Virtual Memory

15-213: Introduction to Computer Systems
Recitation 10, Oct. 28, 2013

Jane Zeng

Recitation I

Update

- **Shell lab is due Tuesday(tomorrow), 11:59PM**
- **Malloc lab is out Tuesday(tomorrow), 11:59PM**
  - Due Thursday Nov. 14
  - Start early

Agenda

- **Virtual memory concepts**
- **Address translation**
  - TLB
  - Multilevel page table

Virtual memory concepts

- **Virtual address:**
  - Each process has its own address space
  - Programs are written with respect to virtual address – “linear array”
  - Page: region of virtual memory

- **Physical address:**
  - Where process ends up in memory
  - Frame: region of physical memory

- **Define a mapping from virtual address(page) to physical address(frame) for each process**

Virtual memory concepts

- **TLB**
  - A cache for page table entries
  - Map virtual address to physical address directly
  - Speed up translation

- **Multi-level page table**
  - Address space is often sparse
  - Use page directory to map large chunk of memory to page table
  - Mark large unmapped regions as non-present in page directory instead of storing page tables full of invalid entries

Agenda

- Virtual memory concepts
- **Address translation**
  - TLB
  - Multilevel page table

Summary of Address Translation Symbols

- **Basic Parameters**
  - **N = 2^n** : Number of addresses in virtual address space
  - **M = 2^m** : Number of addresses in physical address space
  - **P = 2^p** : Page size (bytes)
- **Components of the virtual address (VA)**
  - **TLBI**: TLB index
  - **TLBT**: TLB tag
  - **VPO**: Virtual page offset
  - **VPN**: Virtual page number
- **Components of the physical address (PA)**
  - **PPO**: Physical page offset (same as VPO)
  - **PPN:** Physical page number
  - **CO**: Byte offset within cache line
  - **CI:** Cache index
  - **CT**: Cache tag

Example 1 – Address translation with TLB

- **Addressing**
  - 1MB of virtual memory
  - 256KB of physical memory
  - 4KB Page size
  - TLB is 2-way set associative with 8 entries in total

Question:

(a) How many bits are needed to represent the virtual address space?

(b) How many bits are needed to represent the physical address space?

(c) How many bits are needed to represent VPN?

Address translation with TLB

- **Addressing**
  - 1MB of virtual memory
  - 256KB of physical memory
  - 4KB Page size
  - TLB is 2-way set associative with 8 entries in total

Basics:

(a) How many bits are needed to represent the virtual address space?

Answer: 20 bits

(b) How many bits are needed to represent the physical address space?

Answer: 18 bits

(c) How many bits are needed to represent VPN?

Answer: 8 bits

4 KB Page size -> 12 bits for VPO

20-bit virtual address, one-level page table -> 20 – 12 = 8 bits

Address translation - Case 1:

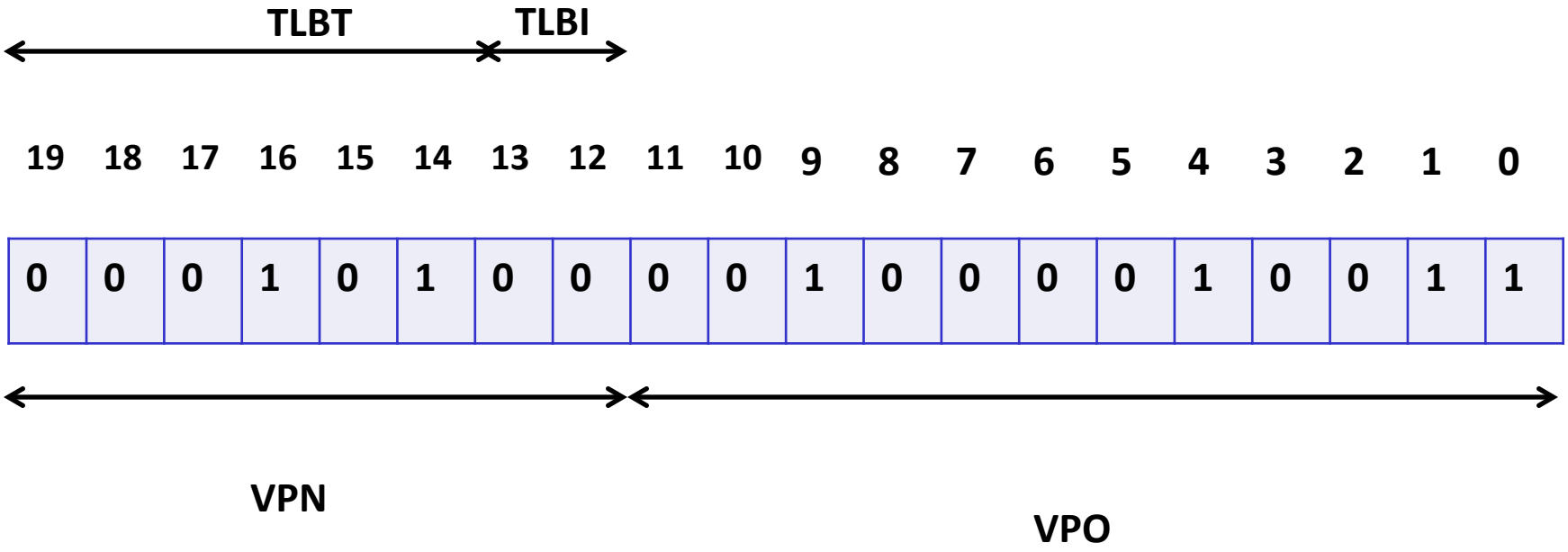
- Translate 0x14213, given the contents of TLB and the first 32 entries of the page table below. (All the numbers are in hexadecimal)

