# CSE 351 Section V

Virtual Memory

# Administrivia - We're in the final stretch!

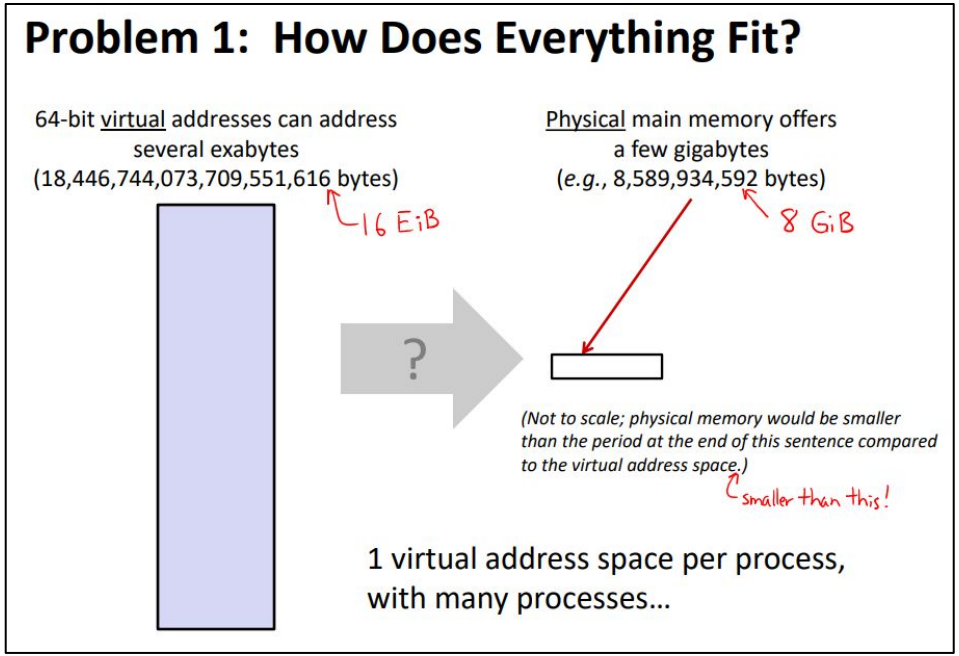
- **Lab 4**
  - Due Mon, Nov 28th
- **Homework 22**
  - Due Wed, Nov 30th
- **Homework 24**
  - Due Fri, Dec 2nd
- **Homework 25**
  - Due Wed, Dec 7th
- **Lab 5**
  - Due Fri, Dec 9th
- **Take-home Final**
  - From Mon, Dec 12th to Wed, Dec 14th


# Virtual Memory

# A summary of virtual memory

**Virtual memory is a huge imaginary region of storage for processes.**

Processes *believe* they have $2^n$ bytes worth of memory available (where n is the word-size).



# A summary of virtual memory

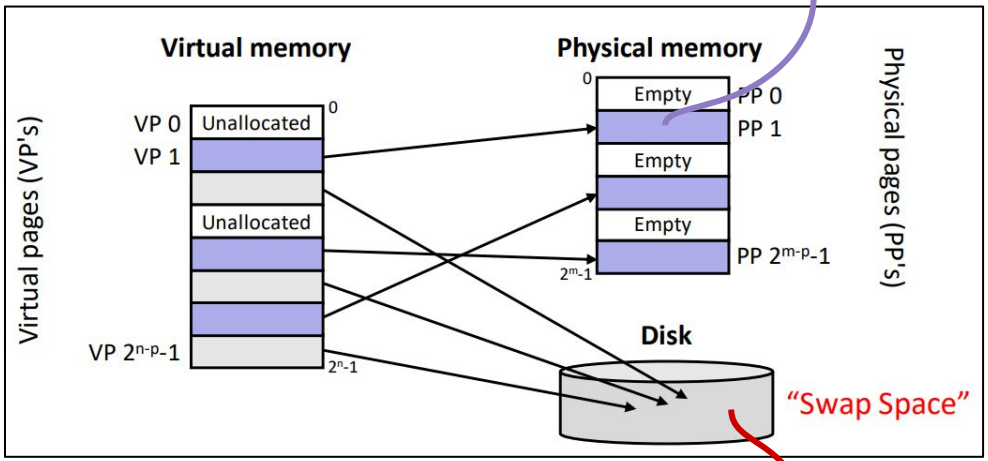
**Virtual memory is a huge imaginary region of storage for processes.**

Processes *believe* they have $2^n$ bytes worth of memory available (where n is the word-size).

**Physical and virtual memory is broken up into fixed-size "pages".**

Virtual pages in-use will map to a physical page.

If we run out of physical memory for processes, then only the most recently-used pages are left in memory.

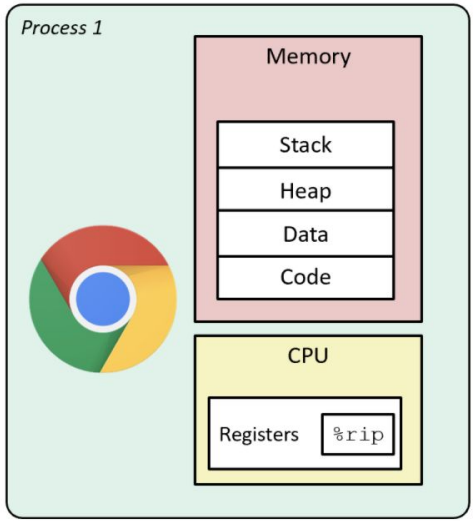



Excess pages in-use are located on the 'swap space' of the disk and 'swapped' in when needed.

