CS 33: Computer Organization

Dis 1B: Week 9 Discussion

Adapted from CS 33 Discussion Slides by Uen-Tao Wang

Agenda

- Instructor Evaluation Survey
- Virtual Memory
- Brief Chat about the Performance Lab

Instruction Evaluation Survey

- Please submit them, both for me and Tony!
- Should only take about 2 mins
- But it will help our teaching staff a lot!

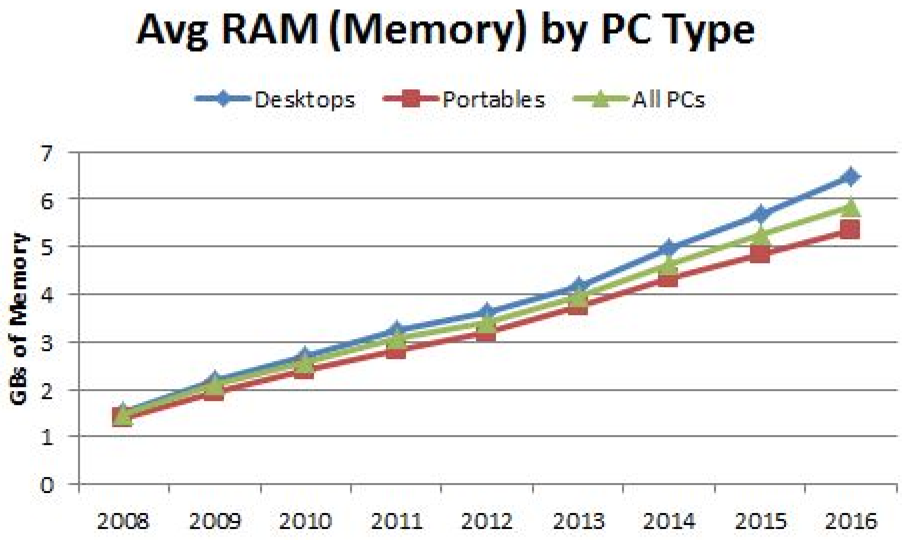
Memory

- We now have an adequate model for describing how a program is executed
- Instruction addresses are fed into the processor and memory accesses generally must be satisfied by the cache
- Memory addresses range from 32~64 bits. If we consider a 32-bit system, this implies that our memory must have a capacity of 2^32 Bytes

Memory

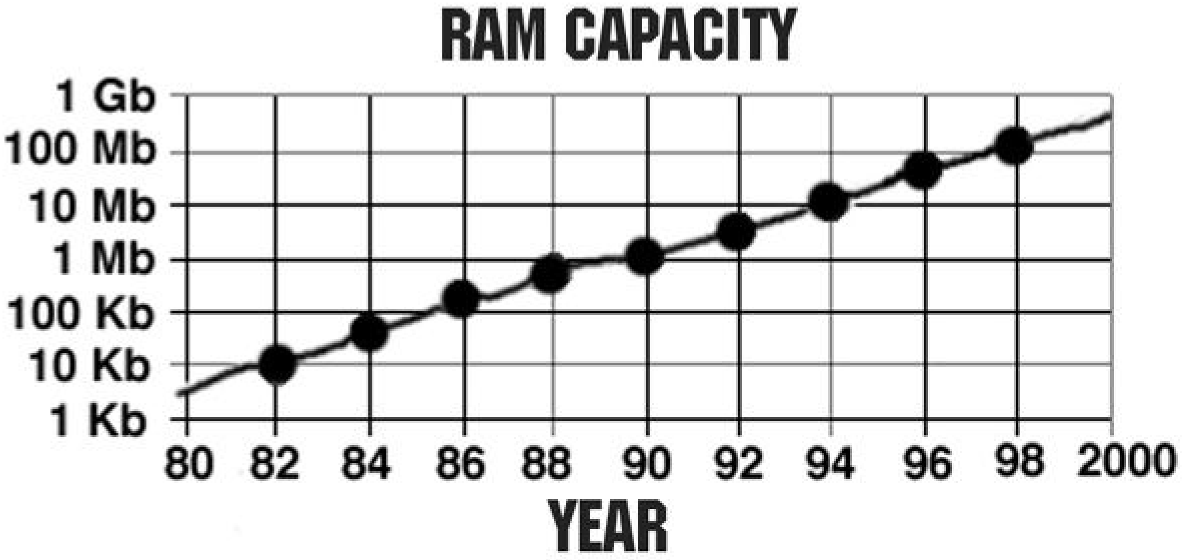
- That suggests that main memory must be at least 2^32 bytes, or approximately 4 GB
- Wait… what?

