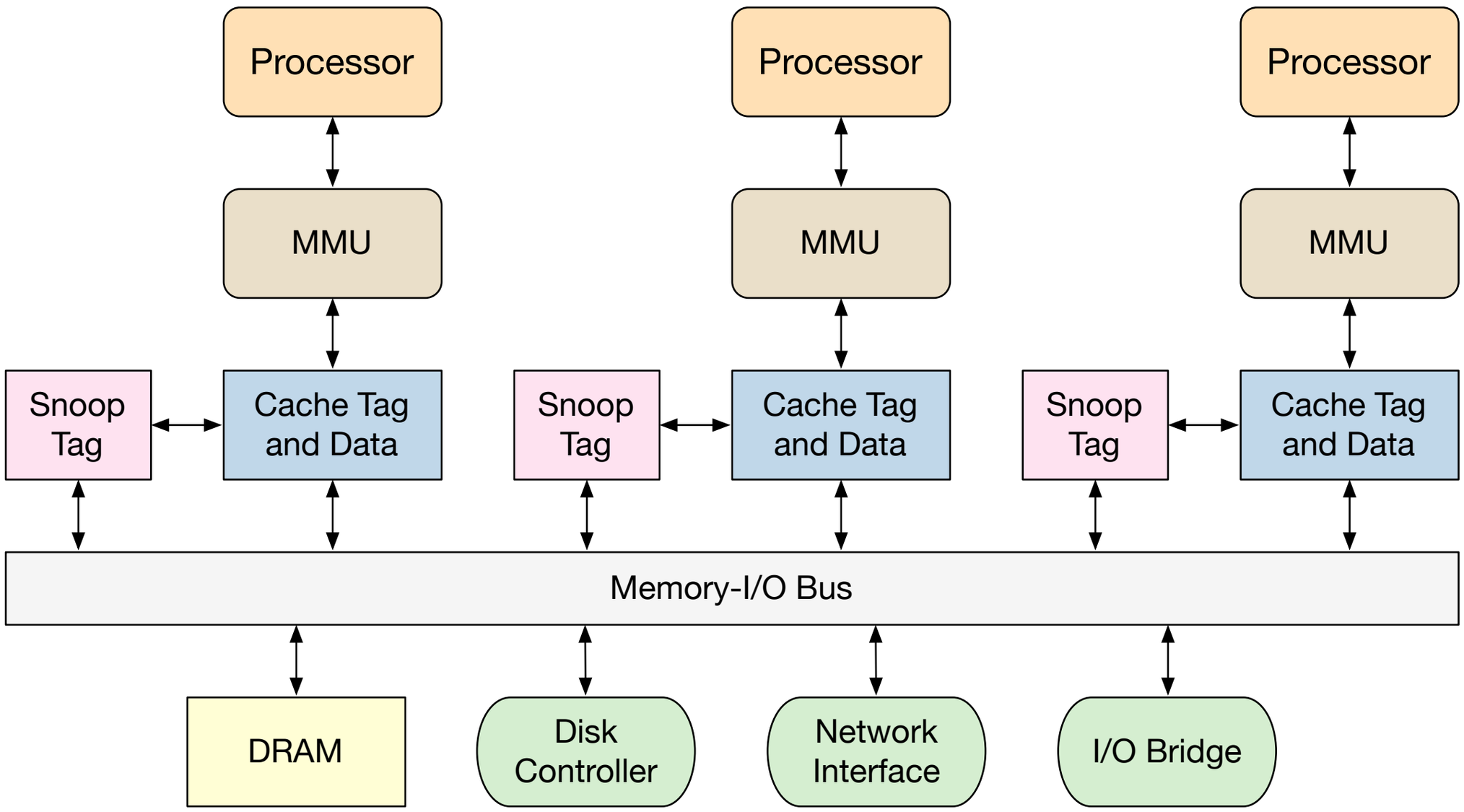
Lecture 22: Address Spaces

Spring 2024
Jason Tang

Topics

- Page table entries
- Memory-mapped I/O
- Direct memory access

System Interconnect



- When the processor generate an address, how does the computer know which device to access on the memory-I/O bus?

