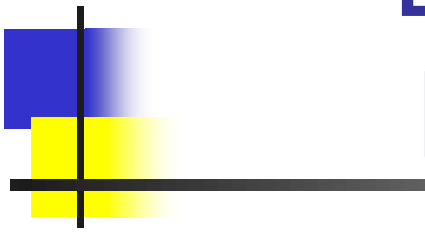
8 : Multiprocessor Systems, Exception Handling, Basic I/O Concepts



Note :

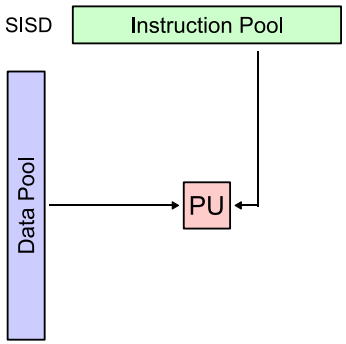
- Slides not covered in detail in the class are left as a self-learning exercise

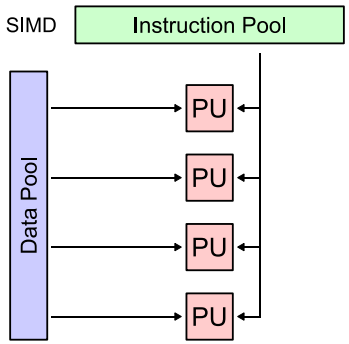
Acknowledgement :

- Relevant Wikipedia articles
- Text by Harris and Harris and companion materials

Flynn's Taxonomy of Computer Arch.

- Single instruction stream single data stream (SISD)
  - A sequential computer which exploits no parallelism in either the instruction or data stream
  - Traditional uniprocessor machines
- Single instruction stream, multiple data streams (SIMD)
  - A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized
  - Exploits Data parallelism
  - For example, an array processor or graphics processing unit (GPU)

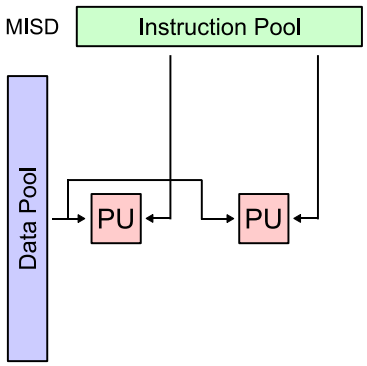


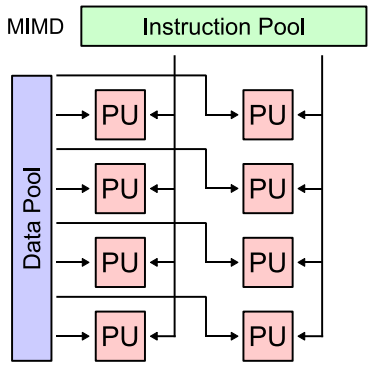


To Do :
Read up about
SSE Instructions

Flynn's Taxonomy of Computer Arch.

- Multiple instruction streams, single data stream (MISD)
  - Multiple instructions operate on one data stream
  - Uncommon architecture which is generally used for fault tolerance (ex : autopilot)
- Multiple instruction streams, multiple data streams (MIMD)
  - Multiple autonomous processors simultaneously executing different instructions on different data
  - MIMD architectures include parallel / distributed systems, using either one shared memory space or a distributed memory space





Multiprocessor Systems

Motivation : Diminishing returns with increased clock frequency / cost

- The memory wall : the increasing gap between processor and memory speeds
- The ILP wall : the increasing difficulty of finding enough parallelism in a single instruction stream to keep a high-performance single-core processor busy
- The power wall : increasing power with operating frequency. It poses manufacturing, system design and deployment problems
- Multiprocessor systems exploit thread-level parallelism instead of trying to increase the performance of a single instruction stream

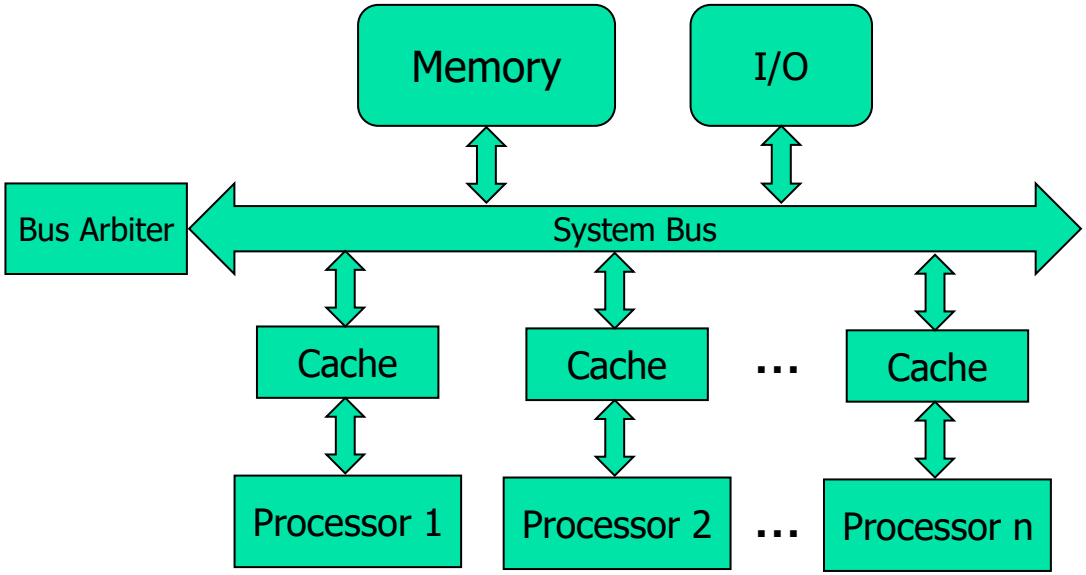
Loosely Coupled Multiprocessor Systems

- Loosely coupled (*distributed*) multiprocessor systems – each node runs different OS instances and communicate by passing messages rather than through a shared memory
- Computer clusters - closely coupled - centralized job management & scheduling
  - Have nodes set to perform the same task, controlled and scheduled by software
  - Typically homogenous in hardware and software
  - Typically housed in the same building/geography, interconnected using a dedicated network, have shared resources
  - Often viewed as a single computer
  - e.g., Supercomputers
- Grid computers – very loosely coupled - distributed job management & scheduling
  - Have each node set to perform a different task to reach a common goal
  - Typically heterogenous and in hardware and autonomous in software
  - Geographically dispersed, diverse computers connected via internet
  - Sometimes just uses free cycles
  - e.g., seti@home, crypto mining pools

Tightly Coupled Multiprocessor Systems

- Tightly coupled (*parallel*) multiprocessor systems – usually mounted on the same mother board or within the same silicon die, communicates via shared memory and (usually) controlled by a single OS
  - Each processor executes different programs and works on different data
  - Each processor usually has one or more levels of private cache
  - They share many resources (higher level caches, memory, I/O device, interrupt system, etc.)
  - All processors connected using a system bus
- Multicore – multiprocessor system on a single die
  - Physical proximity allows the shared circuitry to operate at a much higher clock rate than what is possible if the signals have to travel off-chip

Tightly Coupled Multiprocessor Systems



Processors need not be of the same type

- Shared devices (including the bus) need arbitration mechanisms if two processors attempt to use the same resource simultaneously
- Mutual exclusion mechanisms necessary to protect resources (or a range of memory) which should not be used in a concurrent manner
- Need mechanisms to ensure cache coherence across all levels
  - Bus snooping – the cache controller spies on every transaction on the bus to see if the data inside the cache is affected. Alternative to bus snooping : directory-based mechanisms

Symmetric vs Heterogeneous Multiprocessing

- Symmetric multiprocessing (SMP) systems are systems with two or more identical cores
  - Ex : Intel Core i3, i5 etc.
- Heterogeneous multiprocessing (HMP) refers to systems that use more than one kind of processor or cores
  - Uses dissimilar processors or coprocessors - cores could be of totally different ISAs or different microarchitectures but same ISA
  - Usually incorporating specialized processing capabilities to handle particular tasks
  - Some coprocessors might not be ISA-based
  - Coprocessors are also called hardware accelerators
- Most modern systems have some level of heterogeneity
  - GPUs, Video codecs, cryptography coprocessors, programmable network processors, Neural/AI accelerators, DSPs, etc.
    - Most of these 'accelerators' perform SIMD-style processing