# Where are virtual and physical addresses used?

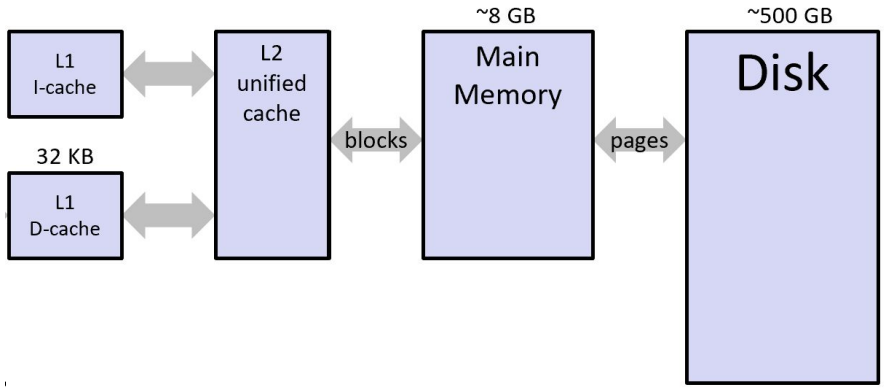
## CPU, Registers, and Processes

Use **virtual** addresses to reference data




## Caches, Physical Memory and Disk

Use **physical** addresses to reference data




*Not drawn to scale*

# How do we convert virtual addresses to physical addresses?

## Page Tables!

- Specific for each process
- Job is to map virtual page numbers (VPN) to physical page numbers (PPN)
- Contains page table entries (PTE)
  - Has a PTE for every possible VPN
  - PTE is made up of PPN + management bits
- Stored in physical memory

Page Table

| VPN | PPN | Valid | Dirty | Read | Write | Execute |
|---|---|---|---|---|---|---|
| 00 | 28 | 1 | 0 | 1 | 1 | 0 |
| 01 | - | 0 | 0 | 0 | 0 | 0 |
| 02 | 33 | 1 | 1 | 1 | 0 | 0 |
| 03 | 02 | 1 | 0 | 1 | 0 | 1 |
| 04 | - | 0 | 0 | 0 | 0 | 0 |
| 05 | 16 | 1 | 1 | 1 | 1 | 0 |
| 06 | - | 0 | 0 | 0 | 0 | 0 |
| 07 | - | 0 | 0 | 0 | 0 | 0 |

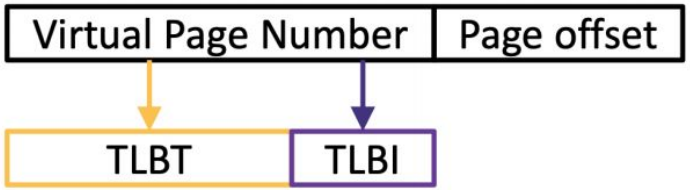
An example of a page table

# Accessing page tables is expensive...

- Accessing page tables is expensive because page tables are located in physical memory
- Page tables store all PTEs for all possible virtual pages
    - Processes don't need access to **all** of those PTEs, only a small subset of the PTEs!

**Solution: Make a cache for our PTEs!**

# Translation Lookaside Buffer (TLB)



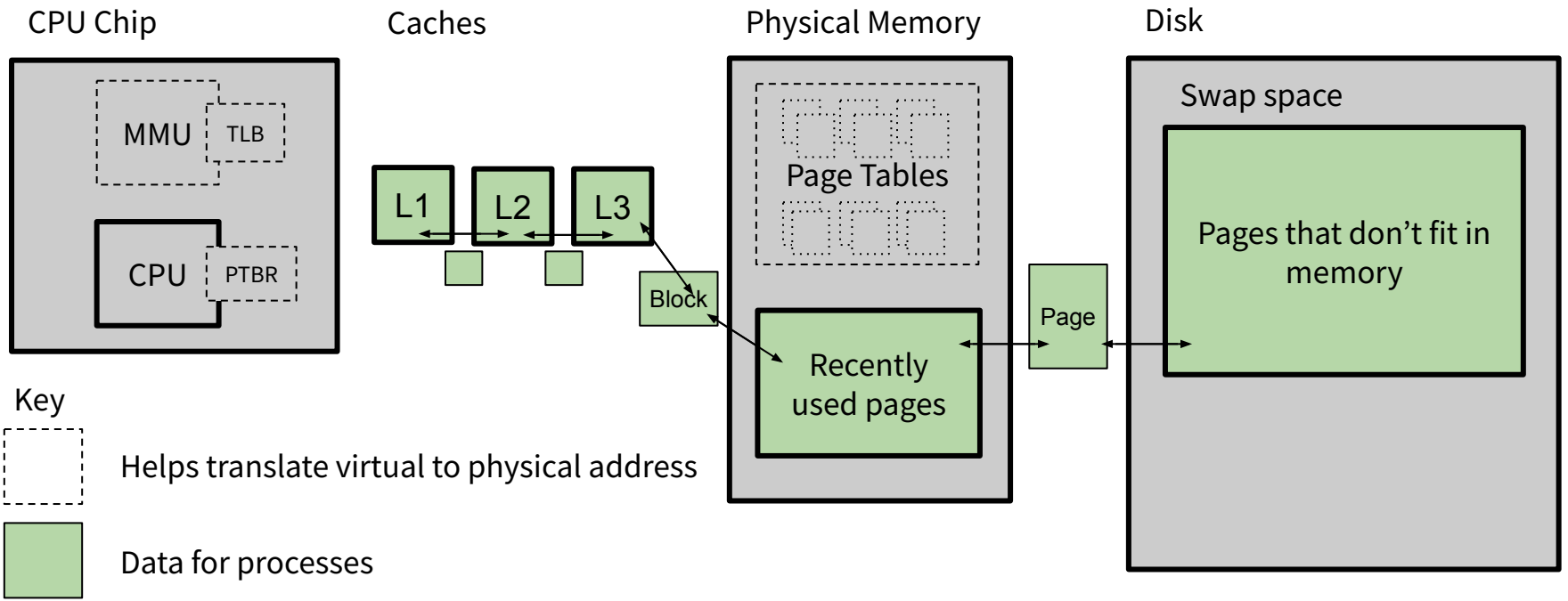

The TLB is a cache of the page table

- Stores the most recently used PTEs
- TLB is super fast to access compared to accessing page tables in memory