Intel Paging

- Traditional Intel x86 (32-bit) had segmented memory; modern (64-bit) has a flat memory model with 64-bit addresses (called long mode)

- Address of the page table is set in the Page-Directory-Table Base Address

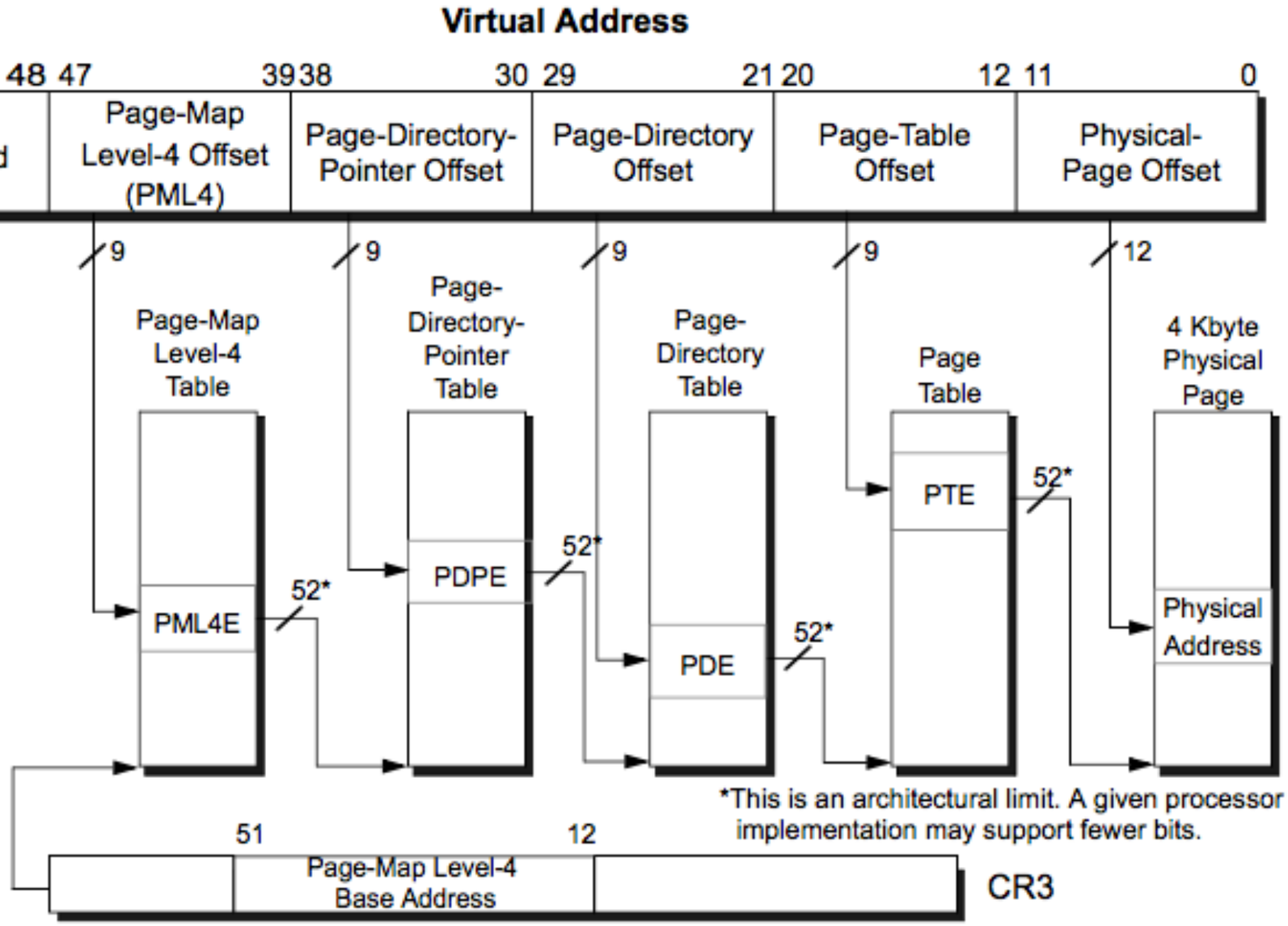
- Long mode supports multiple page sizes:

