CSE 351

Virtual Memory

Virtual Memory

- Very powerful layer of indirection on top of physical memory addressing
  - We never actually use physical addresses when writing programs
  - Every address, pointer, etc you come across is a virtual address
- Why? It gives us several benefits
  - Shared memory spaces
  - Memory isolation
  - Memory R/W/X protection
  - Illusion of large address space despite limited physical memory

Address translation

- How do we convert virtual addresses to physical addresses?
  - Create a table mapping virtual pages to physical pages
- A page table is an array of page table entries (PTEs)
  - An entry exists for every virtual page number (VPN)
  - Each entry stores a physical page number (PPN)
  - Each process gets its own page table
  - The page table is software-defined! It exists in DRAM

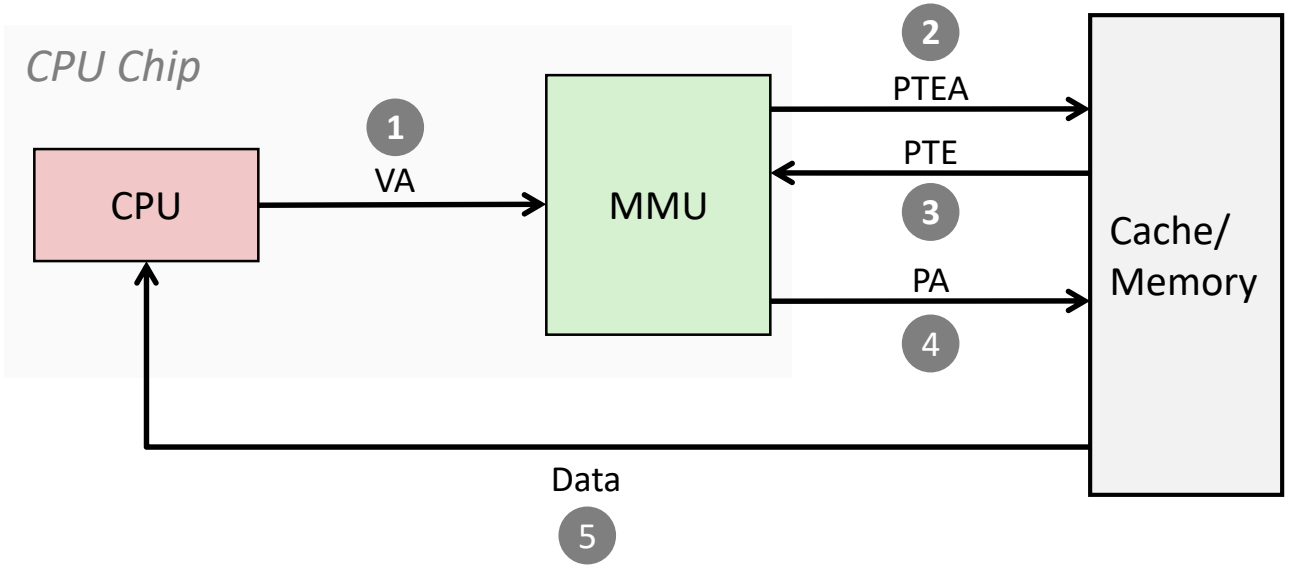
Visualizing virtual memory

- Virtual memory is an enormous, contiguous region for storage
  - It is broken up into fixed-size “pages”
  - When a process needs more memory, the OS will allocated a page of physical memory and insert an entry into that process’s page table
  - If we run out of physical memory for that process, then only the most recently-used pages are left in memory
    - The rest are kept on disk, and “swapped” in when needed
- Thus, we can think of virtual memory as a cache for disk
  - Imagine that the virtual memory space is initially mapped to disk
  - Recently-used pages of virtual memory are cached in physical memory

Page Faults

- When a page table entry points to a page that is on disk, a process will generate a page fault
  - Transfers control to the OS
  - The OS will copy the page of data from disk into memory

Address Translation: Page Hit



1) Processor sends *virtual* address to MMU (*memory management unit*)

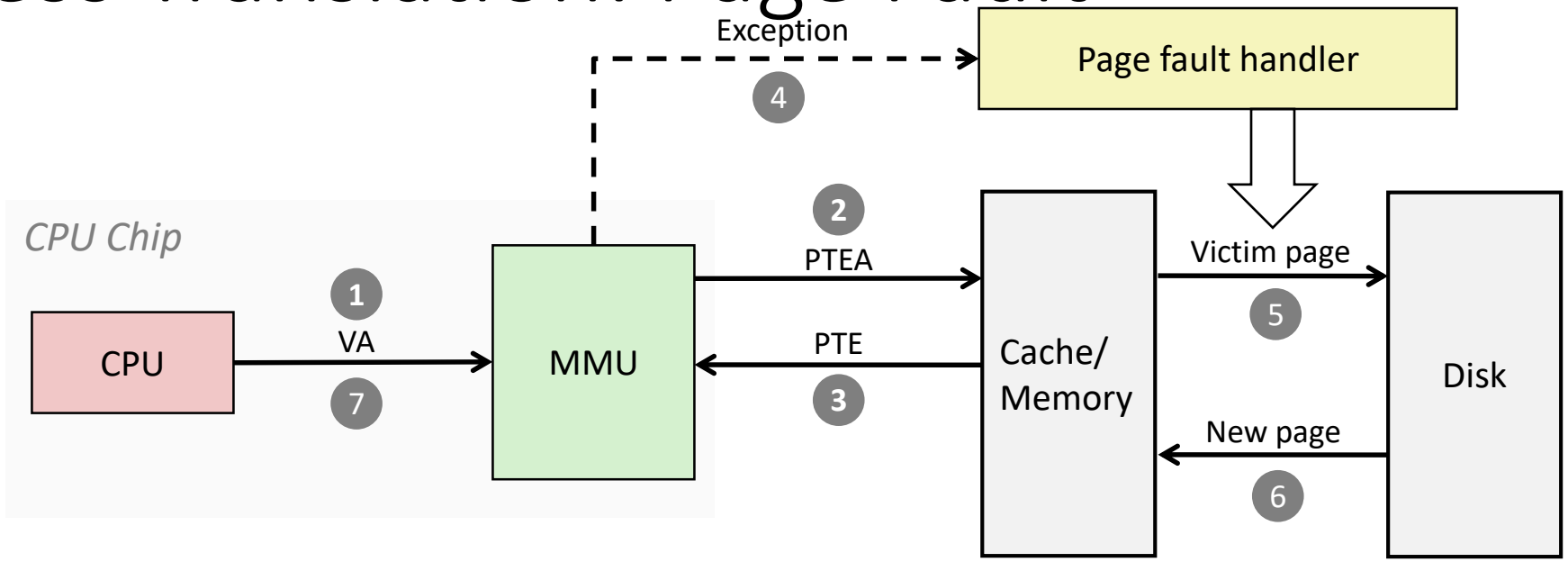
2-3) MMU fetches PTE from page table in cache/memory
(Uses PTBR to find beginning of page table for current process)

