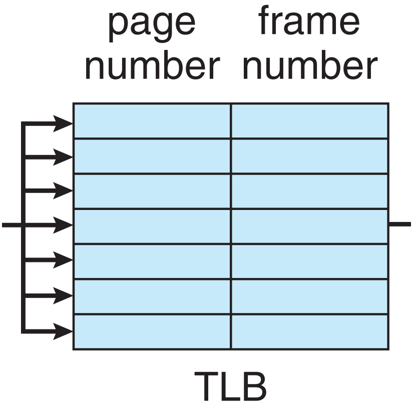


CSL 301

OPERATING SYSTEMS

Lecture 11
TLB

Instructor
Dr. Dhiman Saha

Image Source: Google

Issues?

Space overheads Issue 1

Storing PT in memory wastes valuable memory space.

Issues?

> Factor 2 slow down Issue 2

For every memory reference, paging requires us to perform one extra memory reference in order to first fetch the translation from the page table

Attempt #6: Translation-lookaside Buffer

Faster Paging

How to speed-up Address Translation?

- How can we speed up address translation?
- Avoid the **extra memory reference** that paging seems to require
- What hardware support is required?
- What OS involvement is needed?

TLB Translation-lookaside buffer

- Hardware support
- Part of the chip's memory-management unit
- Basically, a **hardware cache** of popular virtual-to-physical address translations
- A.k.a address-translation cache

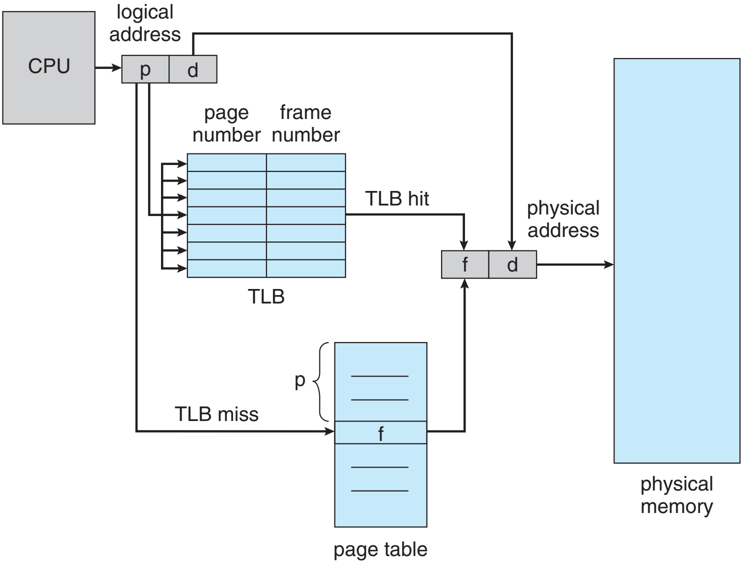
TLB Translation-lookaside buffer

- Hardware support
- Part of the chip's memory-management unit
- Basically, a **hardware cache** of popular virtual-to-physical address translations
- A.k.a address-translation cache

Idea

The hardware first checks the TLB to see if the desired translation is held therein.