Page Size	Page Table Levels
4 KiB	4
2 MiB	3
1 GiB	2

AMD64 Architecture Programmer's Manual, §5.3

Intel Paging

- Even though virtual addresses extend to 64 bits, currently x86-64 only use the lower 48 bits (256 TiB virtual address space)



AMD64 Architecture Programmer's Manual, §5.3

Intel Page Table Entries

- Currently x86-64 supports up to 52 bits of physical memory (4 PiB physical address space)

- Page table entries differ slightly between the different levels, but generally contain the similar information

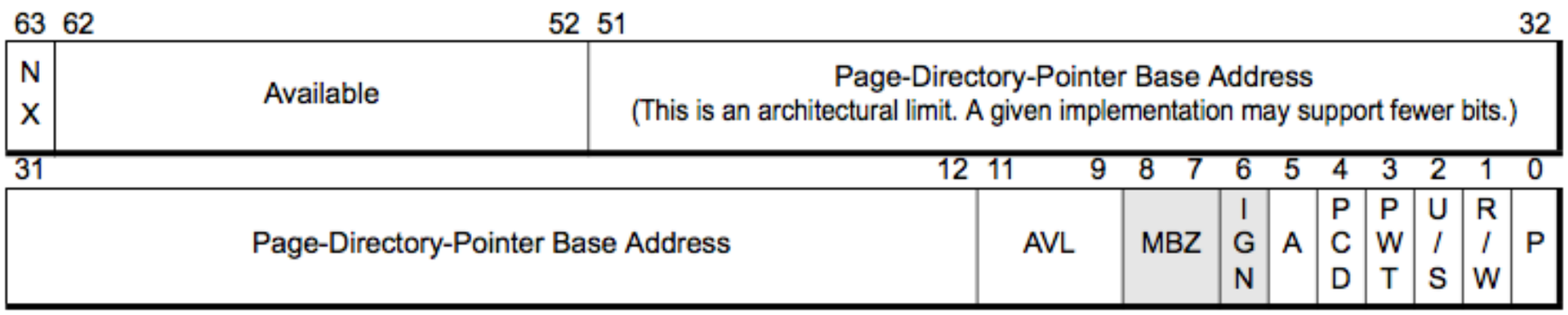


Figure 5-18. 4-Kbyte PML4E—Long Mode

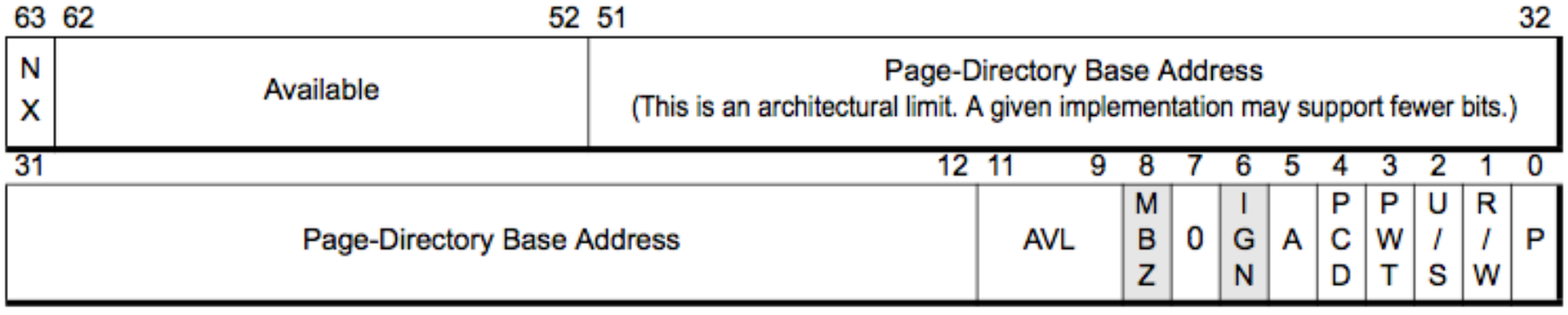


Figure 5-19. 4-Kbyte PDPE—Long Mode

Intel Page Table Entry Fields

Field	Meaning
Present (P)	If set to 1, page is loaded in physical memory
Read/Write (R/W)	If set to 1, both read and write accesses are allowed to page
User/Supervisor (U/S)	If set to 1, both user and supervisor accesses are allowed to page
Page-Level Writethrough (PWT)	If set to 1, page has a writethrough caching policy
Accessed (A)	Processor sets this bit to 1 the first time the page is read from or written to
Dirty (D)	Processor sets this bit to 1 the first time there is a write to the page
No Execute (NX)	If set to 1, code cannot be executed from the page

Meltdown Vulnerability

- Meltdown and Spectre are well publicized vulnerabilities of Intel, AMD, ARM, and other processors

- Meltdown relies upon several properties of modern CPUs:
  - CPU data caches,
  - Protection bits in page table entries, and
  - Out of order execution

- With all of the above combined, an attacker could obtain any value from memory, including passwords and other security credentials

https://meltdownattack.com/, https://hackernoon.com/a-simplified-explanation-of-the-meltdown-cpu-vulnerability-ad316cd0f0de

Meltdown Vulnerability

- Attacker allocates 256 memory pages (256 × 4096 bytes)
    - Attacker does not access memory, keeping data cache cold
- It then executes C code that causes a data cache fill, *even though the instruction never finishes executing*

```
char *receiver_addr = malloc(256 * 4096);
char *victim_value; // pointer to value to exfiltrate
*(0) = 1;           // cause a memory violation
receiver_addr[*victim_value * 4096] = 1;
```

- Then measure how long it takes to read from each of the 256 memory pages
    - The page with fastest access is the secret value

