ARM PROCESSOR AND PROGRAMMING

UNIT – 1

Faculty: Dr. Jisha P

UNIT 1

ARM Processor fundamentals –

Basic Structure of computers- Von Neumann and Harvard Architecture, Basic Processing Unit, Bus Structure, RISC and CISC Architecture, RISC and ARM Design philosophy, ARM core Dataflow model, programming model, processor states and operating modes, ARM pipeline.

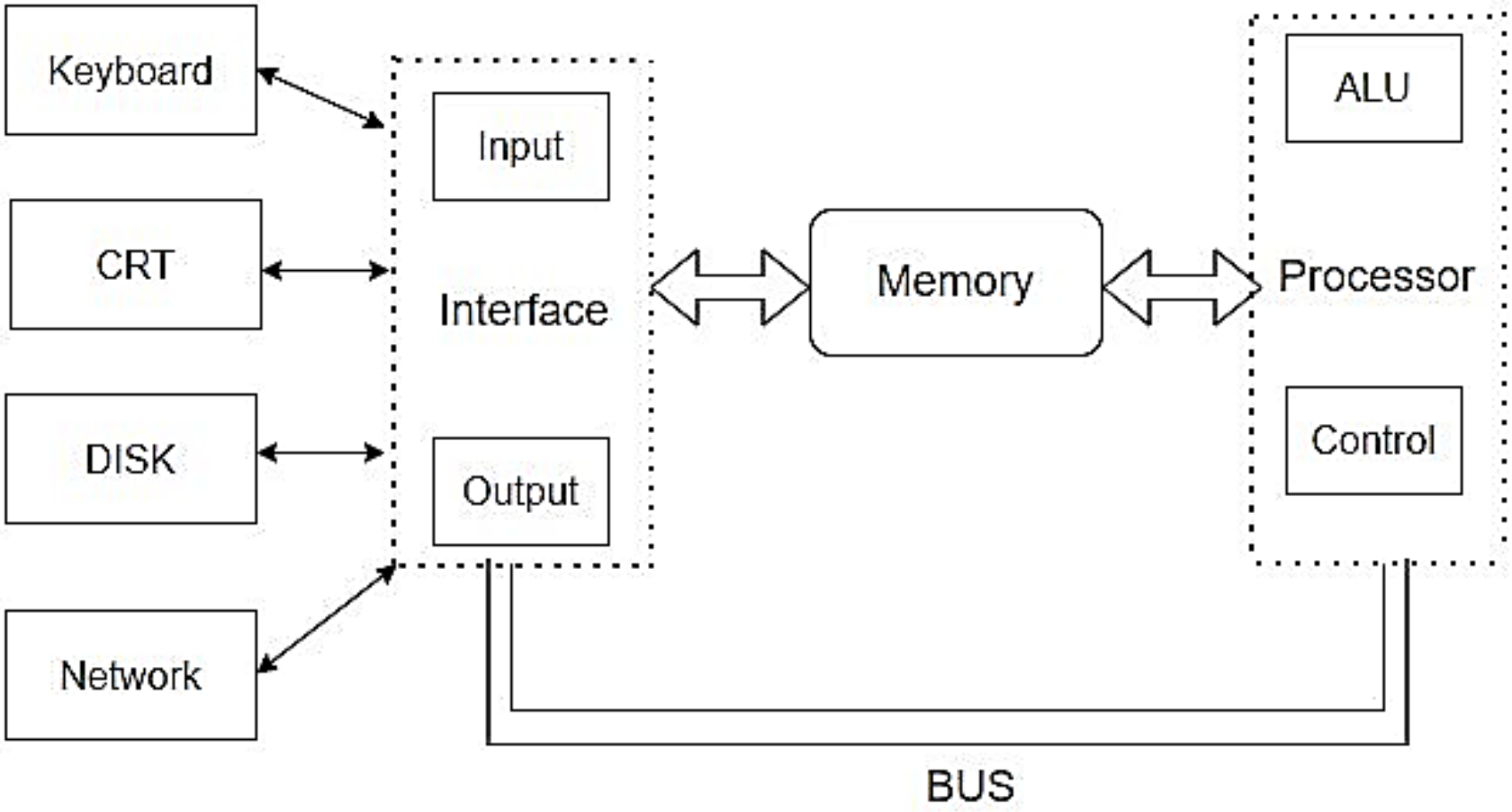
OVERVIEW OF COMPUTING SYSTEMS

- Computing systems are electronic devices or machines that perform various tasks by executing instructions, processing data, and providing outputs.
- A typical computing system consists of hardware components, such as the central processing unit (CPU), memory, storage devices, input/output devices, and a communication network.
- The CPU, often referred to as the brain of the computer, carries out instructions and performs calculations. It consists of an arithmetic logic unit (ALU) and a control unit.
- Memory, or RAM (Random Access Memory), stores data and instructions that are currently being used by the CPU. It provides fast access to data, enabling efficient processing.
- Storage devices, such as hard disk drives (HDDs) and solid-state drives (SSDs), are used for long-term data storage even when the power is turned off.

OVERVIEW OF COMPUTING SYSTEMS

- Input devices, such as keyboards, mice, and touchscreens, allow users to enter data or commands into the computer system.
- Output devices, such as monitors, printers, and speakers, provide users with the results or information processed by the computer system.
- Computing systems are governed by an operating system (OS) that manages hardware resources, provides a user interface, and enables the execution of software applications.
- Software applications are programs that run on computing systems, enabling users to perform specific tasks such as word processing, web browsing, or playing games.
- Computing systems can be classified into different types based on their size, capabilities, and purpose. These include personal computers (PCs), laptops, servers, mainframes, supercomputers, embedded systems, and mobile devices.