4) MMU sends *physical* address to cache/memory requesting data

5) Cache/memory sends data (~1 word) to processor

VA = Virtual Address PTEA = Page Table Entry Address PTE= Page Table Entry
PA = Physical Address Data = Contents of memory stored at VA originally requested by CPU

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in cache/memory

4) Valid bit is zero, so MMU triggers page fault exception

5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction

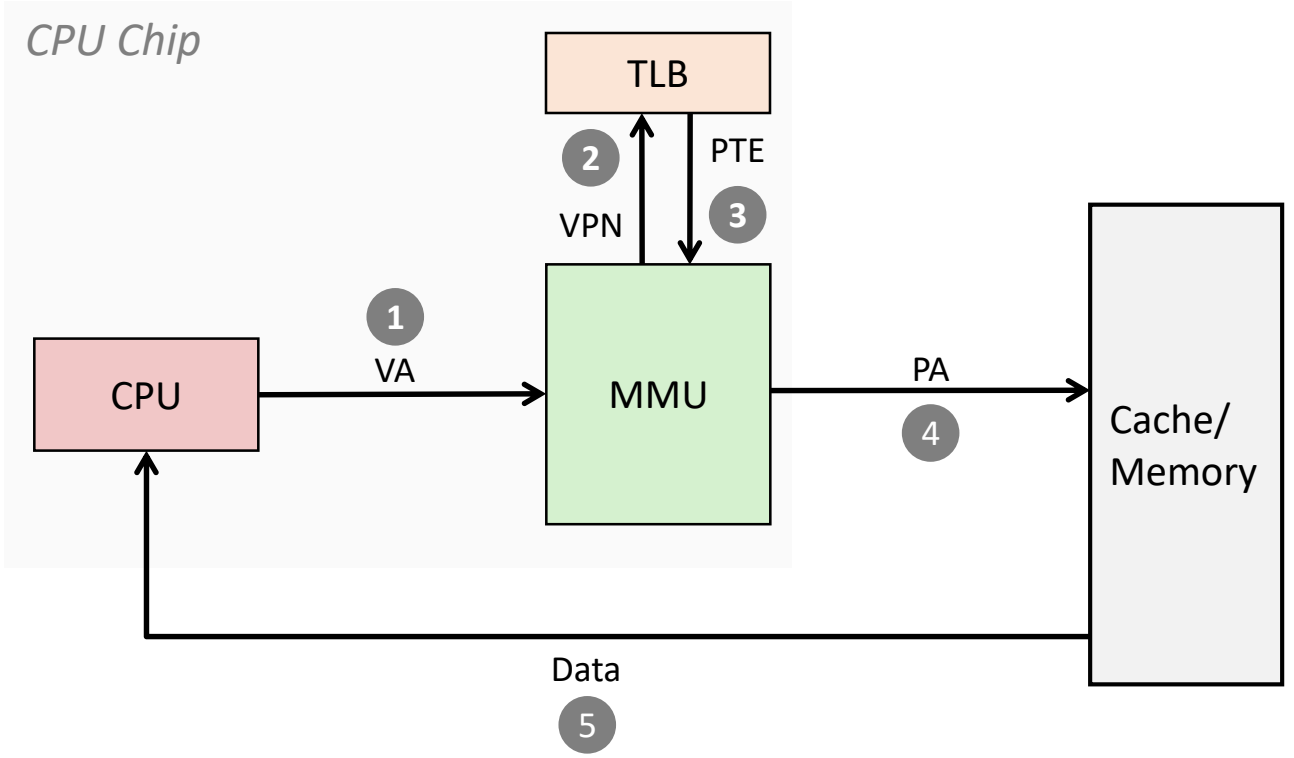
Other benefits

- Shared memory is easy to implement
  - In each process's page table, simply point to the same physical page number
- Memory protection
  - In addition to valid bits in the page table, also store bits for R/W/X
  - If a process attempts to use a page incorrectly, it will generate a segmentation fault

