Lecture 6

Embedded System Architecture

University of Glasgow

UESTC College

Dr Vahid Nabaei

Topics

- Architecture and Hardware of Computer and Embedded System
- Central Processing Unit (CPU)
- Von Neumann and Harvard Architectures
- I/O Interfaces
- Memory-Mapped I/O and I/O-mapped I/O
- Two Types of Bus Cycles
- ARM Processor
- Polling and interrupt

Computer System

Computer Architecture

- **Computer architecture**
  - A description of a computer system specifying its parts and their relations

- **Instruction set architecture**
  - Storage cells (registers, memory)
  - The machine instruction set (set of possible operations)
  - The instruction format (size and meaning of fields within the instruction)

  - Most programming is done in a high-level language, with a compiler being used to convert that program into the binary code (machine code), matching the instruction set of the CPU

Computer Hardware

A Microprocessor

A Large Memory
(Primary and Secondary)
(RAM, ROM and caches)

Input Units
(Keyboard, Mouse, Scanner, etc.)

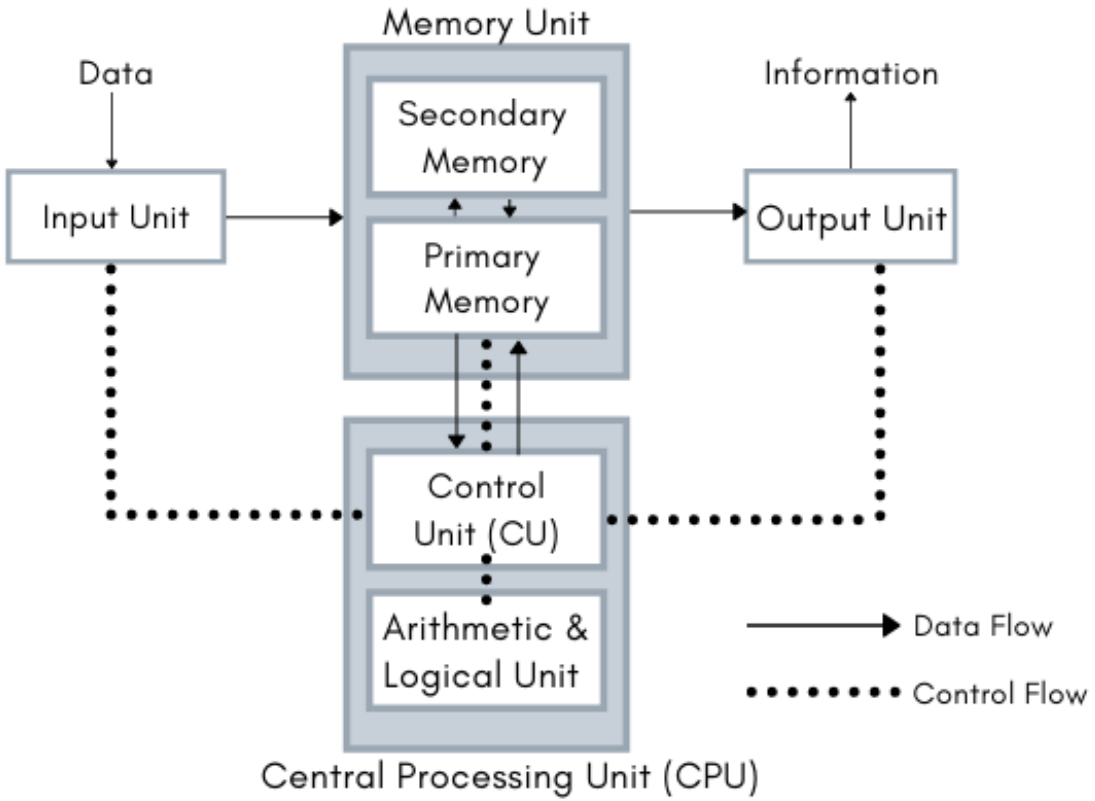
Output Units
(Monitor, printer, etc.)

Networking Units
(Ethernet Card, Drivers, etc.)

I/O Units
(Modem, Fax cum Modem, etc.)

Computer Components Relations

- The basic components of a computer
  - Processor
  - Memory
  - I/O

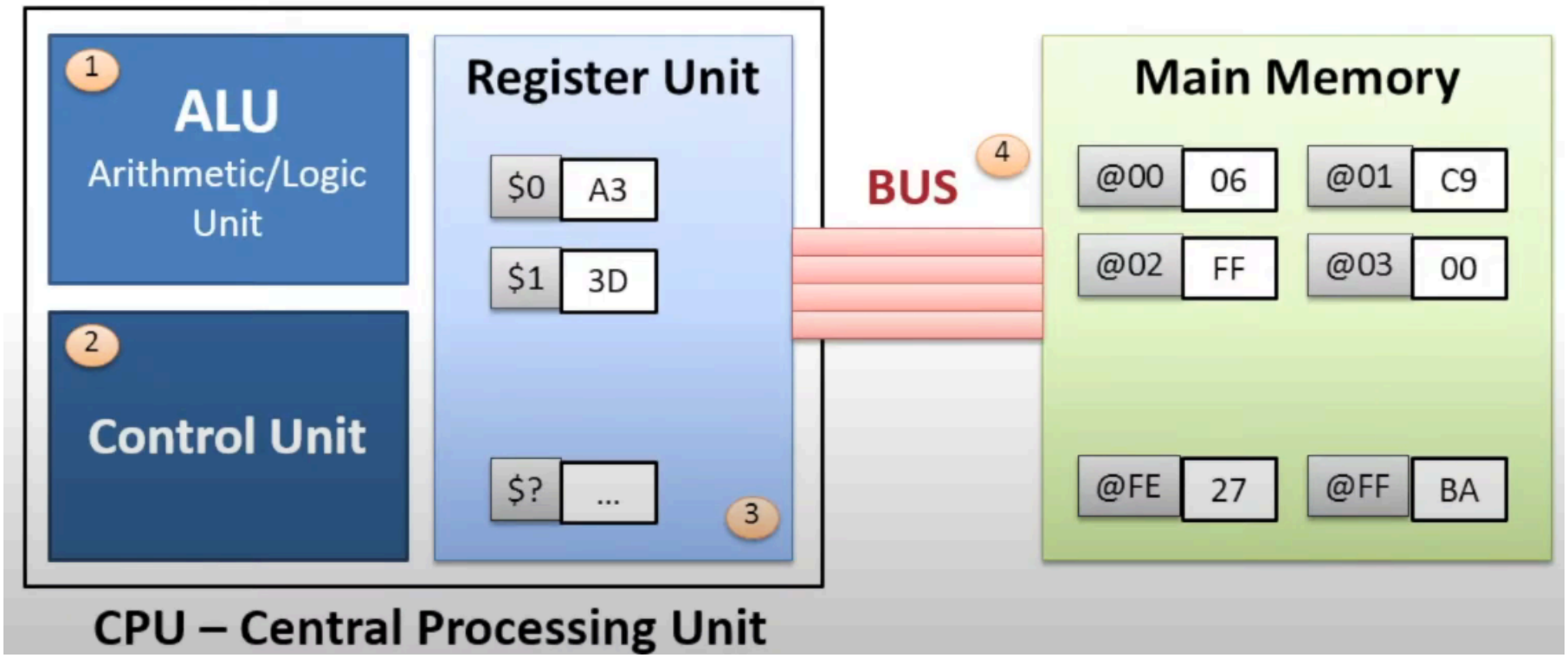


Data bus: One set of wires carries the data itself

Address bus: The other set of wires carries address information

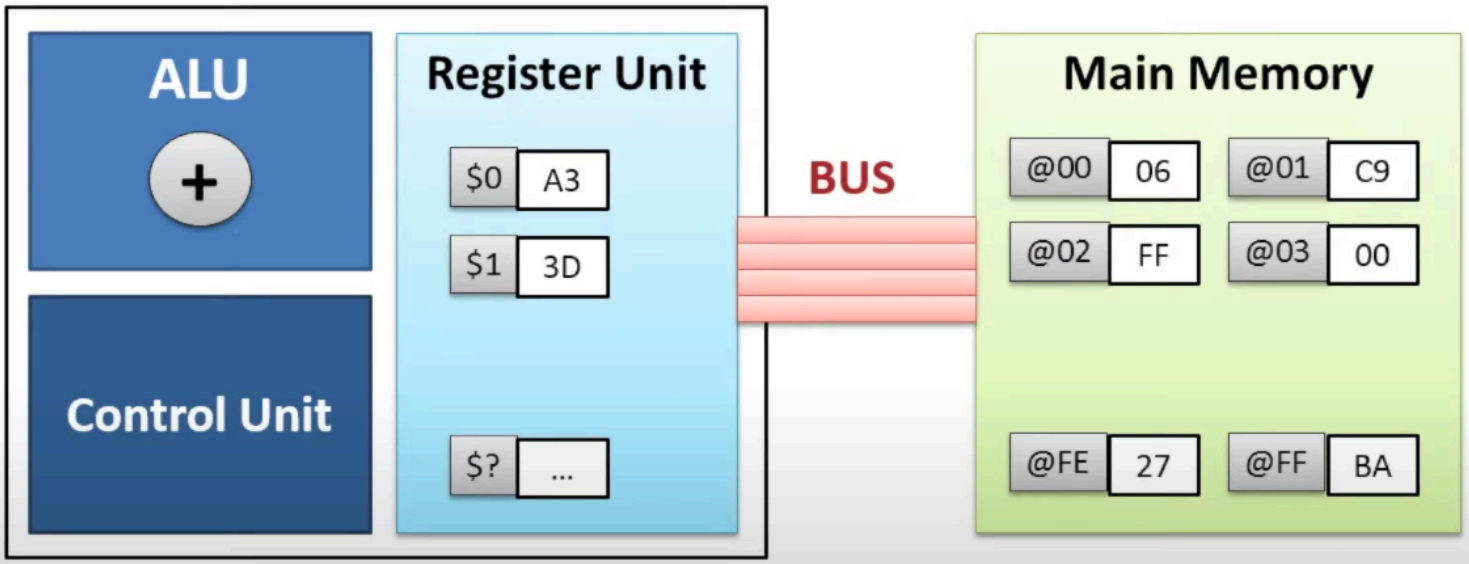
Address: is a digital number that indicates which place in memory the data should be stored in, or retrieved from.

