CS 61C:
Great Ideas in Computer Architecture
Exceptions/Traps/Interrupts

Instructors:
Krste Asanovic, Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa12

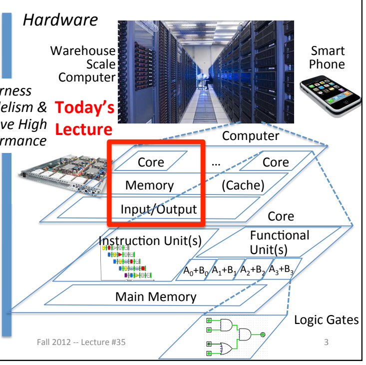
Review

- Programmed I/O versus DMA
- Polling versus Interrupts
- Asynchronous interrupts versus synchronous traps
- Precise interrupt looks like execution stopped at exactly one instruction, every instruction before finished, no instruction after started.
  – Simplify software view of interrupted state

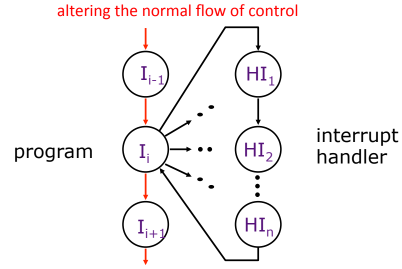
You Are Here!

Software

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages



Interrupts:
altering the normal flow of control

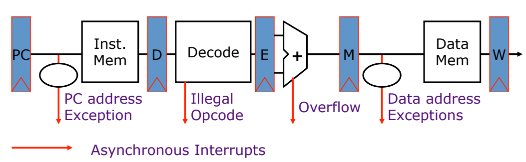


An *external or internal event* that needs to be processed by another (system) program. The event is usually unexpected or rare from program's point of view.

Precise Interrupts

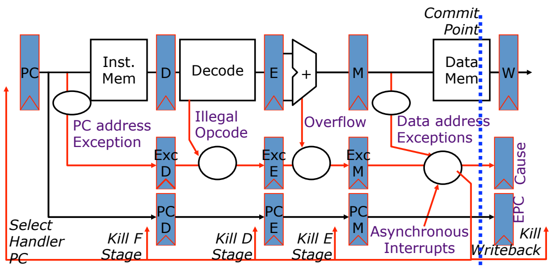
- Interrupt handler's view of machine state is that every instruction prior to the interrupted one has completed, and no instruction after the interrupt has executed.
  – Instruction taking interrupt might have written some special state but can be restarted.
- Implies that handler can return from interrupt by restoring user registers and jumping to EPC
  – Software doesn't need to understand the pipeline of the machine!
- Providing precise interrupts is tricky in a pipelined superscalar out-of-order processor!
  – But handling imprecise interrupts in software is even worse.

Exception Handling in 5-Stage Pipeline



- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

Save Exceptions Until Commit



Handling Exceptions in In-Order Pipeline

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- Inject external interrupts at commit point (override others)
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

Speculating on Exceptions to avoid Control Hazard

- Prediction mechanism
  - Exceptions are rare, so simply predicting no exceptions is very accurate!
- Check prediction mechanism
  - Exceptions detected at end of instruction execution pipeline, special hardware for various exception types
- Recovery mechanism
  - Only write architectural state at commit point, so can throw away partially executed instructions after exception
  - Launch exception handler after flushing pipeline
- Bypassing allows use of uncommitted instruction results by following instructions

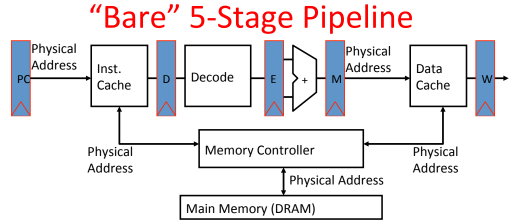
Exception Pipeline Diagram

time

	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	-		*overflow!*		
(I_2) 100: XOR		IF_2	ID_2	EX_2	-	-			
(I_3) 104: SUB			IF_3	ID_3	-	-	-		
(I_4) 108: ADD				IF_4	-	-	-	-	
(I_5) Exc. Handler code					IF_5	ID_5	EX_5	MA_5	WB_5

Virtual Memory

"Bare" 5-Stage Pipeline



- In a bare machine, the only kind of address is a physical address

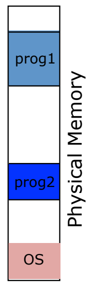
Dynamic Address Translation

Motivation
- In early machines, I/O operations were slow and each word transferred involved the CPU
- Higher throughput if CPU and I/O of 2 or more programs were overlapped.
  - *How?⇒ multiprogramming with DMA I/O devices, interrupts*