Speeding up virtual memory

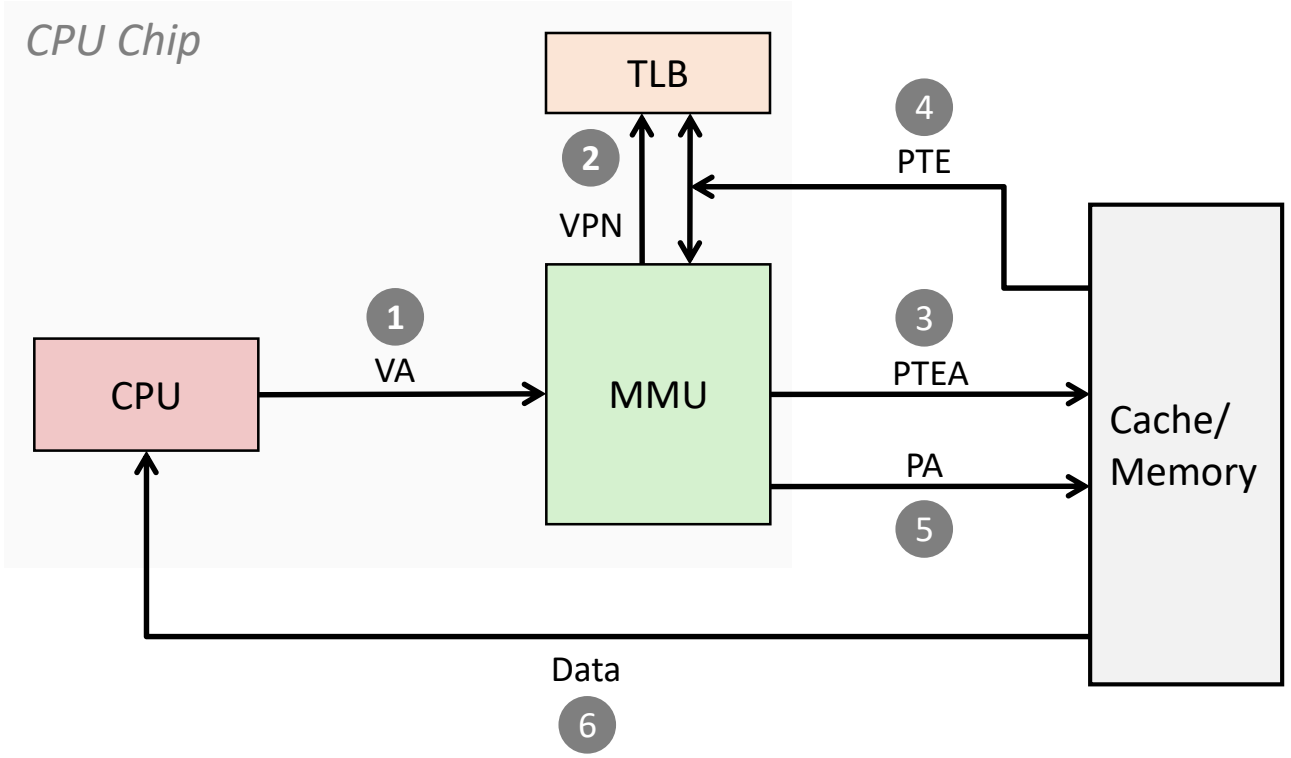
- Reading memory in order to read memory seems…slow
  - It is!
- How have we sped up memory reads already?
  - Caching!
- Translation Lookaside Buffer (TLB)
  - Hardware cache that stores recent VPN->PPN mappings
  - This prevents the MMU from querying DRAM for every address translation
    - However, if a TLB miss occurs, it still has to go to the page table (which is in DRAM)
  - What do we need to do if we context switch to a new process?
    - At the very minimum, we need to flush the TLB (different mappings for different processes)
    - Modern processes rely on tagging TLB entries to avoid needing to flush every context switch
- Now we'll go through some examples using the PT + TLB + cache from lecture

TLB Hit



A TLB hit eliminates a memory access

TLB Miss

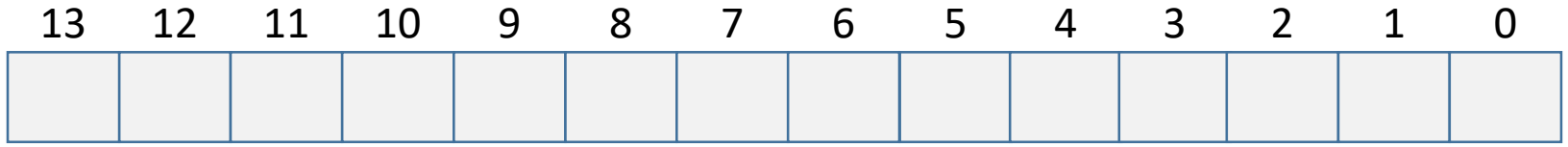


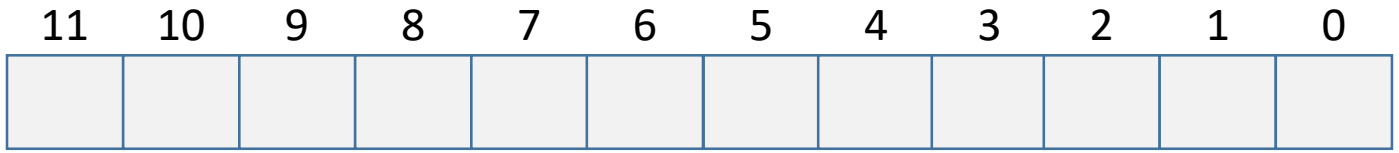
A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare.