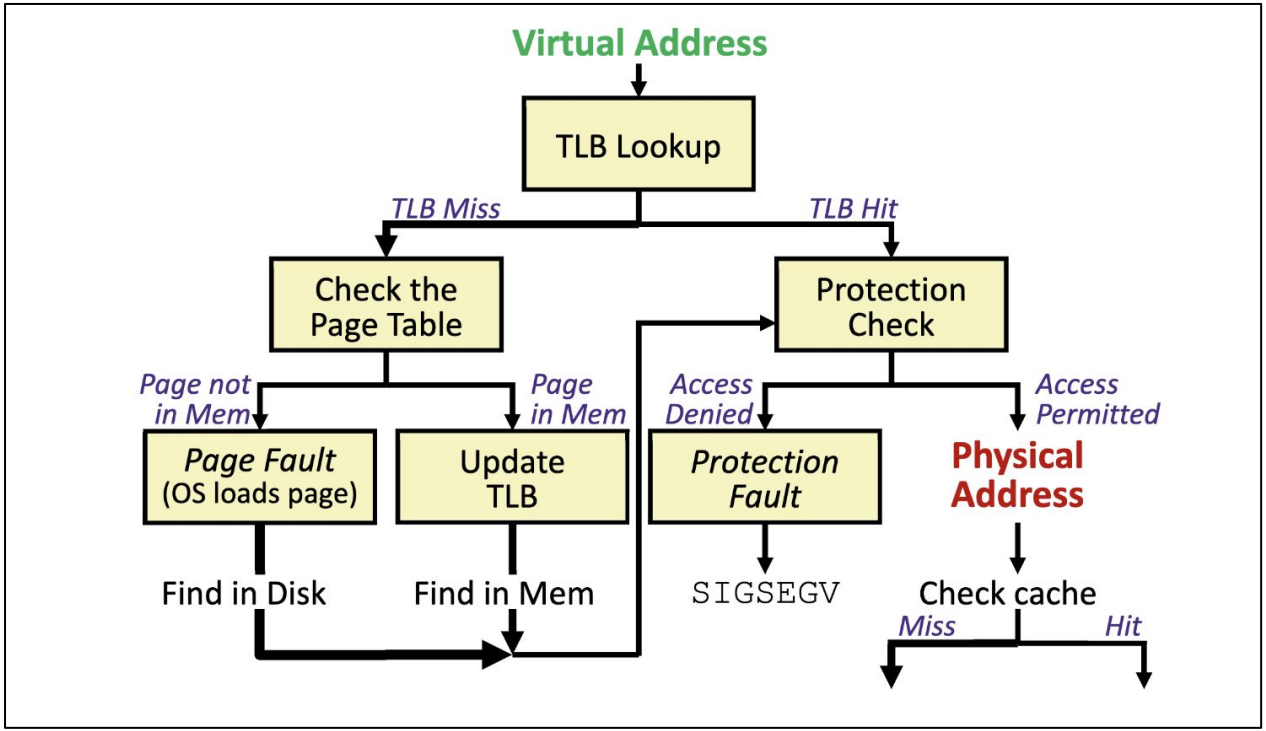
**TLB**

| Set | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 03 | – | 0 | 09 | 0D | 1 | 00 | – | 0 | 07 | 02 | 1 |
| 1 | 03 | 2D | 1 | 02 | – | 0 | 04 | – | 0 | 0A | – | 0 |
| 2 | 02 | – | 0 | 08 | – | 0 | 06 | – | 0 | 03 | – | 0 |
| 3 | 07 | – | 0 | 03 | 0D | 1 | 0A | 34 | 1 | 02 | – | 0 |

# High-level, simplified view of how memory is accessed



# Address Translation Flowchart




What causes the following to occur?

- TLB Hit
- TLB Miss
- Page Table Hit
- Page Fault
- Protection Fault

Note that the width of the arrows is intended to indicate relative latencies of different pathways, though definitely not to scale.

# Address Manipulation

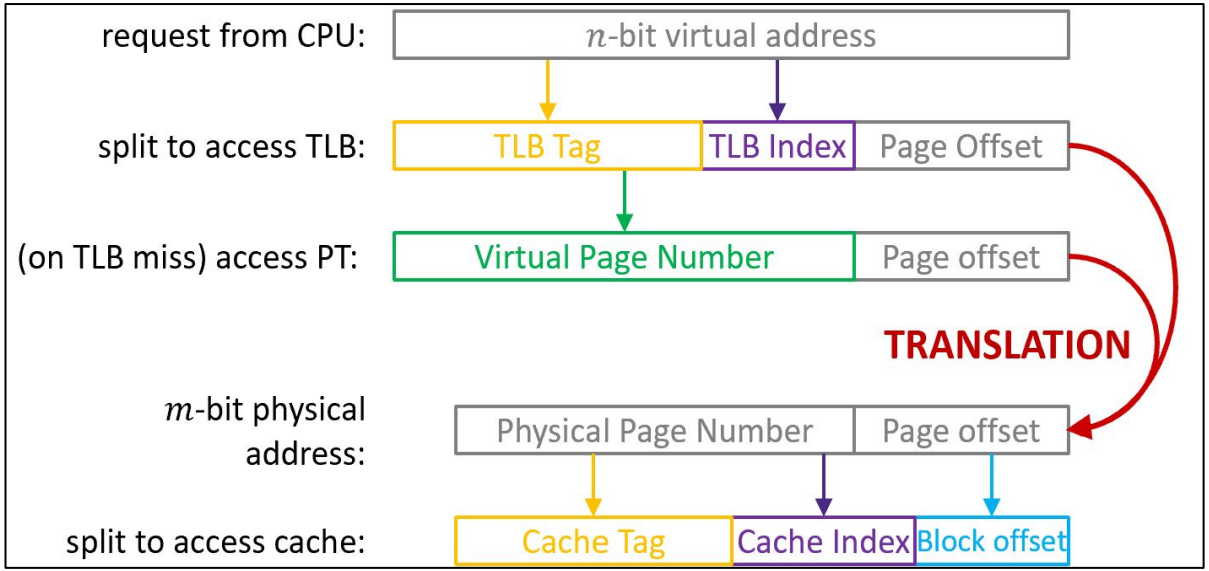
The image below shows the way that the requested address gets manipulated during a data fetch.