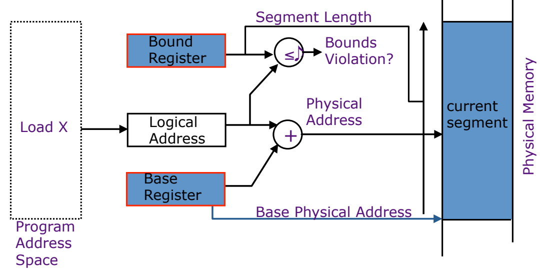
Location-independent programs
- Programming and storage management ease
- ⇒ need for a *base register*

Protection
- Independent programs should not affect each other inadvertently
- ⇒ need for a *bound register*

Multiprogramming drives requirement for resident *supervisor* software to manage context switches between multiple programs



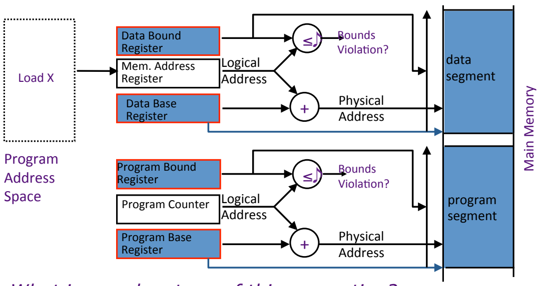
Simple Base and Bound Translation



Base and bounds registers are visible/accessible only when processor is running in *kernel mode*

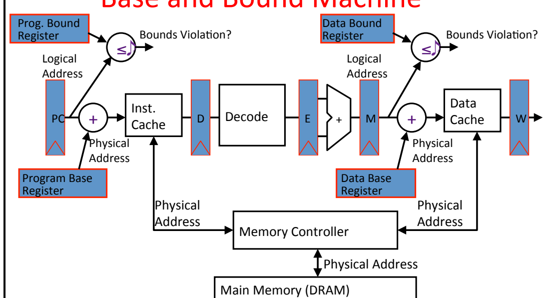
Separate Areas for Program and Data

(Scheme used on all Cray vector supercomputers prior to X1, 2002)



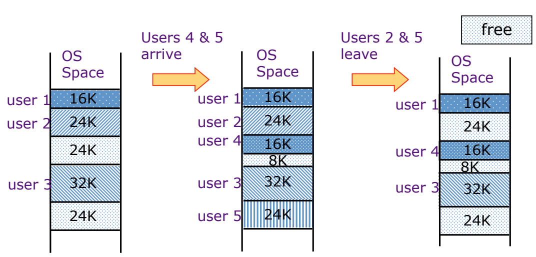
What is an advantage of this separation?

Base and Bound Machine



[Can fold addition of base register into (register+immediate) address calculation using a carry-save adder (sums three numbers with only a few gate delays more than adding two numbers)]

Memory Fragmentation



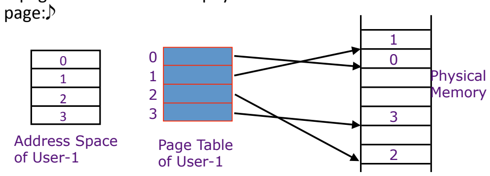
As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

Paged Memory Systems

- Processor-generated address can be split into:

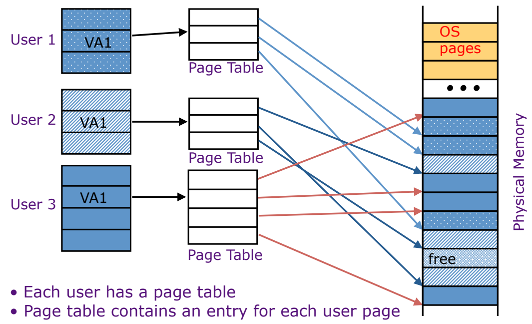
page number	offset

- A page table contains the physical address of the base of each page:



Page tables make it possible to store the pages of a program non-contiguously.