Simple Memory System Example

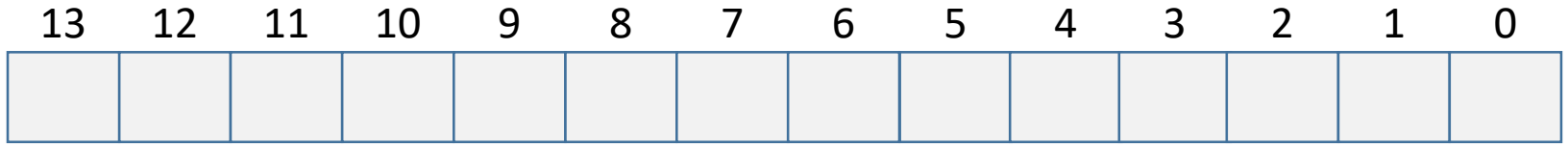
- Addressing
  - 14-bit virtual addresses
  - 12-bit physical addresses
  - Page size = 64 bytes

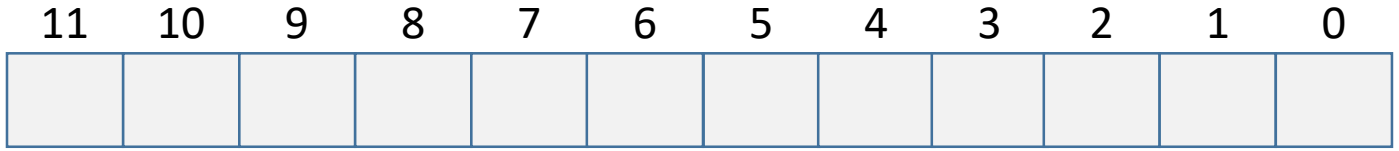




Simple Memory System Example

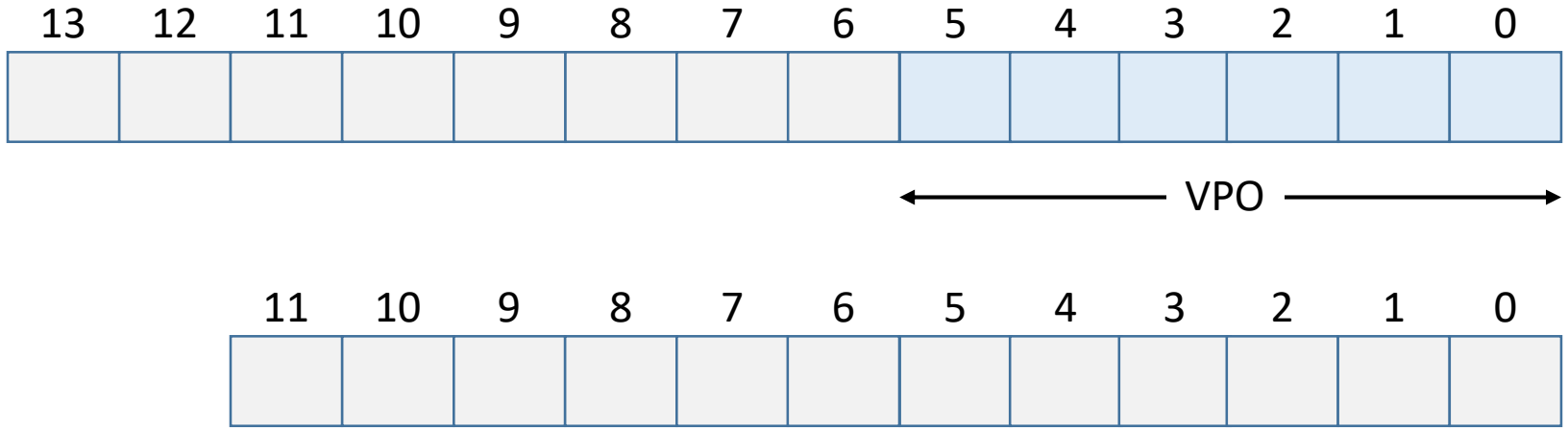
- Step 1: Determine bit ranges for VPN, VPO, PPN, PPO
- The VPO and PPO bits will always be the lower $\log_2 P$, where P is the number of bytes per page
  - We want to be able to address every byte in each page