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Step I: Look up in TLB

8 entries
2-way associative



Step I: Look up in TLB

- TLBI: 0x00
- TLBT: 0x05

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

TLB Hit

PPN: 0x13 (from TLB)
PPO = VPO = 0x213

Physical address:
0x13213

Comments

- TLB hit is fast.
- Again, locality matters.

Address translation - Case 2:

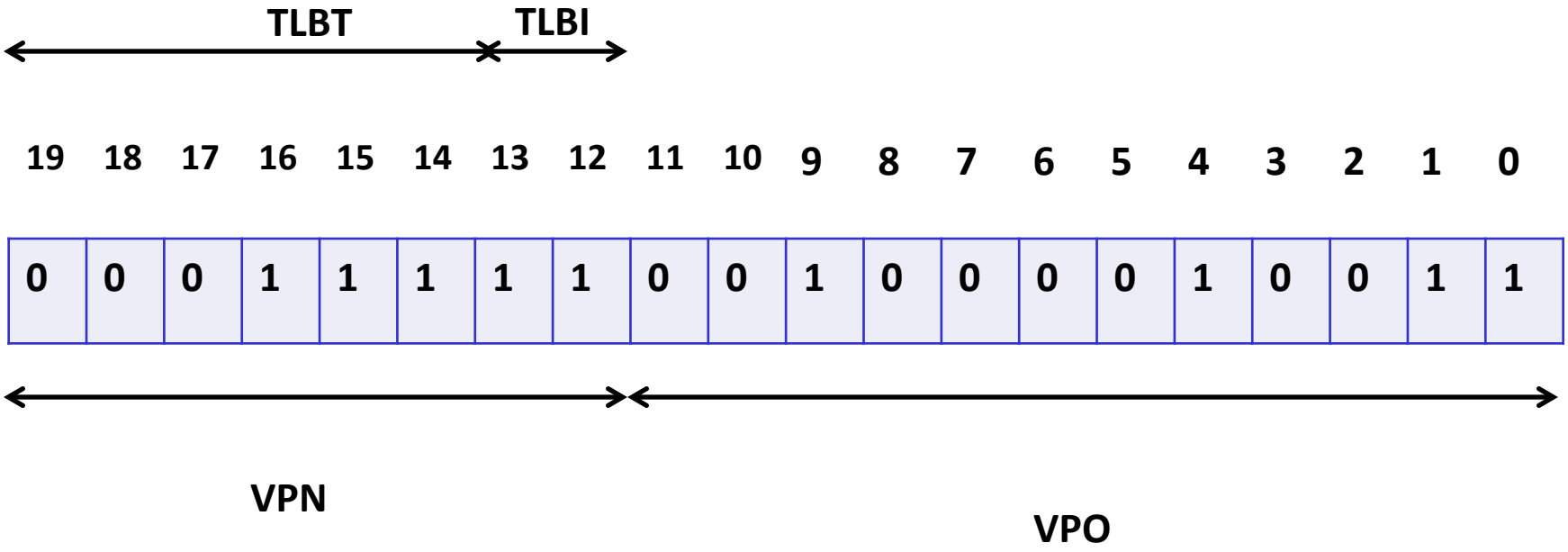
- Translate 0x1F213, given the contents of TLB and the first 32 entries of the page table below.