Central Processing Unit (CPU)

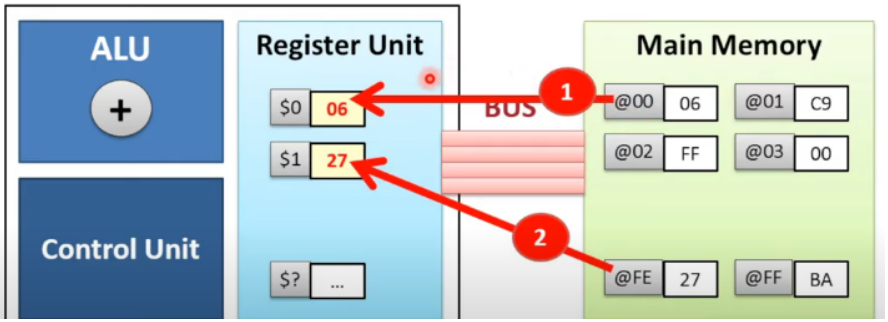


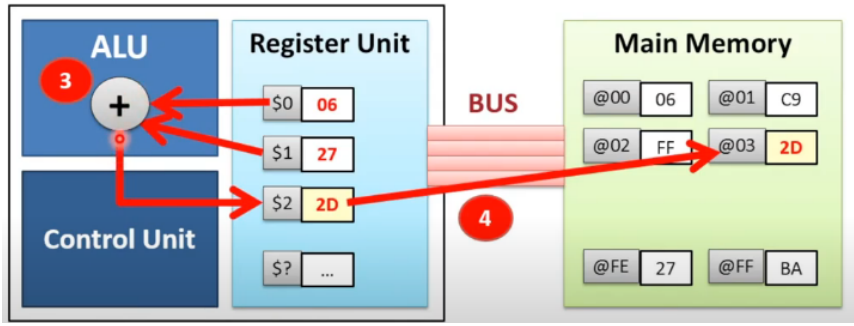
Example of adding two numbers

@00 + @FE → @03

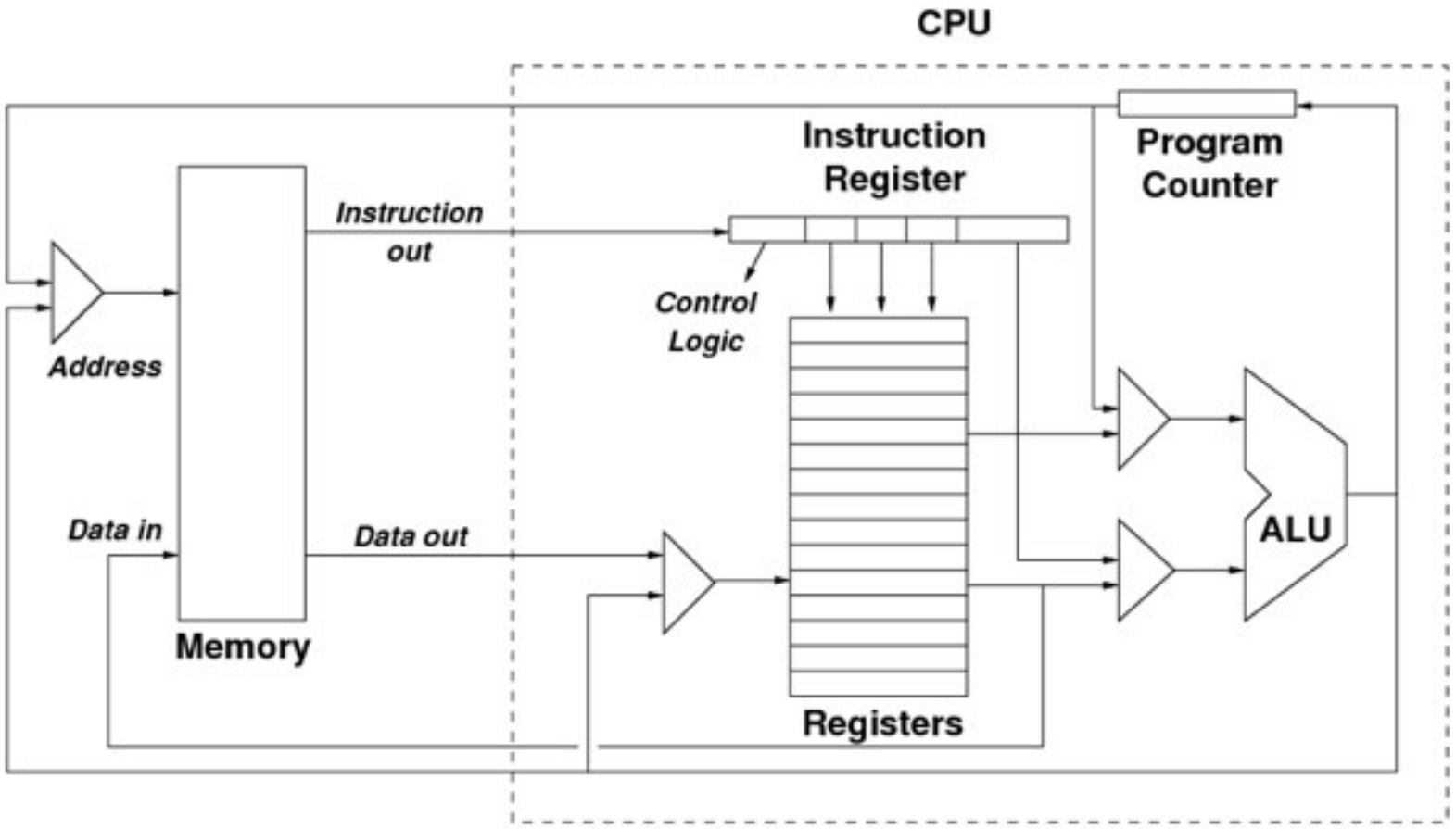


Step 1: Get one of the values to be added from memory and place it in a register
Step 2: Get the other value to be added from memory and place it in another register





CPU Components Interaction with Memory



Arithmetic logic unit: performs arithmetic and logic operations

- Addition, subtraction, multiplication and division
- Or, exclusive or, shift

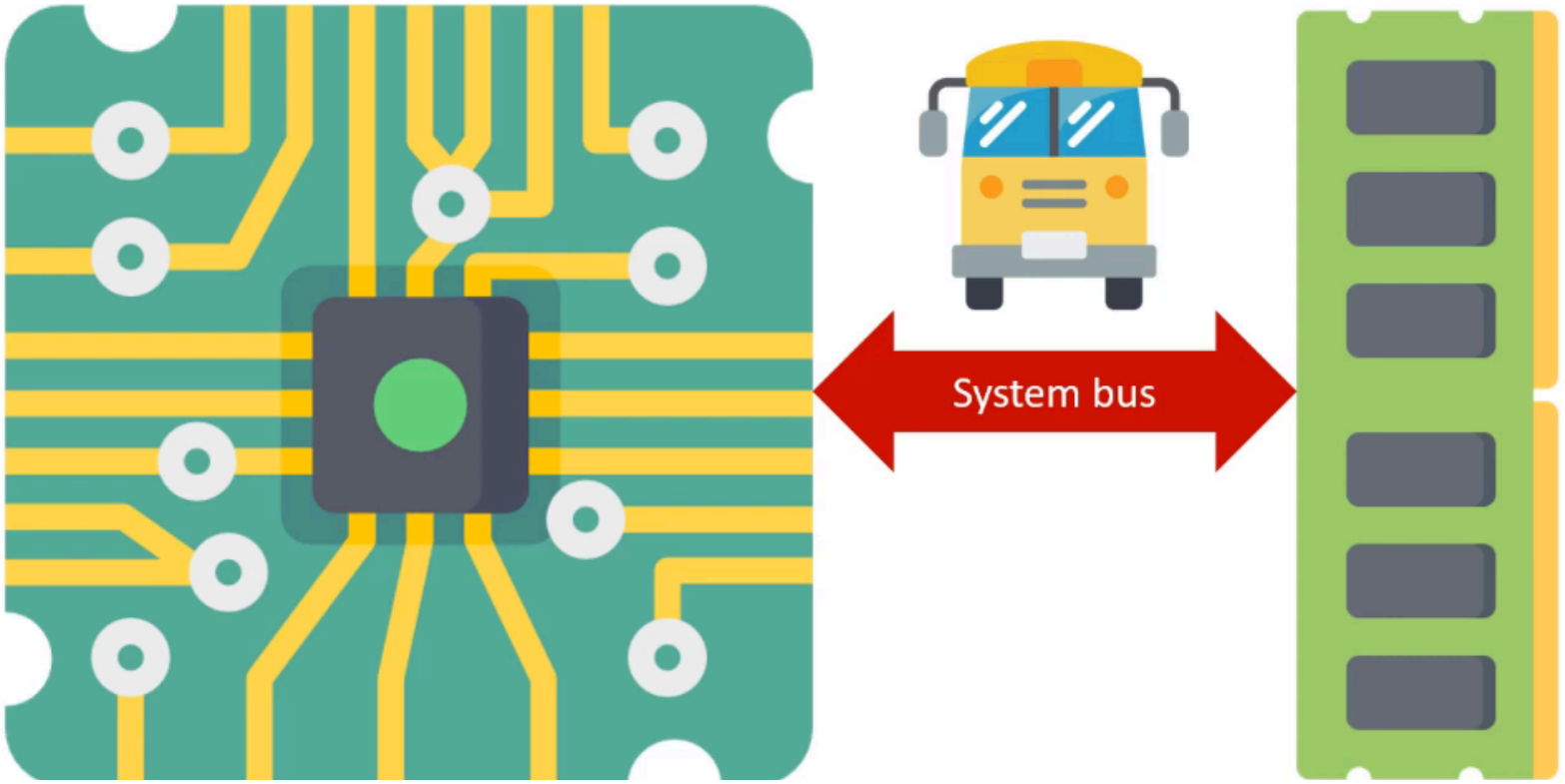
Registers: high speed storage located in the processor