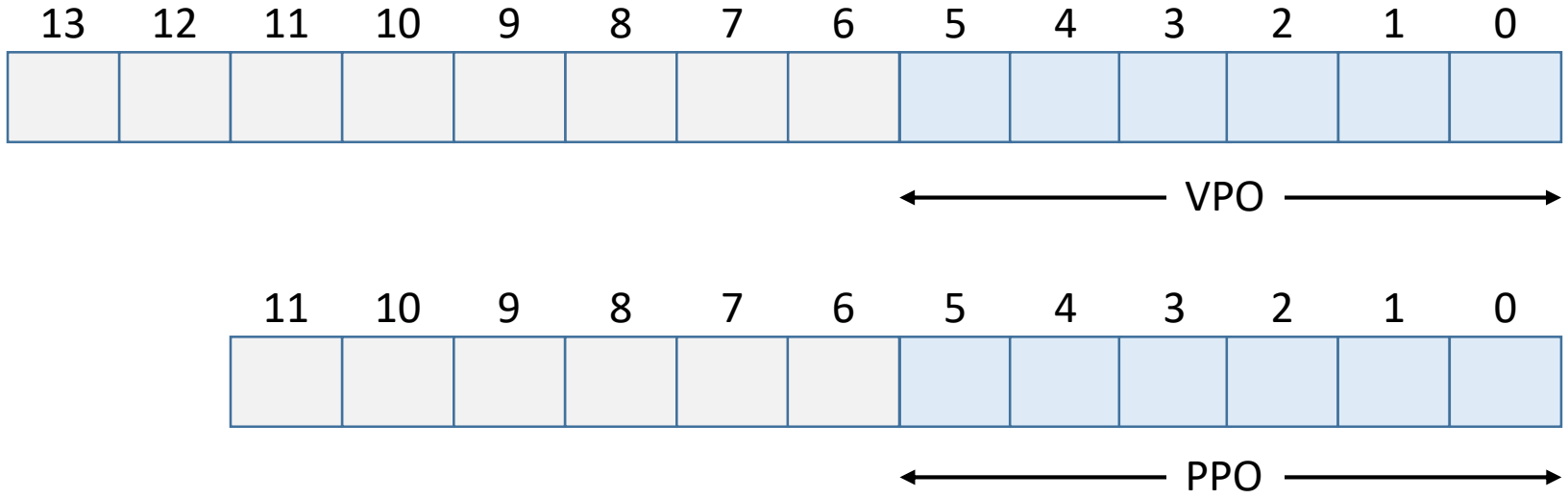
Simple Memory System Example

- In this case, $P = 64$
- $\log_2 64 = 6$
- Thus we will use the lower 6 bits of every virtual address for the VPO



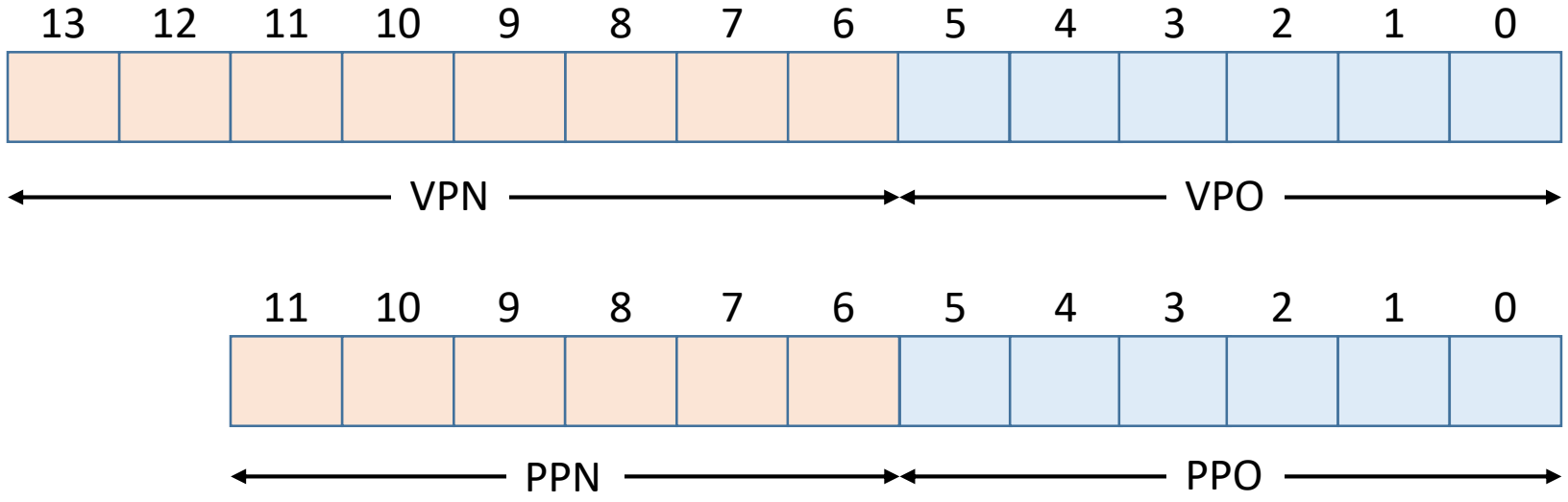
Simple Memory System Example

- The number of bits used for the PPO will always be equal to the number of bits used for the VPO, because the pages are the same size
- Thus, we will use the lower 6 bits of every physical address for the PPO



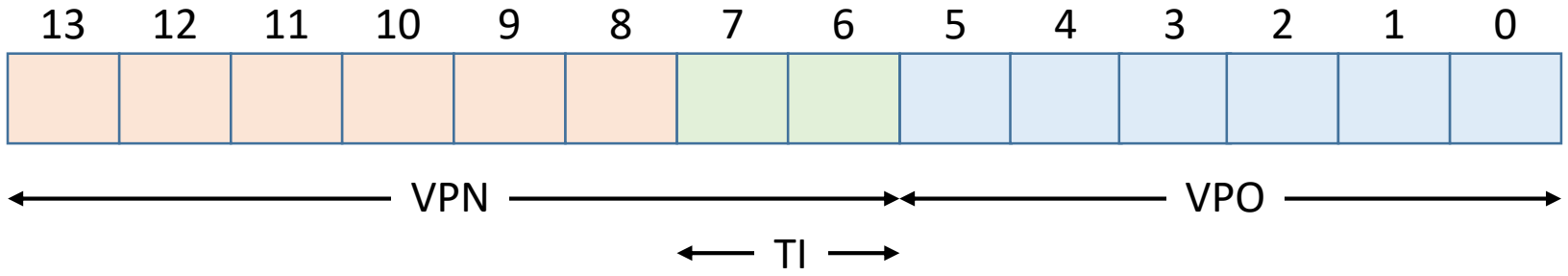
Simple Memory System Example

- The rest of the bits in the virtual address will become the VPN
- Likewise, the rest of the bits in the physical address will become the PPN



