CS 61C Fall 2018

Virtual Memory

Discussion 13: November 26, 2018

1 Addressing

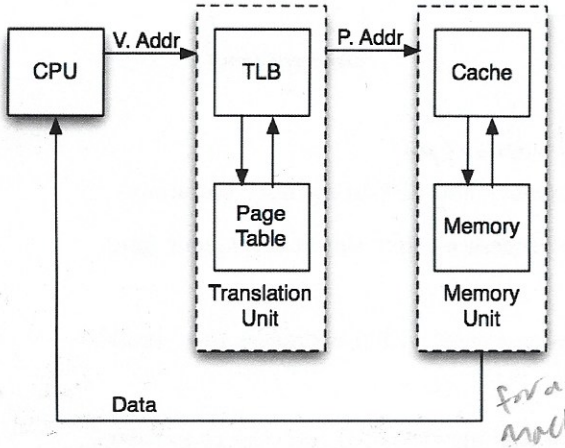
Virtual Address (VA) What your program uses

Virtual Page Number (VPN)	Page Offset

Physical Address (PA) What actually determines where in memory to go

Physical Page Number (PPN)	Page Offset

With 4 KiB pages and byte addresses, $2^{\text{page offset bits}} = 4096$, so there are 12 page offset bits. Translate virtual addresses (VA) to physical addresses (PA) using the translation lookaside buffer (TLB) and page table. Then, use the physical address to access memory as the program intended.



Pages

A chunk of memory or disk with a set size. Addresses in the same virtual page map to addresses in the same physical page. The page table determines the mapping.

Valid	Dirty	Permission Bits	PPN
— *Page entry (VPN: 0)* —			
— *Page entry (VPN: 1)* —			

Each stored row of the page table is called a **page table entry**. The page table is stored in memory: the OS sets a register telling the hardware the address of the first entry of the page table. The processor updates the "dirty" bit in the page table which lets the OS to know whether updating a page on disk is necessary. Each process gets its own page table.

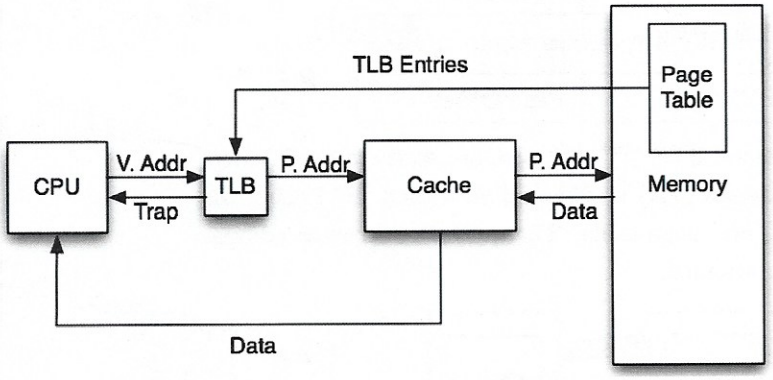
Protection Fault The page table entry for a virtual page has permission bits that prohibit the requested operation.

Page Fault The page table entry for a virtual page has its valid bit set to false. The entry is not in memory.

Translation Lookaside Buffer

A cache for the page table. Each block is a single page table entry. If an entry is not in the TLB, it's a TLB miss. Assuming fully associative:

TLB Valid	Tag (VPN)	Page Table Entry		
		Page Dirty	Permission Bits	PPN
— *TLB entry* —				
— *TLB entry* —				



1.1 What are three specific benefits of using virtual memory?

(well really the full address space).

- Illusion of infinite memory (bridges memory and disk in memory hierarchy).
- Simulates full address space for each process so that the linker/loader dont need to know about other programs.
- Enforces protection between processes and even within a process (e.g. read-only pages set up by the OS).

Isolation

1.2 What should happen to the TLB when a new value is loaded into the page table address register? Invalidate TLB Entrys

The valid bits of the TLB should all be set to 0. The page table entries in the TLB corresponded to the old page table, so none of them are valid once the page table address register points to a different page table

1.3 A processor has 16-bit addresses, 256 byte pages, and an 8-entry fully associative TLB with LRU replacement (the LRU field is 3 bits and encodes the order in which pages were accessed, 0 being the most recent). At some time instant, the TLB for the current process is the initial state given in the table below. Assume that all current page table entries are in the initial TLB. Assume also that all pages can be read from and written to. Fill in the final state of the TLB according to the access pattern below.

2^4 2^8

Question: How many bits does the TLB need to store?

Answer: VPN + PPN + valid + dirty + LRU
$8 + 8 + 1 + 1 + 3 = 21$

Free Physical Pages 0x17, 0x18, 0x19

Access Pattern

1. 0x11f0 (**Read**)
2. 0x1301 (**Write**)
3. 0x20ae (**Write**)
4. 0x2332 (**Write**)
5. 0x20ff (**Read**)
6. 0x3415 (**Write**)

Page = 256 = 2^8 so $\log_2(2^8) = 8$ bit offset.
Address space size − Page offset = Tag
16 − 8 = 8 bit tag.

Initial TLB

VPN	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	0
0x00	0x00	0	0	7
0x10	0x13	1	1	1
0x20	0x12	1	0	5
0x00	0x00	0	0	7
0x11	0x14	1	0	4
0xac	0x15	1	1	2
0xff	0xff	1	0	3

Final TLB

VPN	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	5
0x13	0x17	1	1	3
0x10	0x13	1	1	6
0x20	0x12	1	1	1
0x23	0x18	1	1	2
0x11	0x14	1	0	4
0xac	0x15	1	1	7
0x34	0x19	1	1	0

On miss, we use LRU Replacement to put the new page in.

Steps:
1) Determine Vaddr tag.
2) check TLB for VPN.
3) if exist check valid
↳ Hit
else miss & need new page.
4) translate Vaddr to Paddr.

1. 0x11f0 (**Read**): hit, LRUs: 1, 7, 2, 5, 7, 0, 3, 4
2. 0x1301 (**Write**): miss, map VPN 0x13 to PPN 0x17, valid and dirty, LRUs: 2, 0, 3, 6, 7, 1, 4, 5
   because trying a given free PP ← dirty b/c it was a write. ← gotta evict it
3. 0x20ae (**Write**): hit, dirty, LRUs: 3, 1, 4, 0, 7, 2, 5, 6
4. 0x2332 (**Write**): miss, map VPN 0x23 to PPN 0x18, valid and dirty, LRUs: 4, 2, 5, 1, 0, 3, 6, 7
   ← was write.
5. 0x20ff (**Read**): hit, LRUs: 4, 2, 5, 0, 1, 3, 6, 7
6. 0x3415 (**Write**): miss and replace last entry, map VPN 0x34 to 0x19, dirty, LRUs, 5, 3, 6, 1, 2, 4, 7, 0

not in TLB & TLB full so evict & write to mem page back to disk. LRU.

since LRU is not dirty, we do not need to write to