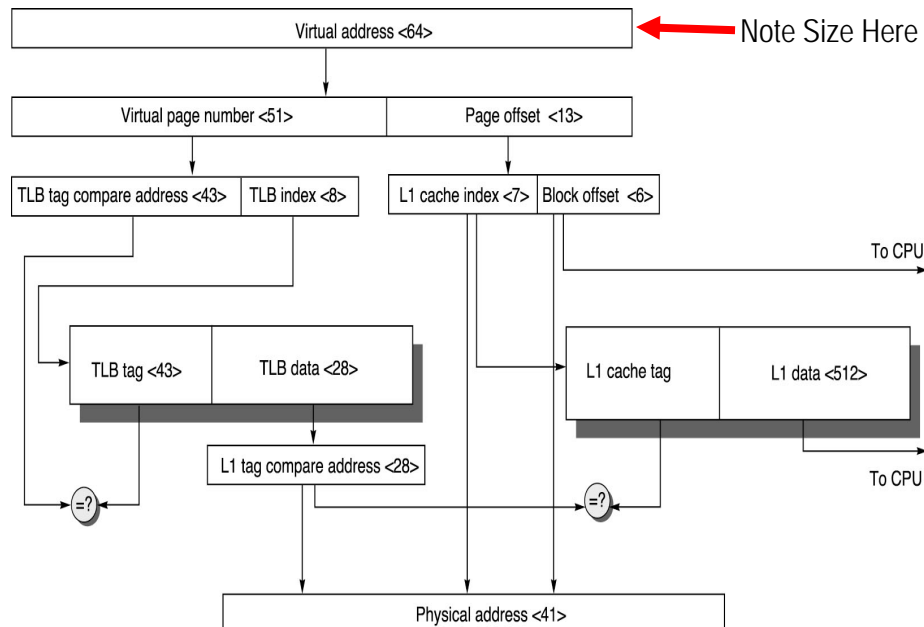
Virtually Memory and Caches: 3 Options

3. Virtually Indexed, Physically Tagged



16

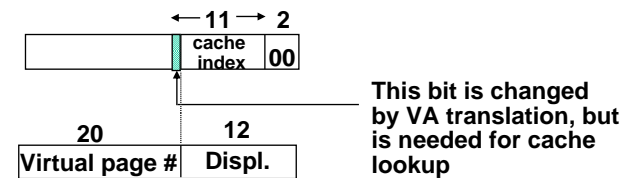
Virtually Indexed, Physically Tagged Example



17

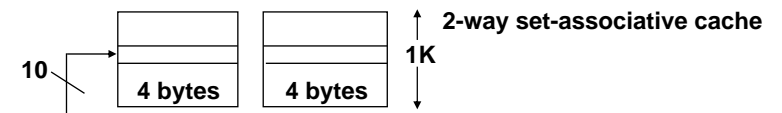
Issues With Overlapped TLB Access

- Limits cache parameters: small caches, large page sizes, or high n-way set-associative caches
- Example: Suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K

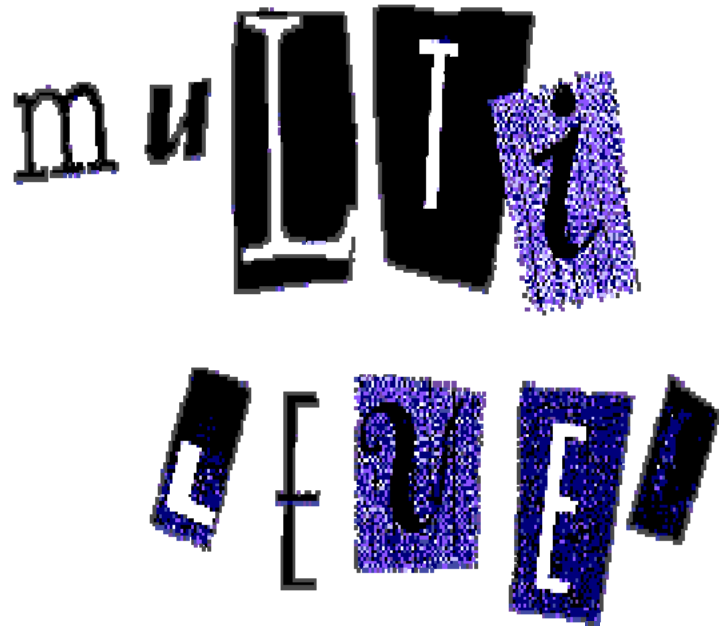


Solutions:

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



18



Some Page Table Math

of page table entries on 64-bit machine with 4K pages:

$$2^{64} / 2^{12} = (\text{only}) 2^{52} \text{ entries}$$

Size of page table:

$$2^{52} * 8 \text{ bytes per table entry} = 2^{55} \text{ bytes}$$

(only 32 petabytes)

kilo- 2^{10} , mega- 2^{20} , giga- 2^{30} , tera- 2^{40} ,
peta- 2^{50} ,
exa- 2^{60} , zetta- 2^{70} , yotta- 2^{80}

20

Some Page Table Math

Size of page table:

$$2^{52} * 8 \text{ bytes per table entry} = 2^{55} \text{ bytes}$$

(only 32 petabytes)

Oh, by the way, that's per process...

21

Solutions

1. Limit Page Table Size

- Keep a limit
- Check limit before going to page

If more entries needed (process needs more memory):

1. Up the limit
2. Add the entries

Good way to do this:

- Double page table size at each step:
- Limit is: $0...01...1$ (number 0 $\rightarrow 2^{n-1}$)

Also, can grow bi-directionally (stack/heap)

22

Solutions

2. Inverted Page Table

!! These things are UGLY !!

Each Physical Frame has an entry.