Private Address Space per User

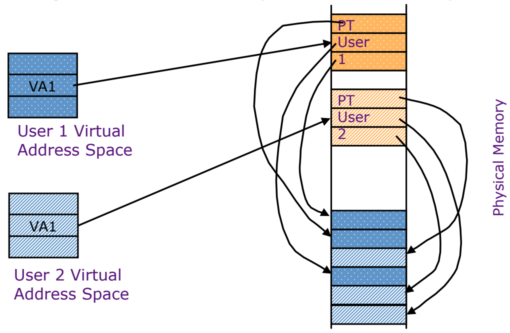


- Each user has a page table
- Page table contains an entry for each user page

Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, ...
  - ⇒ *Too large to keep in registers*
- Idea: Keep PTs in the main memory
  - needs one reference to retrieve the page base address and another to access the data word
    - ⇒ *doubles the number of memory references!*

Page Tables in Physical Memory

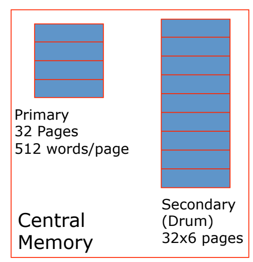


Demand Paging in Atlas (1962)

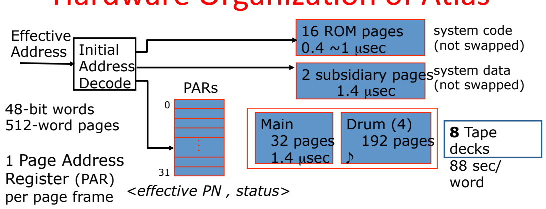
"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."
Tom Kilburn

Primary memory as a *cache* for secondary memory

User sees 32 x 6 x 512 words of storage



Hardware Organization of Atlas



Compare the effective page address against all 32 PARs
match ⇒ normal access
no match ⇒ *page fault*
save the state of the partially executed instruction

Atlas Demand Paging Scheme

- On a page fault:
  - Input transfer into a free page is initiated
  - The Page Address Register (PAR) is updated
  - If no free page is left, a *page is selected to be replaced* (based on usage)
  - The replaced page is written on the drum
    - to minimize drum latency effect, the first empty page on the drum was selected
  - The *page table is updated* to point to the new location of the page on the drum

Administrivia

- Regrade request deadline Monday Nov 26
  - For everything up to Project 4

CS61C In the News:
"Texas Instruments Cuts 1,700 Jobs and Winds Down Tablet Chips", NY Times 11/14/2012

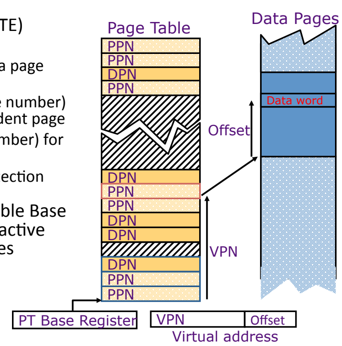
"Texas Instruments is eliminating 1,700 jobs, as it winds down its mobile processor business to focus on chips for more profitable markets like cars and home appliances. Texas Instruments said in September it would halt costly investments in the increasingly competitive smartphone and tablet chip business, leading Wall Street to speculate that part of the company's processor unit, called OMAP, could be sold. The layoffs are equivalent to nearly 5 percent of the Austin, Texas-based company's global workforce.

TI has been under pressure in mobile processors, where it has lost ground to rival Qualcomm Inc. Leading smartphone makers Apple Inc and Samsung Electronics Co Ltd have been developing their own chips instead of buying them from suppliers like TI. Instead of competing in phones and tablets, TI wants to sell its OMAP processors in markets that require less investment, like industrial clients like carmakers. TI is expected to continue selling existing tablet and phone processors for products like Amazon.Com Inc's Kindle tablets for as long as demand remains, but stop developing new chips."

[Rumors of Amazon being interested in buying this OMAP unit from TI]

Linear Page Table

- Page Table Entry (PTE) contains:
  - A bit to indicate if a page exists
  - PPN (physical page number) for a memory-resident page
  - DPN (disk page number) for a page on the disk
  - Status bits for protection and usage
- OS sets the Page Table Base Register whenever active user process changes



Size of Linear Page Table

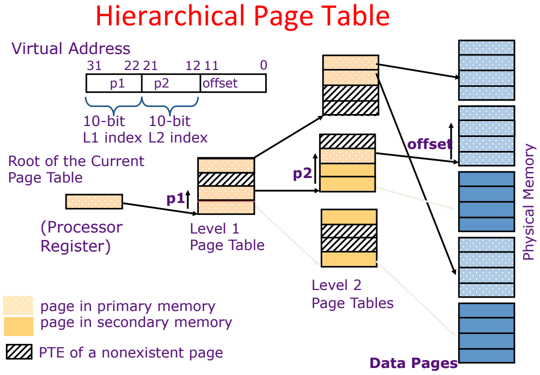
With 32-bit addresses, 4-KB pages & 4-byte PTEs:
- ⇒ 2^{20} PTEs, i.e, 4 MB page table per user
- ⇒ 4 GB of swap needed to back up full virtual address space