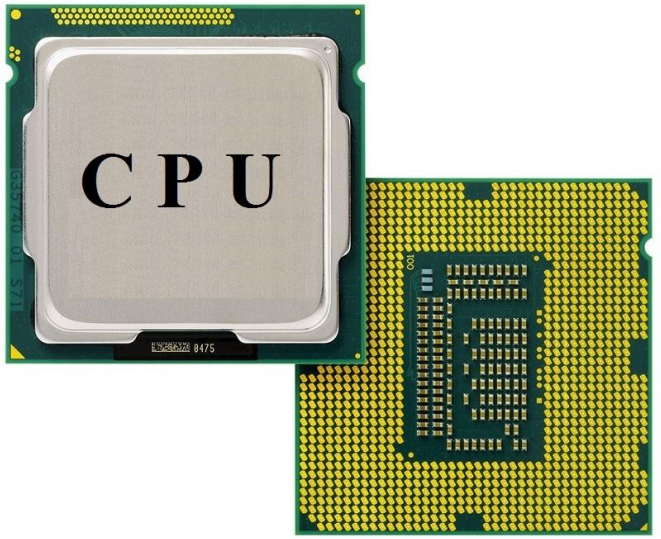
BASIC STRUCTURE OF COMPUTERS - FUNCTIONAL UNITS OF A COMPUTER



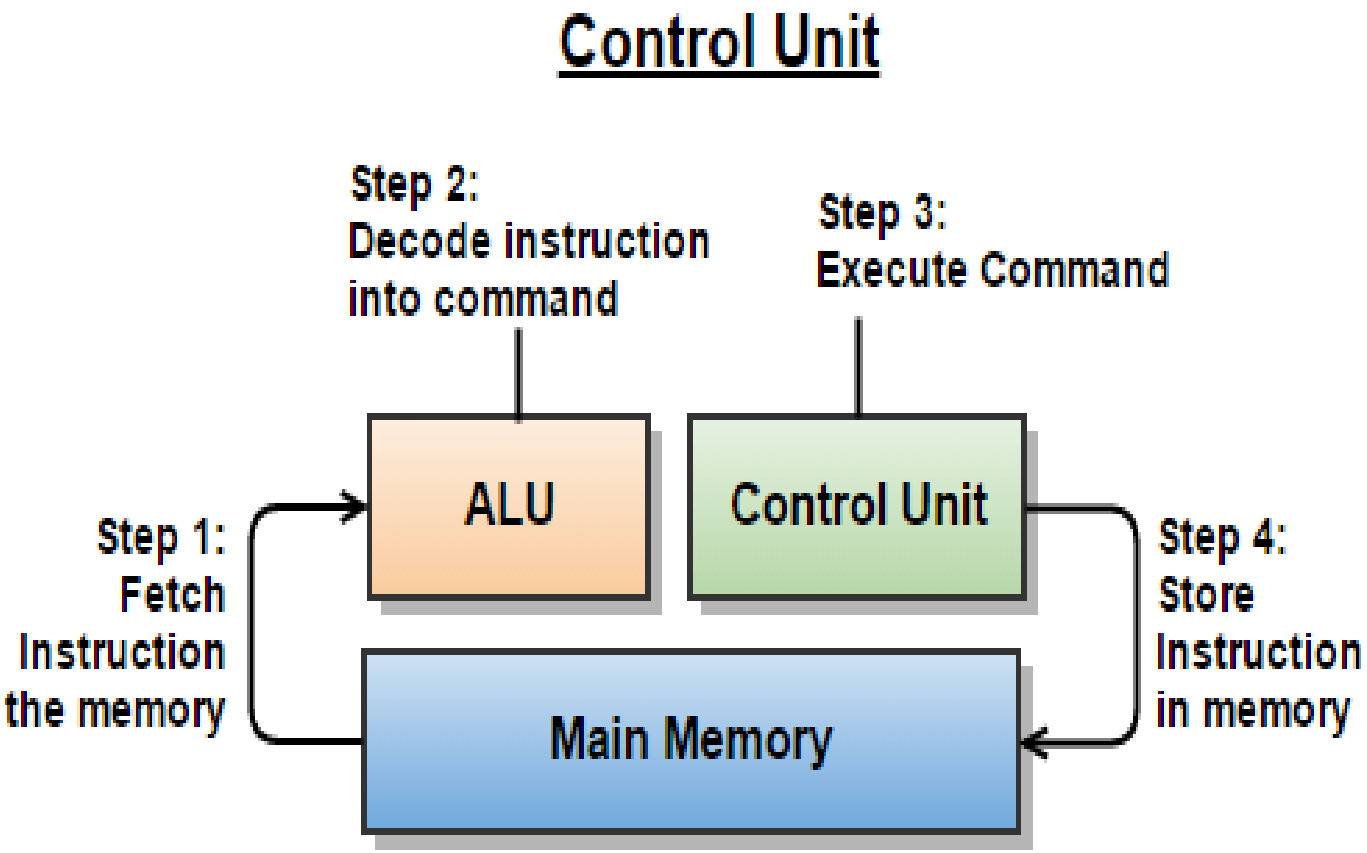
BASIC STRUCTURE OF COMPUTERS

- **Central Processing Unit (CPU):**

The CPU is often referred to as the brain of the computer. It performs the majority of the processing and calculations in a computer system. The CPU consists of two main units: the arithmetic logic unit (ALU) and the control unit. The ALU carries out mathematical operations and logical comparisons, while the control unit coordinates the flow of data and instructions within the CPU and between other components.



