Virtual Memory Overview

Virtual address (VA): What your program uses

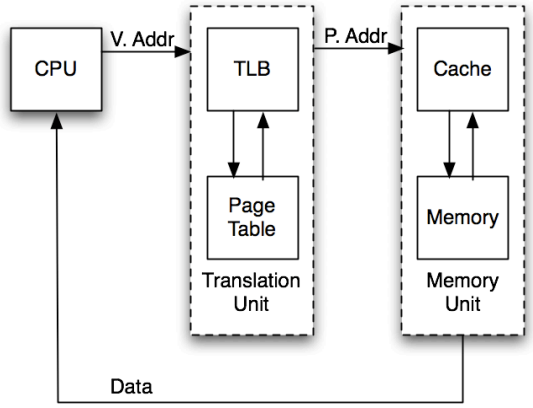
Virtual Page Number	Page Offset

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.

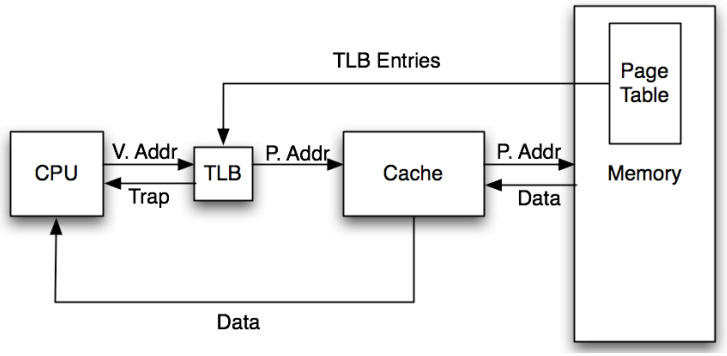
- **Protection Fault**--The page table entry for a virtual page has permission bits that prohibit the requested operation
- **Page Fault**--The page table entry for a virtual page has its valid bit set to false. The entry is not in memory.

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?

Bridges memory and disk in memory hierarchy.
Simulates full address space for each process.
Enforces protection between processes.

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

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.

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

Access pattern:

Read	0x11f0
Write	0x1301
Write	0x20ae
Write	0x2332
Read	0x20ff
Write	0x3415

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	0x16	1	0	3

Read 0x11f0: hit, LRUs: 1,7,2,5,7,0,3,4
Write 0x1301: miss, map VPN 0x13 to PPN 0x17, valid and dirty, LRUs: 2,0,3,6,7,1,4,5
Write 0x20ae: hit, dirty, LRUs: 3,1,4,0,7,2,5,6
Write 0x2332: miss, map VPN 0x23 to PPN 0x18, valid and dirty, LRUs: 4,2,5,1,0,3,6,7
Read 0x20ff: hit, LRUs: 4,2,5,0,1,3,6,7
Write 0x3415: miss and replace last entry, map VPN 0x34 to 0x19, dirty, LRUs, 5,3,6,1,2,4,7,0

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

Hamming ECC

Recall the basic structure of a Hamming code. Given bits 1, . . . , 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	P1	P2	D1	P4	D2	D3	D4	P8	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 00112 to allow single error correction?
   Parity Bits: 3
2. Which locations in 00112 would parity bits be included?
   Using P for parity bits: $PP0P011_2$
3. Which bits does each parity bit cover in 00112?
   Parity bit #1: 1, 3, 5, 7
   Parity bit #2: 2, 3, 6, 7
   Parity bit #3: 4, 5, 6, 7
4. Write the completed coded representation for 00112 to enable single error correction.
   1000011_2
5. How can we enable an additional double error detection on top of this?
   Add an additional parity bit over the entire sequence.
6. Find the original bits given the following SEC Hamming Code: 01101112
   Parity group 1: error
   Parity group 2: okay
   Parity group 4: error
   Incorrect bit: 1 + 4 = 5, change bit 5 from 1 to 0: 0110011_2
   $0110011_2 \rightarrow 1011_2$
7. Find the original bits given the following SEC Hamming Code: 10010002
   Parity group 1: error
   Parity group 2: okay
   Parity group 4: error
   Incorrect bit: 1 + 4 = 5, change bit 5 from 1 to 0: 1001100_2
   $1001100_2 \rightarrow 0100_2$
8. Find the original bits given the following SEC Hamming Code: 0100110100001102
   Parity group 1: okay
   Parity group 2: error
   Parity group 4: okay
   Parity group 8: error
   Incorrect bit: 2 + 8 = 10, change bit 10 from 0 to 1: 0100110101001102
   $0100110101001102 \rightarrow 011001001102$