Memory



Avg RAM (Memory) by PC Type
Desktops
Portables
All PCs
GBs of Memory
0
1
2
3
4
5
6
7
2008
2009
2010
2011
2012
2013
2014
2015
2016

Memory


RAM CAPACITY
1 Gb
100 Mb
10 Mb
1 Mb
100 Kb
10 Kb
1 Kb
80 82 84 86 88 90 92 94 96 98 2000
YEAR

Memory

- What about processes?
  - They assume they have the entire address space
- What if a computer was running 100 processes?
  - For a 32-bit system, we would need 400 GB of RAM
  - 400 GB of RAM?!

Virtual Memory

- Addresses that are known to a program are actually “virtual” addresses in a scheme known as virtual memory
- Each process expects to be the sole owner of 2^32 bytes of memory, but it would be impractical to have that much DRAM
- Instead, we can simply allow them to believe that they own the entire addressable space

Virtual Memory

- In reality, the physical memory must be shared between all of the processes
- The actual contents of each process's memory must actually be stored on the much larger (and much slower) disk
- Main memory acts like a "cache" for the virtual memory on disk

Virtual Memory

- Memory (Both virtual and physical) is split up into a number of “pages”
- Each page is just a contiguous chunk of memory of a set size
    - Word?
- When a process accesses memory, it uses a virtual address. If the physical memory contains the “page” that the virtual address belongs to, get the value from memory
- If not, page fault, you’ll have to find the page on disk, then copy it to physical memory

Virtual Memory

- Does that mean if we have 100 processes, we would need ~400 GB of disk space, just to store the memory spaces of all of the programs?
- Not all pages are actually “allocated”, which means they don’t exist anywhere. This allows processes to only take up as much space as they need

Virtual Memory

- Just like the cache, if the main memory is full and you need to bring a different page from disk into the main memory, some page is going to have to be a victim
- If the virtual page can be anywhere in physical memory, how do we find it?
- Do we just do a linear search?

Page Table

- Each process maintains a page table which maps virtual addresses to physical addresses
- Part of the virtual address is used to index into the table. Each entry in the page table will contain part of the physical address that it maps to