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Step I: Look up in TLB

8 entries
2-way associative



Step I: Look up in TLB

- TLBI: 0x03
- TLBT: 0x07

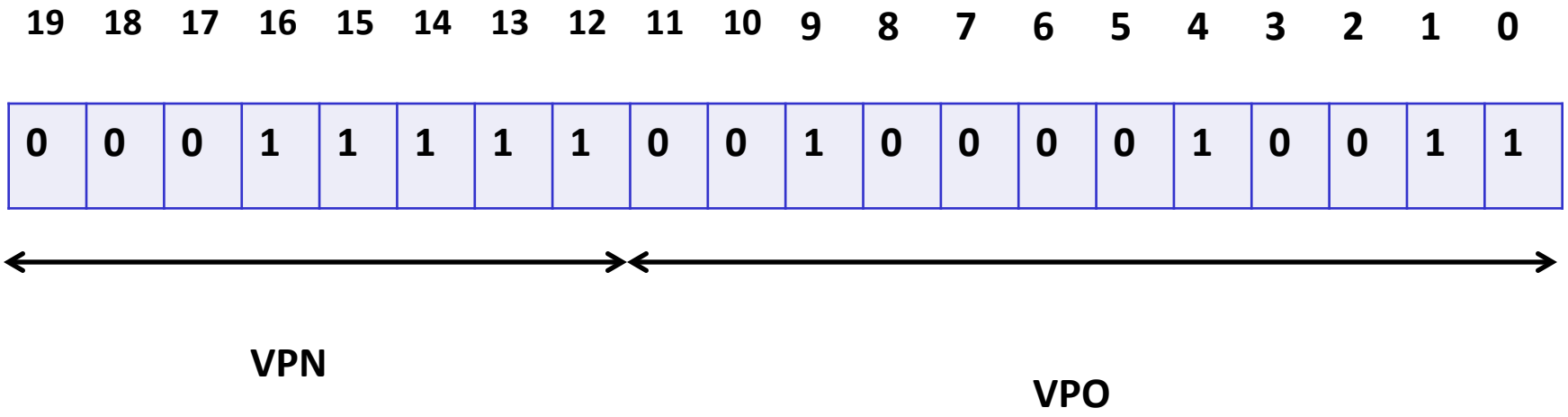
TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

TLB Miss

Step II: Look up in page table

Step II: Look up in page table

VPN: 0x1F
VPO: 0x213



Step II: Look up in page table

- Look up VPN in page table

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

VPN = 15
- Valid Entry
- PPN = 0x15
- PPO = VPO = 0x213
- Physical address is 0x15213

(If entry is not valid, raise page fault exception)

Agenda

- Virtual memory concepts
- **Address translation**
  - TLB
  - Multilevel page table

Example 2: Two-level page table

- **The example deals with virtual memory address translation using Intel IA-32 architecture's page translation mechanism.** **(diagrams in the following slides are from Intel System Programmers Guide)**