SMP Advantages and Disadvantages

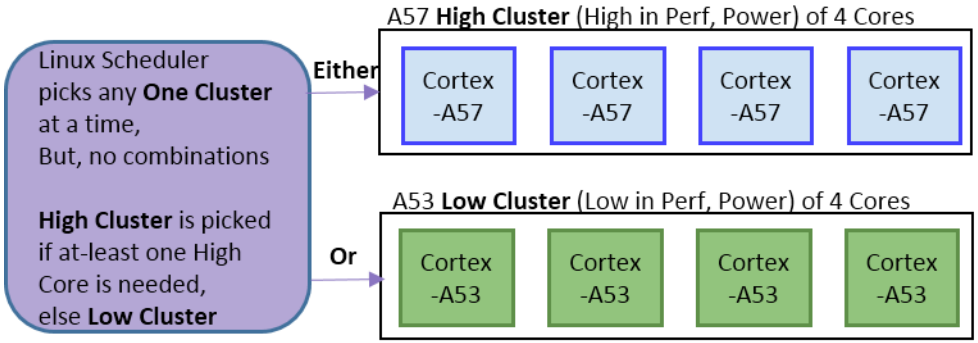
- Binary compatibility – the same binary can be used
- Simplicity of system design, task scheduling and performance evaluation
- Fault tolerance – if a particular core is known to be damaged, the system can still function at a reduced performance rather than failing altogether (also known as graceful degradation)
- Higher power, cost for the same performance compared to HMP
  - Power issues mitigated to some extent by scaling down the voltage and/or frequency of cores based on load - Dynamic voltage and frequency scaling (DVFS)
  - In modern computers, some cores can be turned off altogether if the computational demand of the application(s) is low

HMP Advantages and Disadvantages

- Higher performance for a given cost
- Lower power consumption for a given performance
- Poses difficulty in design, debugging, performance evaluation, task scheduling
  - Open Computing Language (OpenCL) : an attempt to minimize this difficulty
    - A framework for writing programs that execute across heterogeneous platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs) and other processors or hardware accelerators
    - Increasingly supported by many hardware vendors
    - Do not confuse with OpenGL, which is an API for graphics rendering
- Need different binaries for different cores (if ISAs are diff.)
- Limited / no fault-tolerance
- Data format / endianness issues

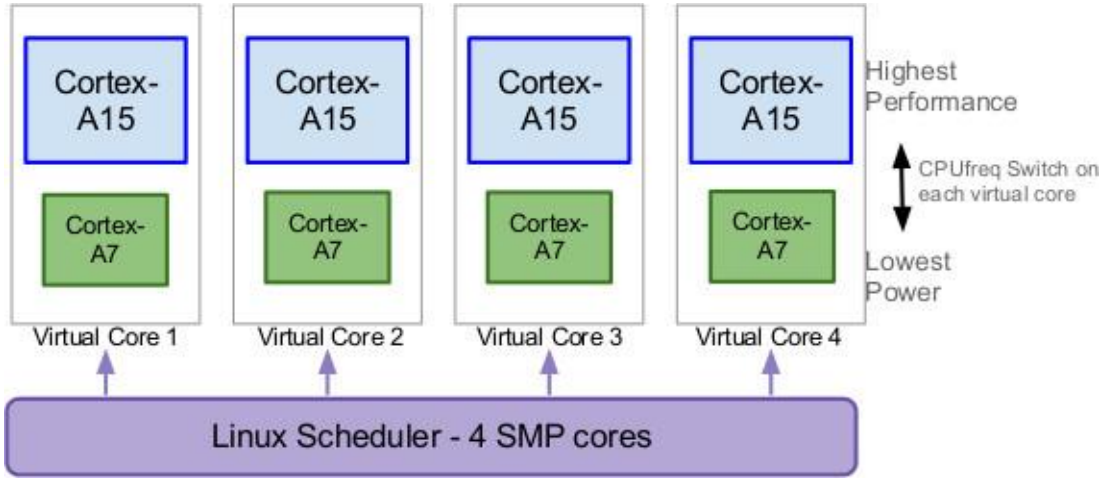
HMP : ARM big.LITTLE

- ARM big.LITTLE is a heterogeneous computing architecture coupling relatively battery-saving and slower processor cores (LITTLE) with relatively more powerful and power-hungry ones (big)
  - A multi-core processor that can adjust better to dynamic computing needs. ARM claims up to 75% savings in power
  - 3 Variants
- Clustered switching
  - Simplest implementation, arranging the processor into identically-sized clusters of 'big' or 'LITTLE' cores
  - Ex : Exynos 5 Octa 5410 (Samsung Galaxy S4), Apple A10 (iPhone 7)



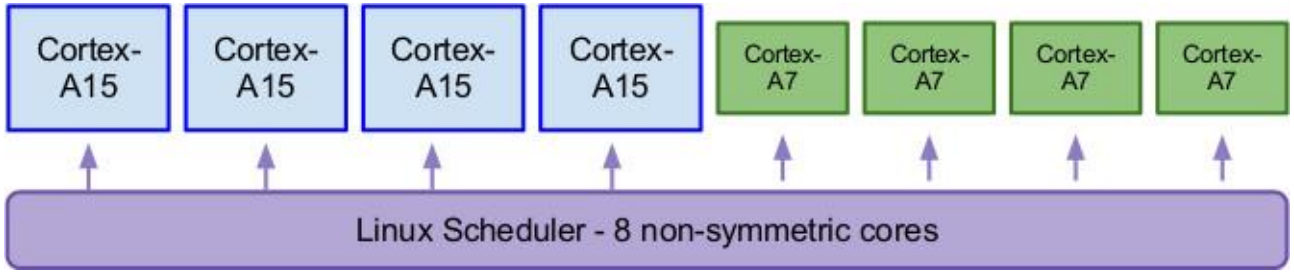
HMP : ARM big.LITTLE

- In-kernel switcher (CPU migration)
  - CPU migration via the in-kernel switcher (IKS) involves pairing up a 'big' core with a 'LITTLE' core
  - Each pair operates as one virtual core, and only one real core is (fully) powered up and running at a time
  - Ex : Nvidia Tegra 3 SoC (Microsoft Surface [non-Pro], LG Optimus 4x HD,..)



HMP : ARM big.LITTLE

- Global task scheduling
  - The most powerful use model - enables the use of all physical cores at the same time
  - Threads with high priority or computational intensity allocated to the 'big' cores while threads with lower priority or less computational intensity, such as background tasks, can be performed by the 'LITTLE' cores
  - Ex : Exynos 5 Octa 5422 (Samsung Galaxy S5), Apple A12 (iPhone XS – 2 big and 4 LITTLE cores)



HMP : Apple A14 SoC

- Apple A14 launched recently and used in iPhone 12 has
  - 11.8 billion transistors, based on 5 nm fabrication process (TSMC) - first commercially available product based on 5 nm tech
  - Two high-performance cores called Firestorm and four energy-efficient cores called Icestorm running ARMv8 ISA
  - 4-core GPU
  - 16-core Neural Engine capable of performing 11 trillion operations per second
  - Machine learning matrix multiplication accelerators (AMX blocks)
  - Image processor for computational photography
  - A motion co-processor (M14) which reads accelerometer, gyroscope, compass, and barometer even if the device is asleep, and applications can retrieve data when the device is woken up
  - Video encoders/decoders, ….

Dealing with Exceptions

- Exceptions (aka interrupts) are just another form of control hazard. Exceptions arise from
  - DP arithmetic errors
  - Trying to execute an undefined instruction
  - An I/O device request
  - An OS service request (e.g., a page fault, TLB exception)
  - A hardware malfunction
- The pipeline has to stop executing the offending instruction in midstream, let all prior instructions complete, flush all following instructions, (sometimes) set a register to show the cause of the exception, save the address of the offending instruction, and then jump to a prearranged address (the address of the exception handler)
- The software (OS) looks at the cause of the exception and "deals" with it

Two Types of Exceptions

- Interrupts – asynchronous to program execution
  - caused by external events
  - may be handled between instructions, so can let the instructions currently active in the pipeline *complete* before passing control to the OS interrupt handler
  - simply suspend and resume user program

