Virtual Memory

Today's topics:

Virtual memory

deeper look at memory hierarchy & management

TLB's for incresed speed and protection

a few examples of approaches to date

Midterm Review

topics you should pay attention to

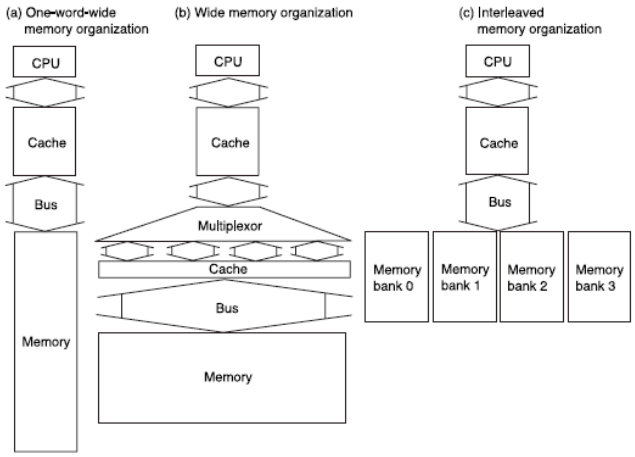
Memory Hierarchy

- 3 physical memory types
  - caches – on chip SRAM
  - main memory – off chip DRAM
    - » fronted by a memory controller
      - lots of details later in the course – for now think slow
  - disk – either SSD or HD
    - » magnetic or slow NVRAM
      - details later for now thing SUPER SLOW
- Common principle
  - similar to multi-level caches
  - miss here?
    - » dig deeper

Main Memory Organization

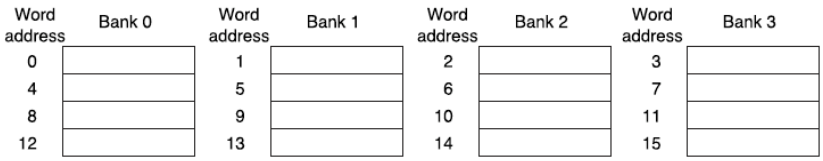
- **Familiar optimizations**
  - **wider memory**
    - » **make a main memory transaction look like a cache line**
      - **handled primarily by the memory controller**
  - **bus width**
    - » **actually a standard**
      - **wider & slower: JEDEC**
      - **skinnier and faster: RAMBUS**
  - **pipeline**
    - » **with synchronous DRAM's**
      - **pipeline extended into DIMM and DRAM chips**
  - **interleaved or phased memory**
    - » **n slow banks – interleave return on higher bandwidth path/bus**
    - » **ultimately the trick being used in DDR1, 2, 3, ...**
  - **optimize for sequential memory accesses**
    - » **capitalize on spatial locality similar to caches**

Hierarchy Options



Striping/Interleaving

- Different for disks
  - ignore this for now
- For caches and main memory
  - exploit concurrency in banks

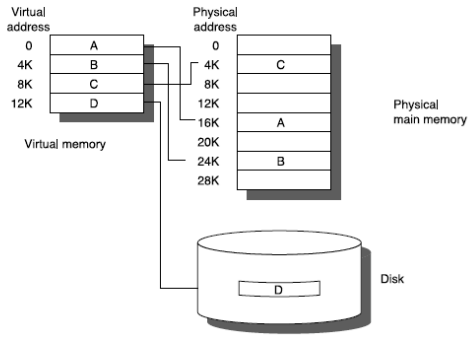


Virtual Memory

- Large virtual address space
  - mapping mechanism to physical main memory
    - » e.g. 64 bit virtual address space
      - smaller physical address
        - 36-40 bits common now
- Multiple process management
  - each process has a "private" and "protected" virtual address space
    - » but share physical memory (caches and main memory)
      - trick is how to manage this private/protected illusion so it's true
      - for caches
        - virtual indexed and tagged via address spaces
  - between DRAM and disk
    - » miss becomes a page or TLB fault
      - TLB is just a cache of recently used page table entries
    - » block becomes a page or segment

Page Relocation

- **Page table allows contiguous virtual addresses to be mapped in a non-contiguous fashion in main memory**



Difficulties

- **TLB is a cache**
  - **usually highly associative**
    - » **conflict miss penalty is huge since miss**
      - **is to main memory (~300 cycles) or disk (~10 msec)**
- **Main memory has 2 masters**
  - **cache line sized blocks move up in the hierarchy**
  - **page sized blocks move down in the the hierarchy**
  - **memory controller has to keep it straight**
    - » **used to be on the Northbridge chipset**
    - » **now moving on chip**
      - **2 of them on Nehalem for example**