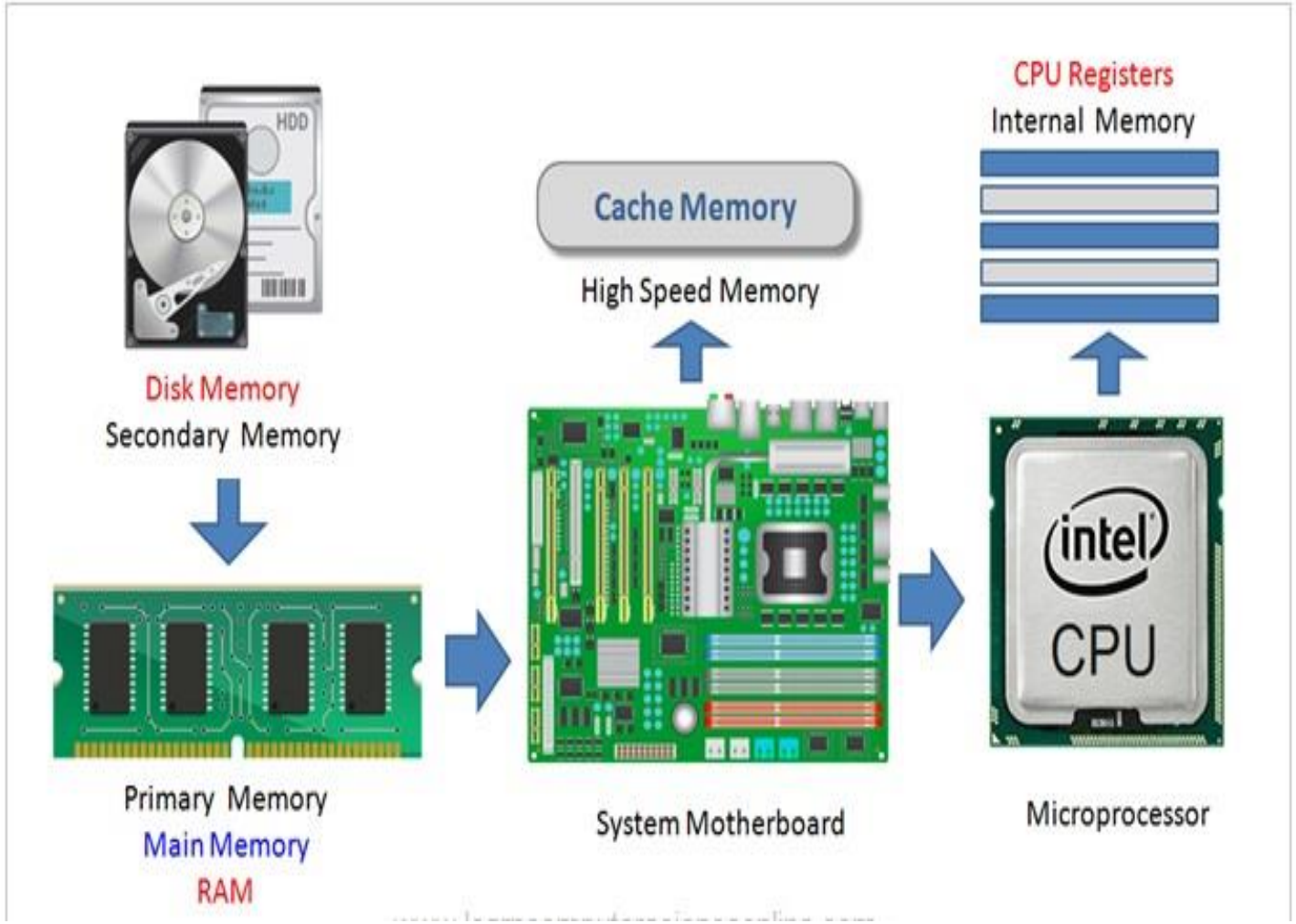
BASIC STRUCTURE OF COMPUTERS



- **Control Unit:**

The control unit coordinates the activities of other function units in the computer. It fetches instructions from memory, decodes them, and controls the flow of data between various function units. The control unit ensures that instructions are executed in the correct sequence and that data is transferred and processed accurately.