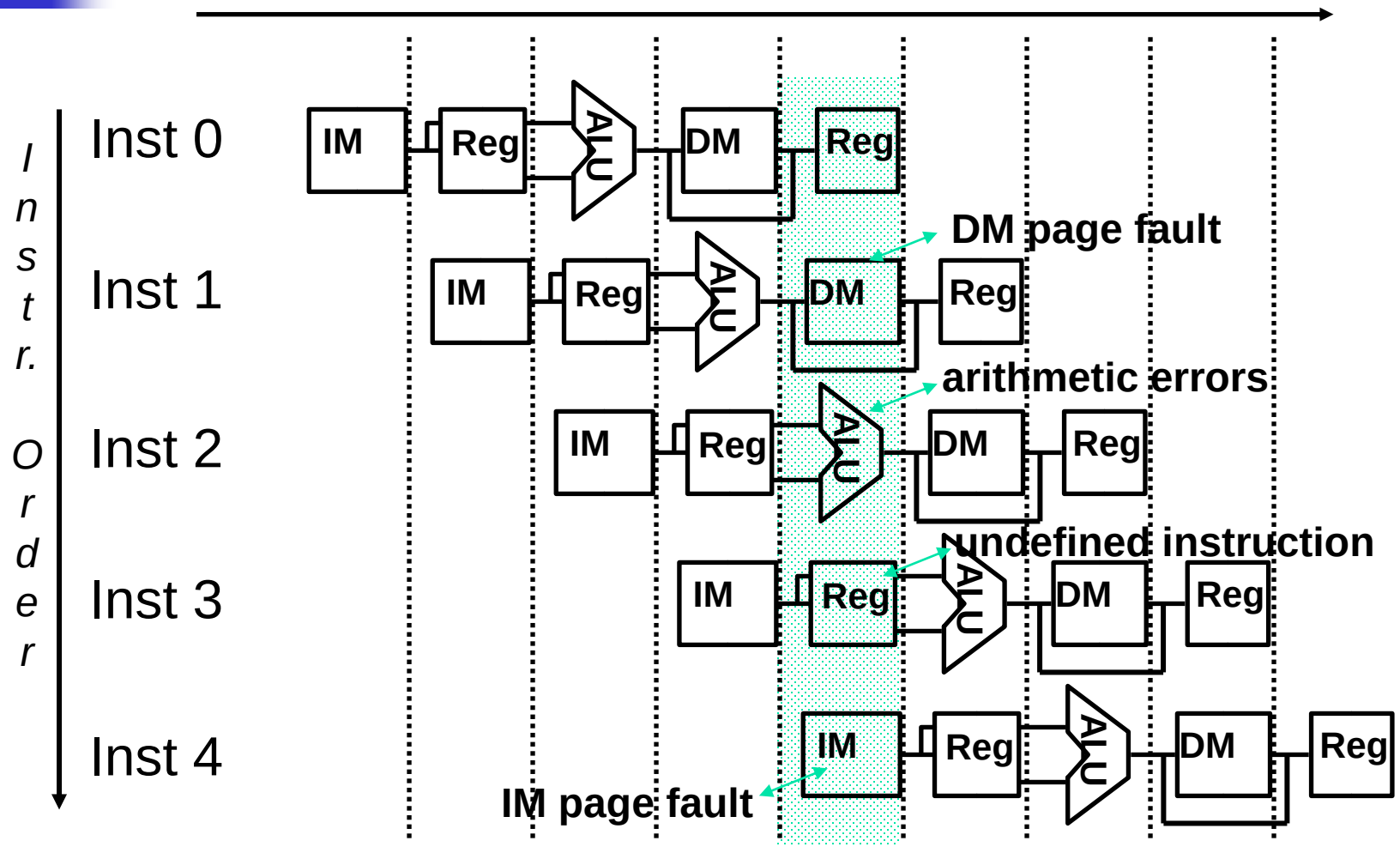
- Traps (Exceptions) – synchronous to program execution
  - caused by internal events
  - condition must be remedied by the trap handler for that instruction, so must stop the offending instruction *midstream* in the pipeline and pass control to the OS trap handler
  - the offending instruction may be retried (or simulated by the OS) and the program may continue or it may be aborted

Where in the Pipeline Exceptions Occur

	Stage(s)?	Synchronous?
Arithmetic error	E	yes
Undefined instruction	D	yes
TLB or page fault	F, M	yes
I/O service request	any	no
Hardware malfunction	any	no

- Beware that multiple exceptions can occur simultaneously in a single clock cycle

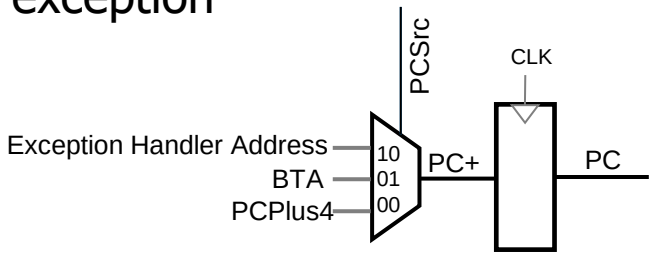
Multiple Simultaneous Exceptions



- Hardware sorts the exceptions so that the earliest instruction is the one interrupted first

Exception Handling Hardware

- Need a mechanism to load the PC with the address of the exception handler. A simple way is to expand the PC input mux where the new input is
  - A fixed exception handler address (non-vectored interrupt)
    - Simpler hardware, but more complicated handler software as it needs to find the cause and deal with it appropriately
    - In some processors (e.g., MIPS), there are registers which record the 'Cause' of exception



Control unit asserts PCSrc = 10 when an interrupt / exception occurs

  - Obtained from a table which has the handler addressees for various interrupts / exceptions stored (vectored interrupt)
    - More complicated hardware, but potentially simpler interrupt handler software, faster interrupt handling

Exception Handling Hardware

- Need to flush offending instruction and the ones that follow it in case of exceptions (flush not needed for interrupts)
- Some processors automatically save some registers to the stack (ARM Cortex M3 pushes xPSR, PC, R0-R3, R12, and LR)
- Save the return address (PC+4) to a register (e.g., LR)
- In some processors, there are separate (banked) copies of some registers for use by the handler, which reduces the amount of data to be saved / restored and eliminates the possibility of corruption / data loss
- Interrupt processing can be complicated if there are many interrupt sources. Many systems include an interrupt controller - a separate unit closely coupled to the processor which contains control registers for enabling / disabling interrupts, setting priority of interrupts etc.

Exception Handler (Software) Steps

- Save additional status / registers etc. if needed
- Check which peripheral / condition within the peripheral caused the interrupt
  - Done by reading interrupt status registers within the peripheral
- Do the BARE MINIMUM required to deal with the interrupt
- Clear the interrupt – tell the interrupting peripheral to de-assert the interrupt request
  - The way it is done is peripheral specific and might also depend on the condition within the peripheral which caused the interrupt
- To do further processing, set a Boolean flag for the main program / wakeup a thread (bottom half of the handler)
- Restore additional status / registers etc if any
- Exit the Handler

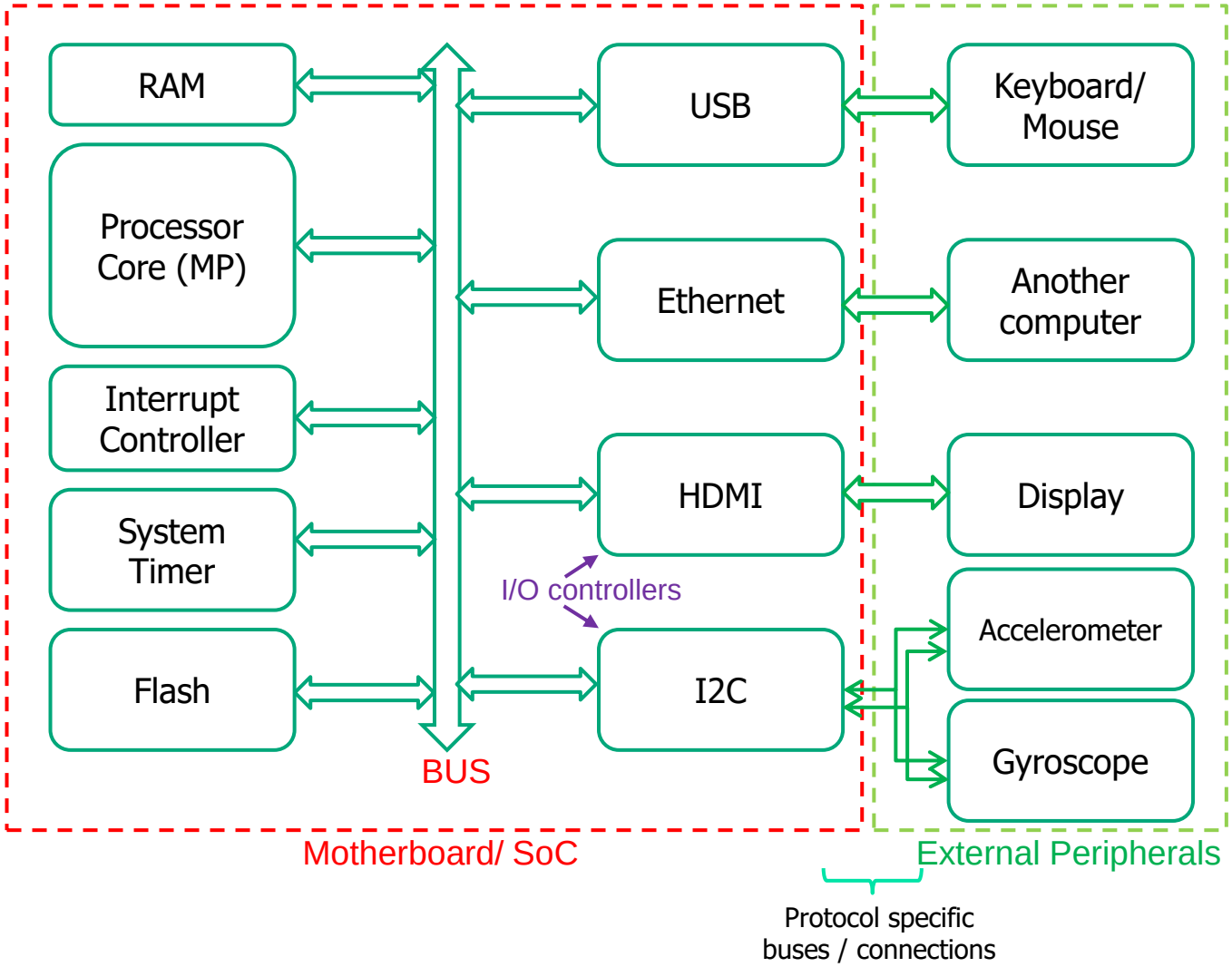
Processor Modes (ARM Cortex M4)

