CS 61C Spring 2019 — Virtual Memory

Discussion 13: April 24, 2019

1 Addressing

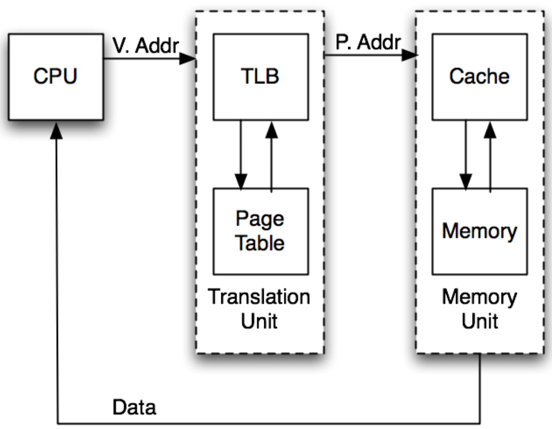
Virtual Address (VA) What your program uses

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, $2^{\text{page offset bits}} = 4096$B, so there are 12 page offset bits. To access some memory location, we first translate virtual addresses (VA) to physical addresses (PA) using the translation lookaside buffer (TLB) and page table. Then, we 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**. There are $2^{\text{VPN bits}}$ such entries in a page table. 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.

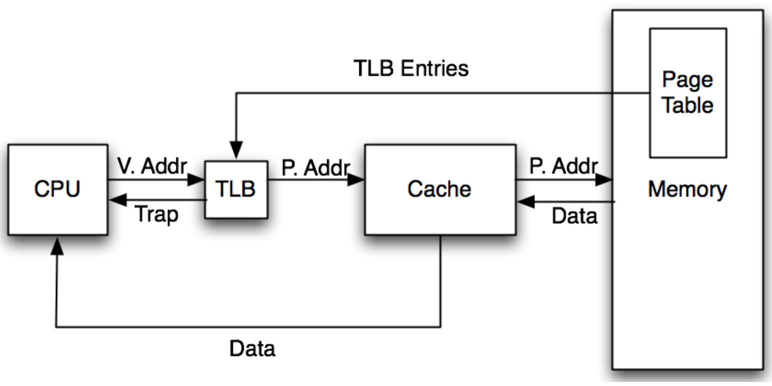
Protection Fault The page table entry for a virtual page has permission bits that prohibit the requested operation.

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:

<table>
<tr><td rowspan="2">TLB Valid</td><td rowspan="2">Tag (VPN)</td><td colspan="3">Page Table Entry</td></tr>
<tr><td>Page Dirty</td><td>Permission Bits</td><td>PPN</td></tr>
<tr><td colspan="5">— TLB entry —</td></tr>
<tr><td colspan="5">— TLB entry —</td></tr>
</table>



1.1 What are three specific benefits of using virtual memory?

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

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.

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**)

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

More MapReduce Practice

Given a person's unique int ID and a list of the IDs of their friends, compute the list of mutual friends between each pair of friends in a social network. You have access to the `intersection` function, which takes in two lists finds the set of elements that appear in both lists.

Declare any custom data types here:

```
FriendPair:
    int friendOne
    int friendTwo
```

```
map(int personID, list<int> friendIDs):
```

```
reduce(________________, ________________):
```