BASIC STRUCTURE OF COMPUTERS

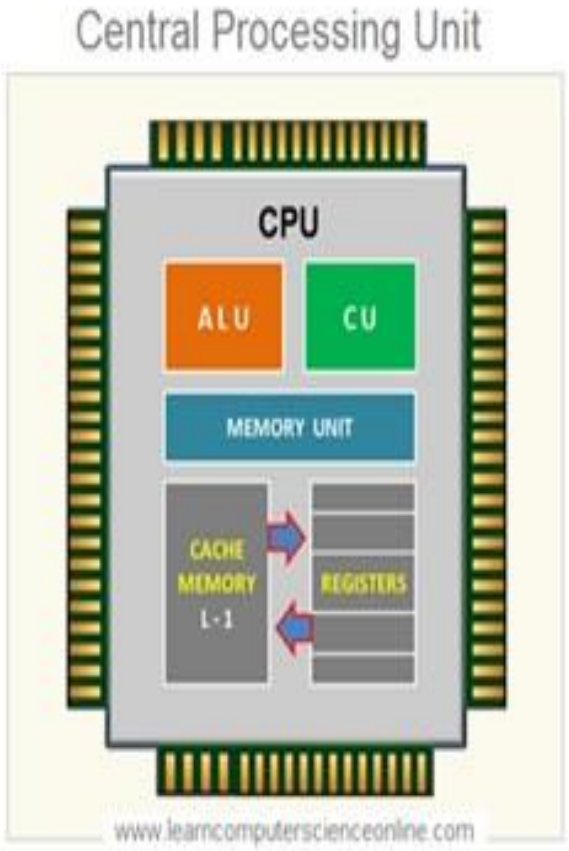


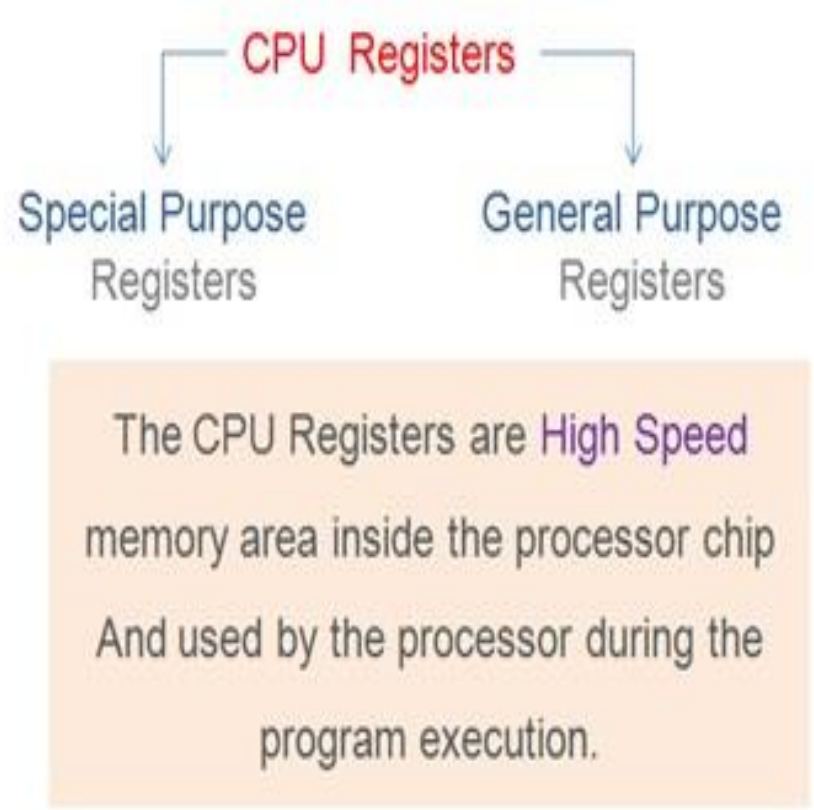
Registers → Cache → RAM → Secondary Storage.

- **Memory:**

Memory, or RAM (Random Access Memory), is a temporary storage space that holds data and instructions that the CPU needs to access quickly. It allows for fast reading and writing of data, making it crucial for the efficient functioning of a computer. RAM is volatile, meaning its contents are lost when the power is turned off. The size of the RAM determines how much data can be stored and accessed simultaneously.