- Thread mode
  - Used to execute application software. The processor enters Thread mode when it comes out of reset
  - Can be privileged or unprivileged
- Handler mode
  - Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing
  - Always privileged

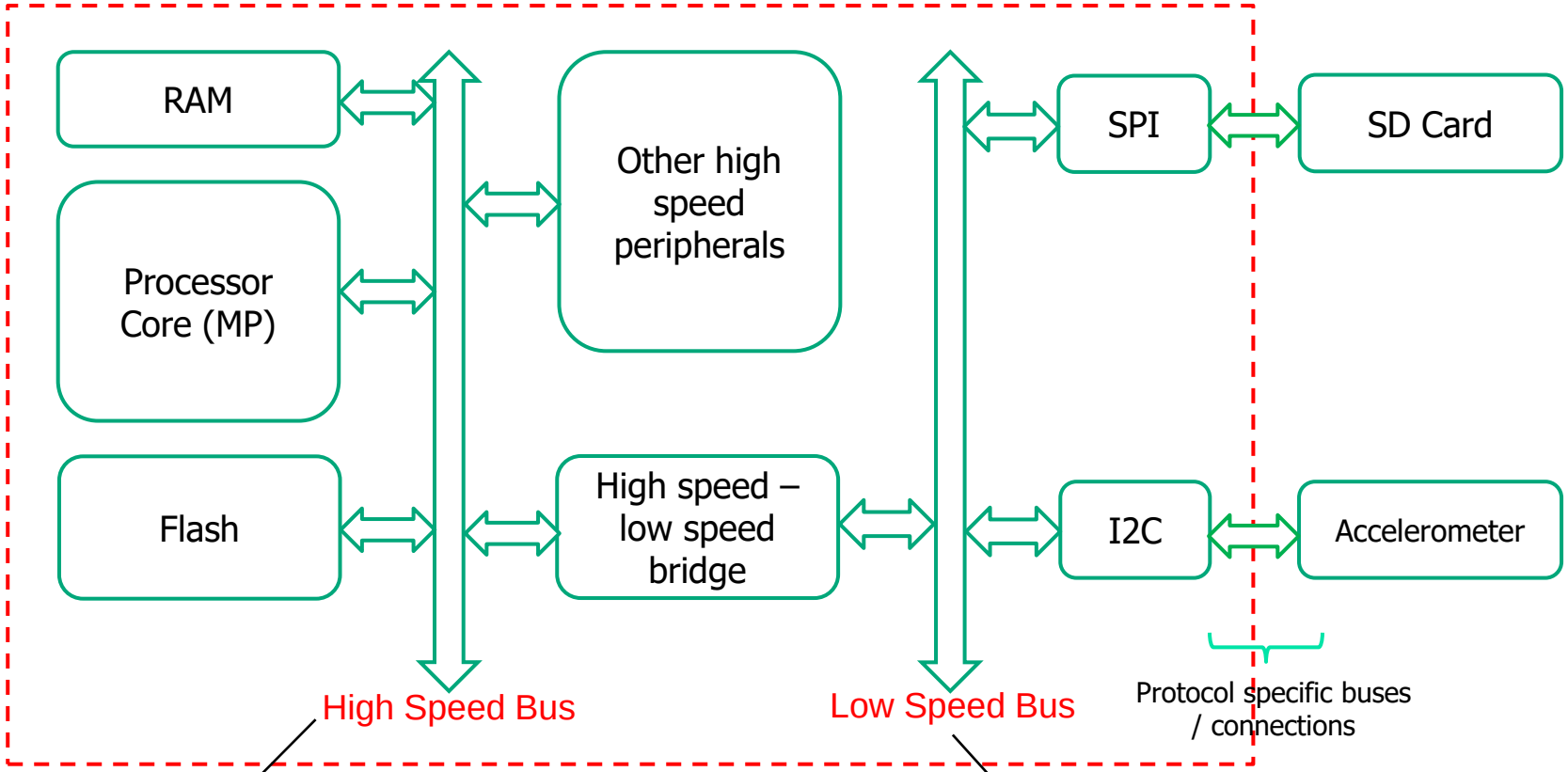
Processor Modes (ARM Cortex M4)

- Unprivileged (User mode)
  - Software has limited access certain instructions, and cannot use the CPS (change processor state) instruction
  - Cannot access the system timer, interrupt controller, and other system control registers
  - Might have restricted access to memory or peripherals
- Privileged (Kernel mode / Supervisor mode)
  - The software can use all the instructions and has access to all resources
  - Only privileged software can change the privilege level for software execution in Thread mode by writing to a CONTROL register
  - Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software

Generic Computer System



Multi-tiered Bus Architecture



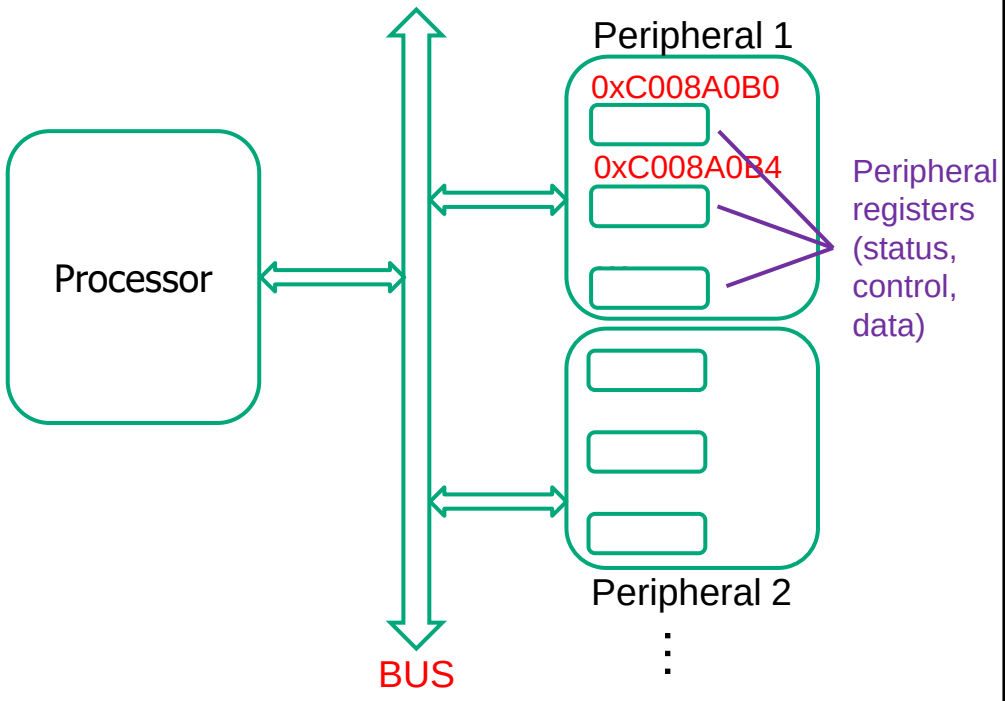
Usually proprietary such as QuickPath Interconnect (Intel), Hypertransport (AMD), AHB* (ARM)
*AHB is an open standard