Virtual Address → Physical Address

1. Do **TLB TIO** breakdown on the virtual address, check TLB
2. On a TLB miss, do a **VPN + PO** breakdown on the virtual address, access the PT

Physical address → Requested Data

1. Do a **Cache TIO** breakdown on the physical address



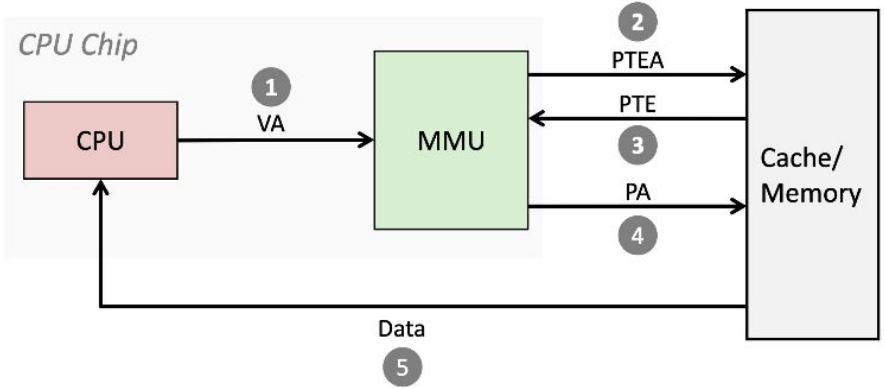
# Other Useful Virtual Memory Slides

Also check out the Edstem lesson for other useful PDFs!

# Address Translation Steps: Page Hit

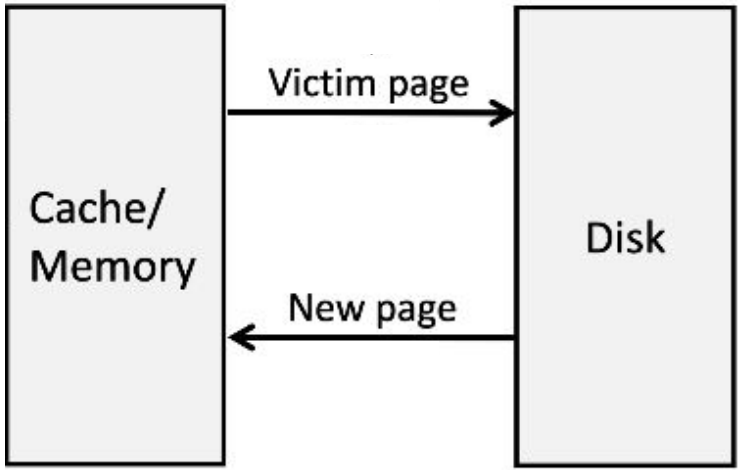
Pretending we don't have a TLB...

1. Processor sends VA to MMU
2. Using the PTBR to find PT, MMU accesses PTE for the respective VPN
3. MMU receives PTE, sees valid bit is 1 (**Page Hit!**), and gets the PA
4. MMU sends PA to cache/memory requesting data
5. Cache/memory returns data to processor



# Page Faults

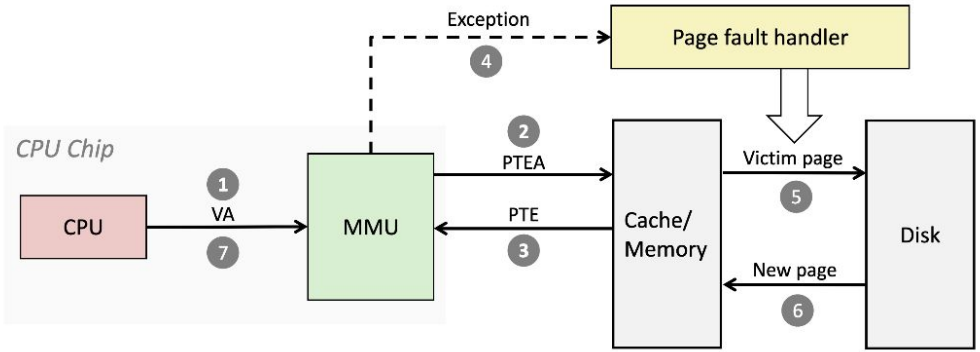
- A **page fault** occurs When a PTE references a page that isn't in physical memory.
  - A page is kicked out of memory
  - The requested page is brought into memory