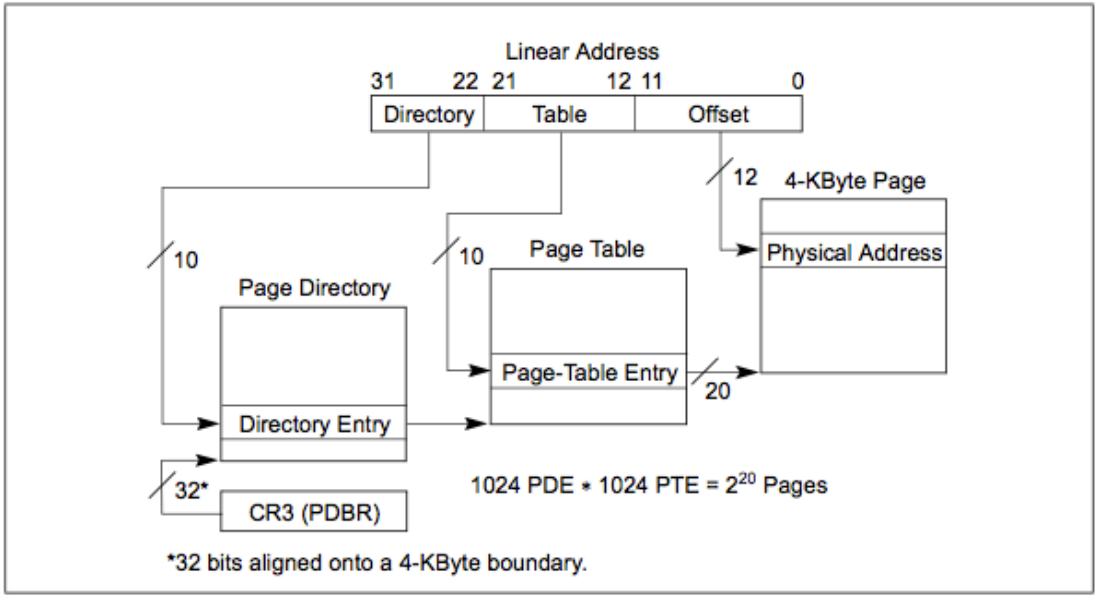
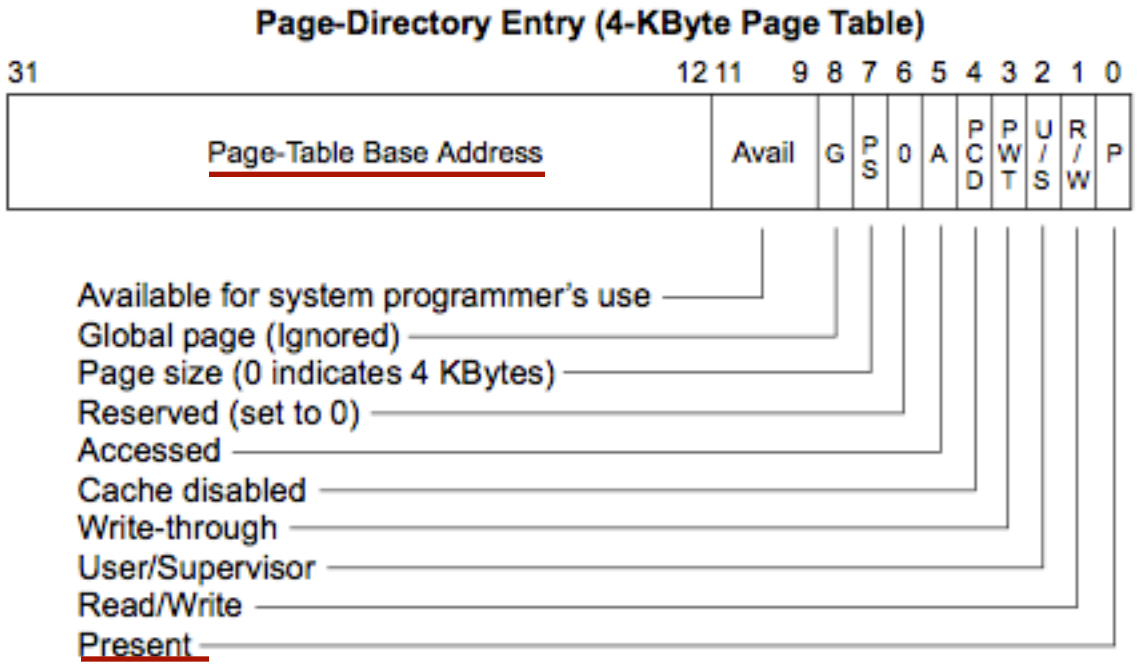
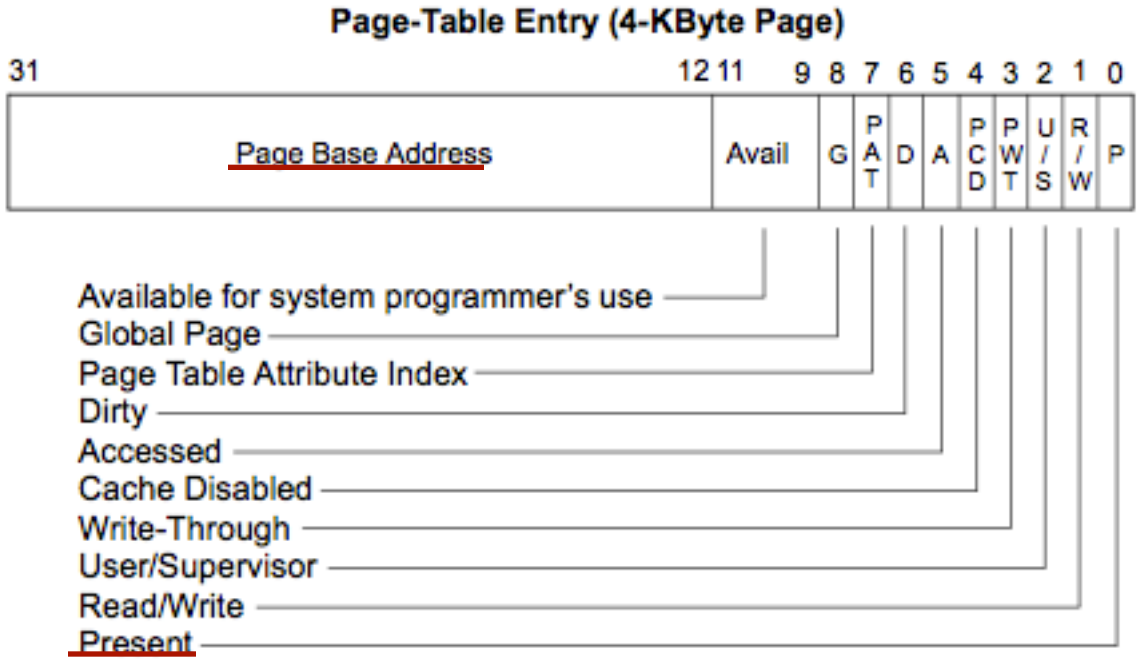


Figure 3-12. Linear Address Translation (4-KByte Pages)

IA-32 Page-Directory Entry Format



IA-32 Page-Table Entry Format



- **The contents of the relevant sections of memory are shown below. All numbers are given in hexadecimal. Any memory not shown can be assumed to be zero.**
- **Page Directory Base Address is 0x0c23b000.**