BASIC STRUCTURE OF COMPUTERS

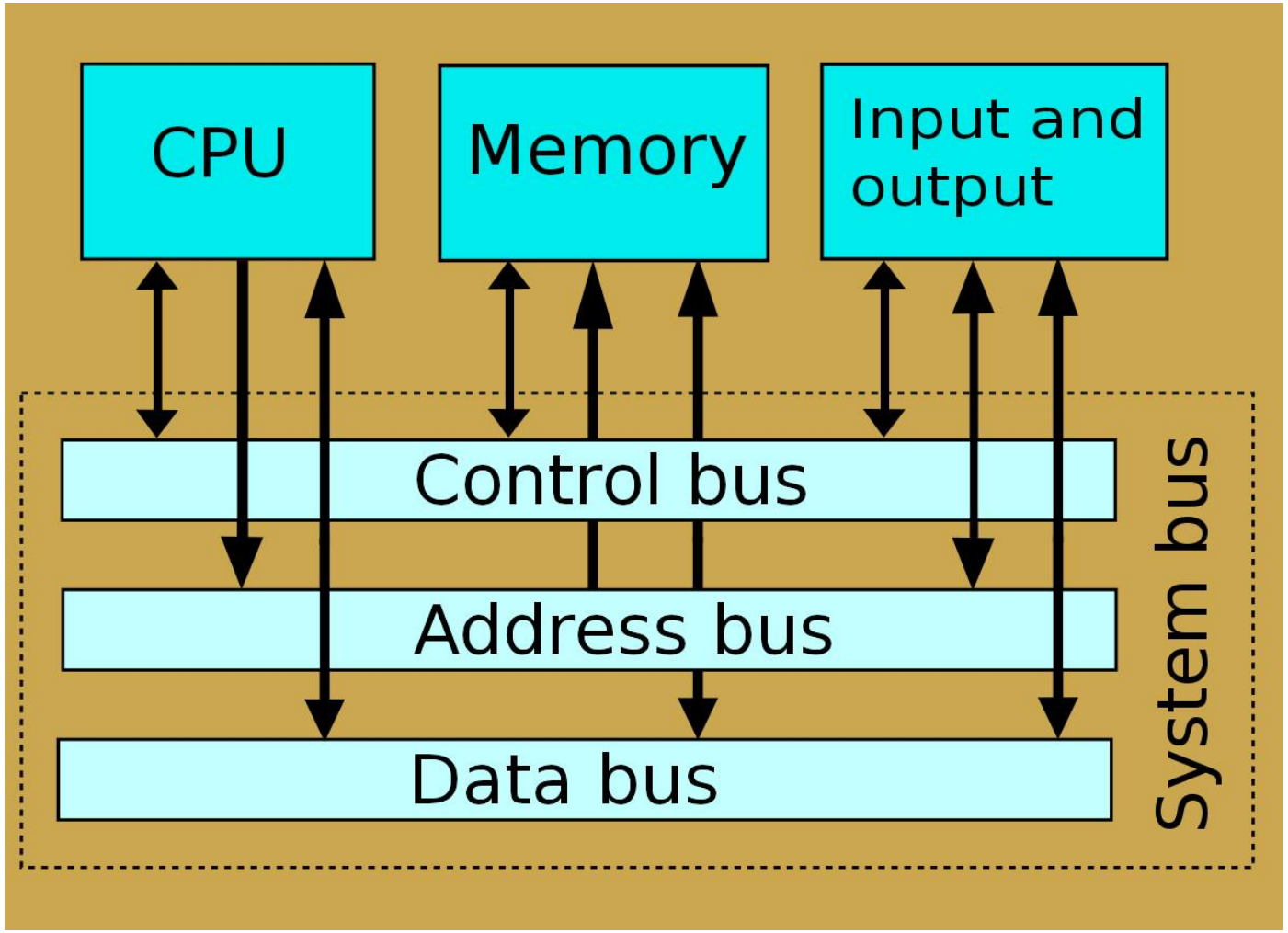




- Registers:

Registers are small, high-speed storage locations within the CPU. They hold instructions, data, and intermediate results during the execution of instructions. The CPU uses different types of registers, such as the program counter (PC), instruction register (IR), accumulator, and general-purpose registers, to perform various tasks efficiently.

BASIC STRUCTURE OF COMPUTERS



- **Bus System:**

The bus system consists of a set of physical pathways that allow data and instructions to be transferred between different function units. It includes the **address bus, data bus, and control bus**. The address bus carries memory addresses, the data bus transfers data, and the control bus carries control signals. The bus system enables communication and coordination among various components of the computer.