# Address Translation Steps: Page Fault

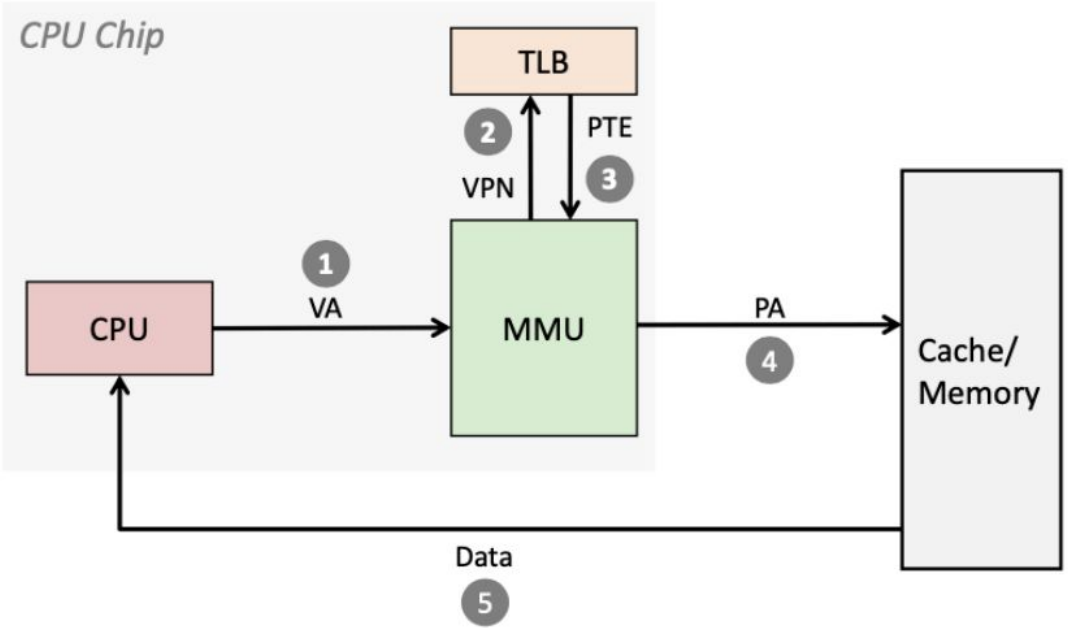
Pretending we don't have a TLB...

1. Processor send VA to MMU
2. Using the PTBR to find PT, MMU accesses PTE for the respective VPN
3. MMU receives PTE, sees valid bit is 0 (**Page Fault!**), and gets the PA
4. MMU triggers page fault exception
5. Handler identifies victim page (if victim is dirty, pages it out to disk)
6. Handler pages in new page and updates PTE in memory
7. Handler restarts faulting instruction for a guaranteed page hit
    a. (see steps for a page hit)



# TLB Hit

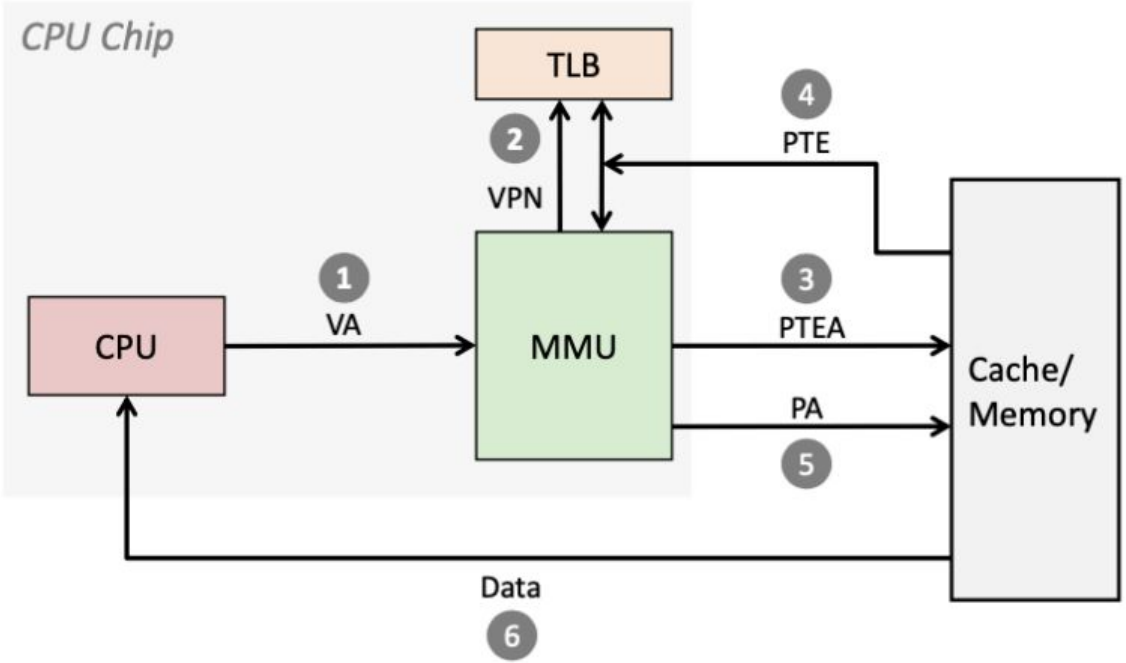
1. Processor sends VA to MMU
2. MMU checks TLB for PTE for respective VPN
3. MMU receives PTE, sees valid bit is 1 (**TLB Hit!**), and gets the PA
   a. Assuming we have access
4. MMU sends PA to cache/memory requesting data
5. Cache/memory returns data to processor




A TLB hit eliminates an expensive memory access to the page table located in memory