Address	Content	Address	Content
00023000	beefbee0	00023120	12fdc883
00023200	debcfd23	00023320	d2e52933
00023fff	bcdeff29	00055004	8974d003
0005545c	457bc293	00055460	457bd293
00055464	457be293	0c23b020	01288b53
0c23b020	01288b53	0c23b040	012aab53
0c23b080	00055d01	0c23b09d	0ff2d303
0c23b274	00023d03	0c23b9fc	2314d222
2314d200	0fdc1223	2314d220	d21345a9
2314d4a0	d388bcbd	2314d890	00b32d00
24aee520	b58cdad1	29de2504	56ffad02

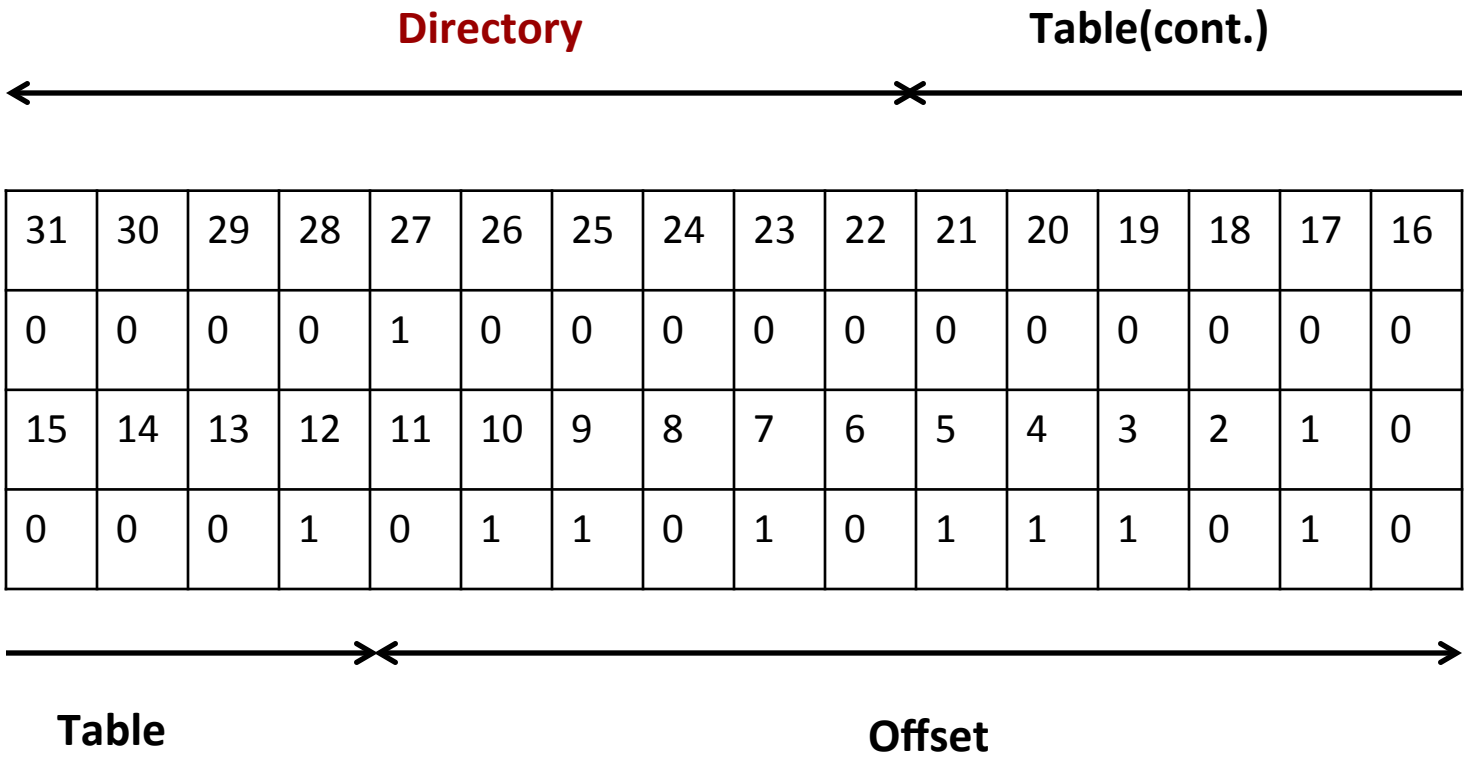
Address translation - Case 1

- **Read from virtual address 0x080016ba**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Case 1 Parse PDE