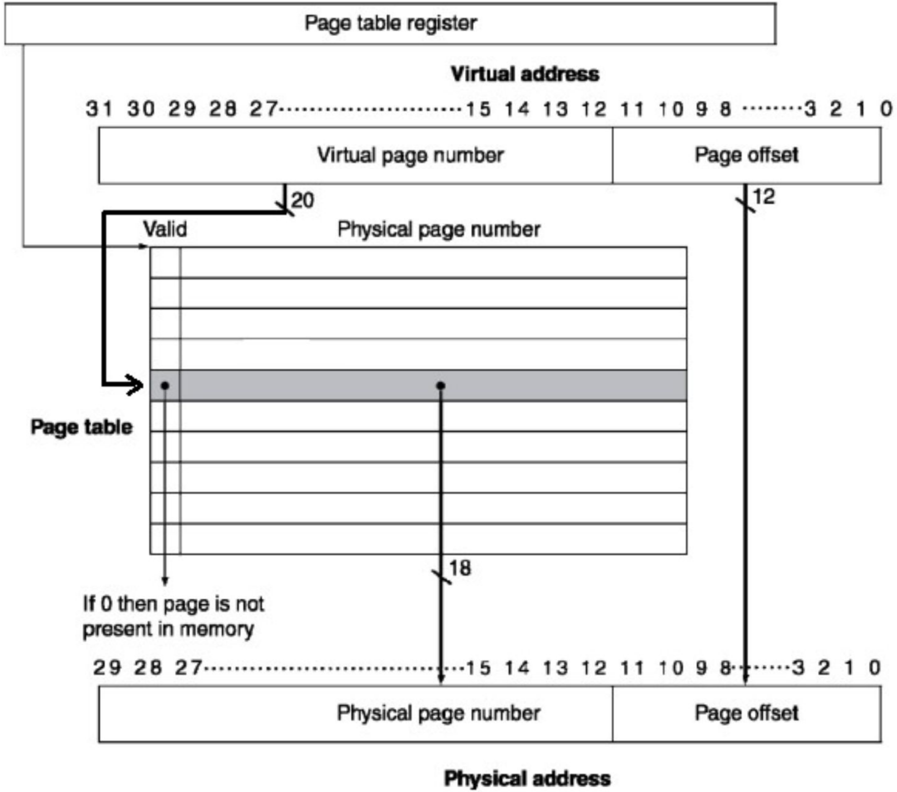
Page Table

- Virtual Address Decomposition
  - [VPN][VPO]
- VPN: Virtual Page Number
  - The index into the page table
- VPO: Virtual Page Offset
  - The byte offset into the page
- Index into the page table with the VPN. The value stored in the page table is the PPN (Physical page number)

Page Table

- Physical Address decomposition
  - [PPN][PPO]
- The VPO and PPO are the same since page size is consistent in both virtual and physical address space

Page Table



Page table register
Virtual address
31 30 29 28 27 ··· 15 14 13 12 11 10 9 8 ··· 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
If 0 then page is not present in memory
29 28 27 ··· 15 14 13 12 11 10 9 8 ··· 3 2 1 0
Physical page number
Page offset
Physical address

Page Table

- Consider a 14-bit virtual address space and a 12-bit physical address space
- Each page is 256 bytes
- How is the Virtual Address splitted in order to index into the Page Table?

Page Table

- How is the Virtual Address splitted in order to index into the Page Table?
- Each page has 256 = 2^8 bytes. This means we need 8 bits to represent the page offset (VPO and PPO)
- This means the VPN is 6-bits and the PPN is 4-bits
- How many entries are in the page table?

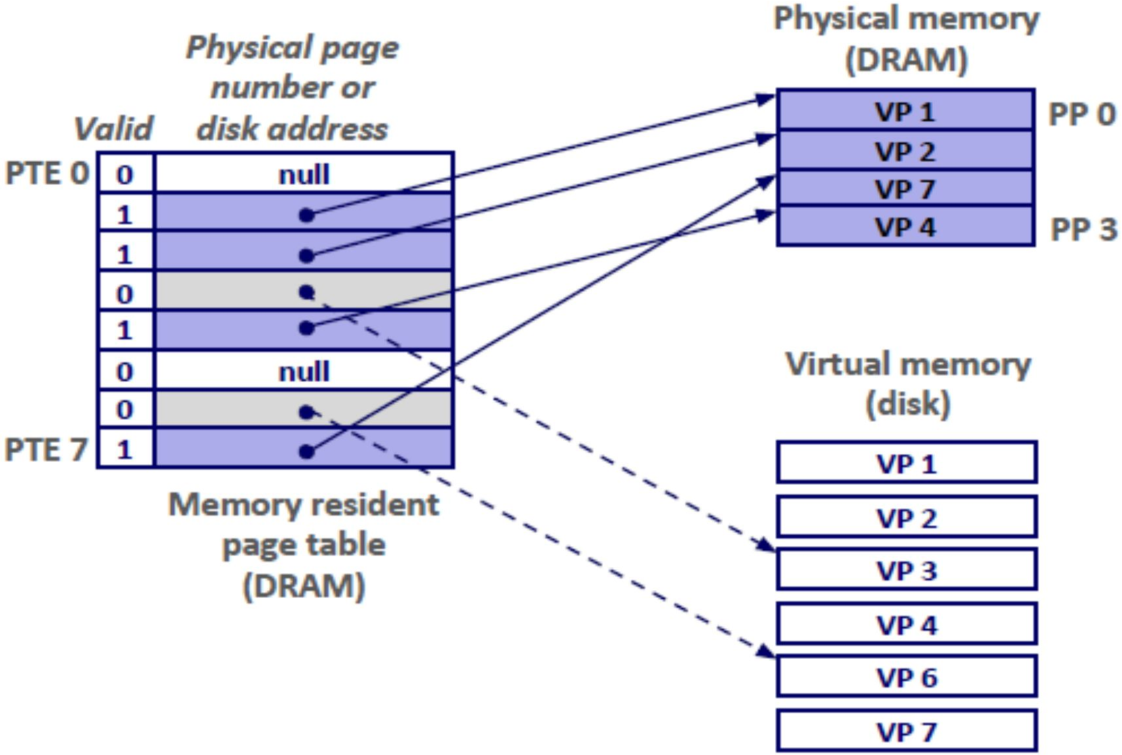
Page Table

- Suppose that you are given the following virtual address
  - 011001 00101000
- Which index of the page table will this address access?