Inverted page table size:

Physical memory size = 8 Gigabytes = 2^{33} bytes

Page frame size = 4K = 2^{12} bytes

$2^{33} / 2^{12} = 2^{21}$ entries

2^{21} entries * 8 bytes per entry (incl. PID) = 2^{24} bytes

16MB, not too bad (not per process!)

23

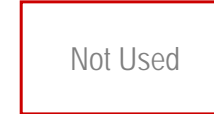
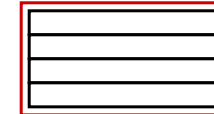
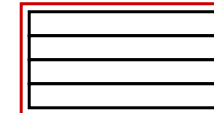
Solutions

3. Multilevel Page Tables

Key Idea: Take advantage of sparse use of virtual memory

Create a hierarchy of pages:

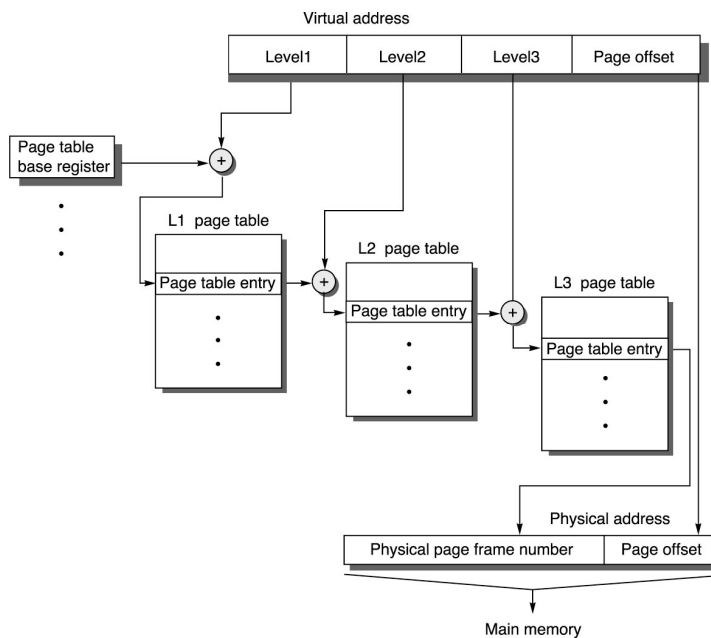
Create a red page table to describe very large pages (coarse cut of virtual address space)



Create a black page table for each red page table entry used (finer cut of superpage)

24

Solution 3: Multi-Level Page Tables Example



25

Solutions

4. Page The Page Table

- Compatible with other methods
- Tricky to get right
- Need to have page portion that refers to rest of page table always in memory

26



Segmentation

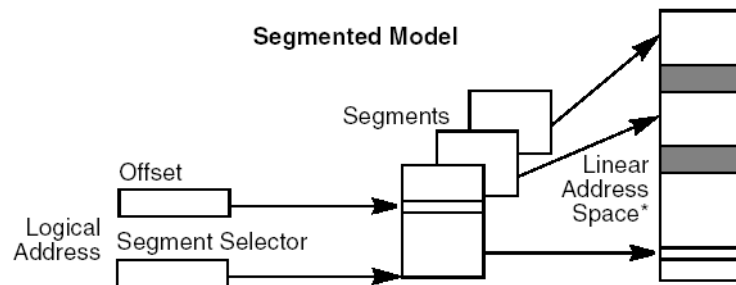
Real Stuff (x86 IA32)

- Segments: Variable-sized pages
- Virtual address are **segment number** + **offset**
- Generally 2 quantities
 - Segment register
 - Offset is address
- Bounds checking
- Nice in some ways:
 - Program fits in one segment - set ReadOnly/Executable
 - Data in another - set ReadWrite/NonExecutable

28

x86: Segmentation

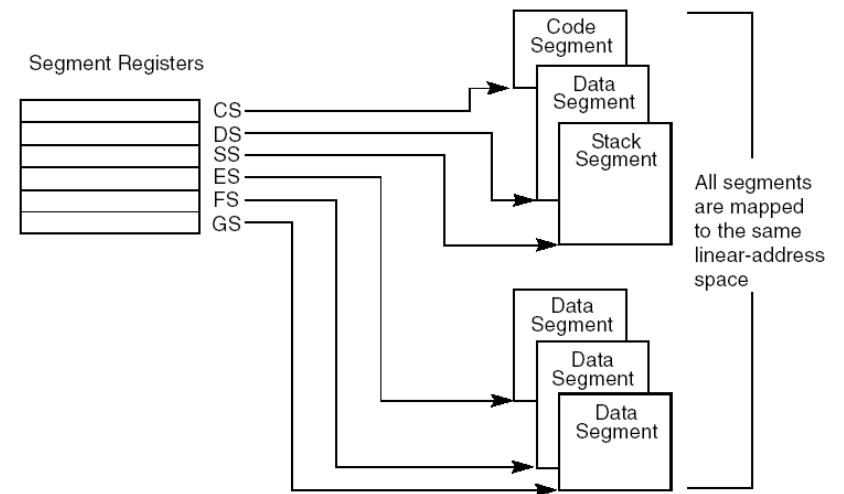
(From: IA-32 Intel® Architecture Software Developers Manual)



29

x86: Segment Registers

(From: IA-32 Intel® Architecture Software Developers Manual)



Pages and Segments Can Co-exist!

30



Relating to the MIPS Pipeline

MIPS R3000 Pipeline

Inst Fetch	Dcd/ Reg	ALU / E.A.	Memory	Write Reg
TLB	I-Cache	RF	Operation	WB
		E.A. TLB	D-Cache	

32

Summary

- Real/Virtual Tag/Index Cache
- Multi Level Page Tables
- Segments
- Pipeline Interaction
- Read book for more real stuff

33