- **Read from virtual address 0x080016ba**



Calculate PDE address

- PDE: 0X20
- Page-directory base address: 0x0c23b000
- Each entry is 4 bytes

  - PDE address = 0x0c23b000 + 4 * 0x20

    = 0x0c23b080

Address	Content	Address	Content
00023000	beefbee0	00023120	12fdc883
00023200	debcfd23	00023320	d2e52933
00023fff	bcdeff29	00055004	8974d003
0005545c	457bc293	00055460	457bd293
00055464	457be293	0c23b020	01288b53
0c23b020	01288b53	0c23b040	012aab53
0c23b080	00055d01	0c23b09d	0ff2d303
0c23b274	00023d03	0c23b9fc	2314d222
2314d200	0fdc1223	2314d220	D21345a9
2314d4a0	D388bcbd	2314d890	00b32d00
24aee520	B58cdad1	29de2504	56ffad02

Step1: Check present bit (last bit according to IA-32 format)

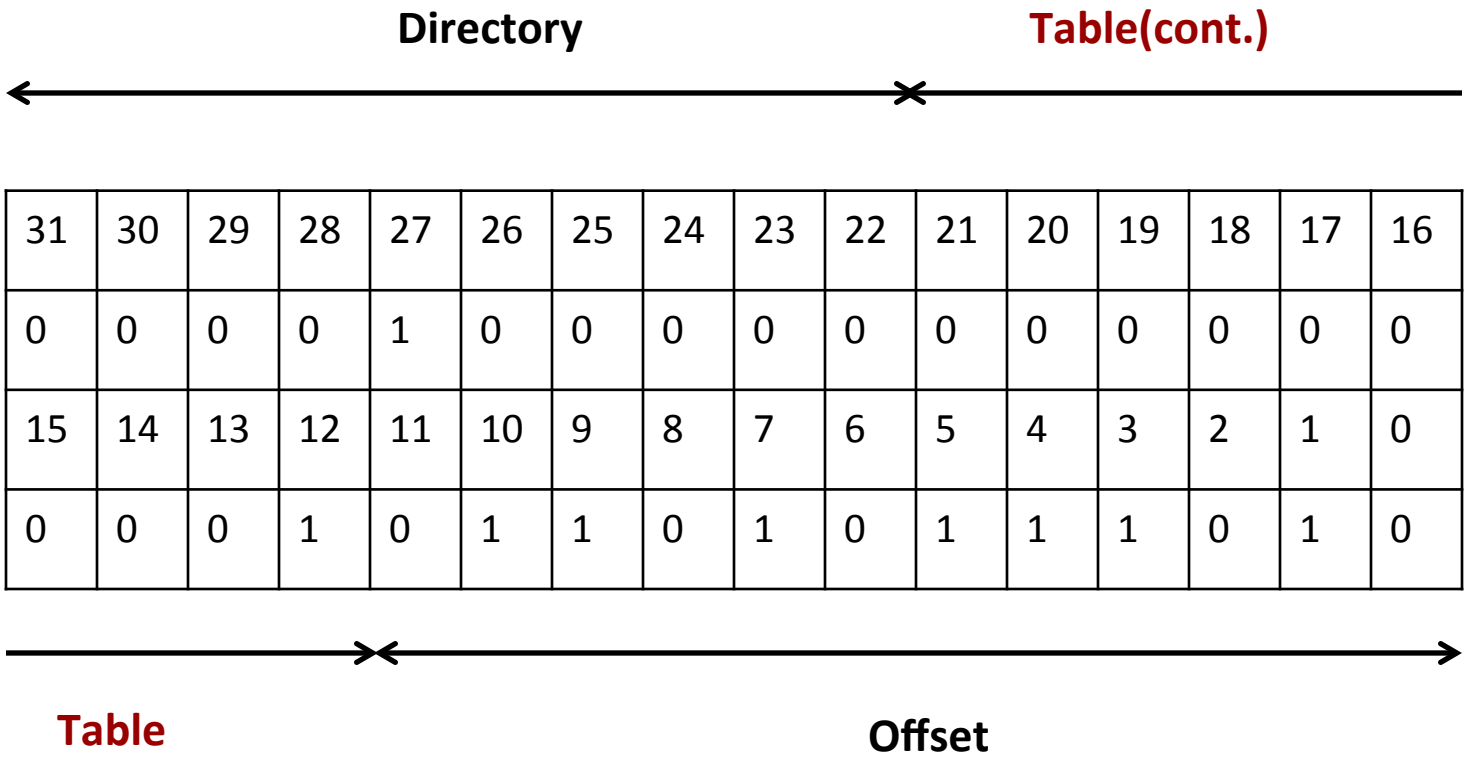
- Present bit is set

Step2: Calculate page table base address (Ignore other flags for now)

- PT base address: 0x00055d01 & 0xFFFFF000=0x00055000

Case 1 Parse PTE

- Read from virtual address 0x080016ba



Calculate PTE address

- PTE: 0x01
- Page table base address: 0x55000
- PTE size = 4 bytes
- PTE address: 0x55000 + 0x1 * 4 = 0x55004