# TLB Miss

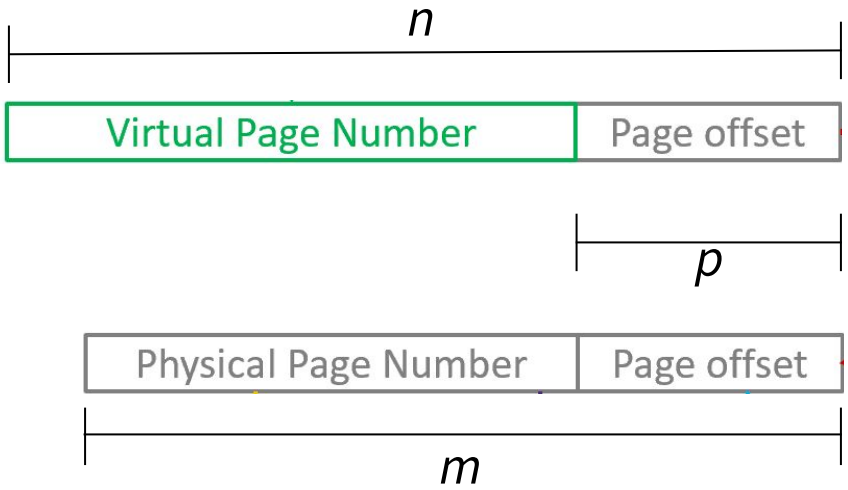
1. Processor sends VA to MMU
2. MMU checks TLB for PTE for respective VPN, valid bit is 0 (**TLB miss!)**
3. MMU fetches PTE from page table
   a. Possible page fault
4. PTE is loaded into TLB
5. MMU sends PA to cache/memory requesting data
   a. Assuming we have access




A TLB miss incurs an additional memory access (the PTE)
Fortunately, TLB misses are rare.

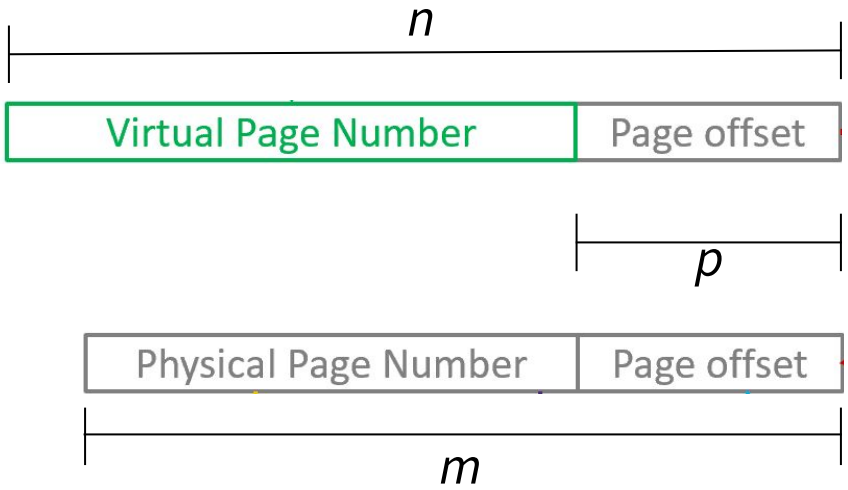
# Virtual Memory Exercise Solutions

# Exercise 4: Fill in the table (row 1)



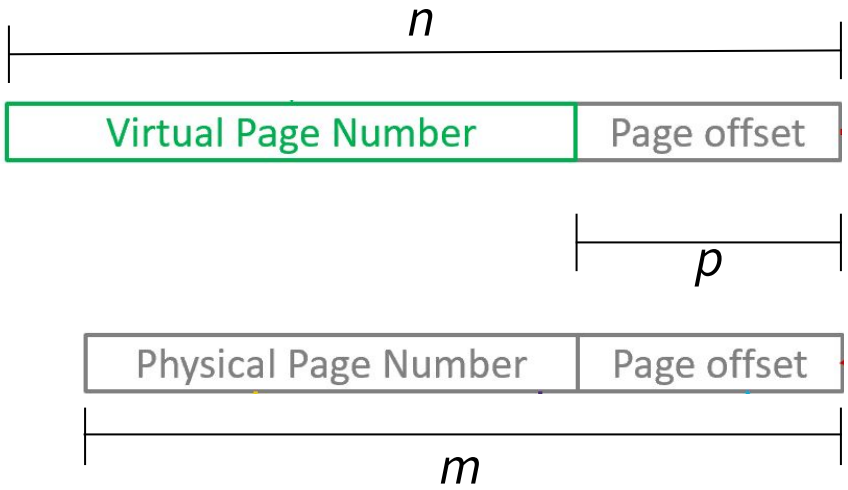

| VA Width ($n$) | PA Width ($m$) | Page Size (P) | VPN Width | PPN Width | Bits in PTE (assume V,D,R,W,X) |
|---|---|---|---|---|---|
| 32 | 32 | 16 KiB<br>16 KiB = $2^{14}$ B<br>p = 14 | 32 - 14 = 18 | 32 - 14 = 18 | 18 + 5 = 23 |

# Exercise 4: Fill in the table (row 2)



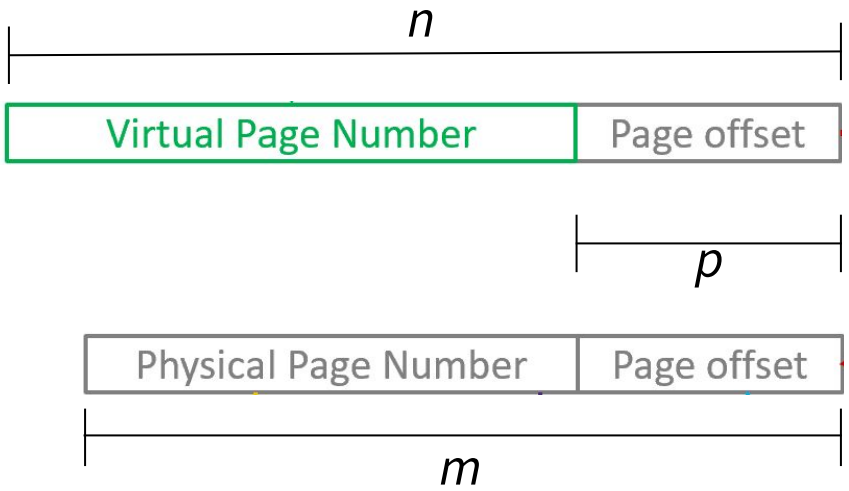

| VA Width ($n$) | PA Width ($m$) | Page Size (P) | VPN Width | PPN Width | Bits in PTE (assume V,D,R,W,X) |
|---|---|---|---|---|---|
| 32 | 26 | p = 26 - 13 = 13<br>$2^{13}$ B = 8 KiB | 32 - 13 = 19 | 13 | 13 + 5 = 18 |