Usually industry standard such as PCIe, APB/AXI

Peripheral Registers

- Software interacts with the hardware by reading / writing peripheral registers
- Peripheral registers are accessed like memory – using LDR/STR or pointers for peripherals within the address space of the processor

- Status registers - *read* by the processor to know the status of the hardware
- Control registers - *written* by the processor to give commands / control signals
- Data registers - *read* (for input) and/or *written* (for output) to perform actual transfer of data



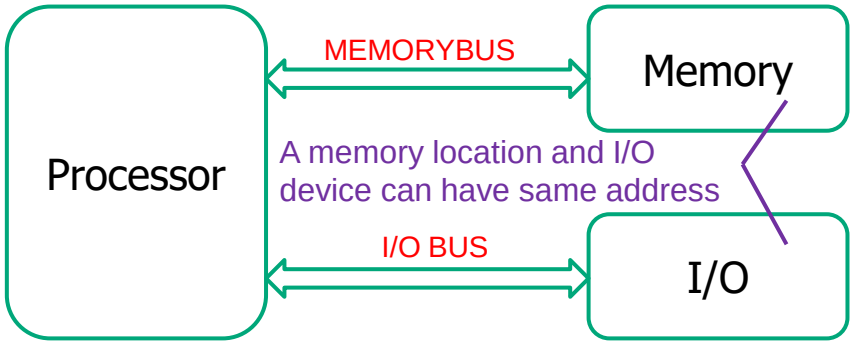
C language, libraries / API / drivers

- Libraries (collection of driver functions for a peripheral / API) encapsulate the process of reading/writing peripheral registers
- Internally use pointers to peripheral registers (i.e., holding peripheral register addresses) to accomplish reading/writing
- Allow ease of programming (higher layer of abstraction – no need to reinvent the wheel)
- Portability – we just need to recompile / re-link with the new library, user program will need minimal/no changes

Port-mapped I/O

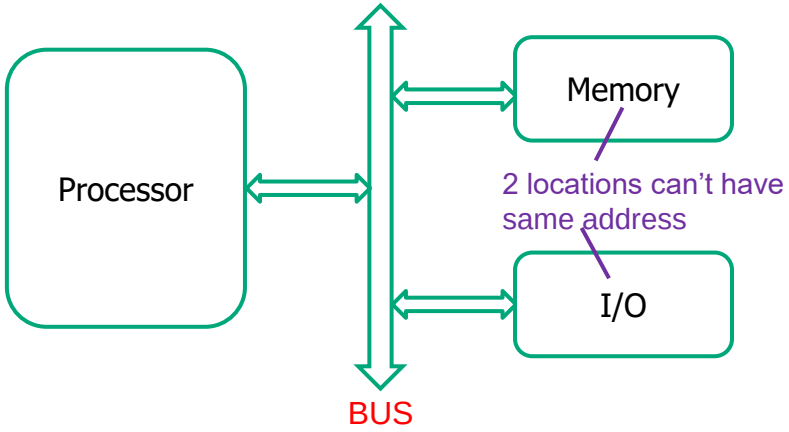
How can the processor address an I/O device/register?

- Port-mapped I/O (PMIO, also called isolated I/O)
  - Dedicated instructions for I/O
  - Ex : Intel x86 instruction set has dedicated IN and OUT instructions, separate from memory access instructions
  - The processor has an output pin specifying whether the address is meant for memory or peripheral
  - Separate address space for memory and peripherals - memory can use the full range of addresses
  - Simpler software
  - Not compliant with RISC philosophy
  - Extra pins



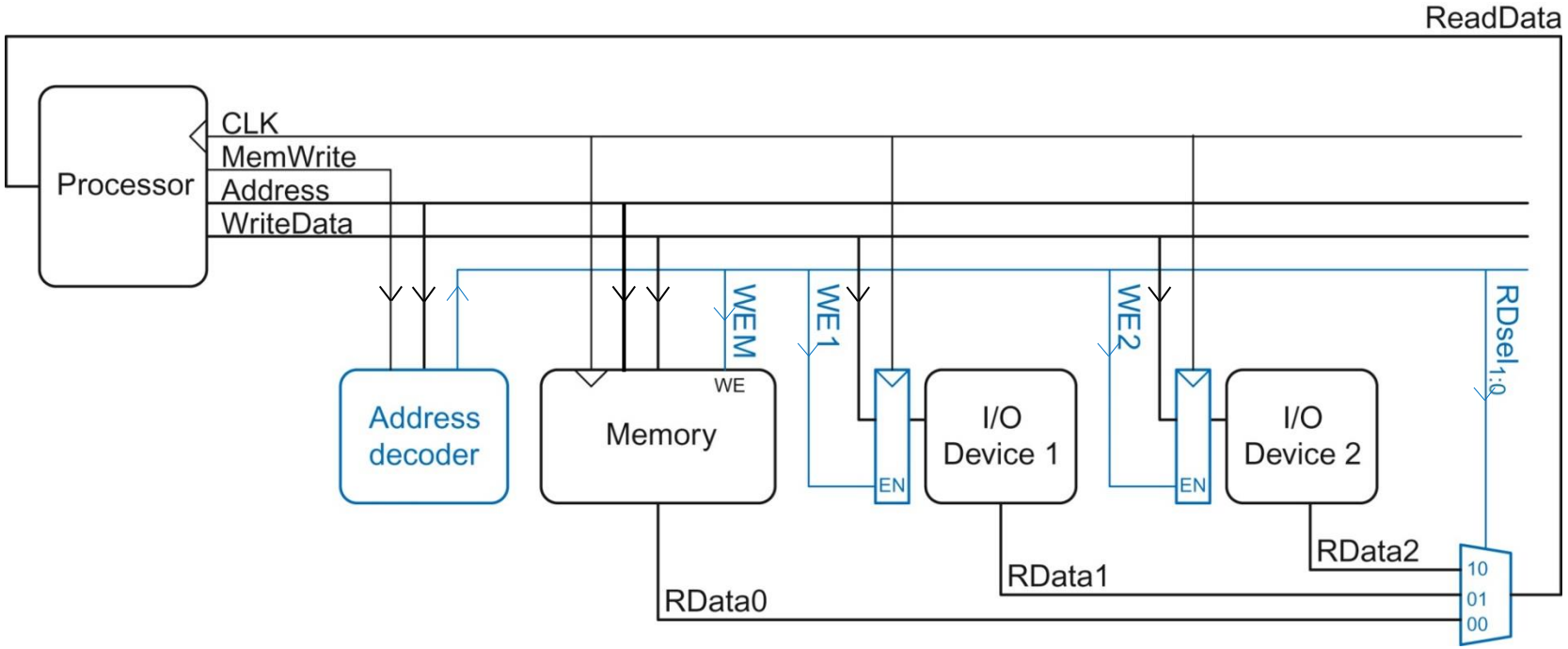
Memory-mapped I/O

- Memory-mapped I/O (MMIO)
  - Same LDR and STR instructions used for I/O, just like memory
  - Address space is shared between memory and I/O
  - An address decoder determines whether memory or I/O, and which I/O device
  - Processor design is simpler
  - Maximum possible memory is reduced, as I/O devices eat into the address space – not a problem with 64-bit systems, but an issue for 32-bit systems
  - Software is more complicated
    - An address range corresponding to an I/O device should not be cached
    - Else, the software needs to do cache invalidation before reading, and needs to flush the (write-back) cache immediately after writing
    - "volatile" keyword is necessary; may not be sufficient (read up about 'fence' instructions)



MMIO : Hardware

- This is the mechanism we used in our labs
  - Go through the Wrapper.v/vhd file!