Page Table

- Say the entry at the page table entry 25 has the following as the PPN:
    - 0110
- What is the reconstructed physical address?

Page Table



Physical page number or disk address
Valid
PTE 0
0
null
1
1
0
1
0
null
0
PTE 7
1
Memory resident page table (DRAM)
Physical memory (DRAM)
VP 1
VP 2
VP 7
VP 4
PP 0
PP 3
Virtual memory (disk)
VP 1
VP 2
VP 3
VP 4
VP 6
VP 7

Page Table

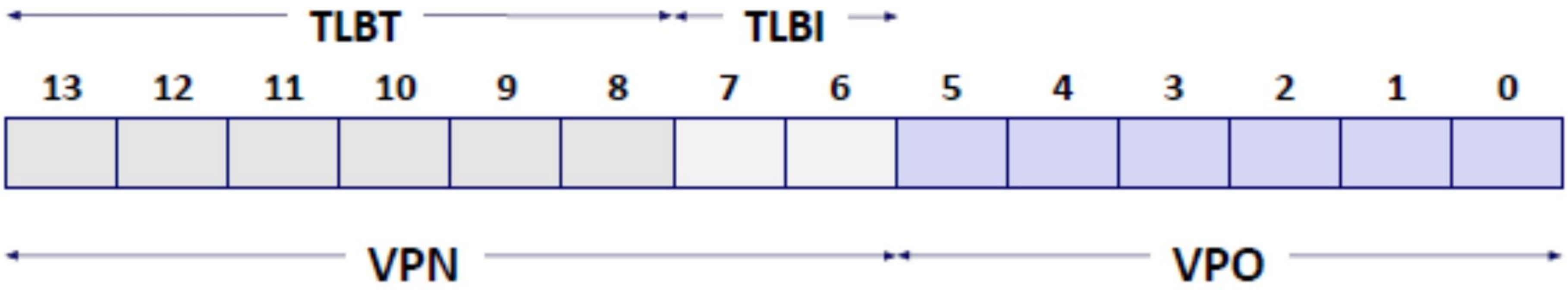
- Page Table is an entity stored in memory
- The physical address of the current process's page table is stored in a register
- Each process has its own page table
- A page table entry will either indicate:
    - Virtual memory is in memory at location [PPN : VPO]
    - Or Virtual page is not in memory (valid bit = 0). Go to disk

TLB

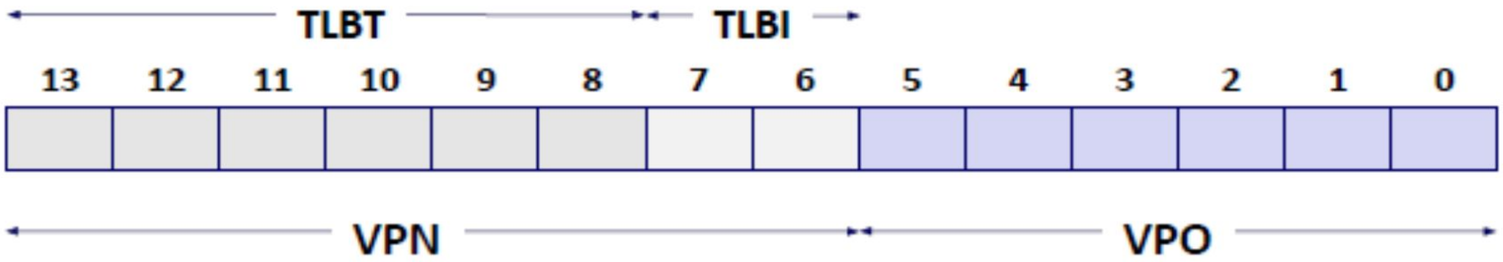
- Instead of having to go to the cache and then memory to get the page table, let’s have a special cache for the page table
- TLB: Translation Lookaside Buffer

TLB

- Accessing the TLB is exactly the same as accessing the cache. The virtual address is split into three components
  - TLBT (tag)
  - TLBI (index)
  - VPO (offset)

