Virtual Memory Overview

Virtual address (VA): What your program uses

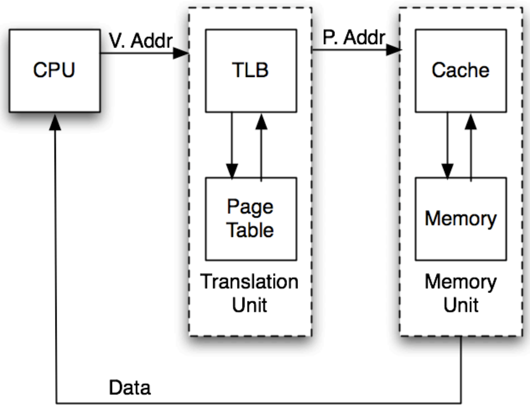
Virtual Page Number	Page Offset

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.

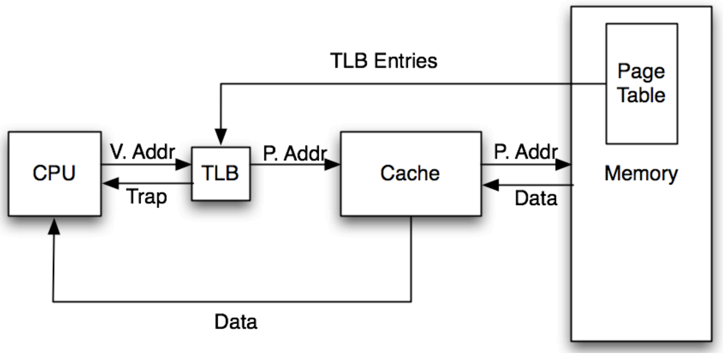
- **Protection Fault**--The page table entry for a virtual page has permission bits that prohibit the requested operation
- **Page Fault**--The page table entry for a virtual page has its valid bit set to false. The entry is not in memory.

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?

Illusion of infinite memory (bridges memory and disk in memory hierarchy).
Simulates full address space for each process so that the linker/loader don't 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).

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.

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:

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
0xac	0x15	1	1	7
0x34	0x19	1	1	0

I/O

1. Fill this table of polling and interrupts.

Operation	Definition	Pro/Good for	Con
Polling	Forces the hardware to wait on ready bit (alternatively, if timing of device is known – the ready bit can be polled at the frequency of the device).	- Low Latency - Low overhead when data is available - Good For: devices that are always busy or when you can't make progress until the device replies	- Can't do anything else while polling - Can't sleep while polling (CPU always at full speed)
Interrupts	Hardware fires an "exception" when it becomes ready. CPU changes $PC to execute code in the interrupt handler when this occurs.	PRO: - Can do useful work while waiting for response - Can wait on many things at once - Good for: Devices that take a long time to respond, especially if you can do other work while waiting.	-nondeterministic when interrupt occurs -interrupt handler has some overhead (e.g. saves all registers, flush pipeline, etc.) - Higher latency/event - Worse throughput

2. Memory Mapped I/O

   Certain memory addresses correspond to registers in I/O devices and not normal memory.

   0xFFFF0000 – Receiver Control:
   LSB is the ready bit, there may be other bits set that we don't need right now.

   0xFFFF0004 – Receiver Data:
   Received data stored at lowest byte.

   0xFFFF0008 – Transmitter Control
   LSB is the ready bit, there may be other bit set that we don't need right now.

   0xFFFF000C – Transmitter Data
   Transmitted data stored at lowest byte.

   Write RISC-V code to read a byte from the receiver and immediately send it to the transmitter.

```
               lui $t0 0xffff
receive_wait:  #poll on ready of receiver
               lw $t1 0($t0)
               andi $t1 $t1 1
               beq $t1 $zero receive_wait
               lb $t2 4($t0)           #load data
transmit_wait: #poll on ready of transmitter
```

```
lw $t1 8($t0)
andi $t1 $t1 1
beq $t1 $zero transmit_wait
#write to transmitter
sb $t2 12($t0)
```