Address	Content	Address	Content
00023000	beefbee0	00023120	12fdc883
00023200	debcfd23	00023320	d2e52933
00023fff	bcdeff29	00055004	8974d003
0005545c	457bc293	00055460	457bd293
00055464	457be293	0c23b020	01288b53
0c23b020	01288b53	0c23b040	012aab53
0c23b080	00055d01	0c23b09d	0ff2d303
0c23b274	00023d03	0c23b9fc	2314d222
2314d200	0fdc1223	2314d220	D21345a9
2314d4a0	D388bcbd	2314d890	00b32d00
24aee520	B58cdad1	29de2504	56ffad02

Step 1: Check present bit (last bit according to PTE format)

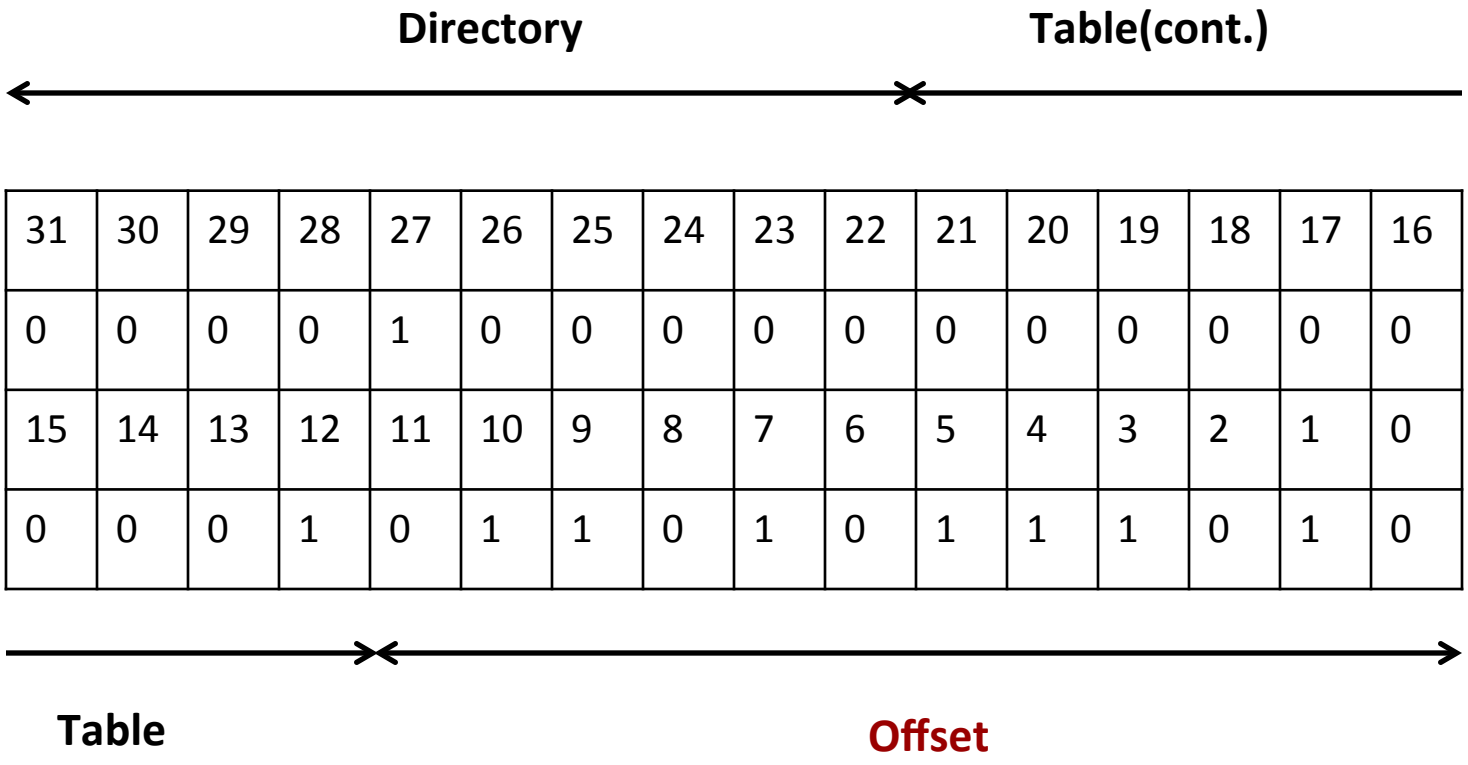
- Present

Step 2: Calculate page base address

- Page base address: 0x8974d003 & 0xFFFFF000=0x8974d000

Case 1 Parse offset

- Read from virtual address 0x080016ba



Calculate physical address

- Offset = 0x6ba (Same for physical and virtual address)
- Page base address = 0x8974d000

- Physical address = 0x8974d6ba
- (If page base is valid, all addresses in that page are considered to be “valid”)