- do not have addresses like regular memory
- Have specific functions explicitly defined by the instruction
- Can contain data or addresses

Von Neumann and Harvard Architecture

Von Neumann

- Shared memory space for instructions and data.
- Instructions and data are stored in the same format.
- A single control unit or processor follows a linear fetch, decode, execute cycle.
- One instruction at a time.
- Registers are used as fast access to instructions and data.

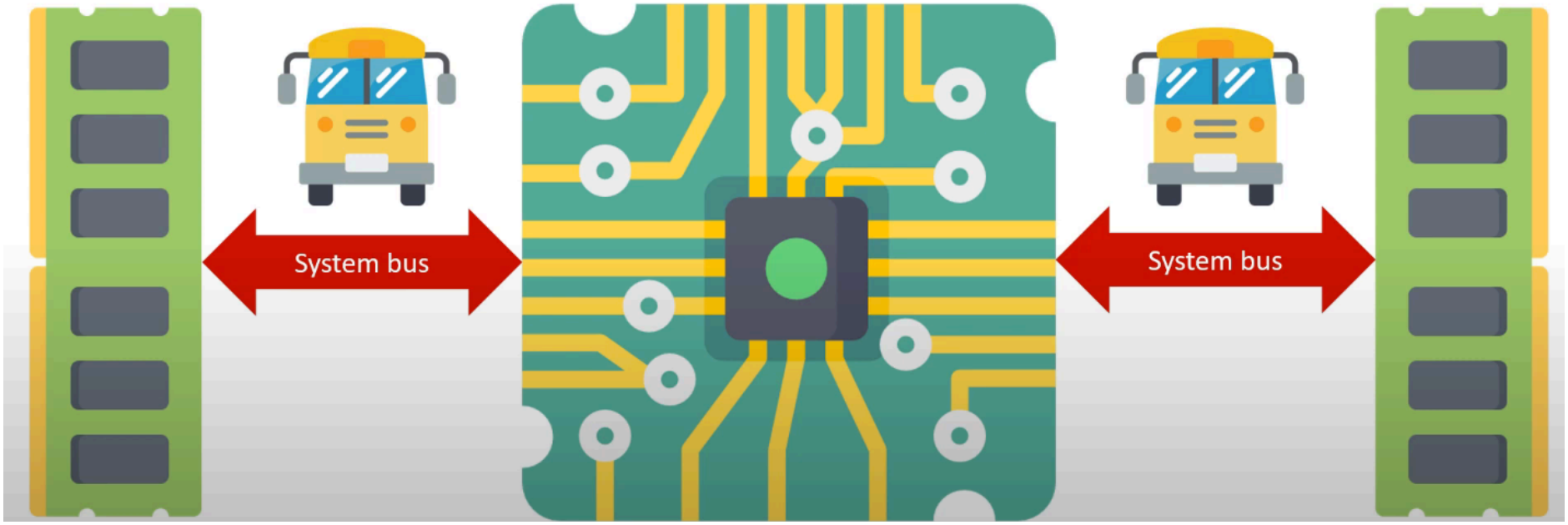


Von Neumann and Harvard Architecture

Harvard

- Instructions and data stored in separate memory units.
- Each has its own bus.
- Reading and writing data can be done at the same time as fetching an instruction.
- Used by RISC processors.

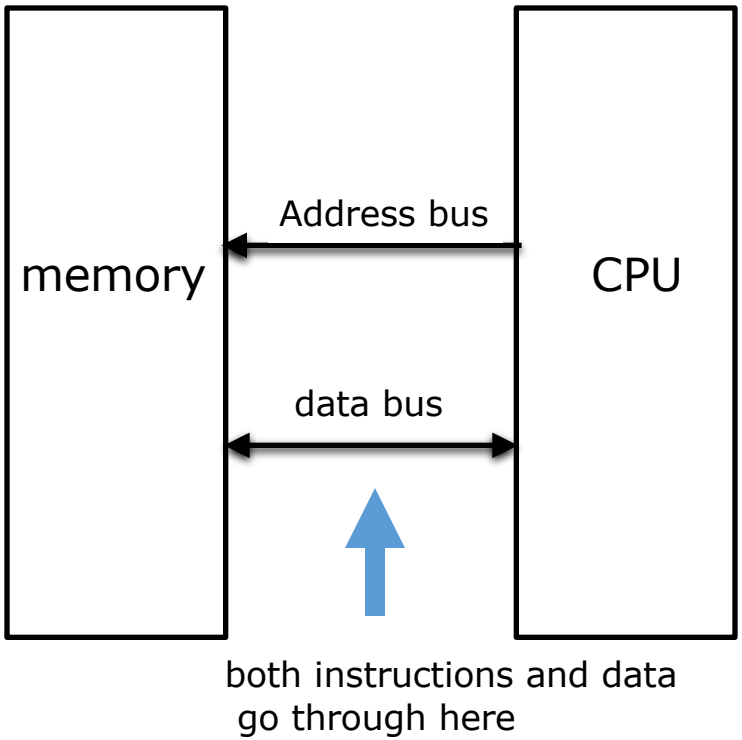
?

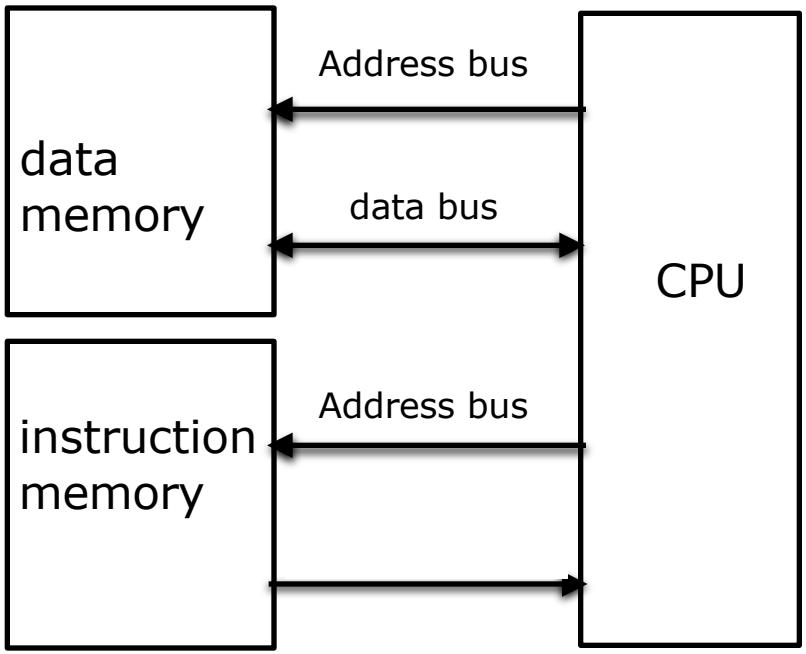


RAM Instruction | Central Processing Unit (CPU) | RAM Data

Von Neumann and Harvard Architecture

- Represent **two different ways of exchanging data between CPU and memory**





Von Neumann Architecture

Harvard Architecture

Von Neumann and Harvard Architecture

von Neumann Architecture

- Used in most general purpose computers
- Use the same bus to access data and instructions

Harvard Architecture

- Many microcontrollers use this architecture
- Separate buses access data and instructions, can operate at the same time
- Different types of memory used for data (RAM, volatile) and instructions (ROM, non-volatile, usually flash memory)