Programmed I/O (PIO)

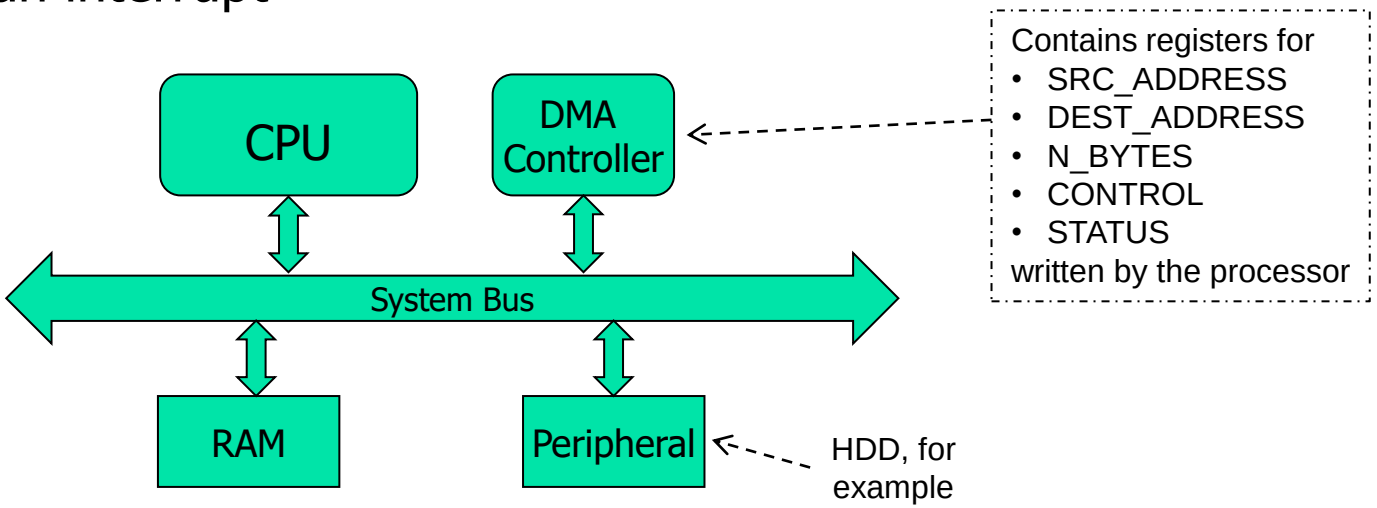
How exactly does the data transfer take place?

- Programmed I/O (PIO)
  - The processor reads the peripheral register and places the data into a location in the memory (or vice versa)
  - Done using a combination of LDR-STR instructions
  - Access initiated through polling or triggered by an interrupt
  - Problem : Processor time is wasted for bulk transfers
  - Solution : Direct Memory Access

- DMA
  - A separate controller is present on the bus, capable of generating addresses, bus signals (R/W etc.) and transferring data
  - CPU's job is limited to initiating the data transfer

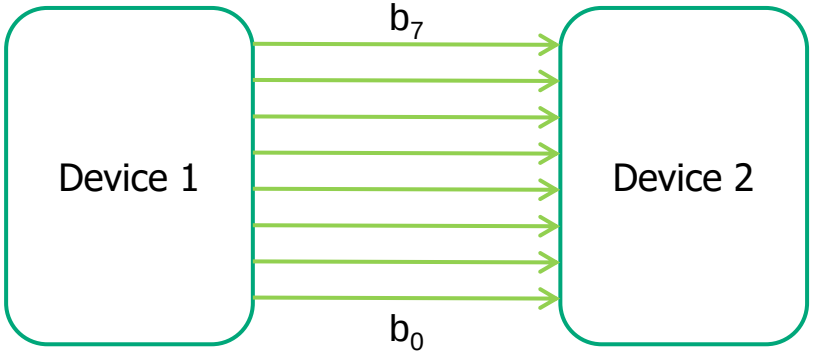
Direct Memory Access (DMA)

- CPU (OS) provides the DMA controller the source address, destination address and the number of bytes/words to be transferred
- CPU then assigns DMA as the bus master
- DMA controller generates the addresses sequentially and performs the transfer
- In the meantime, the CPU can perform computations which do not need memory access (using the data in cache and registers)
- The DMA controller informs the CPU once the transfer is over by raising an interrupt

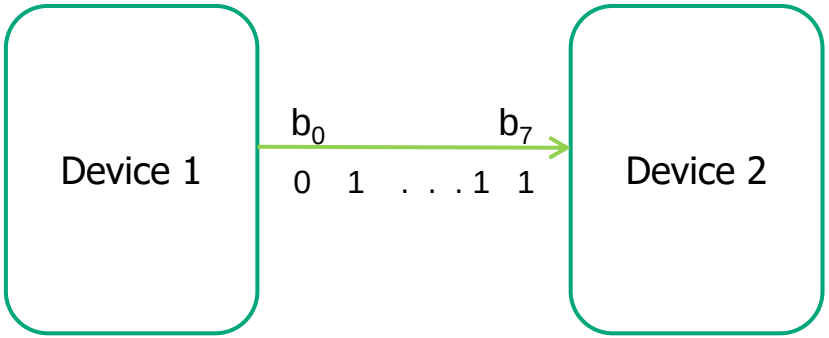


Parallel vs Serial

- Parallel protocols : use a number of parallel wires to transmit bits in parallel
  - Processor-to-memory busses, old (pre-2005) protocols such as PCI, IDE, printer port
- Serial protocols : data is sent through a single wire, one bit at a time
  - Serial ATA (SATA), PCIe, I^2C, SPI, UART



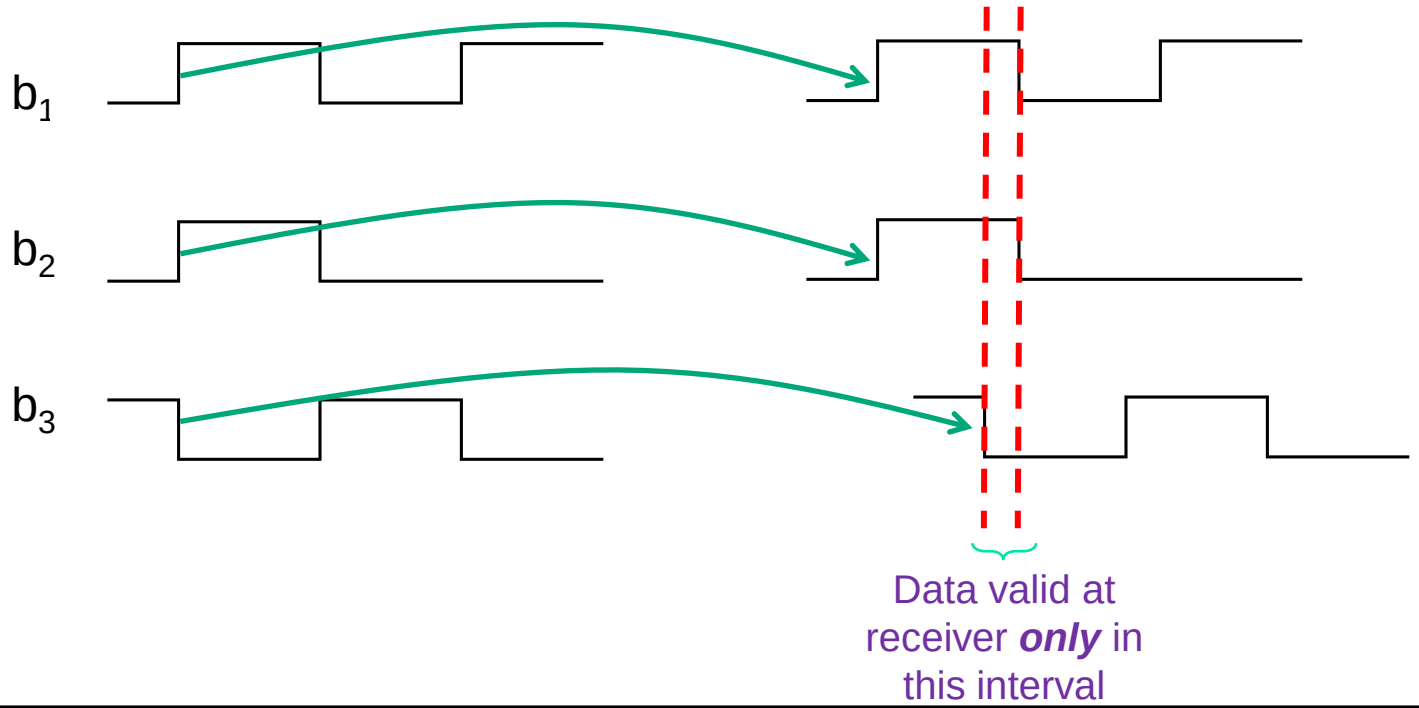
Need not always be 8-bit



MSB First (Need not always be the case)