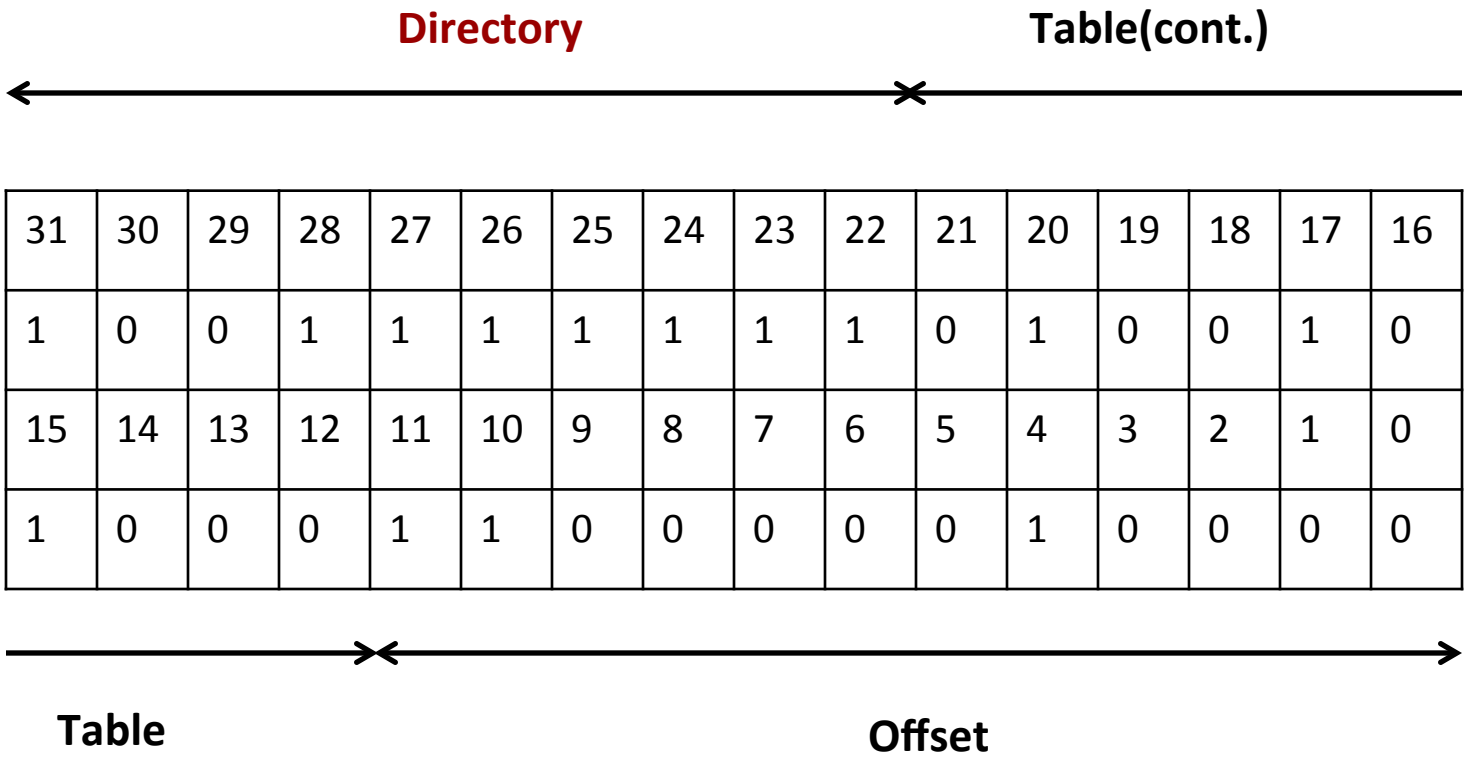
Address translation - Case 2

- **Read from virtual address 0x9fd28c10**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Address translation - Case 2

- Read from virtual address 0x9fd28c10



Calculate PDE address

- PDE: 0X27F
- Page-directory base address: 0x0c23b000
- Each entry is 4 bytes

  - PDE address = 0x0c23b000 + 4 * 0x27F

    = 0x0c23b9FC

Address	Content	Address	Content
00023000	beefbee0	00023120	12fdc883
00023200	debcfd23	00023320	d2e52933
00023fff	bcdeff29	00055004	8974d003
0005545c	457bc293	00055460	457bd293
00055464	457be293	0c23b020	01288b53
0c23b020	01288b53	0c23b040	012aab53
0c23b080	00055d01	0c23b09d	0ff2d303
0c23b274	00023d03	0c23b9fc	2314d222
2314d200	0fdc1223	2314d220	D21345a9
2314d4a0	D388bcbd	2314d890	00b32d00
24aee520	B58cdad1	29de2504	56ffad02

Last bit of PTE 0x2314d222 is not set

- **Non-present page**

Page fault – trap into kernel

Question?