'Volatile' means that the contents are lost when power is removed

Important terms

- **Port**
  - A physical connection between the computer and the outside world; I/O port = input/output port
- **Input Port**
  - Hardware that allows information about the external world to enter the microcontroller
- **Output Port**
  - Hardware to send information out to the world
- **Device Driver**
  - Software that helps you use an I/O port (provides a software interface to hardware devices)
- **Interface**
  - *Hardware*: The collection of the I/O port, external electronics, physical devices
  - *Software*: device driver

Important terms

- **Bus**
  - Used to pass information between modules
- **Address** specifies
  - which module and cell within the module is being accessed
- **Data** contains
  - information that is being transferred
- **Control signals** specify
  - direction of the transfer
  - size of the data
  - timing information
- **The processor** always controls
  - *address* (where to access)
  - *direction* (read or write)
  - *control* (when to access)

I/O Interfaces

We can classify I/O interfaces into four categories

- **Parallel** – binary data are available simultaneously on a group of lines (bus)
  Used mainly inside microcontroller
- **Serial** – Binary data are available one bit at a time on a single line
  Used mainly for external signals (later)
- **Analog** – data are encoded as an electrical voltage, current, or power
- **Time** – data are encoded as a period, frequency, pulse width, or phase shift

Memory-Mapped I/O

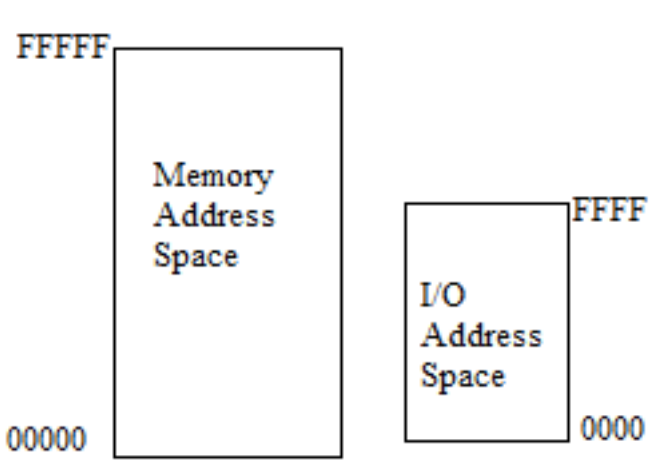
This is the most common way of treating the input/output ports.

- I/O ports are connected like memory
- I/O ports are assigned addresses
- Software accesses I/O using reads and writes to the specific I/O addresses
- **The same instructions** are used for accessing I/O ports and memory
  - Inputs from an input device: same instructions as a memory read
  - Outputs to an output device: same instructions as a memory write

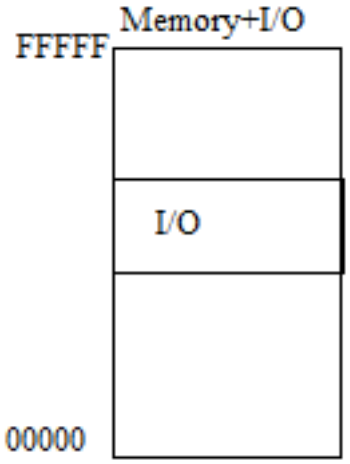
This is simple to use.

Memory-Mapped I/O vs. I/O-mapped I/O

- **I/O-mapped I/O** (used in older Intel devices)
  - The control bus signals that activate the I/O are separate from those that activate the memory devices
  - Separate address space
  - Separate instructions to access the I/O devices
- **Comparison**
  - I/O-mapped I/O: Software can not inadvertently access I/O when it thinks it is accessing memory
  - Memory-mapped I/O: Easier to design, and software is easier to write



I/O-mapped I/O



memory-mapped I/O

Two Types of Bus Cycles

Read cycle: Transfer data into the processor

- Processor places the address on the address signals
- Processor issues a read command on the control signals
- The slave module responds by placing the contents at that address on the data signals
- Processor will accept the data and disable the read command

Type	Address Driven	Data Driven by	Transfer to
Read Cycle	processor	RAM, ROM, Input	processor
Write Cycle	processor	processor	Output or RAM

Simple computers generate two types of cycles.

Two Types of Bus Cycles

Write cycle: Store data into memory or I/O

- Processor places the address on the address signals
- Processor places the information it wishes to store on the data signals
- Processor issues a write command on the control signals
- The memory or I/O will respond by storing the information into the proper place
- After the processor is sure the data has been captured, it will disable the write command

EMBEDDED SYSTEM (ES)

Components of Embedded Systems

- **It has Hardware**

  Processor, Timers, Interrupt controller, I/O Devices, Memories, Ports, etc.

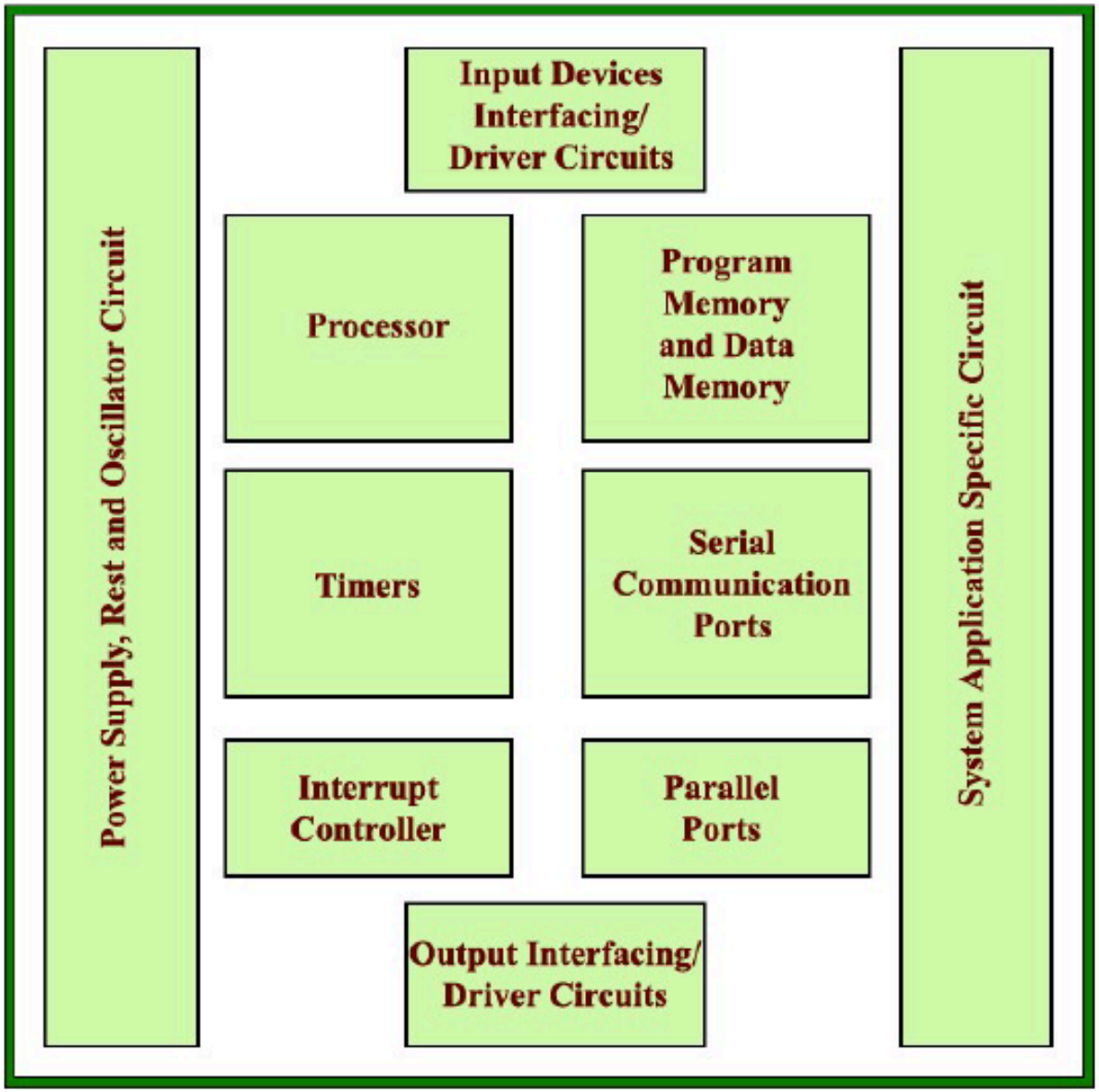
- **It has main Application Software**

  Which may perform concurrently the series of tasks or multiple tasks.

- **It has Real Time Operating System (RTOS)**

  RTOS defines the way the system work. Which supervise the application software. It sets the rules during the execution of the application program. A small scale embedded system may not need an RTOS.

Embedded Systems Hardware



Embedded Systems Constraints

An embedded system is software designed to keep in view three constraints:

- **Available system memory**
- **Available processor speed**
- **The need to limit the power dissipation**

When running the system continuously in cycles of wait for events, run, stop and wakeup.

What Make ES Different?