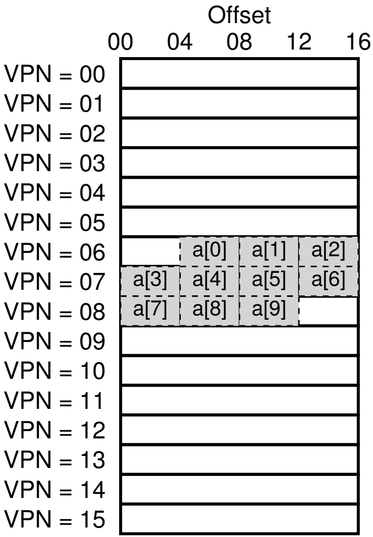
Paging hardware with TLB



HW Managed TLB Control Flow Algorithm

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)   // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                  // TLB Miss
    PTEAddr = PTBR + (VPN * sizeof(PTE))
    PTE = AccessMemory(PTEAddr)
    if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
    else if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
    else
        TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
        RetryInstruction()
```

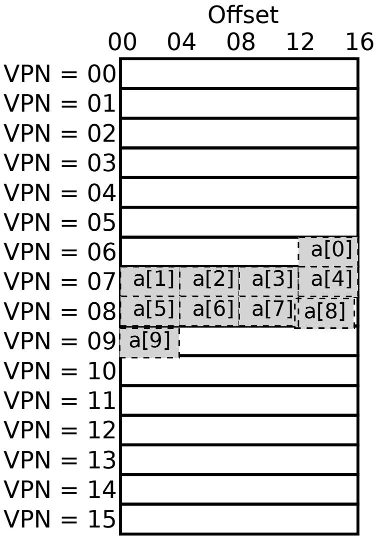
Example — Integer Array of 10 elements



VPN = 00				
VPN = 01				
VPN = 02				
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				

What is the TLB activity pattern?

Example — Integer Array of 10 elements



What is the TLB activity pattern?

Points to Ponder

- TLB Hit Vs TLB Miss

Points to Ponder

- TLB Hit Vs TLB Miss
- Hit rate

Points to Ponder

- TLB Hit Vs TLB Miss
- Hit rate
- Effect of page-size

Points to Ponder

- TLB Hit Vs TLB Miss
- Hit rate
- Effect of page-size
- Spatial Locality

Points to Ponder

- TLB Hit Vs TLB Miss
- Hit rate
- Effect of page-size
- Spatial Locality
- Temporal Locality

Points to Ponder

- TLB Hit Vs TLB Miss
- Hit rate
- Effect of page-size
- Spatial Locality
- Temporal Locality

Effective Memory Access Time

How to calculate?

Locality is Key

Temporal Locality

If a program accesses a memory location, it is likely to access that same location again soon. *Example: Instructions in a loop, loop variables.*

Spatial Locality

If a program accesses a memory location, it is likely to access nearby memory locations soon. *Example: Accessing elements of an array sequentially.*

Caches, including the TLB, are effective because most programs exhibit both temporal and spatial locality.

Who Handles The TLB Miss?

- Hardware Vs Software

CISC Complex-instruction set computers

- Early systems
- TLB miss handled by hardware
- via a page-table base register

RISC Reduced-instruction set computers

- Modern architectures
- Software-managed TLB
- Hardware simply raises an exception
- Trap handling mechanism takes over

Who Handles a TLB Miss?

Hardware-Managed TLB

- e.g., x86, ARM
- Hardware knows the page table format and location.
- On a miss, the hardware "walks" the page table to find the translation.
- Updates the TLB and retries the instruction.
- **Fast but inflexible.**

Software-Managed TLB

- e.g., MIPS, SPARC
- On a miss, hardware raises a TLB miss exception.
- The OS trap handler finds the translation in its own data structures (page tables).
- Uses special, privileged instructions to update the TLB.
- **Flexible but slower miss handling.**

OS Handled TLB Control Flow Algorithm

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)   // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                  // TLB Miss
    RaiseException(TLB_MISS)
```

Points to Ponder

Does the return-from-trap instruction in a TLB miss needs to be a little different than the return-from-trap when servicing a system call?

Points to Ponder

Can an infinite chain of TLB misses occur with a OS handled TLB miss?

- Possible Solutions

Points to Ponder

What are the advantages of a software-managed TLB-miss

TLB Contents

- Typically 32, 64, 128 entries
- Fully Associative. What does this mean?

TLB Contents

- Typically 32, 64, 128 entries
- Fully Associative. What does this mean?

TLB Entry

VPN | PFN | other bits

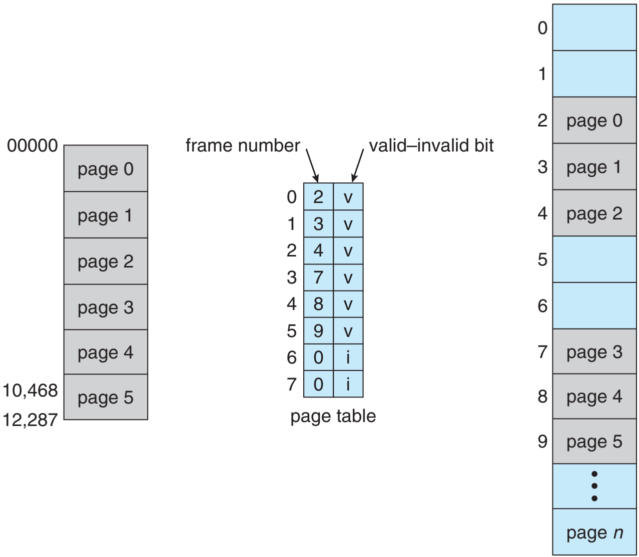
TLB Contents

- Typically 32, 64, 128 entries
- Fully Associative. What does this mean?

