TCSS 422: OPERATING SYSTEMS

Translation Lookaside Buffer (TLB)

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

OBJECTIVES

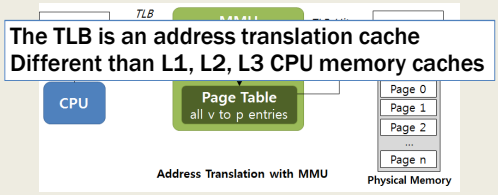
- TLB Algorithm
- TLB Tradeoffs
- TLB Context Switch

TRANSLATION LOOKASIDE BUFFER

- Legacy name…
- Better name, "Address Translation Cache"
- TLB is an on CPU cache of address translations
  - virtual → physical memory

TRANSLATION LOOKASIDE BUFFER (TLB)

- Part of the CPU's Memory Management Unit (MMU)
- Goal: Reduce accesses to the page tables



TLB BASIC ALGORITHM

- For: array based page table
- Hardware managed TLB

```
1: VPN = (VirtualAddress & VPN_MASK ) >> SHIFT
2: (Success , TlbEntry) = TLB_Lookup(VPN)
3:    if(Success == True){ // TLB Hit
4:    if(CanAccess(TlbEntry.ProtectBits) == True ){
5:        Offset = VirtualAddress & OFFSET_MASK
6:        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:        AccessMemory( PhysAddr )
8:    }else RaiseException(PROTECTION_ERROR)
```

Generate the physical address to access memory

TLB BASIC ALGORITHM - 2

```
11:    }else{ //TLB Miss
12:        PTEAddr = PTBR + (VPN * sizeof(PTE))
13:        PTE = AccessMemory(PTEAddr)
14:        (…)  // Check for, and raise exceptions…
15:    }else{
16:        TLB_Insert( VPN , PTE.PFN , PTE.ProtectBits)
17:        RetryInstruction()
18:    }
19:}
```

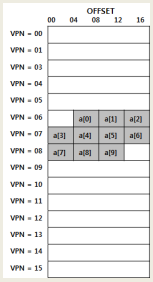
Retry the instruction… (requery the TLB)

TLB – ADDRESS TRANSLATION CACHE

- Key detail:
- For a TLB miss, we access the page table to populate the TLB… we then **requery** the TLB
- **All address translations go through the TLB**

TLB EXAMPLE

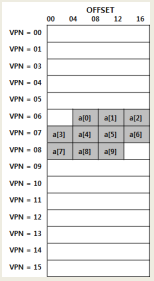
```
0:      int sum = 0 ;
1:      for( i=0; i<10; i++){
2:              sum+=a[i];
3:      }
```



- Consider:
- Address space: 256-byte
  - Addressable using 8 total bits
  - 4 bits for the VPN
- Page size: 16 bytes
  - Offset is addressable using 4-bits
- Array: 10 x 4-byte integers

TLB EXAMPLE - 2

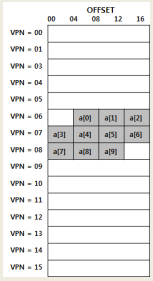
```
0:      int sum = 0 ;
1:      for( i=0; i<10; i++){
2:              sum+=a[i];
3:      }
```



- Consider the code above:
- Initially the TLB does not know where a[] is
- Consider the accesses:
- a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9]
- **How many pages are accessed?**
- **What happens when accessing a page not in the TLB?**

TLB EXAMPLE - 3

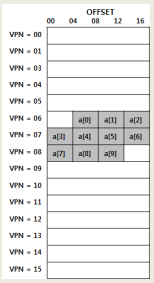
```
0:      int sum = 0 ;
1:      for( i=0; i<10; i++){
2:              sum+=a[i];
3:      }
```



- For the accesses: a[0], a[1], a[2], a[3], a[4],
- a[5], a[6], a[7], a[8], a[9]
- How many are hits?
- How many are misses?
- What is the hit rate? (%)

TLB EXAMPLE - 4

```
0:      int sum = 0 ;
1:      for( i=0; i<10; i++){
2:              sum+=a[i];
3:      }
```



- What factors affect the hit/miss rate?

TLB TRADEOFFS

- **Page size**
- Larger page sizes increase the probability of a TLB hit
- Example: 16-bytes (very small), 4096-bytes (common)
- Larger sizes increase memory requirement of offset

TLB TRADEOFFS - 2

- **Spatial locality**
- Accessing addresses local to each improves the hit rate.
- Intel example - nehalem architecture: two-level TLB
- First level TLB: split into data and code instruction cache
- Second level TLB: shared TLB for code and data
- Each level may have a small and large page cache

TLB TRADEOFFS - 3

- **Temporal locality**
- Higher cache hit ratios are expected for repeated memory accesses close in time
- Can dramatically improve performance for "second iteration"

HW CACHE TRADEOFF

- **Speed vs. size**

Speed ⟷ Size

- In order to be fast, caches must be small
- Too large of a cache will mimic physical memory
- Limitations for on chip memory

HANDLING TLB MISS

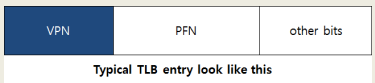
- Historical view
- CISC – Complex instruction set computer
  - Intel x86 CPUs
  - Traditionally have provided on CPU HW instructions and handling of TLB misses
  - HW has a page table register to store location of page table

HANDLING TLB MISS - 2

- RISC – Reduced instruction set computer
  - ARM CPUs
  - Traditionally the OS handles TLB misses
  - HW raises an exception
  - Trap handler is executed to handle the miss
- Advantages
  - HW Simplicity: simply needs to raise an exception
  - Flexibility: OS provided page table implementations can use different data structures, etc.