Larger pages?
- Internal fragmentation (Not all memory in page is used)
- Larger page fault penalty (more time to read from disk)

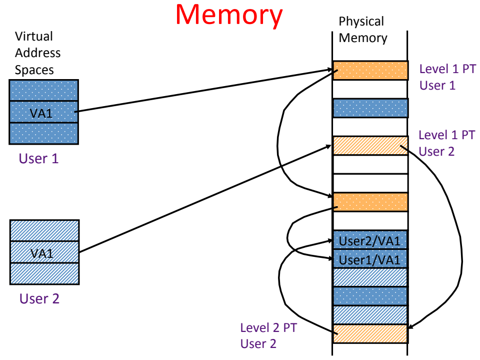
What about 64-bit virtual address space???
- Even 1MB pages would require 2^{44} 8-byte PTEs (35 TB!)

What is the "saving grace" ?

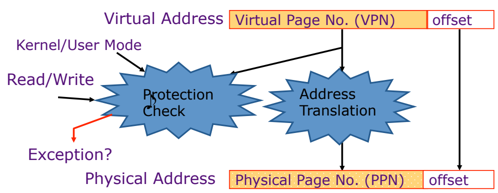
Hierarchical Page Table



Two-Level Page Tables in Physical Memory



Address Translation & Protection



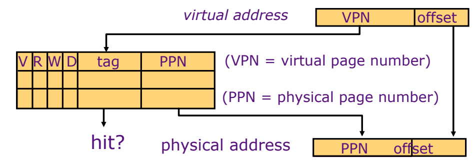
- Every instruction and data access needs address translation and protection checks

A good VM design needs to be fast (~ one cycle) and space efficient

Translation Lookaside Buffers (TLB)

Address translation is very expensive!
In a two-level page table, each reference becomes several memory accesses

Solution: *Cache translations in TLB*
TLB hit ⇒ *Single-Cycle Translation*
TLB miss ⇒ *Page-Table Walk to refill*



TLB Designs

- Typically 32-128 entries, usually fully associative
  - Each entry maps a large page, hence less spatial locality across pages ➔ more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative
  - Larger systems sometimes have multi-level (L1 and L2) TLBs
- Random or FIFO replacement policy
- No process information in TLB?
- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KB pages, one page per entry

TLB Reach = ____?

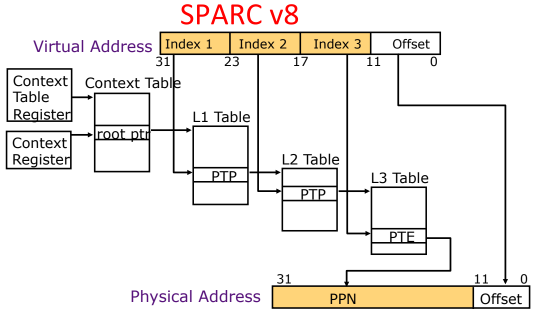
Handling a TLB Miss

Software (MIPS, Alpha)
TLB miss causes an exception and the operating system walks the page tables and reloads TLB. *A privileged "untranslated" addressing mode used for walk*

Hardware (SPARC v8, x86, PowerPC, RISC-V)
A memory management unit (MMU) walks the page tables and reloads the TLB

If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

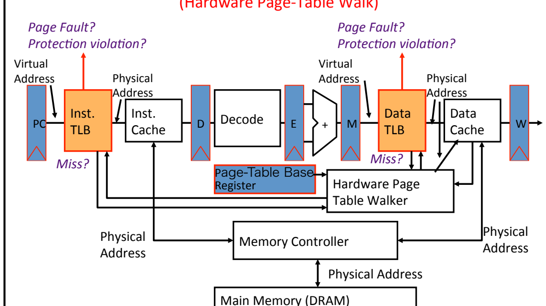
Hierarchical Page Table Walk: SPARC v8



MMU does this table walk in hardware on a TLB miss

Page-Based Virtual-Memory Machine

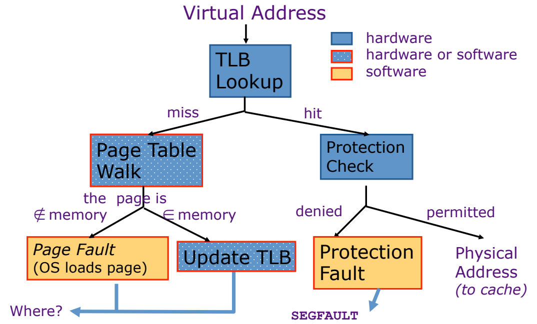
(Hardware Page-Table Walk)



- Assumes page tables held in untranslated physical memory

Address Translation:

putting it all together



Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252