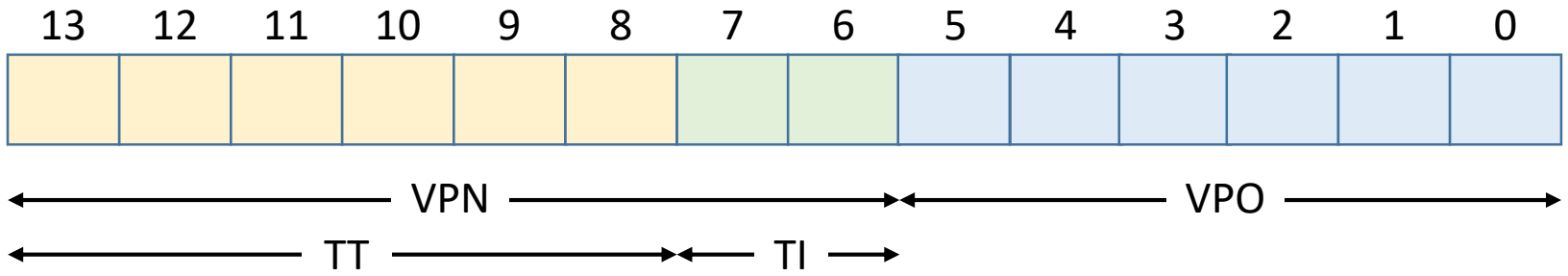
Simple Memory System Example

- Step 2: Look at the architecture of the TLB to decide how to use the bits in the VPN to index into the TLB
- For this problem, our TLB holds 16 entries and is 4-way set associative
  - Thus, there are $S = \frac{16}{4} = 4$ sets
- We know that the number of bits used to represent TI is $\log_2 S$, so in our case $TI = 2$
- The index bits are the low-order bits of the VPN, as shown below



Simple Memory System Example

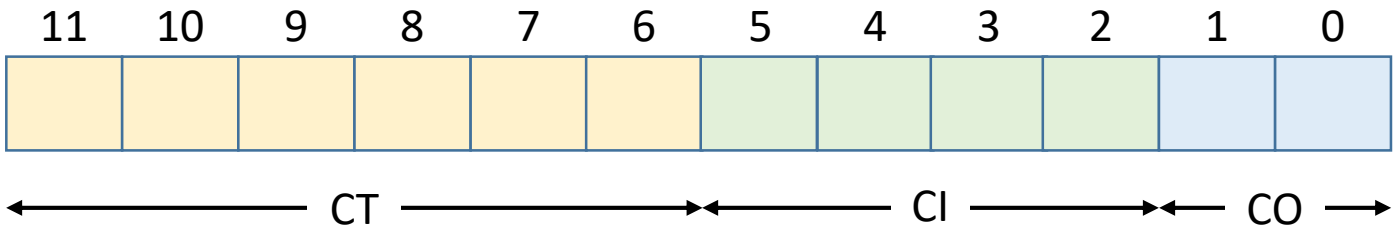
- The remaining bits of the VPN are used as the TLB tag, TT



- Now we know how to calculate the VPN, VPO, PPN, PPO, TI, TT

Simple Memory System Example

- For the following problems, we will be using a cache that with a size of 64 bytes, is direct-mapped, and has a block size of 4 bytes
- The topic of today is virtual memory, so we will skip over the process of determining the CT, CI, and CO bits
  - This is what they will look like for our 12-bit physical addresses:



Simple Memory System Example

- Now we will do some sample reads of virtual addresses
- The TLB has 16 entries and is 4-way set associative
- The page table is obviously massive, but this problem only includes a small portion of it
  - If a VPN is outside the range of the page table, then the result is undefined
  - Could be a page fault or a page table hit
- The cache has a size of 64 bytes, is direct-mapped, and has a block size of 4 bytes

Simple Memory System Example

- Here is the state of the 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

Simple Memory System Example

- Here is the state of the page table:

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	-	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	0C	-	0
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

Simple Memory System Example

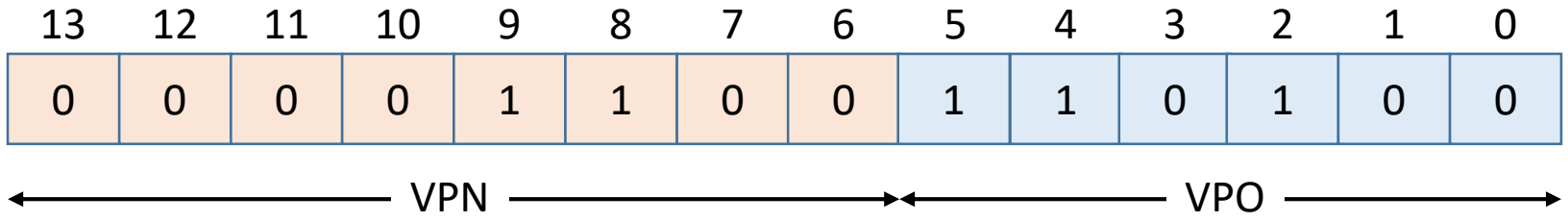
- Here is the state of the cache:

Index	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	-	-	-	-
2	1B	1	00	02	04	08
3	36	0	-	-	-	-
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	-	-	-	-
7	16	1	11	C2	DF	03

Index	Tag	Valid	B0	B1	B2	B3
8	24	1	3A	00	51	89
9	2D	0	-	-	-	-
A	2D	1	93	15	DA	3B
B	0B	0	-	-	-	-
C	12	0	-	-	-	-
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	-	-	-	-