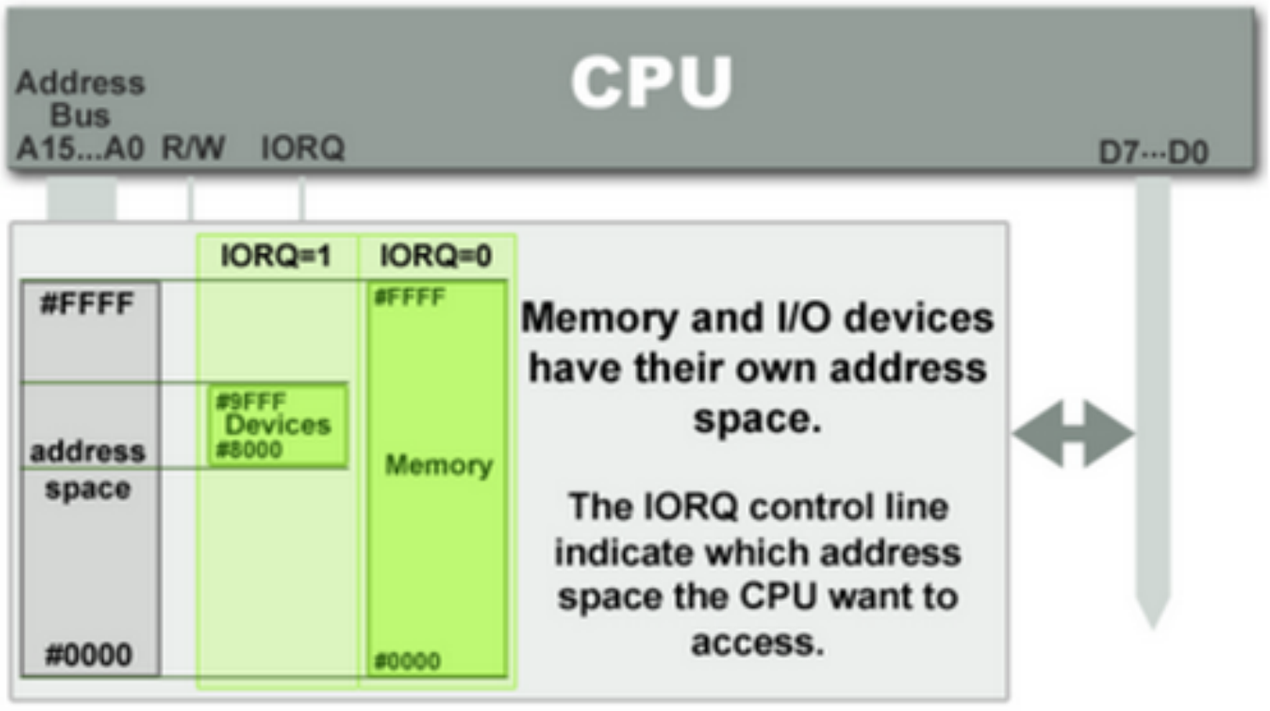
I/O Access

- Processor-to-memory access is handled via loads and stores
  - Processor generates a virtual address, which is translated by MMU into a memory physical address, and then the memory controller fetches data from main memory
- Two ways to handle processor-to-device:
  - Port I/O (PMIO): special instructions to access devices