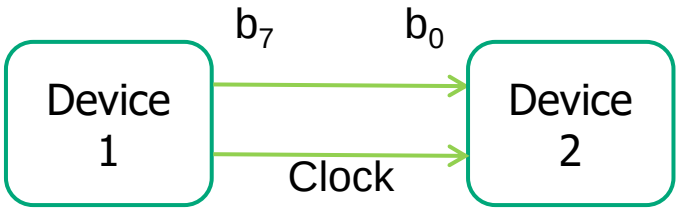
Parallel Buses : Bus Skew

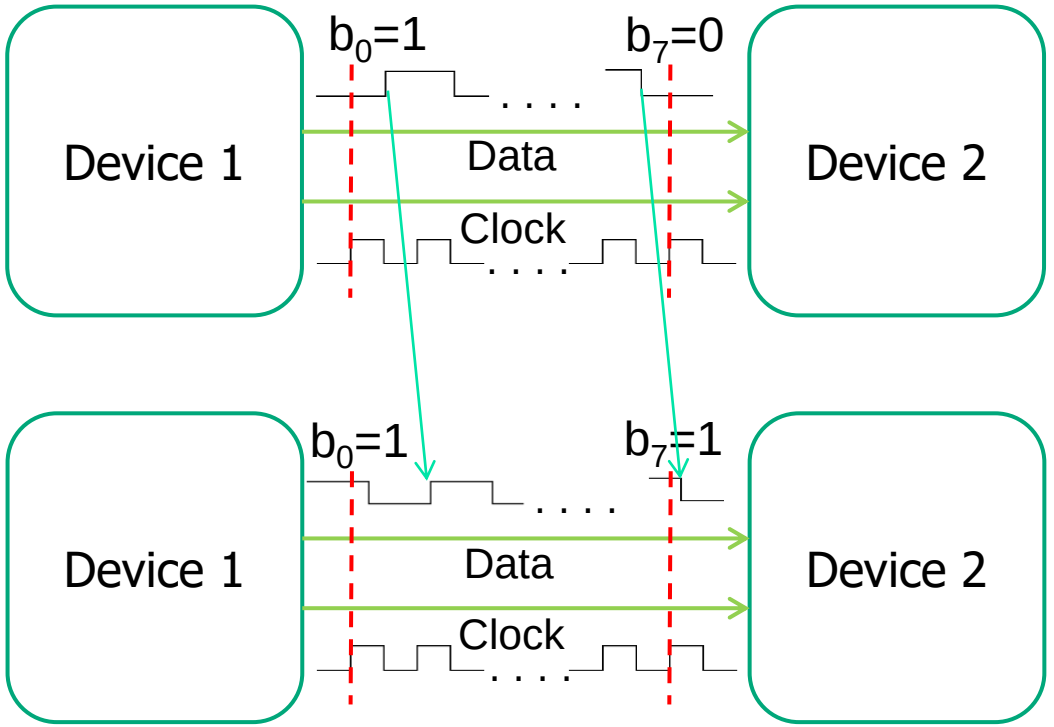
- Parallel : used in high-speed, low distance
- Suffers from bus-skew : different signals travel at diff. speeds
- Suffers from cross-talk : signals from diff. wires interfere with each other



Synchronous vs Asynchronous

- Synchronous : one of the wires carry clock - allows transmitter and receiver to have a common time reference
- Faster, but suffers from the same issue as the parallel protocols (cross-talk, skew etc.)
  - SPI, I^2C

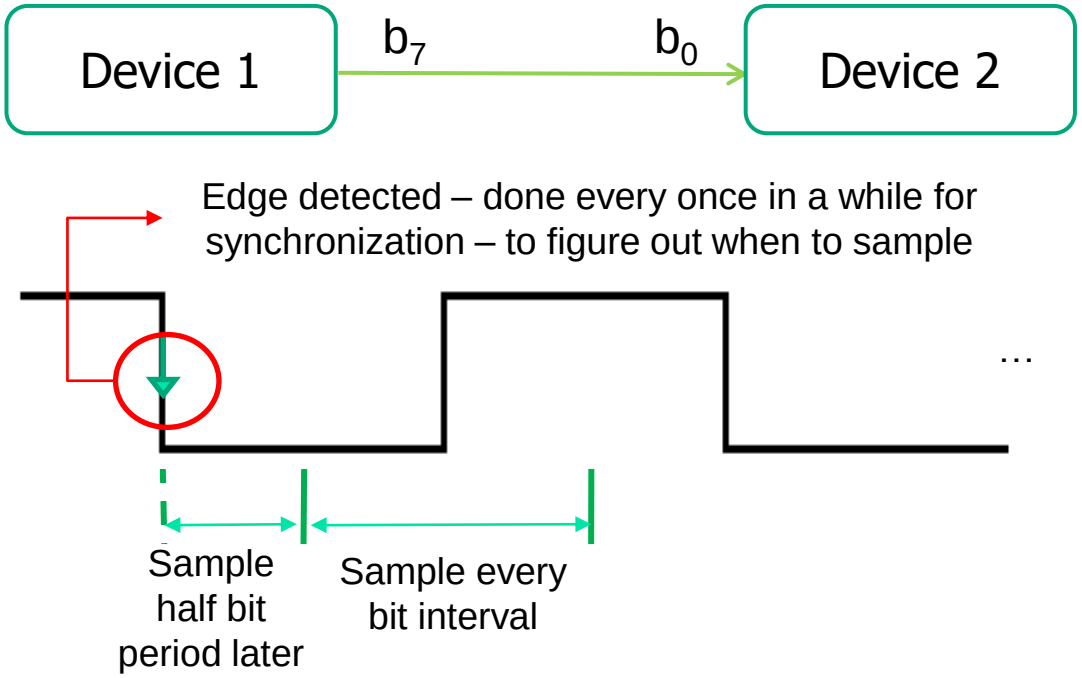




Note: Clock need not always be generated by the transmitting device

Synchronous vs Asynchronous

- Asynchronous : only data is transmitted, not clock
- Receiver needs to recover timing info from data – slower, complicated hardware
  - UART, USB*

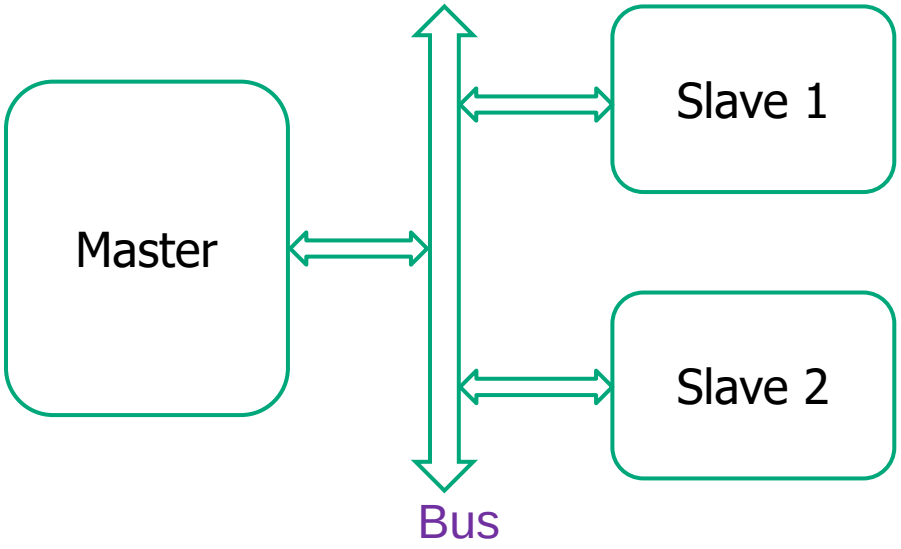


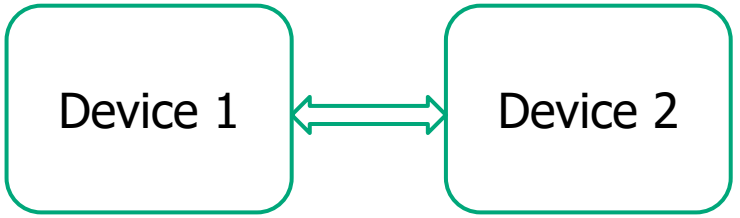
A handshaking mechanism with additional control lines may also be used instead of edge detection based scheme. Here, the transmitting party will assert a 'ready' signal and keeps the data unchanged until it receives an 'acknowledgement' from the receiving party.

