TCSS 422: OPERATING SYSTEMS

Translation Lookaside Buffer (TLB)

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

OBJECTIVES

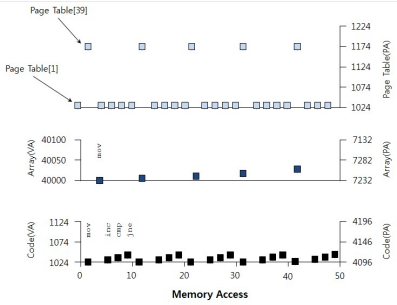
- Chapter 19
  - TLB Algorithm
  - TLB Tradeoffs
  - TLB Context Switch

TRANSLATION LOOKASIDE BUFFER

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

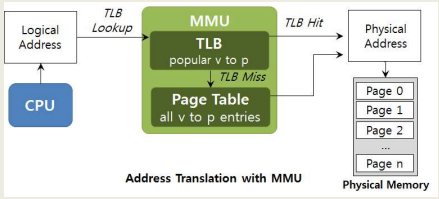
TRANSLATION LOOKASIDE BUFFER - 2

- Goal: Reduce access to the page tables
- Example: 50 accesses for 5 for-loop iterations
- Move lookups from RAM to TLB



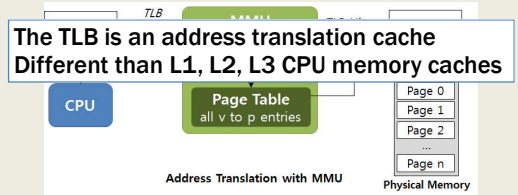
TRANSLATION LOOKASIDE BUFFER (TLB)

- Part of the CPU's Memory Management Unit (MMU)
- Address translation cache



TRANSLATION LOOKASIDE BUFFER (TLB)

- Part of the CPU's Memory Management Unit (MMU)
- Address translation cache



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:
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:      }
```

- Example:
- Program address space: 256-byte
  - Addressable using 8 total bits (2^8)
  - 4 bits for the VPN (16 total pages)
- Page size: 16 bytes
  - Offset is addressable using 4-bits
- Store an array: of (10) 4-byte integers

	OFFSET 00	04	08	12	16
VPN = 00					
VPN = 01					
VPN = 03					
VPN = 04					
VPN = 05					
VPN = 06		a[0]	a[1]	a[2]	
VPN = 07	a[3]	a[4]	a[5]	a[6]	
VPN = 08	a[7]	a[8]	a[9]		
VPN = 09					
VPN = 10					
VPN = 11					
VPN = 12					
VPN = 13					
VPN = 14					
VPN = 15					

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?**

	OFFSET 00	04	08	12	16
VPN = 00					
VPN = 01					
VPN = 03					
VPN = 04					
VPN = 05					
VPN = 06		a[0]	a[1]	a[2]	
VPN = 07	a[3]	a[4]	a[5]	a[6]	
VPN = 08	a[7]	a[8]	a[9]		
VPN = 09					
VPN = 10					
VPN = 11					
VPN = 12					
VPN = 13					
VPN = 14					
VPN = 15					

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? (%)
  - 70% (3 misses one for each VP, 7 hits)

	OFFSET 00	04	08	12	16
VPN = 00					
VPN = 01					
VPN = 03					
VPN = 04					
VPN = 05					
VPN = 06		a[0]	a[1]	a[2]	
VPN = 07	a[3]	a[4]	a[5]	a[6]	
VPN = 08	a[7]	a[8]	a[9]		
VPN = 09					
VPN = 10					
VPN = 11					
VPN = 12					
VPN = 13					
VPN = 14					
VPN = 15					

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?
  - Page size
  - Data locality
  - Temporal locality

	OFFSET 00	04	08	12	16
VPN = 00					
VPN = 01					
VPN = 03					
VPN = 04					
VPN = 05					
VPN = 06		a[0]	a[1]	a[2]	
VPN = 07	a[3]	a[4]	a[5]	a[6]	
VPN = 08	a[7]	a[8]	a[9]		
VPN = 09					
VPN = 10					
VPN = 11					
VPN = 12					
VPN = 13					
VPN = 14					
VPN = 15					

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 other improves the hit rate.
- Consider random vs. sequential array access
- What happens when the data size exceeds the TLB size?
  - E.g. 1st level TLB caches 64 4KB page addresses
  - Single program can cache data lookups for 256 KB

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"

EXAMPLE: LARGE ARRAY ACCESS

- Example: Consider an array of a custom struct where each struct is 64-bytes. Consider sequential access for an array of 8,192 elements stored contiguously in memory:
- 64 structs per 4KB page
- 128 total pages
- TLB caches stores a maximum of 64 - 4KB page lookups
- How many hits vs. misses for sequential array iteration?
  - 1 miss for every 64 array accesses, 63 hits
  - Complete traversal: 128 total misses, 8,064 hits (98.4% hit ratio)

TLB EXAMPLE IMPLEMENTATIONS

- Intel Nehalem microarchitecture 2008 – multi level TLBs
  - First level TLB: separate cache for data (DTLB) and code (ITLB)
  - Second level TLB: shared TLB (STLB) for data and code
  - Multiple page sizes (4KB, 2MB)
  - Page Size Extension (PSE) CPU flag for larger page sizes

Cache		Page Size	
Name	Level	4 KB	2 MB
DTLB	1st	64	32
ITLB	1st	128	7 / logical core
STLB	2nd	512	none

- Intel Haswell microarchitecture 22nm 2013
  - Two level TLB
  - Three page sizes (4KB, 2MB, 1GB)
- Without large page sizes consider the # of TLB entries to address 1.9 MB of memory…

Cache		Page size		
Name	Level	4 KB	2 MB	1 GB
DTLB	1st	64	32	4
ITLB	1st	128	8 / logical core	none
STLB	2nd	1024		none

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

VPN	PFN	other bits

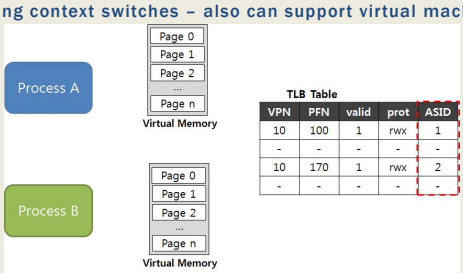
Typical TLB entry look like this

TLB: ON 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 not in CPU for long, the TLB may not get reloaded
- **Alternative**: be lazy…
  - Don't flush TLB on C/S
  - Share TLB across processes during C/S
  - Use address space identifier (ASID) to tag TLB entries by process

TLB: CONTEXT SWITCH - 2

- Address space identifier (ASID): enables TLB data to persist during context switches – also can support virtual machines



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
-	-	-	-	-

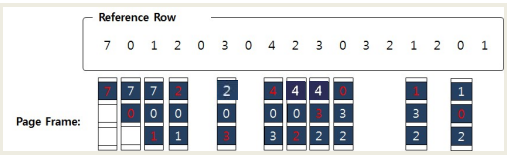
- When processes share a code page
  - Shared libraries ok
  - Code pages typically are RX, not RWX

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

CACHE REPLACEMENT POLICIES

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

LEAST RECENTLY USED

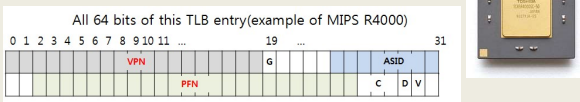


- RED – miss
- WHITE – hit
- For 3-page TLB, observe replacement

11 TLB miss, 5 TLB hit

EXAMPLE TLB ENTRY – MIPS R4000

- Early 64-bit RISC processor



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