BASIC STRUCTURE OF COMPUTERS

- **Input Devices:**

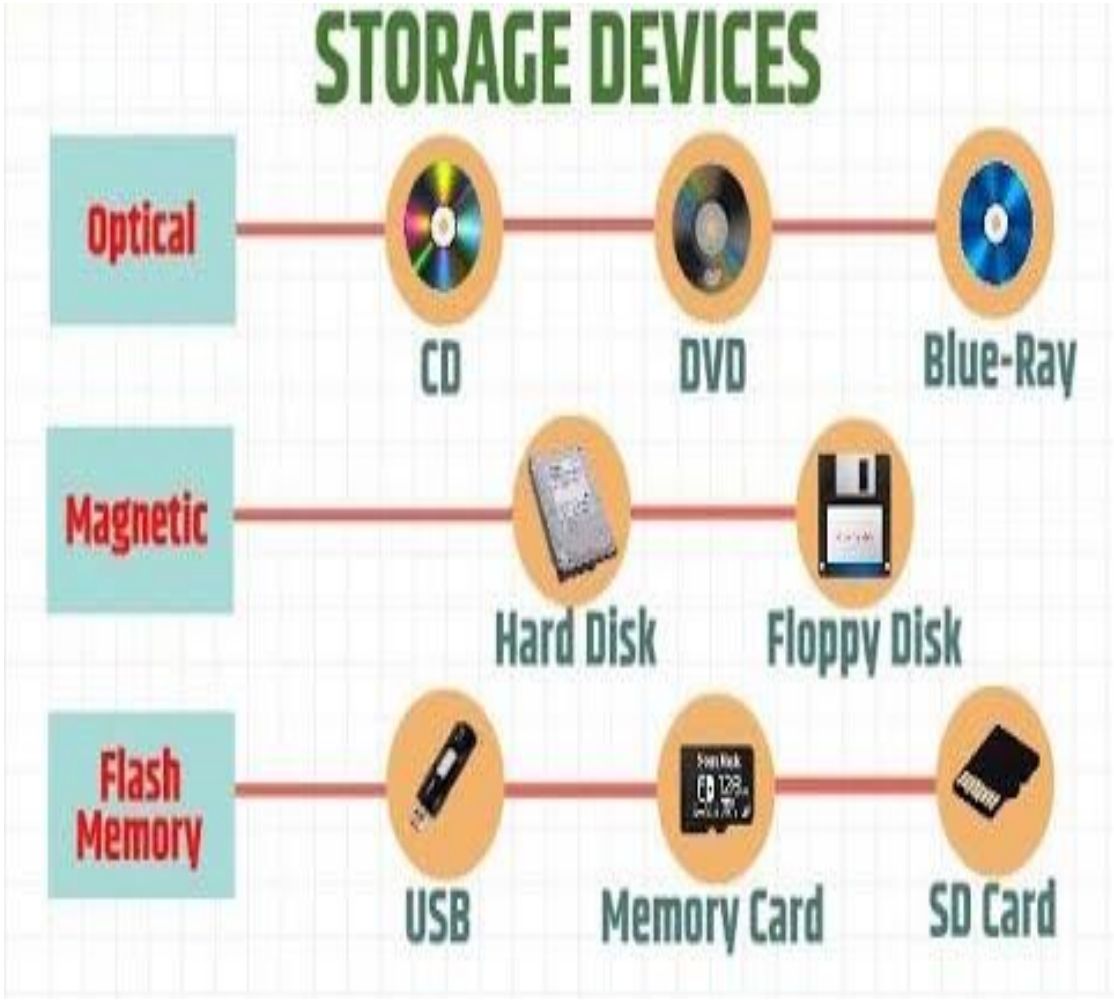
Input devices allow users to enter data and commands into the computer system. Common input devices include keyboards, mice, touchscreens, scanners, and microphones. These devices convert user input into signals that can be processed by the CPU. Input devices play a crucial role in interacting with the computer and providing it with the necessary instructions and information.

BASIC STRUCTURE OF COMPUTERS

- **Output Devices:**

Output devices provide users with the results or information processed by the computer system. They present the processed data in a human-readable form. Common output devices include monitors, printers, speakers, and projectors. These devices receive signals from the computer and convert them into a format that can be perceived by humans.

BASIC STRUCTURE OF COMPUTERS



- **Storage Devices:**

Storage devices are used for long-term data storage even when the power is turned off. They store the operating system, software applications, user data, and other files. The two primary types of storage devices are hard disk drives (HDDs) and solid-state drives (SSDs). HDDs use spinning disks and magnetic heads to read and write data, while SSDs use flash memory chips for faster data access.

BASIC STRUCTURE OF COMPUTERS

- **Secondary Storage Unit:**

The secondary storage unit provides long-term storage for data, even when the power is turned off. It typically includes hard disk drives (HDDs), solid-state drives (SSDs), optical drives, and other storage devices. The secondary storage unit stores the operating system, software applications, user files, and other data that are not actively used by the CPU.