  - Memory-mapped I/O (MMIO): same normal load and store instructions, but address decoder redirects request to a device instead of main memory

http://www.bogotobogo.com/Embedded/memory_mapped_io_vs_port_mapped_isolated_io.php

Port I/O

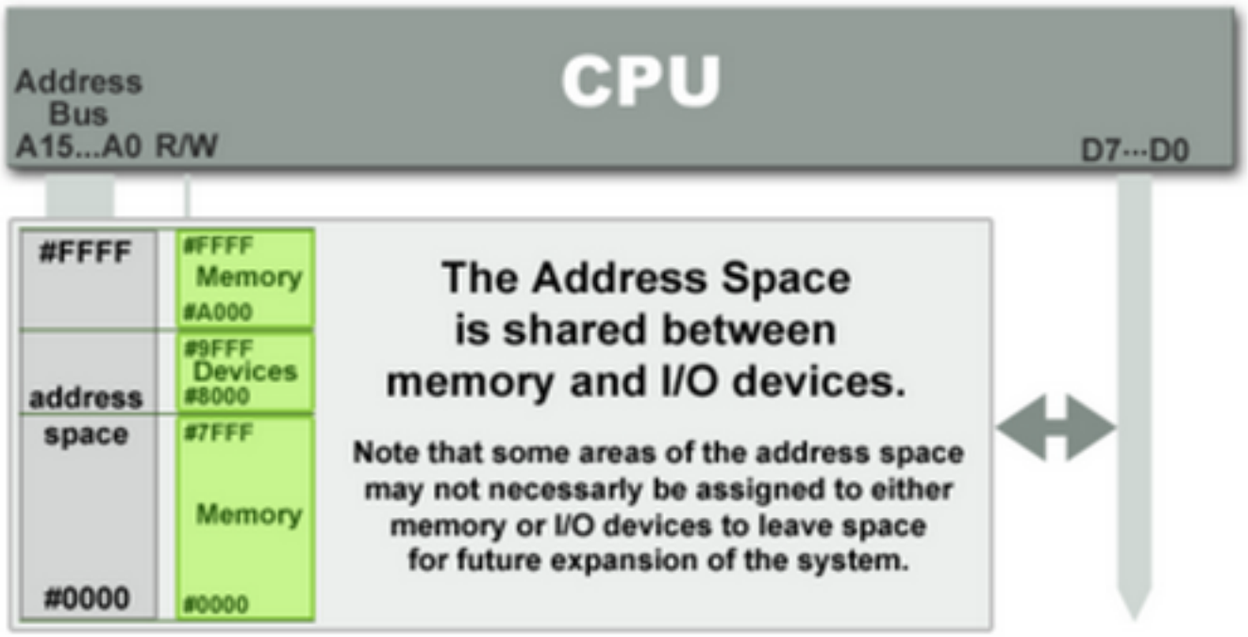
- Dedicated address space for I/O devices
  - Necessary on older systems, where physical [memory] address space is only 16 or 32 bits