Line vs. Page Differences

- **Replacement**
  - **page fault handled by OS**
    - » **time to access disk + context switch is large**
    - » **hence more exotic replacement (LRU'ish) policy is tractable**
- **Capacity**
  - **cache size choice is unrelated to either physical or virtual address size**
  - **physical address size specifies maximum main memory size**
    - » **smaller is OK but mask exists to based on existing configuration**
  - **virtual address size specifies the minimum swap space size**
    - » **multiply by how many processes you'd like to be partially resident**
- **What's on the disk**
  - **SWAP partition**
  - **File system partitions**

2 VM Styles: Main Memory

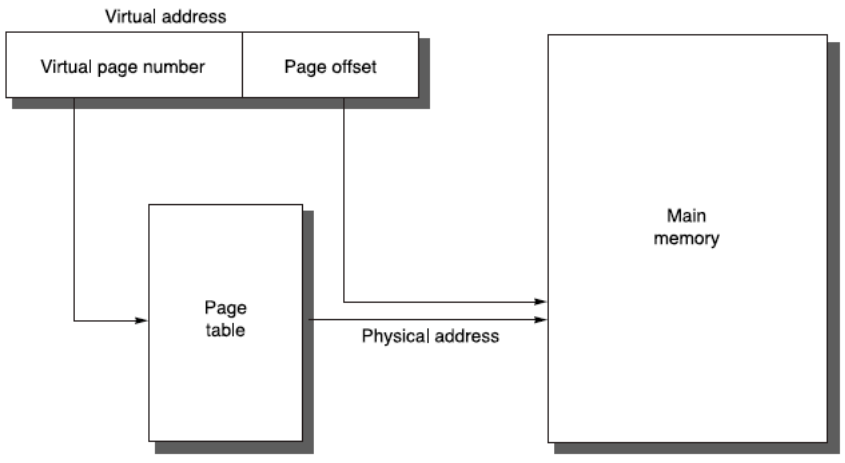
- **Pages are fixed size**
  - **super-page options exist to increase TLB reach**
- **Segments**
  - **variable sized – hence base pointer and offset addressing**

Aspect	Page	Segment
Words/Address	One - contains page and offset	Two - possible large max-size hence need Seg and offset address words
Programmer visible	No	Sometimes yes
Replacement	Trivial - due to fixed size	Hard - need to find contiguous space ==> GC necessary or wasted memory
Memory Inefficiency	Internal fragmentation - wasted part of a page	External fragmentation - due to variable size blocks
Disk Efficiency	Yes - adjust page size to balance access and transfer time	Not always - segment size varies

VM's & Same 4 Questions

- **Placement**
  - **lower miss rates vs. complex placement**
    - » **large miss penalty**
      - **choose low miss rate → place anywhere**
        - **similar to fully associative cache but on a page granularity in main mem**
- **Addressing**
  - **pages via a page table**
    - » **VPN → PPN and catenate page offset**
      - **page table or TLB cache does translation**
      - **valid bit needed as a minimum to indicate presence in main mem**
  - **segmentation**
    - » **segment table**
      - **segment # → offset in segment table**
        - **pointer to head of segment table required**
      - **lots of segments → bigger segment table required**

VPN → PPN Mapping Basics



Normal Page Tables

- Size
  - # entries = number of virtual pages
- Role
  - VPN → PPN translation
    - » enables page relocation
  - still need status tags
    - » valid
    - » protection: priv'd, R, W, Xeq, ...
- Potential problem
  - 64-bit virtual address space, 34 bit physical address & 4 KB page
    - » page table has 2^{52} entries
      - YOW that's more than physical memory
    - » ideas of how to fix this?