Other bits

- Valid. Is this same as page valid bit (Refer next slide)?
- Protection
- ASID (Described next)
- Dirty

Valid bit in Paging



00000

page 0

page 1

page 2

page 3

page 4

10,468

page 5

12,287

frame number

valid–invalid bit

	frame number	valid–invalid bit
0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

page table

0	
1	
2	page 0
3	page 1
4	page 2
5	
6	
7	page 3
8	page 4
9	page 5
	⋮
	page *n*

What happens on Context Switches?

How to handle this?

- When context-switching between processes, the translations in the TLB for the last process are **not meaningful to** the about-to-be-run process.

VPN	PFN	valid	prot
10	100	1	rwx
—	—	0	—
10	170	1	rwx
—	—	0	—

Solution #1

Flushing the TLB

Flushing the TLB

How?

- Set all valid bits to 0

Solution #1 Flushing the TLB

How?

- Set all valid bits to 0

When

- In SW use explicit instruction
- In HW while page-table base reg is updated

Flushing the TLB

How?

- Set all valid bits to 0

When

- In SW use explicit instruction
- In HW while page-table base reg is updated

Issue

- Performance overhead
- Each time a process runs, it **must incur TLB misses** as it touches its data and code pages.

Solution #2 Address Space Identifier (ASID)

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
—	—	0	—	—
10	170	1	rwx	2
—	—	0	—	—

- Analogous to PID
- Smaller in size
- With ASID the TLB can hold translations from different processes at the same time without any ambiguity.

Solution #2 Address Space Identifier (ASID)

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
—	—	0	—	—
10	170	1	rwx	2
—	—	0	—	—

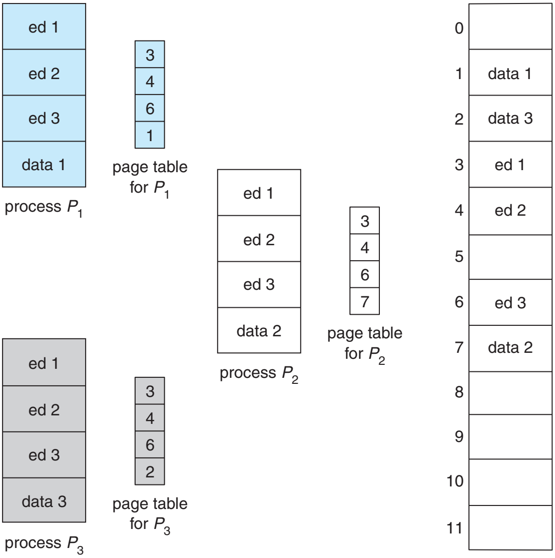
- Analogous to PID
- Smaller in size
- With ASID the TLB can hold translations from different processes at the same time without any ambiguity.

OS-HW Support

OS must, on a context switch, set some privileged register to the ASID of the current process.

Recall

Shared Pages



What would the same reflect in the TLB?

What would the same reflect in the TLB?

VPN	PFN	valid	prot	ASID
10	101	1	r-x	1
—	—	0	—	—
50	101	1	r-x	2
—	—	0	—	—

TLB Entry Replacement

Which TLB entry should be replaced when we add a new TLB entry?

Goal

Minimize the miss rate (or increase hit rate)

Least Recently Used

Idea

It is likely that an entry that has **not recently been used** is a good candidate for eviction

- Takes advantage of **locality** in the memory-reference stream

Classwork

- Can you devise a program that would perform terribly with LRU?
- What would be the alternative policy?

Random Replacement

- Evicts a TLB mapping at random
- Simple to implement
- Ability to avoid corner-case behaviors like LRU

HomeWork

Study MIPS TLB Entry

Issues?

Solves one issue with Paging

- Effective access time is decreased

Issues?

Solves one issue with Paging

- Effective access time is decreased

However

- Issue of TLB Coverage

Issues?

Solves one issue with Paging

- Effective access time is decreased

However

- Issue of TLB Coverage

Does not solve

- Issue of storing large PT in memory

Attempt #7:
Smaller Page Tables

Next Lecture