- Separate lines for memory and devices
  - Allows each component run at different clock speeds
- Harder for software to transfer data between memory and devices

Memory-Mapped I/O

- Memory and devices share same physical address space
    - MMU still translates a virtual address to physical address
    - Physical addresses may refer to devices or main memory
    - Common on modern systems, with 32-bit and 64-bit physical addresses
- Same bus connects memory and devices
    - Data cache **must be** disabled when accessing devices



Address Decoder

- Component that takes a physical address and routes the load or store request to memory or to a given I/O device

- For a MMIO system, physical memory only part of physical address space; the rest of address space is used by I/O devices

- Example: the original Raspberry Pi has a Broadcom BCM2835 system on chip (SoC):

> Physical addresses start at 0x00000000 for RAM.
> …
> Physical addresses range from 0x20000000 to 0x20FFFFFF for peripherals. The bus addresses for peripherals are set up to map onto the peripheral bus address range starting at 0x7E000000. Thus a peripheral advertised here at bus address 0x7Ennnnnn is available at physical address 0x20nnnnnn.

BCM2835 ARM Peripherals, §1.2.3

Datasheets

- Manufacturer describes device's layout in its published datasheets

- Example: BCM2835 has a memory mapped UART device:

Address	Register Name	Description	Size
0x7E21 5000	AUX_IRQ	Auxiliary Interrupt status	3
0x7E21 5004	AUX_ENABLES	Auxiliary enables	3
0x7E21 5040	AUX_MU_IO_REG	Mini Uart I/O Data	8
0x7E21 5044	AUX_MU_IER_REG	Mini Uart Interrupt Enable	8
0x7E21 5048	AUX_MU_IIR_REG	Mini Uart Interrupt Identify	8
0x7E21 504C	AUX_MU_LCR_REG	Mini Uart Line Control	8
0x7E21 5050	AUX_MU_MCR_REG	Mini Uart Modem Control	8

- Reading from a device register usually returns the device's status; writing to a device register usually causes some physical reaction

BCM2835 ARM Peripherals, §2.1

Direct Memory Access

- Memory controller handles processor-to-memory, and address decoder handles processor-to-device, but what about between memory and devices?

