

Virtual Memory Overview

Virtual address (VA): What your program uses

Virtual Page Number	Page Offset
---------------------	-------------

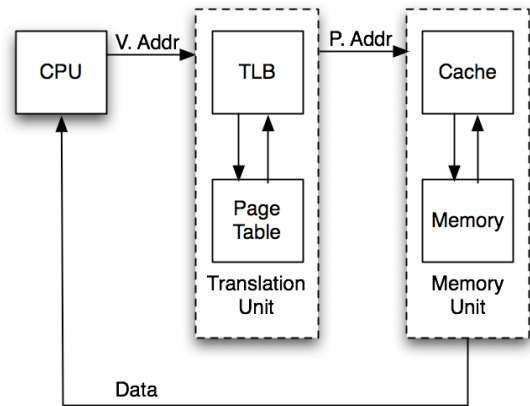
Physical address (PA): What actually determines where in memory to go

Physical Page Number	Page Offset
----------------------	-------------

With 4 KiB pages and byte addresses, $2^{(page\ offset\ bits)} = 4096$, so page offset bits = 12.

The Big Picture: Logical Flow

Translate VA to PA using the TLB and Page Table. Then use PA to access memory as the program intended.



Pages

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

The Page Table

Index = Virtual Page Number (VPN) (not stored)	Page Valid	Page Dirty	Permission Bits (read, write, ...)	Physical Page Number (PPN)
0				
1				
2				
...				
(Max virtual page number)				

Each stored row of the page table is called a **page table entry** (the grayed section is the first 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 “page dirty” in the page table: “page dirty” bits are used by the OS to know whether updating a page on disk is necessary. Each process gets its own page table.

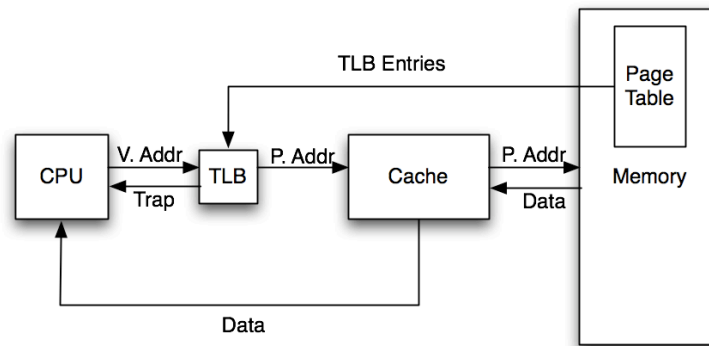
- **Protection Fault**--The page table entry for a virtual page has permission bits that prohibit the requested operation
- **Page Fault**--The page table entry for a virtual page has its valid bit set to false. The entry is not in memory.

The Translation Lookaside Buffer (TLB)

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 Entry Valid	Tag = Virtual Page Number	Page Table Entry		
		Page Dirty	Permission Bits	Physical Page Number
...

The Big Picture Revisited



Exercises

- 1) What are three specific benefits of using virtual memory?
- 2) What should happen to the TLB when a new value is loaded into the page table address register?
- 5) 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

Hamming ECC

Recall the basic structure of a Hamming code. Given bits $1, \dots, m$, the bit at position 2^n is parity for all the bits with a 1 in position n . For example, the first bit is chosen such that the sum of all odd-numbered bits is even.

Bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	<u>P1</u>	<u>P2</u>	D1	<u>P4</u>	D2	D3	D4	<u>P8</u>	D5	D6	D7	D8	D9	D10	D11
P1	X		X		X		X		X		X		X		X
P2		X	X			X	X			X	X			X	X
P4				X	X	X	X					X	X	X	X
P8								X	X	X	X	X	X	X	X

1. How many bits do we need to add to 0011_2 to allow single error correction?
2. Which locations in 0011_2 would parity bits be included?
3. Which bits does each parity bit cover in 0011_2 ?
4. Write the completed coded representation for 0011_2 to enable single error correction.
5. How can we enable an additional double error detection on top of this?
6. Find the original bits given the following SEC Hamming Code: 0110111_2
7. Find the original bits given the following SEC Hamming Code: 1001000_2
8. Find the original bits given the following SEC Hamming Code: 010011010000110_2