- Real-time operation
- size
- cost
- time
- reliability
- safety
- energy
- security

Various Processors

1. General Purpose processor (GPP)

- Microprocessor
- Microcontroller
- Embedded Processor
- Digital signal Processor

2. Application Specific System Processor (ASSP)

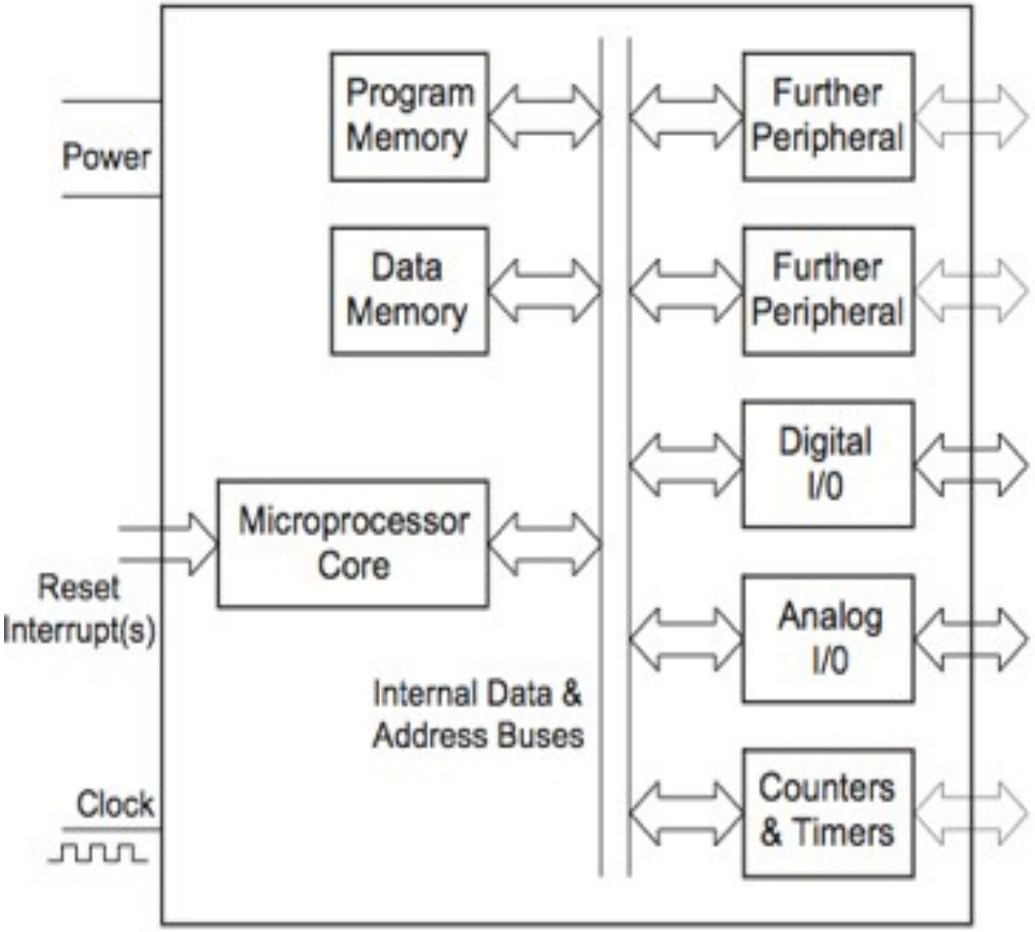
3. Multi Processor System using GPPs

The Microcontroller

- A **microcontroller** is a functional computer system-on-a-chip. It contains a processor, memory, and programmable input/output peripherals.

- Microcontrollers include an integrated CPU, memory (a small amount of RAM, program memory, or both) and peripherals capable of input and output.

The Microcontroller



Various Microcontroller

INTEL

8031,8032,8051,8052,8751

PIC

8-bit PIC16, PIC18,

16-bit DSPIC33 / PIC24,

PIC16C7x

Motorola

MC68HC11

Features of an example microcontroller:
core + **memory** + **peripherals**

Microprocessor Vs. Microcontroller

MICROPROCESSOR	MICROCONTROLLER
The functional blocks are ALU, registers, timing & control units	It includes functional blocks of microprocessors & in addition has timer, parallel i/o, RAM, EPROM, ADC & DAC
Bit handling instruction is less, One or two type only	Many type of bit handling instruction
Rapid movements of code and data between external memory & MP	Rapid movements of code and data within MC
It is used for designing general purpose digital computers system	They are used for designing application specific dedicated systems

Embedded Processor

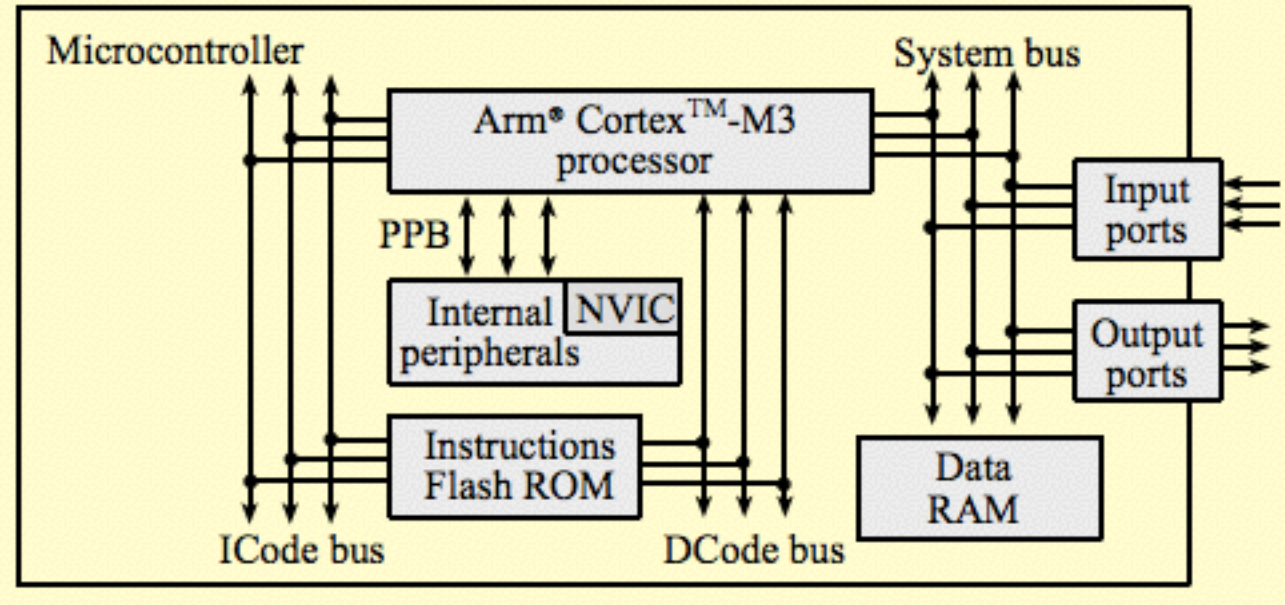
- **Special microprocessors & microcontrollers often called, Embedded processors.**

- **An embedded processor is used when fast processing fast context-switching & atomic ALU operations are needed.**

Examples : ARM 7, INTEL i960, AMD 29050.

ARM PROCESSOR

ARM Cortex-M3 Architecture



- ARM Cortex-M3 processor
- **computer architecture?**
  - Different buses for instructions and data – Harvard architecture
  - **ICode bus**: **fetch instructions** from flash ROM
    - Contains 32 bits of data
    - May be one or two instructions
  - **DCode bus**: fetches data or debug information from flash ROM
  - **System bus**: **Read/write data from RAM or I/O ports**
  - **Private peripheral bus** (PPB): access some of the common peripherals like the interrupt controller

Instruction Set Architecture (ISA) for Simple Computer

- Instructions are stored in RAM or ROM as a program
- The address for the next instruction is provided by a **program counter** (PC)
  - This counts up after each instruction
  - Or loads a new address after an **if … else** statement
- The PC and associated control logic are part of the Control Unit
- A typical instruction specifies:
  - Operands to use
  - Operation to be performed
  - Where to place the result, or which instruction to execute next
- The **Program status register** (PSR) contains information about the result of the instruction, whether the result was zero for example