# Exercise 4: Fill in the table (row 3)



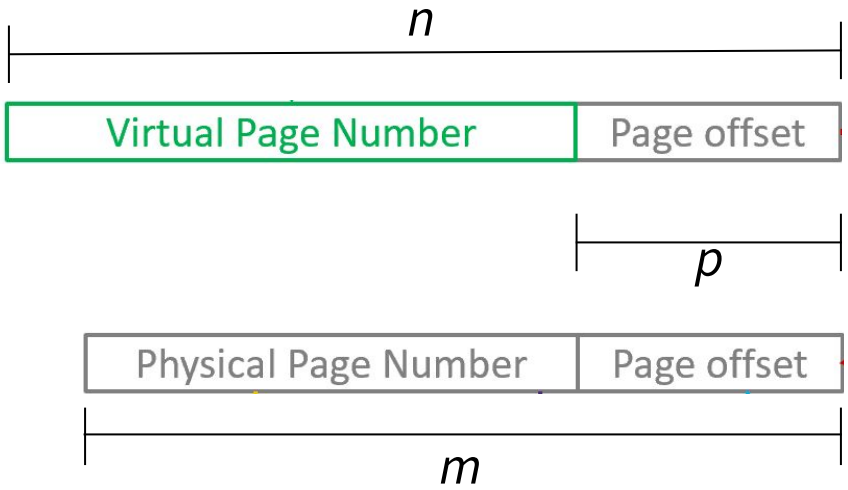

| VA Width (*n*) | PA Width (*m*) | Page Size (P) | VPN Width | PPN Width | Bits in PTE (assume V,D,R,W,X) |
|---|---|---|---|---|---|
| 21 + 15 = 36 | 32 | p = 32 - 17 = 15<br>$2^{15}$ B = 32 KiB | 21 | 22 - 5 = 17 | 22 |

# Exercise 4: Fill in the table (row 4)




| VA Width ($n$) | PA Width ($m$) | Page Size (P) | VPN Width | PPN Width | Bits in PTE (assume V,D,R,W,X) |
|---|---|---|---|---|---|
| 25 + 15 = 40 | 21 + 15 = 36 | 32 KiB<br>$P = 2^{15}$ B<br>p = 15 | 25 | 26 - 5 = 21 | 26 |

# Exercise 4: Fill in the table (row 5)

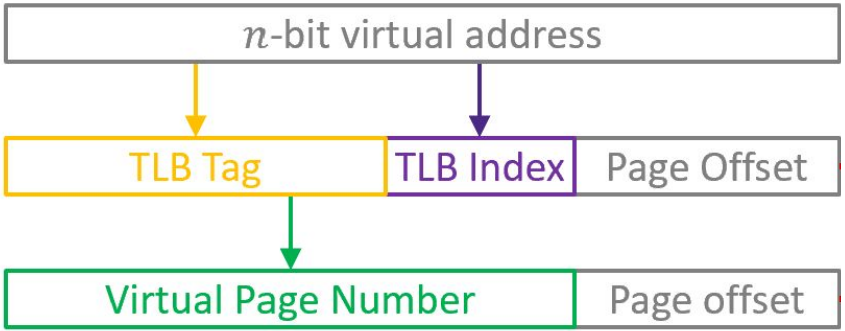



| VA Width (*n*) | PA Width (*m*) | Page Size (P) | VPN Width | PPN Width | Bits in PTE (assume V,D,R,W,X) |
|---|---|---|---|---|---|
| 64 | 24 + 16 = 40 | p = 64 - 48 = 16<br>$2^{16}$ B = 64 KiB | 48 | 29 - 5 = 24 | 29 |

# Exercise 5

- Processor: 16-bit addresses, 256-byte pages
- TLB: 8-entry fully associative with LRU replacement
  - Track LRU (shown in decimal) using 3 bits to encode the order in which pages were accessed, with 0 being the most recent
- Assume that all page table entries that are not in the initial TLB have read and write permissions, but no execute permission (i.e. R = 1, W = 1, X = 0).
  - OS will assign new pages starting at PPN 0x20, with read and write permissions but no execute permission (i.e. R = 1, W = 1, X = 0).
- At some time instant, the TLB for the current process is in the initial state given.
- Fill in the final state of the TLB according to the access pattern below. For each access, indicate if it leads to a:
  - a) TLB hit? b) TLB miss? c) Page fault? d) Protection fault?

# Exercise 5




First, let's figure out the relevant numbers:

- 16-bit addresses means ***n* = 16**
- 256-byte pages means ***P* = 256**
  - so $p = \log_2(256) = 8$
- **VPN width = *n* - *p* = 8 bits**
- TLB is fully associative, so $S = 1$, which means **TLBI bits = $\log_2(1) = 0$**
  - **so TLBT = VPN**
  - Because the VPN width is 8 bits, we **can just read the first 2 hex digits as both the VPN and TLBT**
- LRU column will help us figure out which entries to kick out
  - also need to make sure we update LRU every time the TLB is accessed

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0x02 | 0x18 | 1 | 1 | 0 | 0 | 0 | 6 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 0 | 5 |
| 0x00 | 0x00 | 0 | 0 | 0 | 0 | 0 | 7 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 4 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 2 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 3 |

1) **Read 0x11F0**

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0x02 | 0x18 | 1 | 1 | 0 | 0 | 0 | 6 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 2 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 0 | 5 |
| 0x00 | 0x00 | 0 | 0 | 0 | 0 | 0 | 7 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | **0** |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 3 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 4 |