TLB CONTENTS

- TLB typically may have 32, 64, or 128 entries
- HW searches the entire TLB in parallel to find the translation
- Other bits
  - Valid bit: valid translation?
  - Protection bit: read/execute, read/write
  - Address-space identifier: identify entries by process
  - Dirty bit



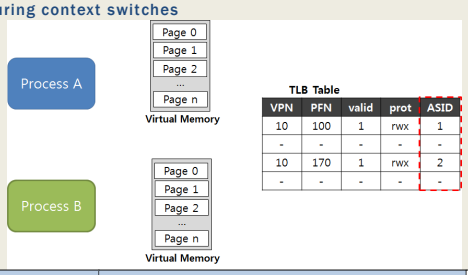
Typical TLB entry look like this

TLB: CONTEXT SWITCH

- TLB stores address translations for current running process
- A context/switch to a new process invalidates the TLB
- Must "switch" out the TLB
- TLB flush
  - Flush TLB on context switches, set all entries to 0
  - Requires time to flush
  - TLB must be reloaded for each C/S
  - If process doesn't reside in the CPU for long, the TLB may not get reloaded
- Share TLB across processes during C/S
  - Use address space identifier (AID) to tag TLB entries by process

TLB: CONTEXT SWITCH - 2

- Address space identified (AIS) enables data state to perist during context switches



TLB Table

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
-	-	-	-	-
10	170	1	rwx	2
-	-	-	-	-

SHARED MEMORY SPACE

VPN	PFN	valid	prot	ASID
10	101	1	rwx	1
-	-	-	-	-
50	101	1	rwx	2
-	-	-	-	-

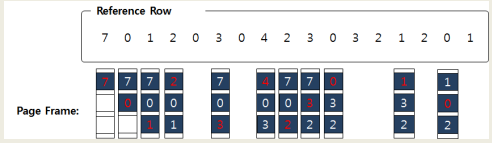
- When processes share a code page
  - Shared libraries ok
  - Code pages typically are RX, not RWX

Sharing of pages is useful as it reduces the number of physical pages in use.

CACHE REPLACEMENT

- When TLB cache is full, how add a new address translation to the TLB?
- Least Recently Used (LRU)
  - Evict the oldest entry
- Note how the TLB is loaded / unloaded…
- Goal minimize miss rate, increase hit rate
- Random policy
  - Pick a candidate at random to free-up space in the TLB

LEAST RECENTLY USED



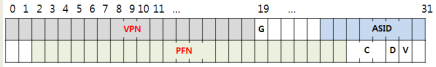
- RED – miss
- WHITE – hit
- For 3-page TLB, observe replacement

Total 11 TLB miss

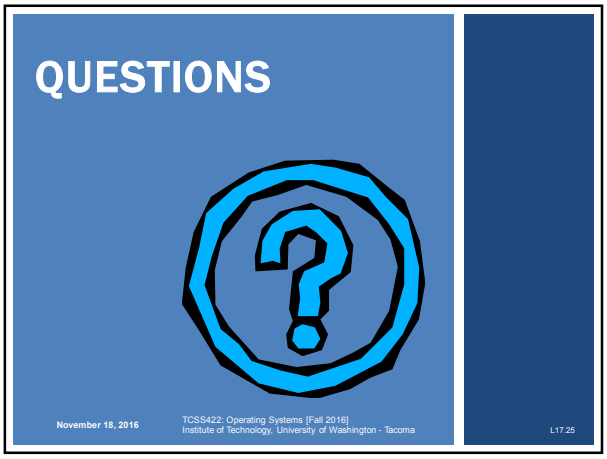
EXAMPLE TLB ENTRY – MIPS R4000

- Early 64-bit RISC processor

All 64 bits of this TLB entry(example of MIPS R4000)



Flag	Content
19-bit VPN	The rest reserved for the kernel.
24-bit PFN	Systems can support with up to 64GB of main memory($2^{24} * 4KB$ pages).
Global bit(G)	Used for pages that are globally-shared among processes.
ASID	OS can use to distinguish between address spaces.
Coherence bit(C)	determine how a page is cached by the hardware.
Dirty bit(D)	marking when the page has been written.
Valid bit(V)	tells the hardware if there is a valid translation present in the entry.


QUESTIONS
November 18, 2016
TCSS422: Operating Systems [Fall 2016]
Institute of Technology, University of Washington - Tacoma
L17.25