Steps in executing an instruction

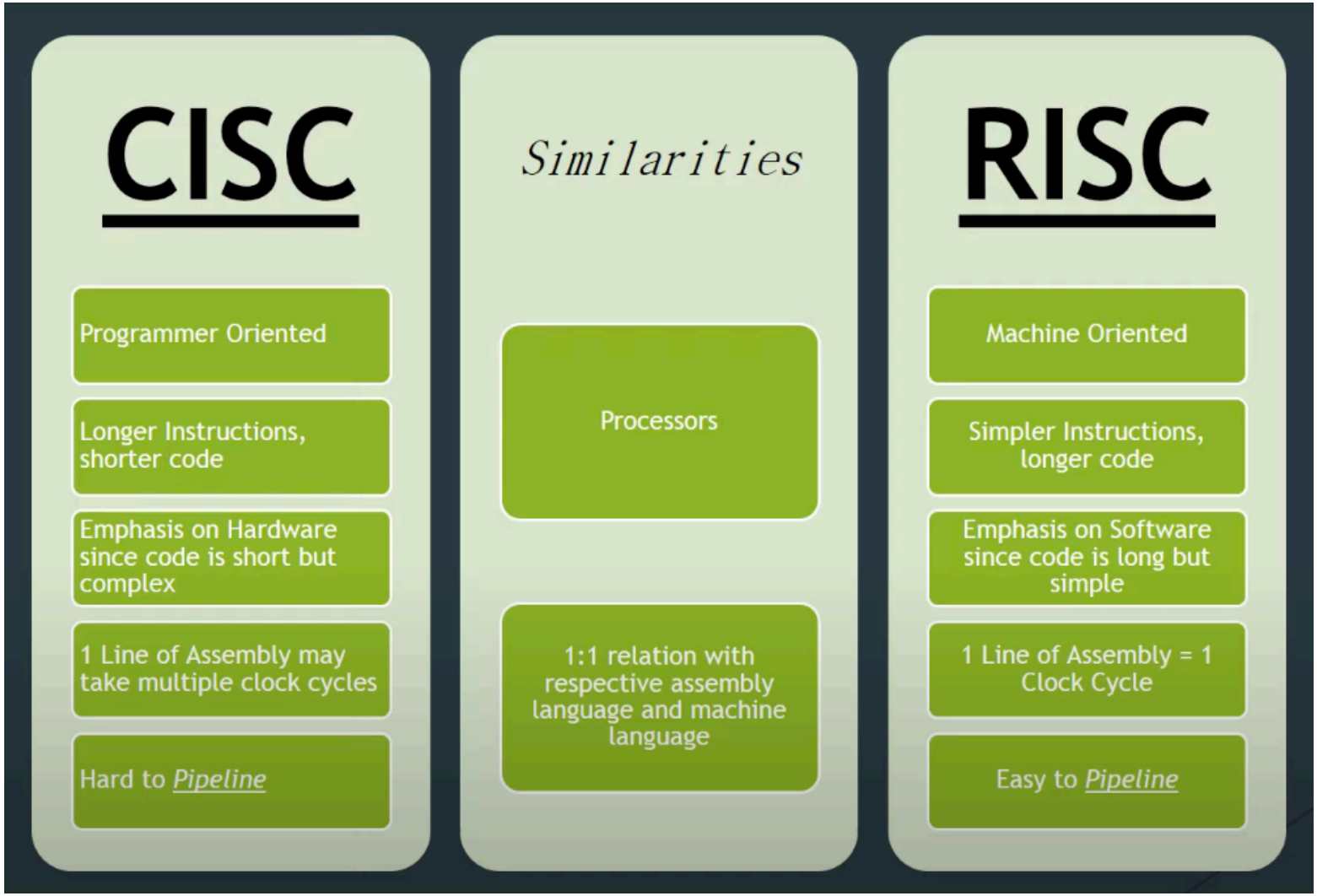
Phase	Function	Bus	Address	Comment
1	Instruction fetch	Read	PC++	Put in Instruction Register
2	Data read	Read	EAR	Data passes through ALU
3	Operation	-	-	ALU operations, set PSR
4	Data store	Write	EAR	Results stored in memory

Effective address register (EAR): contains the memory address used to fetch the data needed for the current instruction

Processor and Different Architectures

- Another term for Central Processing Unit (CPU)
- Primary functions include: Fetching, Decoding, Executing, Writeback
- 2 Architecture Types:
  - Complex Instruction Set Computing (CISC)
  - Reduced Instruction Set Computing (RISC)
    - Each Architecture has machine code that corresponds with their own unique Assembly Language (One to One)

RISC vs. CISC


CISC
Programmer Oriented
Longer Instructions, shorter code
Emphasis on Hardware since code is short but complex
1 Line of Assembly may take multiple clock cycles
Hard to Pipeline
Similarities
Processors
1:1 relation with respective assembly language and machine language
RISC
Machine Oriented
Simpler Instructions, longer code
Emphasis on Software since code is long but simple
1 Line of Assembly = 1 Clock Cycle
Easy to Pipeline

RISC versus CISC

RISC

- Instructions cannot read and write memory in the same bus cycle
- Only load and store instructions can access memory
- Small instruction set
- Fixed-size instructions
- Very few addressing modes
- Many identical general purpose registers

CISC

- Complex instructions with varying lengths
- Large instruction set
- Many addressing modes
- Many instructions can access memory
- Instructions can read and write memory in the same bus cycle
- Fewer and more specialized registers (data only or address only)

ISA Examples

RISC — Reduced Instruction Set Computer

- ARM
- Sun SPARC
- MIPS RX000 (used in 32-bit microcontrollers)
- IBM PowerPC (used in Macintosh in past)
- Microchip PIC16 (popular 8-bit microcontroller, unusual design)

CISC — Complex Instruction Set Computer

- Intel x86 and Pentium families
- Motorola 68000 (used in original Macintosh)
- DEC VAX (famous minicomputer, around 1980)

Microprocessors

	von Neumann	Harvard
RISC	ARM7	ARM9
CISC	Pentium	SHARC (DSP)

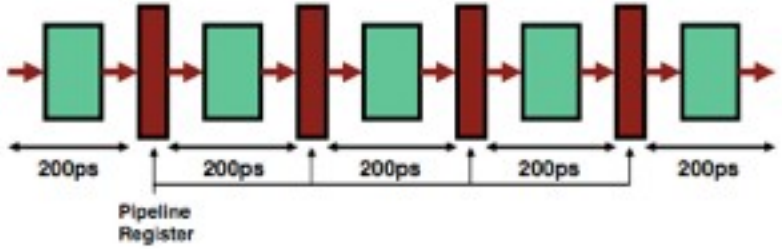
The SHARC is an example of a **Digital Signal Processor** (DSP) from Analog Devices. This is a specialised device, which does calculations for DSP very fast.

Pipelining a Digital System

Key ideas: break big computation into pieces

Non-pipelined:
1 operation finishes every
1ns = 1000ps

Split into 5 steps and separate each piece with a pipeline register



Pipelined:
1 operation finishes every
200ps

Why do this? It gives multiple operations to perform

- Pipelining increases throughput, but does not reduce total computation time of a task
  - An answer is available every 200ps, but
  - A single computation still takes 1ns
- Limitations:
  - Computations must be divisible into stage size
  - Pipeline registers add overhead

It is like a production line.

ARM Architecture

- **A family of instruction set architectures** for computer processors based on RISC architecture developed by British company ARM Holdings (now part of SoftBank Group Corp)
- ARM Ltd design and license ARM core designs but do not fabricate products themselves

Why ARM?

One of the most licensed and widely-used **processor cores** in the world

- The cores are intended for different types of embedded system
- Used in PDA, cell phones, multimedia players, Handheld game console, digital TV and cameras
- ARM7: GBA, iPod, cellphones around 2000
- ARM9: NDS, PSP, Sony Ericsson, BenQ
- ARM11: Apple iPhone, Nokia N93, N800
- 75% of 32-bit embedded processors

Used especially in portable devices due to its low power consumption and high performance

A little history

The original ARM processor was designed for a desktop computer. It used 32-bit instructions.

Most instructions in an embedded system do not need 32 bits so ARM introduced a 16-bit instruction set called **Thumb**.

This was added to the ARM7. A Multiplier was also added and the result was the **ARM7TDMI**, introduced in 1994.

This was a highly successful design and was used in many phones made in the years after 2000, such as the Nokia 5110.

The ARM7TDMI was also used in general purpose microcontrollers.

Image from wikipedia

A little history

The ARM7TDMI had some problems in microcontrollers.

- Although most code used 16-bit Thumb instructions, some operations had to be done in ARM 32-bit code
- Embedded systems rely heavily on **interrupts** (later in this course) and every designer had to add a ‘vector interrupt controller’ to the ARM core

ARM solved these problems when the **Cortex-M** series was introduced around 2005.

- It includes a vector interrupt controller
- It also includes a timer, essential for an operating system
- The instruction set was upgraded to **Thumb-2**, which is mostly 16-bit with a few 32-bit instructions: it is no longer necessary to switch between 16-bit and 32-bit modes

Cortex-M series

The Cortex-M family has several members:

- Cortex-M0 – smallest, only 56 instructions
- Cortex-M0+ – more powerful but MCUs can cost below 40¢
- Cortex-M3 – original version, used in our mbed platform
- Cortex-M4 – more powerful for digital signal processing; may include floating point (M4F)
- Cortex-M7 – even more powerful

New designs are simpler and use less power. *Rough* comparison:

- ARM7TDMI had around 45k gates and used 1 mW per MHz of clock speed
- Cortex-M0+ needs only 12k gates minimum and down to 3 µW per MHz of clock speed (partly due to fewer gates and modern design, but mainly smaller transistors)

ARM versions

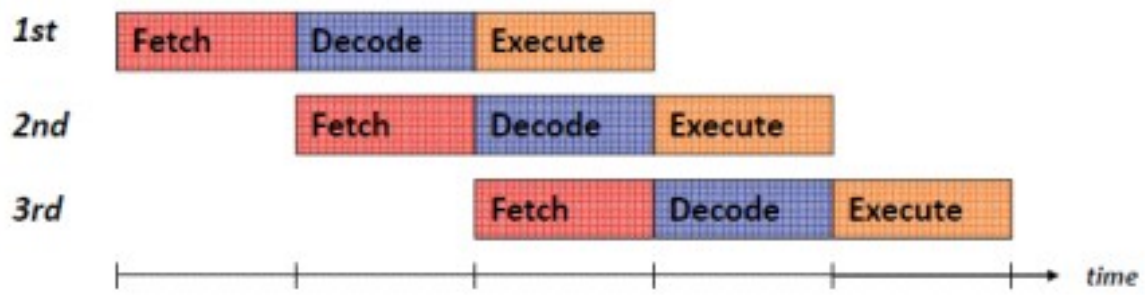
- ARM architecture has been extended over several versions.
  - ARM7TDMI
  - ARM9 — includes "Thumb" instruction set
  - ARM10 — for multimedia (graphics, video, etc.)
  - ARM11 — high performance + Jazelle (Java)
  - SecureCore — for security app's (smart cards)
  - **Cortex-M** — Optimized for microcontrollers
  - Cortex-A — High performance (multimedia systems)
  - Cortex-R — Optimized for real-time app's
  - StrongARM — portable communication devices