Simple Memory System Example

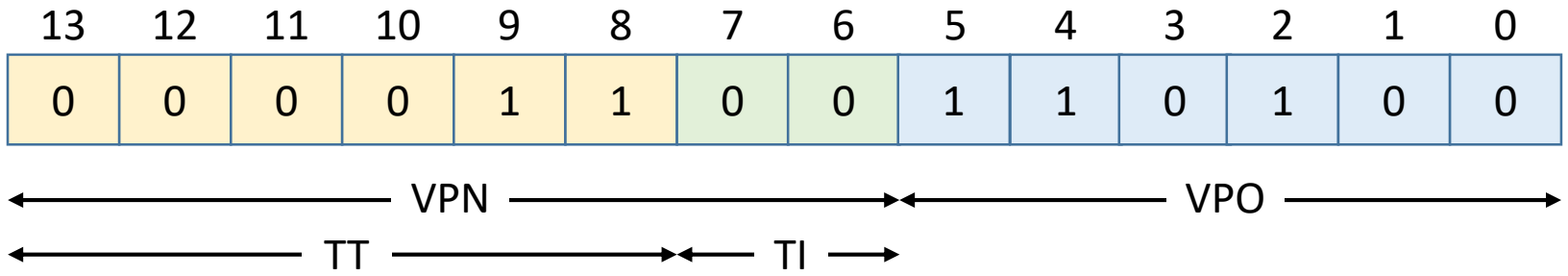
- Read virtual address 0x0334
- First, let's break the address into VPN and VPO:



- VPN: 0x0C
- VPO: 0x34

Simple Memory System Example

- Next, we should break that VPN into TT and TI:



- TT: 0x03
- TI: 0

Simple Memory System Example

- Using TT (0x03) and TI (0) we can index into the 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

- This resulted in a miss, so we go to the page table

Simple Memory System Example

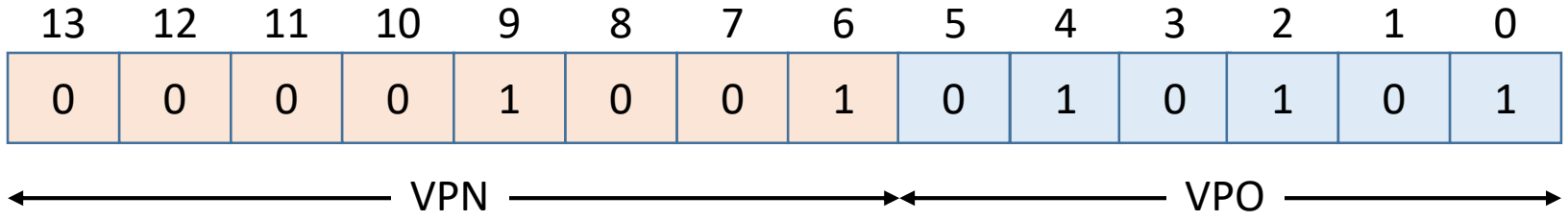
- Because we missed in the TLB, we must use the VPN (0x0C) to look up a PPN in the page table:

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	-	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	**0C**	**-**	**0**
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

- We missed in the page table too, so it is a page fault

Simple Memory System Example

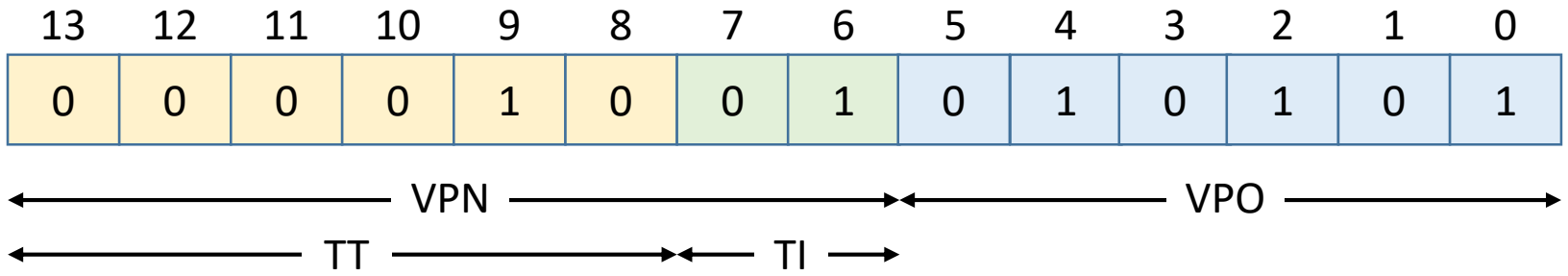
- Read virtual address 0x0255
- First, let's break the address into VPN and VPO:



- VPN: 0x09
- VPO: 0x15

Simple Memory System Example

- Next, we should break that VPN into TT and TI:



- TT: 0x02
- TI: 1

Simple Memory System Example

- Using TT (0x02) and TI (1) we can index into the 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

- This resulted in a miss, so we go to the page table

Simple Memory System Example

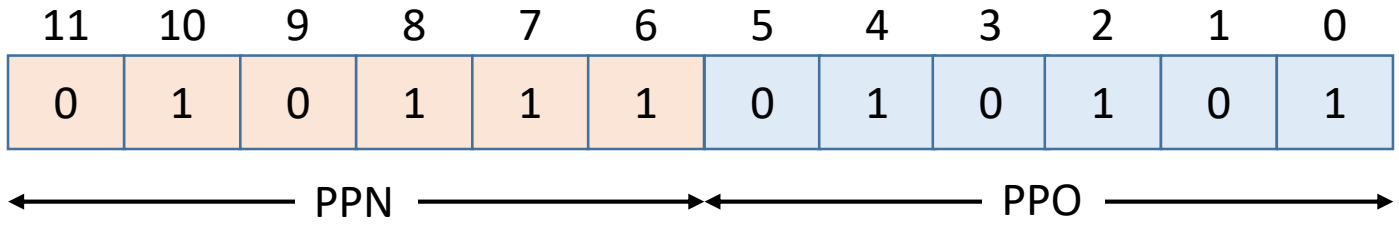
- Because we missed in the TLB, we must use the VPN (0x09) to look up a PPN in the page table:

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	-	0	**09**	**17**	**1**
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	0C	-	0
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