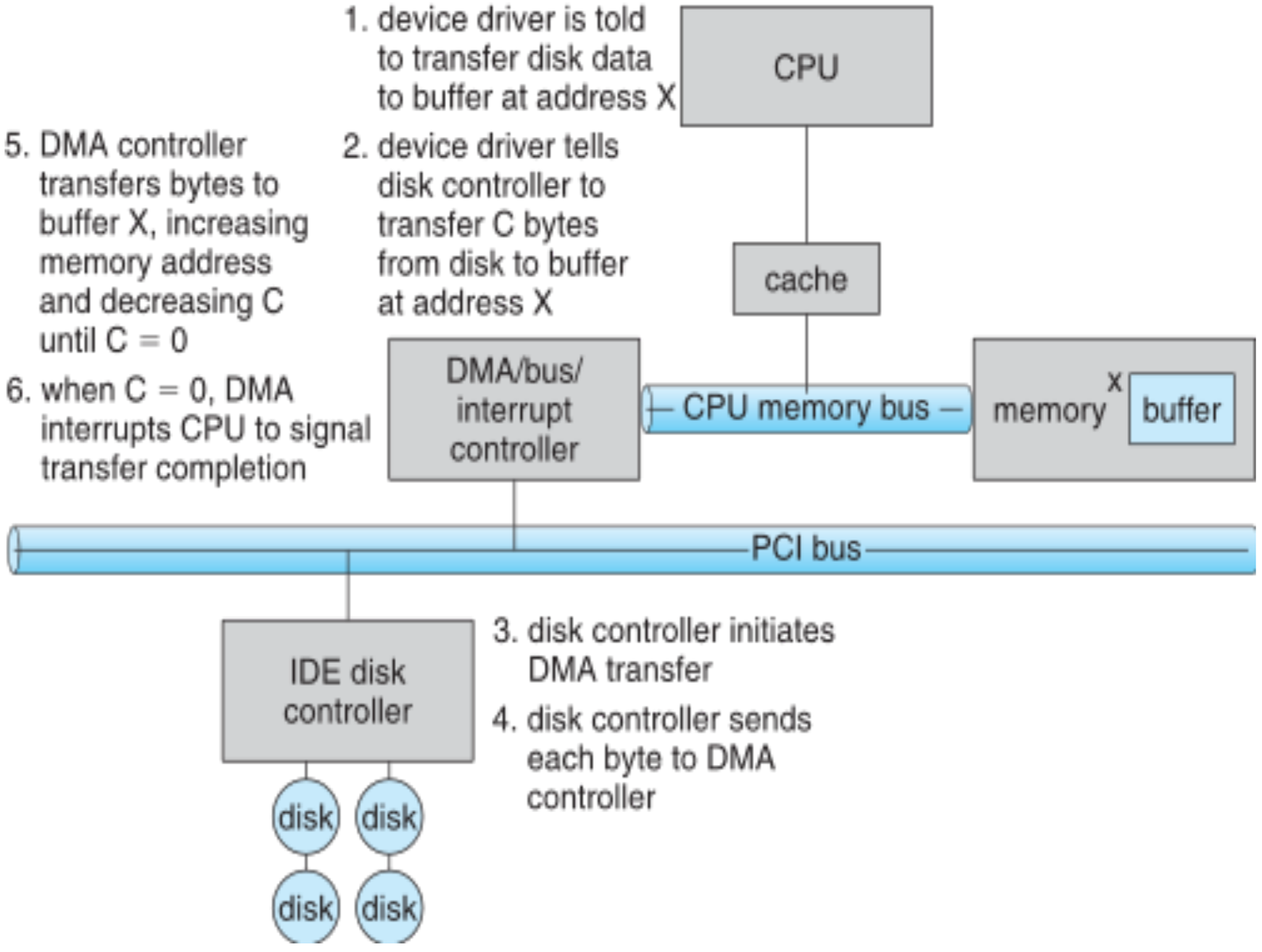
- Programmed I/O (PIO): processor responsible for copying data from device to CPU register, and then from CPU register to memory

- Direct Memory Access (DMA): allows devices to read and write directly to memory, bypassing processor
  - DMA controller handles DMA requests
  - Allows devices to work in the background, while processor continues executing main software

DMA Controllers

- In simple case, processor writes a transaction descriptor (TxD) to DMA controller
    - TxD, at a minimum, consists of a source address (memory or device), destination address (memory or device), and number of bytes to transfer
    - TxD are bus addresses, neither physical nor virtual
- Processor then commands DMA controller to start processing TxD
    - Usually, DMA controller raises an interrupt when it completes transaction

Simple DMA Transfer



DMA Transfer Types

- Single-cycle: write a single word to/from device
  - Good for slow-speed devices, like UARTs
- Burst: transfer a block of data, over many clock cycles
  - Used when writing to GPU memory, network cards, and hard disks
  - During transfer, processor is unable to use bus
- Cyclic: repeatedly transfer data
  - Used for sound cards

http://www.pebblebay.com/direct-memory-access-embedded-systems/

DMA Coherency

- Processor's data cache is usually not updated when DMA controller transfers from device to memory