ARM ISA: Thumb2 Instruction Set

- Variable-length instructions
  - Traditional ARM instructions are a fixed length of 32 bits
  - **Thumb** instructions are a fixed length of 16 bits
  - **Thumb-2** instructions can be either 16-bit or 32-bit
- Thumb-2 gives approximately 26% improvement in code density over ARM
- Thumb-2 gives approximately 25% improvement in performance over Thumb
- **ARM Cortex-M3 supports the Thumb-2 instruction set**
  - Note that **not all the instructions in the Thumb-2** instruction set are implemented on the Cortex-M3
  - Cortex-M3 utilizes a mixture of 32-bit and 16-bit instructions (mostly the latter)
  - **Only a few instructions (such as load and store) can access memory**

Cortex-M3 Pipeline

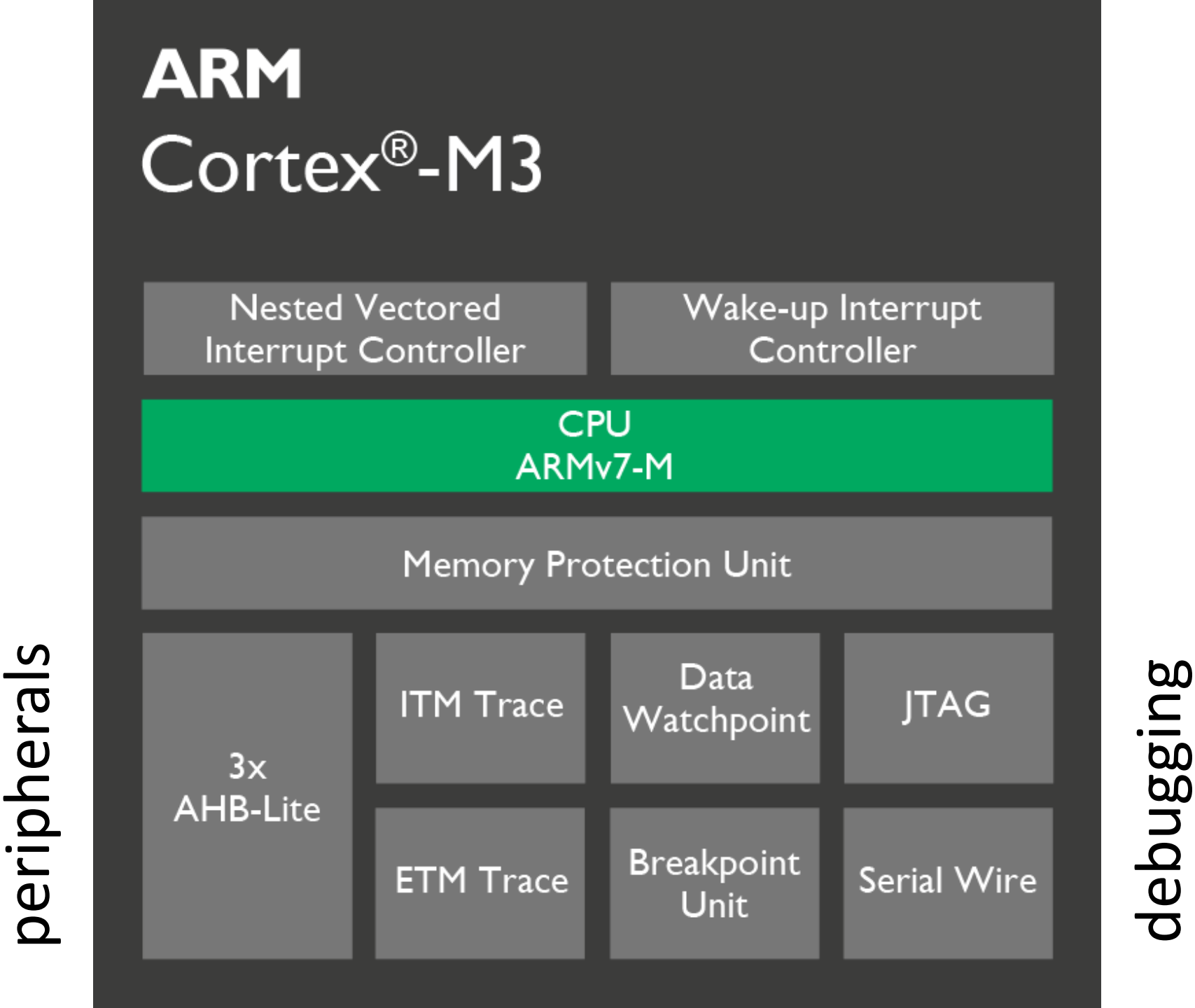
- The Cortex-M3 uses the 3-stage pipeline for instruction executions
  - Fetch => Decode => Execute
  - Pipeline design allows effective throughput to increase to one instruction per clock cycle
  - Allows the next instruction to be fetched while still decoding or executing the previous instructions

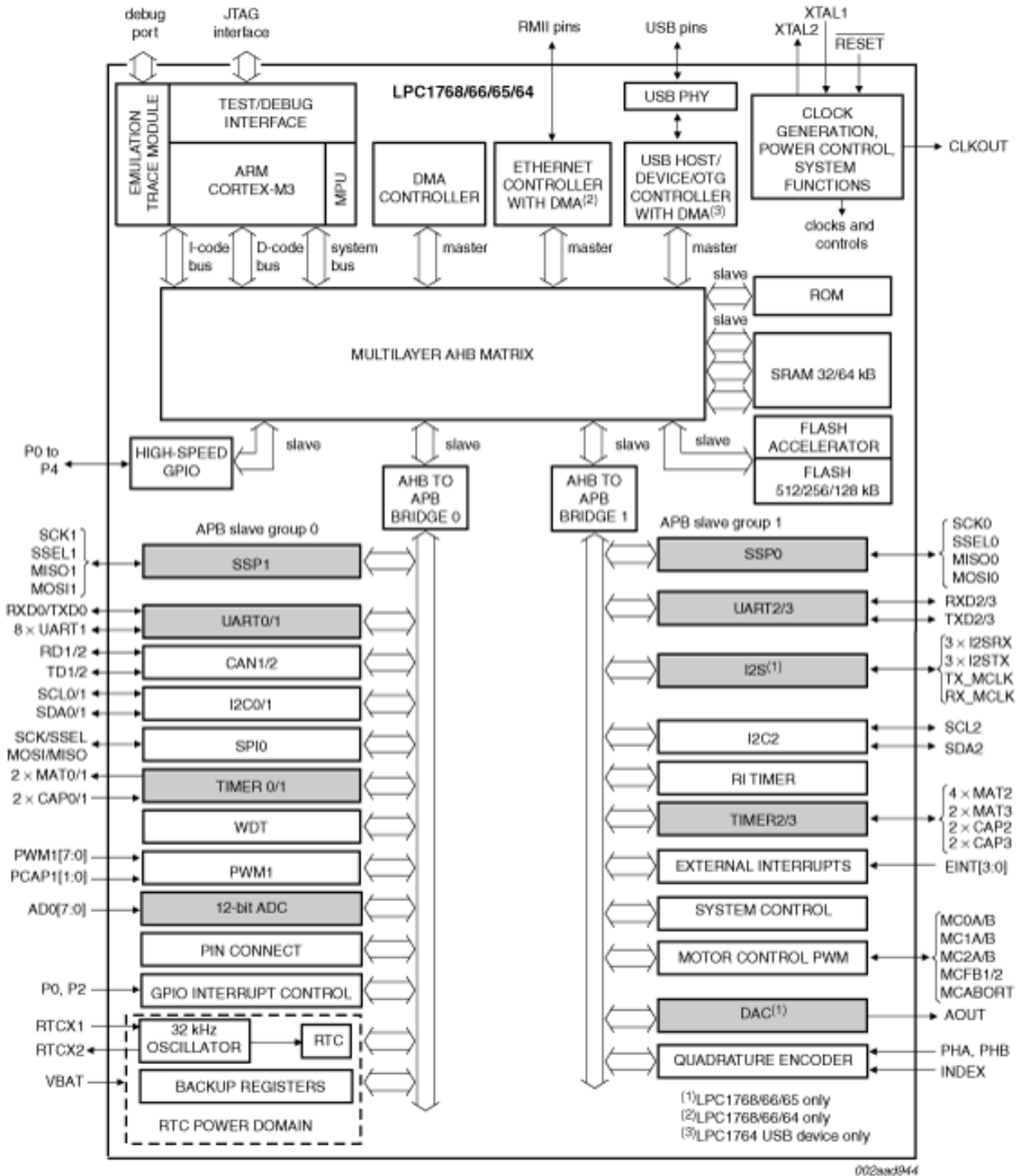


(The M0+ has only a 2-stage pipeline)

Cortex-M3 and LPC1768

- Cortex-M3 differs from previous generations of ARM processors by defining a number of key peripherals as part of the core:
  - Interrupt controller
  - System timer
  - Debug and trace hardware (including external interfaces)
- This enables for real-time operating systems and hardware development tools such as debugger interfaces be common across the family of processors
- Various Cortex-M3 based microcontroller families differ significantly in terms of **hardware peripherals and memory**
- At the heart of the mbed is the LPC1768 microcontroller
  - LPC17xx (of NXP) is an ARM Cortex-M3 based micro controller
  - The Cortex-M3 is also the basis for microcontrollers from other manufacturers including TI, ST, Toshiba, Atmel, etc.