Inverted Page Table

- Make page table reflect what's in physical memory
  - use a hash mechanism
    - » create index into inverted page table
  - compare VPN with tag to make sure of the hit
    - » similar to caches
  - if you don't find it you go to disk
    - » double jeopardy
      - disk access to get the page table
      - disk access to get the page you want
      - plus update the IPT
- The good bit
  - caches miss rarely
  - IPT miss is even more rare

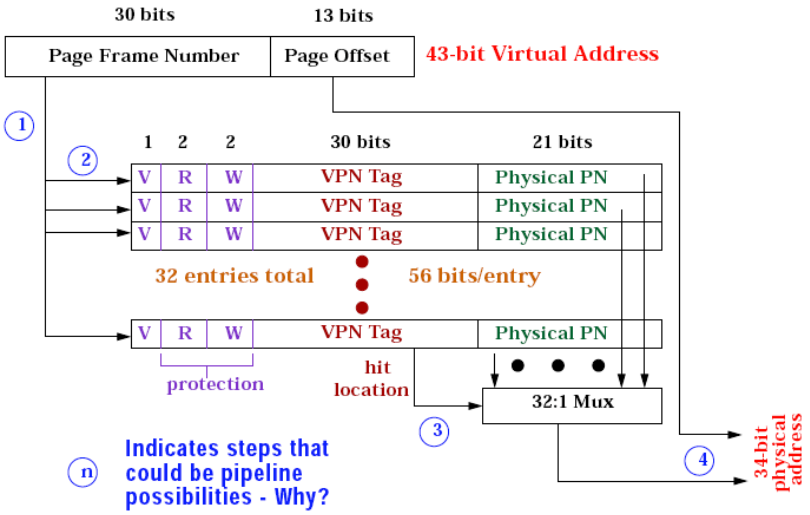
Page Policies

- **Replacement**
  - LRU best but same story – expensive
  - hence "use" bit idea is employed
    - » rarer OS wake up makes this closer to LRU than it is for caches
    - » strategy
      - spend a few OS cycles to reduce miss rate and horrific page miss penalty
- **Write strategy**
  - always write back – so dirty bit required
    - » write-through to disk is silly
  - write buffering works as with caches
    - » larger grain size → larger buffer size
    - » get the requested one first then do the write when you can

Page Size Dilemma

- **Large pages are good**
  - reduces page table size
    - » increases TLB reach
  - amortizes long disk latencies
  - works well when spatial locality is in play
- **Large pages are bad**
  - more internal fragmentaion
    - » last page of text, heap, and control stack is 50% wasted
  - process start up delay
    - » at least the first 3 pages of each type is required
    - » big pages → longer transfer time delay

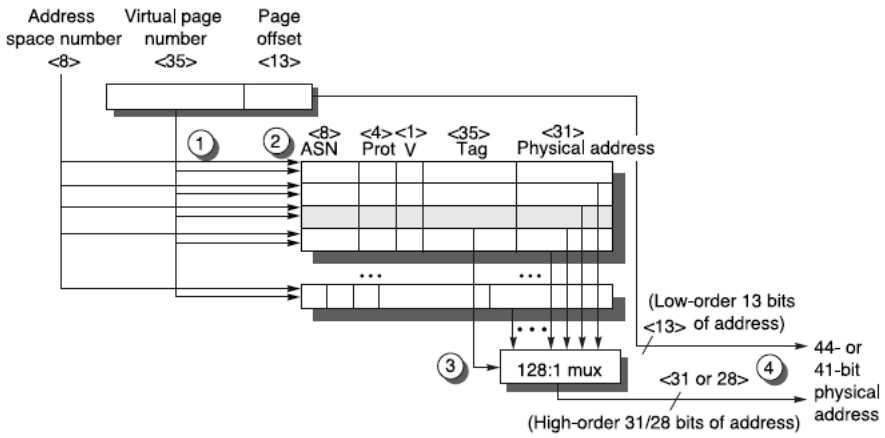
AXP 21064 TLB Example



Improvements

- **What's chanaged**
  - **multple process' data can co-reside in memory**
    - » **technology side-effect**
      - **larger memories, more processes, higher context switch overhead**
  - **virtual address alias problem**
    - » **use PID's is an option but there are too many of them**
    - » **use address space numbers instead**
      - **similar mechanism discussed for caches**
        - **1 per process**
        - **no need for TLB flush if valid ASN**
        - **if start a process without a current ASN**
          - **remap and flush TLB with evicted ASN**
            - **no write back needed – just invalidate**
      - **HP-PA series used this idea from the start**

AXP 21264



Protection Options

- **Base and bound – segmented VM**
  - **check that address falls between 2 register values**
    - » **registers can only be changed by OS/priv'd instructions**
    - » **PID or ASN idea can be used for finer grain ACL control**
      - **read, write, execute**
- **Paged VM**
  - **check as part of the VA→PA translation**
    - » **held in TLB on a page based privilege**
- **Other options**
  - **ring based**
    - » **MULTICS (late 60's) and now Pentium**
      - **inner ring is most priv'd – outer is user**
        - **allows greater distinction: kernel, OS, loadable module, user**
      - **capabilities (aka the ill-fated Intel I432)**
        - **key or password based model**
        - **OS hands them out so difficult to forge**
        - **apps can pass them around which could be dangerous**