*perspective dependent. While the transmitter and the receiver do not share a clock explicitly, the clocks are synchronized through sync fields and remain in sync through appropriate coding of data

Bus-based vs Point-to-point

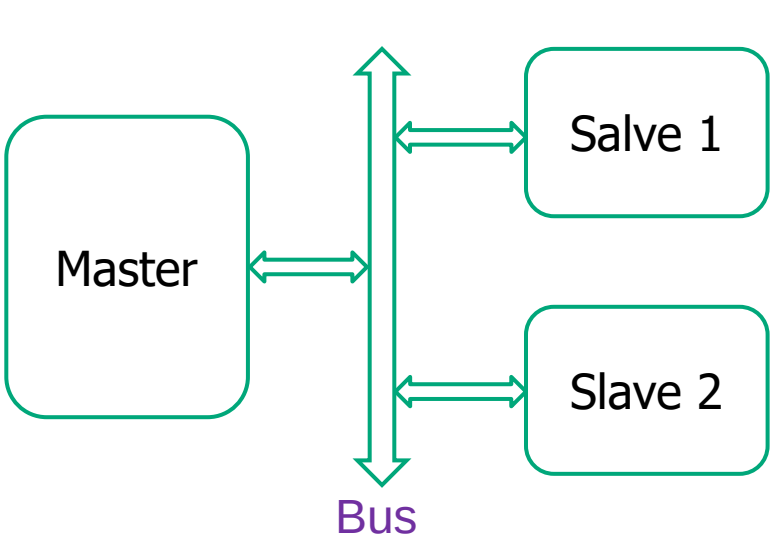
- Bus : a number of devices are connected to the same set of wires (called a bus) – each device has a unique address
  - I^2C, SPI, USB
- Point-to-point : two devices have a dedicated link between them – no addressing
  - UART

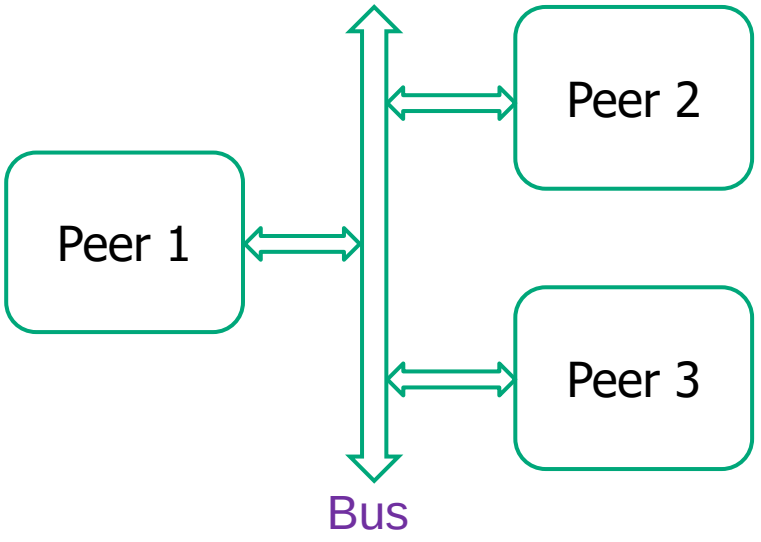




Master-Slave vs Peer-to-peer

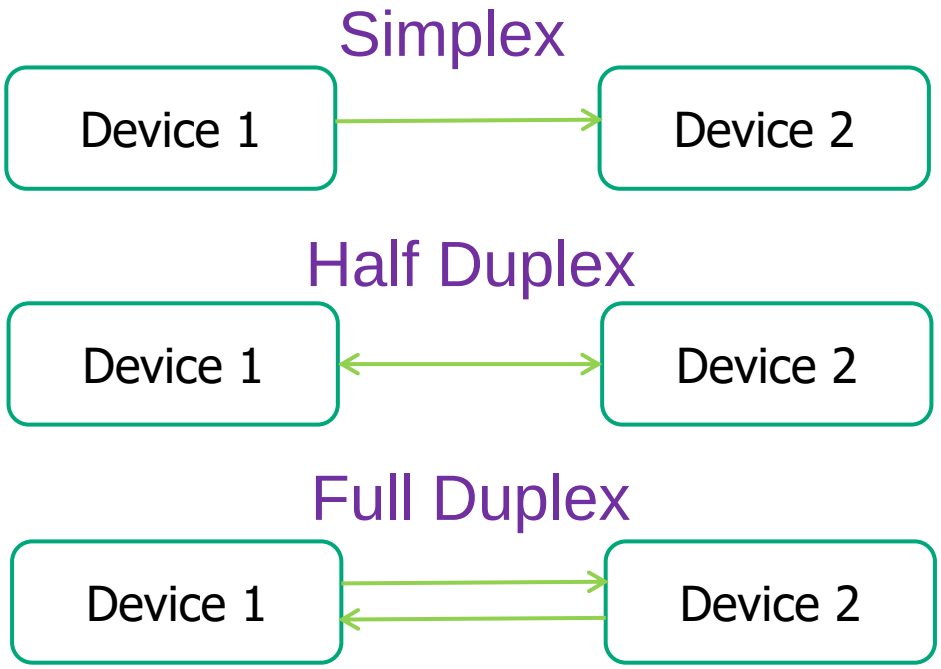
- Master-slave : only a master device can initiate communication, decides which slave should respond, and (in some synchronous protocols) generates the clock
  - I^2C, SPI, USB, AHB
- Peer-to-peer : any device can initiate communication
  - Ethernet (configuration dependent)





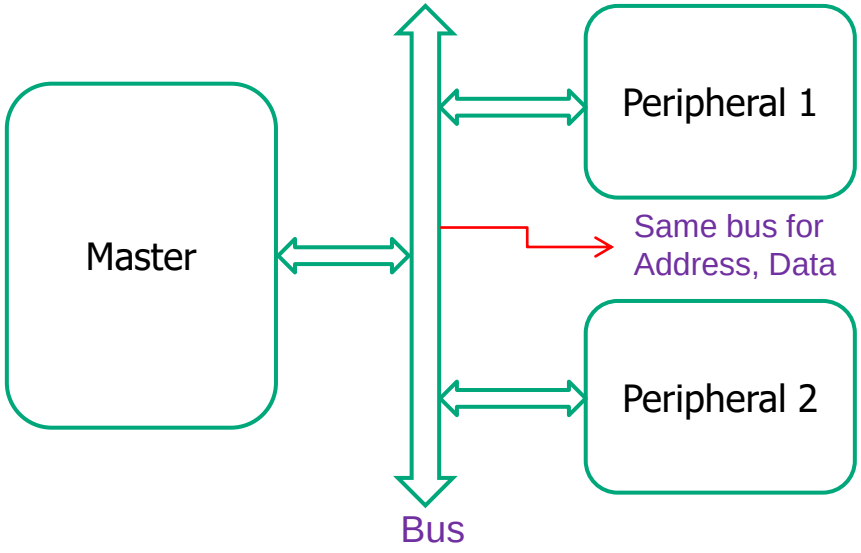
Simplex vs Half Duplex vs Full Duplex

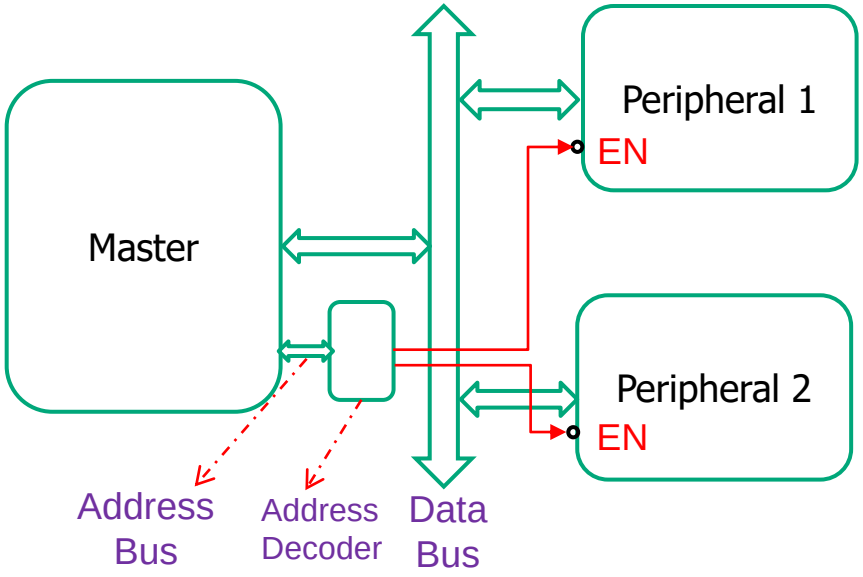
- Simplex : single, unidirectional link, one-way interaction
  - Connection between processor and LEDs / switches
- Half-duplex : one bi-directional link, devices take turns
  - I^2C, USB 2.0
- Full-duplex : at least two links, can transmit and receive at the same time
  - SPI, UART, USB 3.x



Addressing : In-band vs Out-of-band

- In-band : address of the device being accessed (to select the device), as well as the data (read from or written to the device) are sent through the same bus
  - I^2C, USB
- Out-of-band : separate address bus, a decoder enables (activates) the device
  - SPI





Concluding Remarks

- Computer Architecture is a very fast evolving field
- Go through various tech websites to be in sync with the state of the art
- "We cannot predict the future, but we can invent it" - Dennis Gabor

Signing off,

Rajesh