ARM
Cortex®-M3
Nested Vectored Interrupt Controller
Wake-up Interrupt Controller
CPU ARMv7-M
Memory Protection Unit
3x AHB-Lite
ITM Trace
Data Watchpoint
JTAG
ETM Trace
Breakpoint Unit
Serial Wire
peripherals
debugging


debug port
JTAG interface
RMII pins
USB pins
XTAL1
XTAL2
RESET
LPC1768/66/65/64
EMULATION TRACE MODULE
TEST/DEBUG INTERFACE
ARM CORTEX-M3
MPU
USB PHY
CLOCK GENERATION, POWER CONTROL, SYSTEM FUNCTIONS
CLKOUT
clocks and controls
DMA CONTROLLER
ETHERNET CONTROLLER WITH DMA(2)
USB HOST/ DEVICE/OTG CONTROLLER WITH DMA(3)
I-code bus
D-code bus
system bus
master
master
master
slave
ROM
MULTILAYER AHB MATRIX
slave
SRAM 32/64 kB
FLASH ACCELERATOR
FLASH 512/256/128 kB
P0 to P4
HIGH-SPEED GPIO
slave
slave
slave
slave
AHB TO APB BRIDGE 0
AHB TO APB BRIDGE 1
APB slave group 0
APB slave group 1
SCK1
SSEL1
MISO1
MOSI1
SSP1
SCK0
SSEL0
MISO0
MOSI0
SSP0
RXD0/TXD0
8 × UART1
UART0/1
UART2/3
RXD2/3
TXD2/3
RD1/2
TD1/2
CAN1/2
I2S(1)
3 × I2SRX
3 × I2STX
TX_MCLK
RX_MCLK
SCL0/1
SDA0/1
I2C0/1
I2C2
SCL2
SDA2
SCK/SSEL
MOSI/MISO
SPI0
2 × MAT0/1
2 × CAP0/1
TIMER 0/1
RI TIMER
WDT
TIMER2/3
4 × MAT2
2 × MAT3
2 × CAP2
2 × CAP3
PWM1[7:0]
PCAP1[1:0]
PWM1
EXTERNAL INTERRUPTS
EINT[3:0]
AD0[7:0]
12-bit ADC
SYSTEM CONTROL
PIN CONNECT
MOTOR CONTROL PWM
MC0A/B
MC1A/B
MC2A/B
MCFB1/2
MCABORT
P0, P2
GPIO INTERRUPT CONTROL
DAC(1)
AOUT
RTCX1
RTCX2
32 kHz OSCILLATOR
RTC
QUADRATURE ENCODER
PHA, PHB
INDEX
VBAT
BACKUP REGISTERS
RTC POWER DOMAIN
(1)LPC1768/66/65 only
(2)LPC1768/66/64 only
(3)LPC1764 USB device only
002aad944

Polling and interrupt

I/O devices

- External devices attached to the CPU: mouse, keyboard, scanner, printer, etc. These devices need the CPU attention. Supposing that, a CPU is busy in displaying a PDF and you click the window media player icon on the desktop. Though the CPU does not have any idea when an event like this would occur, but it has to respond to such inputs from the I/O devices.

Interrupt and Polling

- **Interrupt and Polling** are the two ways to handle the events generated by the devices that can happen at any moment while CPU is busy in executing another process.

- **Polling and Interrupt** let CPU stop what it is currently doing and respond to the more important task.

Interrupt and Polling

- **Polling and Interrupt** are different from each other in many aspects.
- In polling CPU keeps on checking I/O devices at regular interval whether it needs CPU service.
- In interrupt, the I/O device interrupts the CPU and tell CPU that it need CPU service.

Comparison of interrupt and polling

BASIS FOR COMPARISON	INTERRUPT	POLLING
Basic	Device notify CPU that it needs CPU attention.	CPU constantly checks device status whether it needs CPU's attention.
Mechanism	An interrupt is a hardware mechanism.	Polling is a Protocol.
Servicing	Interrupt handler services the Device.	CPU services the device.
Indication	Interrupt-request line indicates that device needs servicing.	Command-ready bit indicates the device needs servicing.
CPU	CPU is disturbed only when a device needs servicing, which saves CPU cycles.	CPU has to wait and check whether a device needs servicing which wastes lots of CPU cycles.

Comparison of interrupt and polling

Occurrence	An interrupt can occur at any time.	CPU polls the devices at regular interval.
Efficiency	Interrupt becomes inefficient when devices keep on interrupting the CPU repeatedly.	Polling becomes inefficient when CPU rarely finds a device ready for service.
Example	Let the bell ring then open the door to check who has come.	Constantly keep on opening the door to check whether anybody has come.

Definition of Interrupt

- An interrupt is a **hardware mechanism** that enables CPU to detect that a device needs its attention. The CPU has a wire **interrupt-request line** which is checked by CPU after execution of every single instruction. When CPU senses an interrupt signal on the interrupt-request line, CPU stops its currently executing task and respond to the interrupt send by I/O device by passing the control to **interrupt handler**. The interrupt handler resolves the interrupt by servicing the device.

- Although CPU is not aware when an interrupt would occur as it can occur at any moment, but it has to respond to the interrupt whenever it occurs.

Definition of Interrupt

- When the interrupt handler finishes executing the interrupt, then the CPU **resumes** the execution of the task that it has stopped for responding the interrupt. **Software**, **hardware**, **user**, **some error in the program**, etc. can also generate an interrupt. Interrupts handling nature of CPU leads to **multitasking**, i.e. a user can perform a number of different tasks at the same time.

- If more than one interrupts are sent to the CPU, the interrupt handler helps in managing the interrupts that are waiting to be processed. As interrupt handler gets **triggered** by the reception of an interrupt, it **prioritizes** the interrupts waiting to be processed by the CPU and arranges them in a **queue** to get serviced.

Definition of Polling

- As we have seen in interrupts, the input from I/O device can arrive at any moment requesting the CPU to process it. Polling is a protocol that notifies CPU that a device needs its attention. Unlike in interrupt, where device tells CPU that it needs CPU processing, in polling CPU keeps asking the I/O device whether it needs CPU processing.

- The CPU continuously test each and every device attached to it for detecting whether any device needs CPU attention. Every device has a command-ready bit which indicates the status of that device i.e. whether it has some command to be executed by CPU or not.

- If command bit is set 1, then it has some command to be executed else if the bit is 0, then it has no commands. CPU has a busy bit that indicates the status of CPU whether it is busy or not. If the busy bit is set 1, then it is busy in executing the command of some device, else it is 0.

Algorithm for polling

- When a device has some command to be executed by CPU it continuously checks the busy bit of CPU until it becomes clear (0).
- As the busy bit becomes clear, the device set write-bit in its command register and writes a byte in data-out register.
- Now the device sets (1) the command-ready bit.
- When CPU checks the devices command-ready bit and finds it set (1), it sets (1) its busy bit.
- The CPU then reads the command register of the device and executes the command of the device.
- After command execution, CPU clears(0) the command-ready bit, error bit of the device to indicate successful execution of the command of the device and further it clears (0) its busy bit also to indicate that the CPU is free to execute the command of some other device.

Key Differences Between Interrupt and Polling in OS

1. In interrupt, the device notifies the CPU that it needs servicing whereas, in polling CPU repeatedly checks whether a device needs servicing.

2. Interrupt is a **hardware mechanism** as CPU has a wire, **interrupt-request line** which signal that interrupt has occurred. On the other hands, Polling is a **protocol** that keeps checking the **control bits** to notify whether a device has something to execute.

3. Interrupt handler handles the interrupts generated by the devices. On the other hands, in polling, **CPU** services the device when they require.

4. Interrupts are signaled by the **interrupt-request line**. However, **Command-ready** bit indicate that the device needs servicing.

Key Differences Between Interrupt and Polling in OS

5. In interrupts, CPU is only disturbed when any device interrupts it. On the other hand, in polling, CPU waste lots of CPU cycles by repeatedly checking the command-ready bit of every device.

6. An interrupt can occur at **any instant of time** whereas, CPU keeps polling the device at the **regular intervals**.

7. Polling becomes inefficient when CPU keeps on polling the device and rarely finds any device ready for servicing. On the other hands, interrupts become inefficient when the devices keep on interrupting the CPU processing repeatedly.

Conclusion

- Both Polling and Interrupts are efficient in attending the I/O devices. But they can become inefficient at a certain condition as discussed.