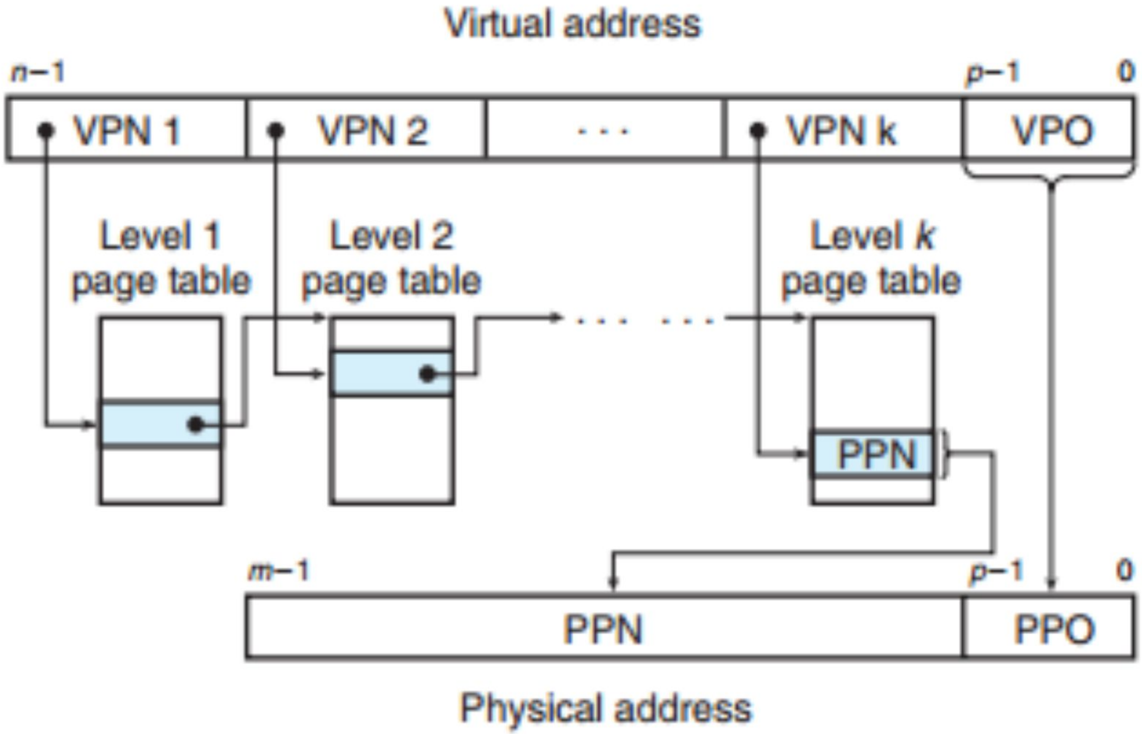


TLB



Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Hierarchical Page Table



Hierarchical Page Table

- Assume our VPN is 5-bit long
- In the non-hierarchical case, we would have 2^5 = 32 entries
- What if we are using a 2-level hierarchical page table with 1 L1 table and 1 L2 table?
- 2^3 entries (1 L1 table) + 1 * 2^2 (1 L2 table) = 12 total entries
- Hierarchical page tables are helpful if we are not using up all of our page table entries, which is most of the time

Performance Lab

- Any progress?

Performance Lab Common Questions

- How fast can you get using single-thread optimization?
  - I would say about ~2x
  - Some people have got up to ~4x
  - But it should be really hard to get better than ~4x
  - Multithreading is necessary to get full credit

Performance Lab Common Questions

- Why is my code not working?
    - Because what you want to do is actually different from what you are telling the computer to do
    - https://www.youtube.com/watch?v=cDA3_5982h8&feature=youtu.be

Performance Lab Common Questions

- Why am I getting segmentation fault?
    - You are most likely accessing an element that you are not supposed to
        - Array index out of bounds
        - Dereferencing an uninitialized pointer
    - How are we supposed to know what the problem is?
        - Add a lot of printf statements in different places in your code to find out exactly where the segmentation fault is happening

Performance Lab Common Questions

- How can I debug my program?
  - Try using gdb!
  - Or simply print out the variables and see if they have the correct values
  - printf(“Thread ID: %d\n”, thread_id);
  - printf(“Start index: %d\n”, start);
  - printf(“End index: %d\n”, end);