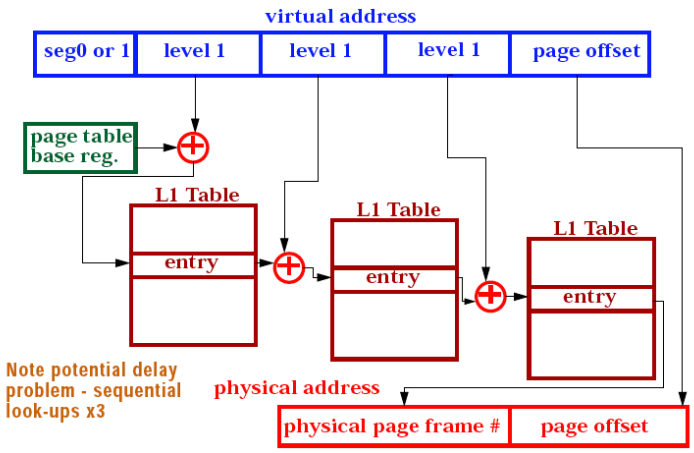
AXP 21264

- **Both segmented and paged**
  - 64 bit address space (only 48 were used)
    - » $addr_{63:46}$
      - all 0's
        - seg0 for text and heap – grows upward
        - seg 1 for stack grows downward
      - all 1's
        - kseg – reserved for OS kernel
        - uniformly protected space – no memory management
        - Idea is to keep kernel resident w/ no perterbation from seg0 & seg1
- **Advantages**
  - segmentation conserves page table space
  - paging provides VM, protection and relocation
    - » paging happens within each segment
    - » split page tables
  - best of both worlds
    - » how about cost?

21264 VM Problems

- **Big page table**
  - big memories are slow
  - go hierarchical here too
    - » 3 levels of page table
      - each table is 1 page in size
      - 8KB page but support for super pages 16, 32, 64KB in 21364
      - 34 bit physical address
- **Virtual addr = [seg index, lvl1, lvl2, lvl3, offset]**
  - mapping
    - » LVL1-TBL[lvl1]+lvel2 points to LVL2-TBL entry
      - and so forth
    - » LVL3-TBL entry provides PPN
    - » PPN##offset → physical address for main memory

21264 Mapping



Concluding Remarks

- **Pentium**
  - **both paged and segmented w/ table based translation**
    - » **VA's mapped to segments and physical addresses**
      - **contains protection bits – 4 level ring based**
        - **wrinkle – depends on who calls who**
        - **allows user code to safe call OS and use shared memory**
        - **possible Trojan horse problem**
          - **fix by not allowing OS to provide an indirect reference**
          - **OS and user stack are separate – hence copy required**
            - **no parameter passing via registers**
    - » **example of not trusting the OS much (oh that would be MS)**
    - » **lots of other cruft that we'll ignore**
- **Things are getting hairier**
  - **multiple cores, VM's, hypervisors**
    - » **open question is what is the right HW support for protection**
      - **mantra – common case goes to HW, flexibility is best done in SW**

Midterm Review 1

- Topics
  - cost and market segment issues
  - quantitative analysis ala HW1 and HW2
  - ISA issues
    - » RISC vs. CISC, memory modes
  - Pipelining
    - » performance issues, hazards, forwarding, laminarity
  - ILP
    - » various optimizations in both HW and SW/compiler
      - know the trade-offs
  - Branch prediction (HW3 should have focused some attention)
    - » static vs. dynamic
    - » local, global, tournament
    - » predict what
      - taken-not taken
      - address
      - instruction

Midterm Review 2

- More topics
  - dynamic issue superscalar
    - » Scoreboarding and Tomasulo
      - know how they work
  - static issue superscalar
    - » VLIW and EPIC
      - understand group and bundle idea from EPIC
  - ILP limitations
    - » there are good reasons for going multi-core and multi-threaded
    - » know why and the basic multi-threading approaches
  - Caches
    - » we've covered the basics
    - » conceptual question only is possible
    - » understand
      - basic organizations: direct-mapped, set-associative, associative
      - basic tradeoffs and organizations
      - miss types what what helps reduce each type
      - basic memory performance equations