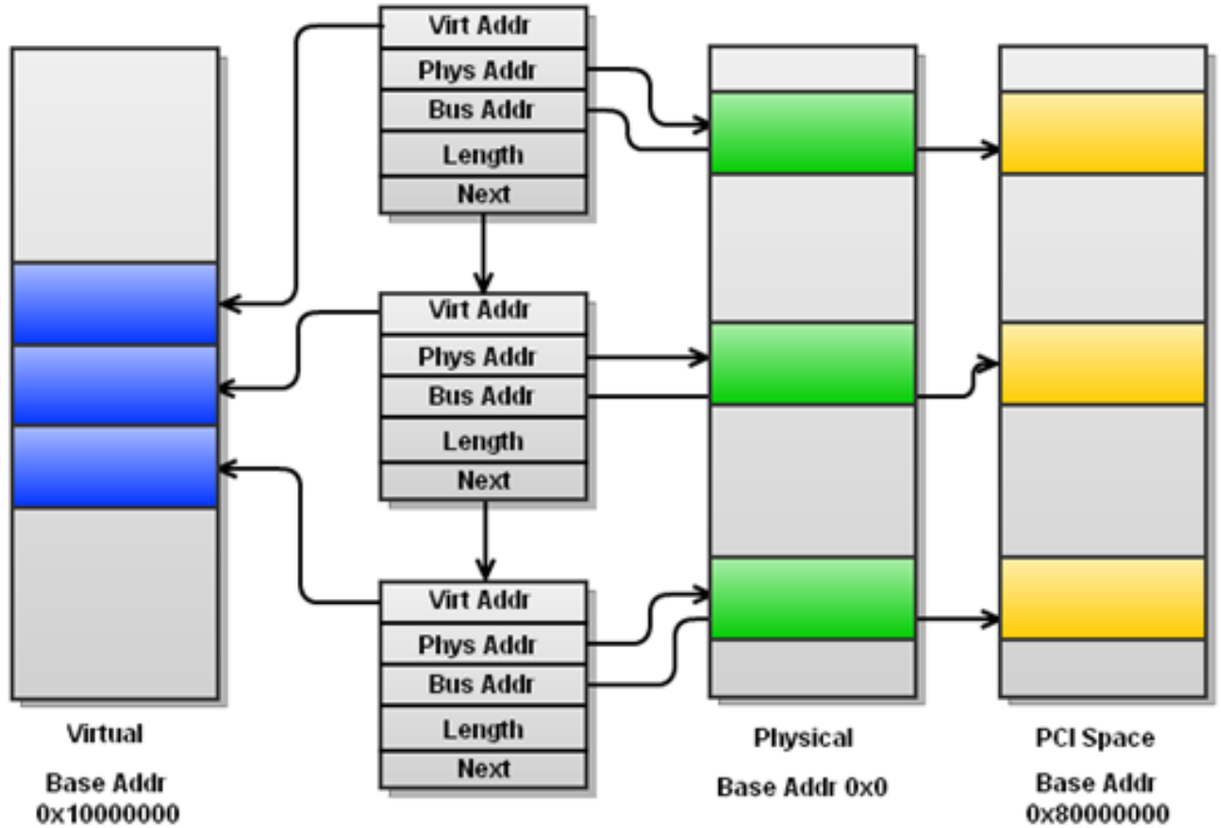
- If data cache is still dirty, DMA controller will transfer stale contents from memory to device

- Some fancier systems allow DMA controller to snoop the memory bus (same resolution as SMP caches)

- If DMA controller cannot snoop, software must explicitly flush and/or invalidate the data cache, at the *virtual address* associated with memory's *physical address*

Bus Address

- Whereas processors mostly run within virtual addresses, and memory is accessed with physical addresses, devices have their own address space



- When building TxDs, software needs to be careful when writing source and destination addresses

  - Often, a bus address is equal to physical address, plus some offset

  - Example: For BCM2835, “a peripheral advertised here at bus address 0x7Ennnnnn is available at physical address 0x20nnnnnn.”

I/O MMU

- Just as the MMU translates virtual to physical addresses, and has protection bits, a I/O MMU translates I/O virtual addresses to bus addresses

- Without I/O MMU, a malicious device could initiate a DMA to overwrite memory with malware, bypassing all software security

- I/O MMU blocks attempts to overwrite unmapped memory

- I/O MMU permits virtualization of I/O, useful for virtual machines

https://thehackernews.com/2019/02/thunderbolt-peripheral-dma-attacks.html