CSE 351 Virtual Section – Virtual Memory

Sorry for assigning this during Thanksgiving, but we want you to practice while it's still fresh in your minds. ☺

See the "Virtual Memory Overview" for material review

Exercises:

1) Name three specific benefits of using virtual memory:

   Bridges memory and disk in memory hierarchy (treats Mem as a cache for Disk).
   Simulates full address space for each process (simplifies memory management for programmers).
   Enforces protection (and sharing) between processes.

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

   Updating the PTBR means that we are switching processes. The page table entries currently in the TLB corresponded to the old page table & process, so none of them are valid once we context switch. **The valid bits of the TLB should all be set to 0 (invalidated).**

3) Fill in the formulas below using *descriptions*, not variables:

   Page offset bits = $\log_2$(bytes in a page)

   Virtual address bits = virtual page number bits + page offset bits

   Physical address bits = physical page number bits + page offset bits

   Virtual page number bits = $\log_2$(number of virtual pages)

   Entries in a page table = number of virtual pages

4) Fill in the following table:

VA width (n)	PA width (m)	Page size (P)	VPN width	PPN width	Bits in PTE (assume V, D, R, W, X)
32	32	16 KiB	18	18	23
32	26	8 KiB	19	13	18
36	32	32 KiB	21	17	22
40	36	32 KiB	25	21	26
64	40	64 KiB	48	24	29

$p = \log_2 P$, VPN width $= n - p$, PPN width $= m - p$, bits in PTE = PPN width + 5

5) **Processor:** 16-bit addresses, 256-byte pages

TLB: 8-entry fully associative with LRU replacement

- Track LRU using 3 bits to encode the order in which pages were accessed, with 0 being the most recent

At some time instant, the TLB for the current process is in the initial state given below.

Assume that all page table entries NOT in the initial TLB start as invalid.

- OS will assign new pages at the lowest available PPN starting at 0x17

Assume all pages can be read from and written to (ignore protection).

Fill in the final state of the TLB according to the access pattern below:

Access pattern:

1. Read `0x11F0`
2. Write `0x1301`
3. Write `0x20AE`
4. Write `0x2332`
5. Read `0x20FF`
6. Write `0x3415`

Initial TLB:

TLBT	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

n = 16, p = 8, so VPN width = $n - p$ = 8 bits. TLB is fully associative, so S = 1 and TLBT = VPN.

1. Read `0x11F0` → TLBT = `0x11`, TLB Hit

TLBT	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	1
0x00	0x00	0	0	7
0x10	0x13	1	1	2
0x20	0x12	1	0	5
0x00	0x00	0	0	7
0x11	0x14	1	0	**0**
0xAC	0x15	1	1	3
0xFF	0x16	1	0	4

2. Write `0x1301` → TLBT = `0x13`, TLB Miss
   Map `0x13` to PPN `0x17` & fill invalid slot

TLBT	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	2
0x13	0x17	1	1	**0**
0x10	0x13	1	1	3
0x20	0x12	1	0	6
0x00	0x00	0	0	7
0x11	0x14	1	0	1
0xAC	0x15	1	1	4
0xFF	0x16	1	0	5

3. Write `0x20AE` → TLBT = `0x20`, TLB Hit

TLBT	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	3
0x13	0x17	1	1	1
0x10	0x13	1	1	4
0x20	0x12	1	1	**0**
0x00	0x00	0	0	7
0x11	0x14	1	0	2
0xAC	0x15	1	1	5
0xFF	0x16	1	0	6

4. Write `0x2332` → TLBT = `0x23`, TLB Miss
   Map `0x23` to PPN `0x18` & fill invalid slot

TLBT	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	4
0x13	0x17	1	1	2
0x10	0x13	1	1	5
0x20	0x12	1	1	1
0x23	0x18	1	1	**0**
0x11	0x14	1	0	3
0xAC	0x15	1	1	6
0xFF	0x16	1	0	7

5. Read 0x20FF → TLBT = 0x20, TLB Hit

TLBT	PPN	Valid	Dirty	LRU
0x01	0x11	1	1	4
0x13	0x17	1	1	2
0x10	0x13	1	1	5
0x20	0x12	1	1	0
0x23	0x18	1	1	1
0x11	0x14	1	0	3
0xAC	0x15	1	1	6
0xFF	0x16	1	0	7

6. Write 0x3415 → TLBT = 0x34, TLB Miss
   Map 0x34 to PPN 0x19 and replace LRU

TLBT	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

Final TLB:

TLBT	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