**1) Read 0x11F0**

- → TLBT = 0x11
- → TLB Hit
  - PTE w/ matching tag is present and valid
- → No protection fault
  - has Read access

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0x02 | 0x18 | 1 | 1 | 0 | 0 | 0 | 6 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 2 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 0 | 5 |
| 0x00 | 0x00 | 0 | 0 | 0 | 0 | 0 | 7 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 3 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 4 |

**2) Write 0x0301**

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 2 |
| 0x02 | 0x18 | 1 | 1 | 0 | 0 | 0 | 7 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 3 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 0 | 6 |
| **0x03** | **0x17** | **1** | **1** | **1** | **0** | **1** | **0** |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 4 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 5 |

Page Table (partial):

| VPN | Valid | PPN |
|---|---|---|
| 0x0 | 0 | 0x00 |
| 0x1 | 1 | 0x19 |
| 0x2 | 1 | 0x18 |
| 0x3 | 1 | 0x17 |
| 0x4 | 0 | - |
| 0x5 | 0 | - |
| 0x6 | 1 | 0x1A |
| 0x7 | 0 | - |

**2) Write 0x0301**

- → TLBT = 0x03
- → TLB Miss
  - Tag is not in TLB
- → No page fault
  - PTE exists in memory
  - Load PPN 0x17 into the invalid/LRU entry of the TLB
- → No protection fault
  - has Write access
  - update Dirty bit

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 2 |
| 0x02 | 0x18 | 1 | 1 | 0 | 0 | 0 | 7 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 3 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 0 | 6 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 4 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 5 |

**3) Write 0x20AE**

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 3 |
| 0x02 | 0x18 | 1 | 1 | 0 | 0 | 0 | 7 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 4 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | **1** | **0** |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 2 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 5 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 6 |

**3) Write 0x20AE**

- → TLBT = 0x20
- → TLB Hit
  - PTE w/ matching tag is present and valid
- → No protection fault
  - has Write access
- update Dirty bit

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 3 |
| 0x02 | 0x18 | 1 | 1 | 0 | 0 | 0 | 7 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 4 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 2 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 5 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 6 |

**4) Write 0x0532**

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 4 |
| **0x05** | **0x20** | **1** | **1** | **1** | **0** | **1** | **0** |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 5 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 2 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 3 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 6 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 7 |

Page Table (partial):

| VPN | Valid | PPN |
|---|---|---|
| 0x0 | 0 | 0x00 |
| 0x1 | 1 | 0x19 |
| 0x2 | 1 | 0x18 |
| 0x3 | 1 | 0x17 |
| 0x4 | 0 | - |
| 0x5 | 0 | - |
| 0x6 | 1 | 0x1A |
| 0x7 | 0 | - |

**4) Write 0x0532**

- → TLBT = 0x05
- → TLB Miss
  - Tag is not in TLB
- → Page Fault
  - PTE is not valid for VPN=0x05
  - Assign PPN 0x20 to VPN 0x05
- Update TLB PTE, replace LRU
- Update Dirty Bit

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 4 |
| 0x05 | 0x20 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 5 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 2 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 3 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 6 |
| 0x34 | 0x16 | 1 | 1 | 1 | 0 | 1 | 7 |

**5) Read 0x0E15**

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 5 |
| 0x05 | 0x20 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 6 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 1 | 2 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 3 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 4 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 7 |
| **0x0E** | **0x21** | **1** | **1** | **1** | **0** | **0** | **0** |

| VPN | Valid | PPN |
|---|---|---|
| 0x8 | 1 | 0x1C |
| 0x9 | 1 | 0x1D |
| 0xA | 0 | 0x1E |
| 0xB | 1 | 0x1F |
| 0xC | 0 | - |
| 0xD | 1 | 0x09 |
| 0xE | 0 | - |
| 0xF | 1 | 0x1B |

**5) Read 0x0E15**

- → TLBT = 0x0E
- → TLB Miss
  - Tag is not in TLB
- → Page Fault
  - PTE is not valid for VPN=0x0E
  - Assign PPN 0x21 to VPN 0x05
- Update TLB PTE, replace LRU

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 5 |
| 0x05 | 0x20 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 6 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 1 | 2 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 3 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 4 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 7 |
| 0x0E | 0x21 | 1 | 1 | 1 | 0 | 0 | 0 |

**6) Write 0xACFF**

# Exercise 5

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 6 |
| 0x05 | 0x20 | 1 | 1 | 1 | 0 | 1 | 2 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 7 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 1 | 3 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 4 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 5 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | **0** |
| 0x0E | 0x21 | 1 | 1 | 1 | 0 | 0 | 1 |

**6) Write 0xACFF**

Write 0xACFF → TLBT = 0xAC, TLB Hit, Protection fault

- → TLBT = 0xAC
- → TLB Hit
  - PTE w/ matching tag is present and valid
- → Protection fault
  - does not have Write access
- → Still update LRU

# Exercise 5

**Final TLB:**

| TLBT | PPN | Valid | R | W | X | Dirty | LRU |
|---|---|---|---|---|---|---|---|
| 0x01 | 0x11 | 1 | 1 | 1 | 0 | 1 | 6 |
| 0x05 | 0x20 | 1 | 1 | 1 | 0 | 1 | 2 |
| 0x10 | 0x13 | 1 | 1 | 1 | 1 | 1 | 7 |
| 0x20 | 0x12 | 1 | 0 | 1 | 0 | 1 | 3 |
| 0x03 | 0x17 | 1 | 1 | 1 | 0 | 1 | 4 |
| 0x11 | 0x14 | 1 | 1 | 0 | 0 | 0 | 5 |
| 0xAC | 0x15 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0x0E | 0x21 | 1 | 1 | 1 | 0 | 0 | 1 |

# That’s All, Folks!

Thanks for attending section! Remember to post on Ed or go to OH if you have questions.

See you all next week and good luck on lab 4.