- It’s a hit! We now know that the PPN is 0x17

Simple Memory System Example

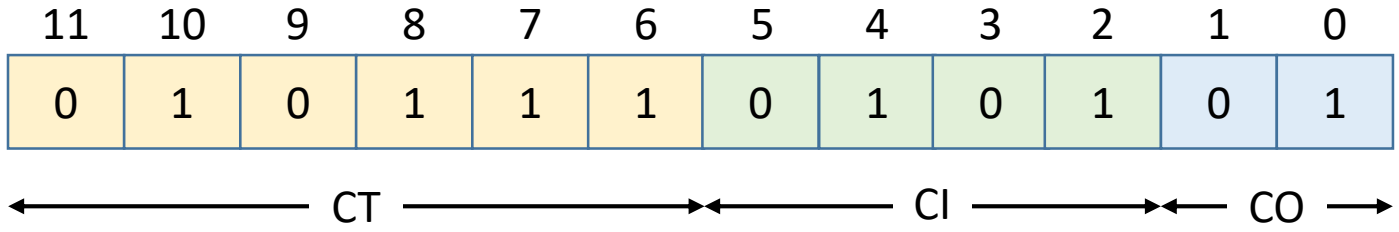
- The next step is to combine the PPN and PPO to form the physical address
  - We know from the page table lookup that the PPN is 0x17
  - The VPO we calculated earlier (0x15) is identical to the PPO



- By combining the two, we see that the physical address is 0x5D5

Simple Memory System Example

- Using a physical address of 0x5D5, we can now do a cache lookup to see if the data we want is in the cache
  - First, we need CT, CI, and CO



- From the diagram above, we get the following:
  - CT = 0x17
  - CI = 5
  - CO = 1

Simple Memory System Example

- Using the values of CT (0x17), CI (5), and CO (3) that we calculated, let's look into set 5 of the cache for a tag matching 0x17:

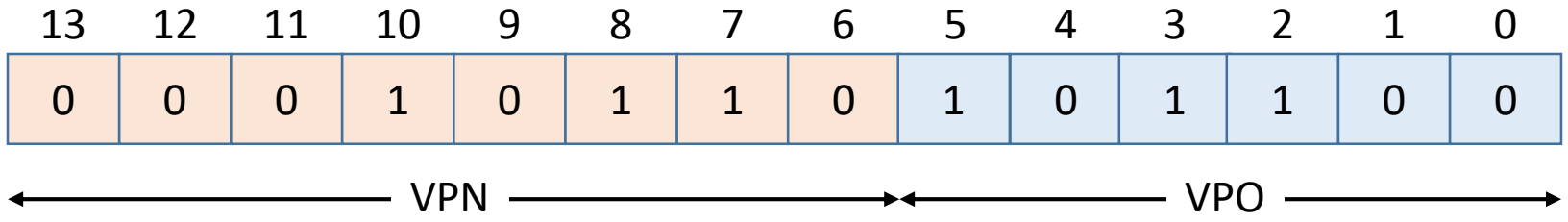
Index	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	-	-	-	-
2	1B	1	00	02	04	08
3	36	0	-	-	-	-
4	32	1	43	6D	8F	09
5	**0D**	**1**	**36**	**72**	**F0**	**1D**
6	31	0	-	-	-	-
7	16	1	11	C2	DF	03

Index	Tag	Valid	B0	B1	B2	B3
8	24	1	3A	00	51	89
9	2D	0	-	-	-	-
A	2D	1	93	15	DA	3B
B	0B	0	-	-	-	-
C	12	0	-	-	-	-
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	-	-	-	-

- The tag in set 5 is 0x0D (not 0x17), so we get a cache miss

Simple Memory System Example

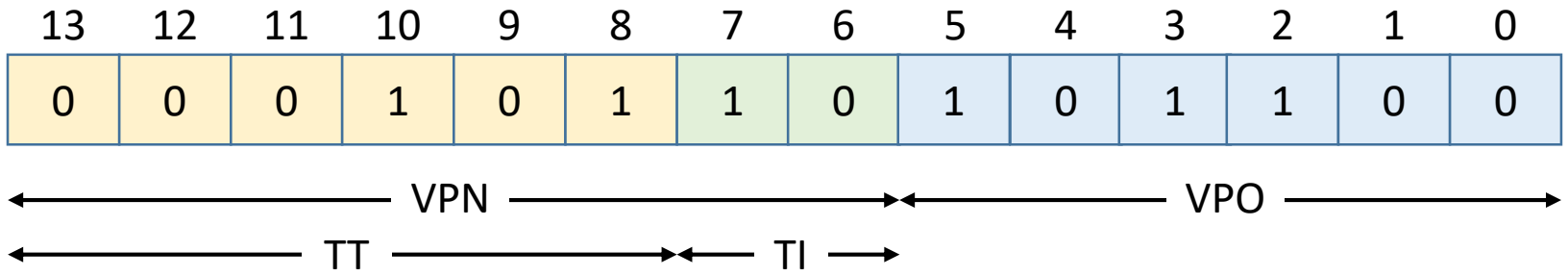
- Read virtual address 0x05AC
- First, let's break the address into VPN and VPO:



- VPN: 0x16
- VPO: 0x2A

Simple Memory System Example

- Next, we should break that VPN into TT and TI:



- TT: 0x05
- TI: 2

Simple Memory System Example

- Using TT (0x05) and TI (2) we can index into the 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

- This resulted in a miss, so we go to the page table

Simple Memory System Example

- Because we missed in the TLB, we must use the VPN (0x16) to look up a PPN in the page table:

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	-	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	0C	-	0
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

- Unfortunately, the PTE for 0x16 is not given, so our answer to the question “did a page fault occur?” is unknown