BASIC STRUCTURE OF COMPUTERS - FUNCTIONAL UNITS OF A COMPUTER

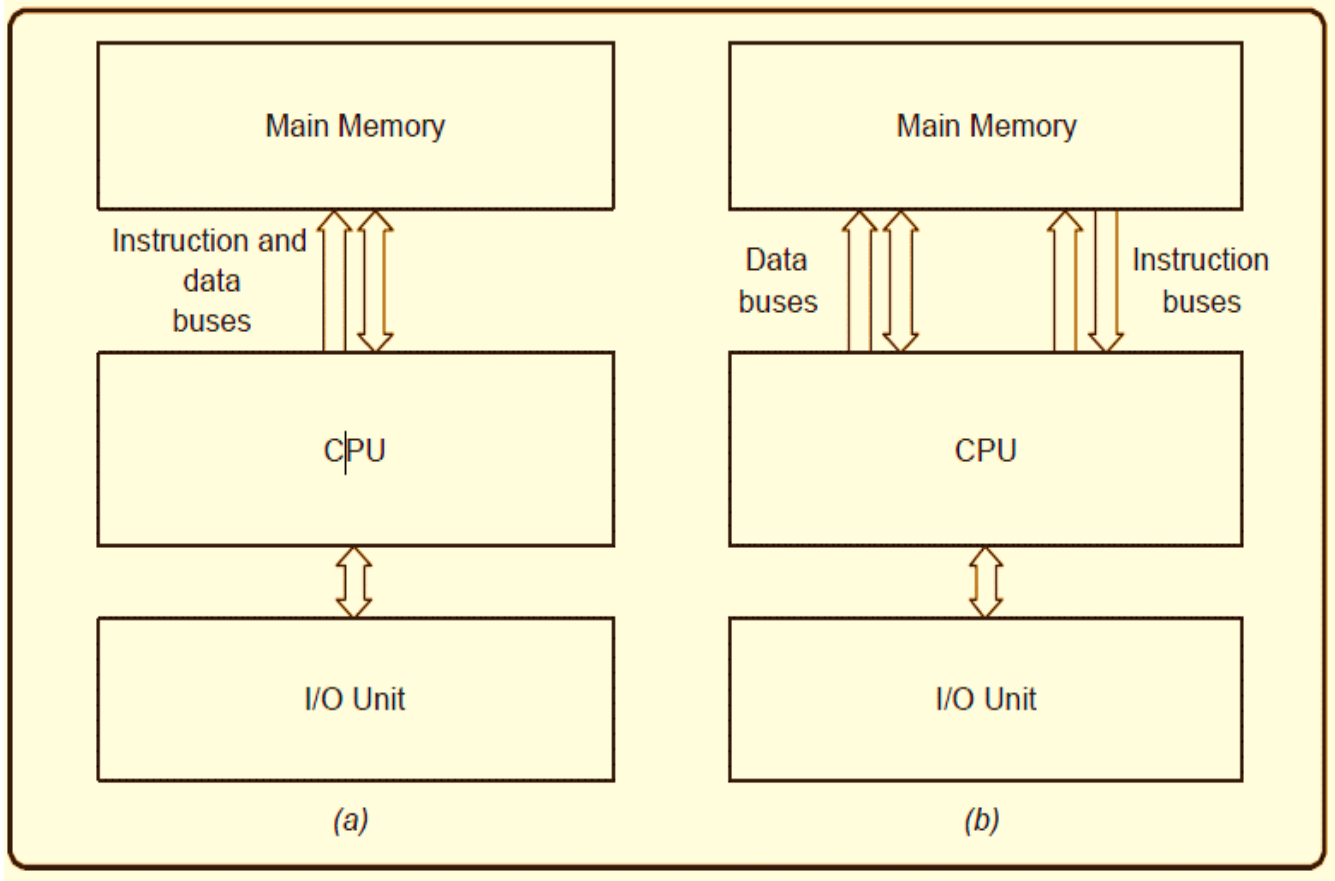
- These function units work together in a computer system, with the CPU as the central processing unit orchestrating the execution of instructions, the ALU performing calculations, the control unit managing instruction flow, the memory unit providing temporary storage, the I/O unit facilitating communication with external devices, registers holding data during processing, the bus system enabling data transfer, and the secondary storage unit providing long-term data storage.
- The coordination and interaction of these function units enable the computer to execute programs, process data, and perform a wide range of tasks.

COMPUTER ARCHITECTURE

- *Computer architecture determines how a computer's components exchange electronic signals to enable input, processing, and output.*
- Computer architecture is defined as the end-to-end structure of a computer system that determines how its components interact with each other in helping execute the machine's purpose (i.e., processing data) often avoiding any reference to the actual technical implementation.
- It specifies the machine interface for which programming languages and associated processors are designed.
- Complex instruction set computer (CISC) and reduced instruction set computer (RISC) are the two predominant approaches to the architecture that influence how computer processors function.

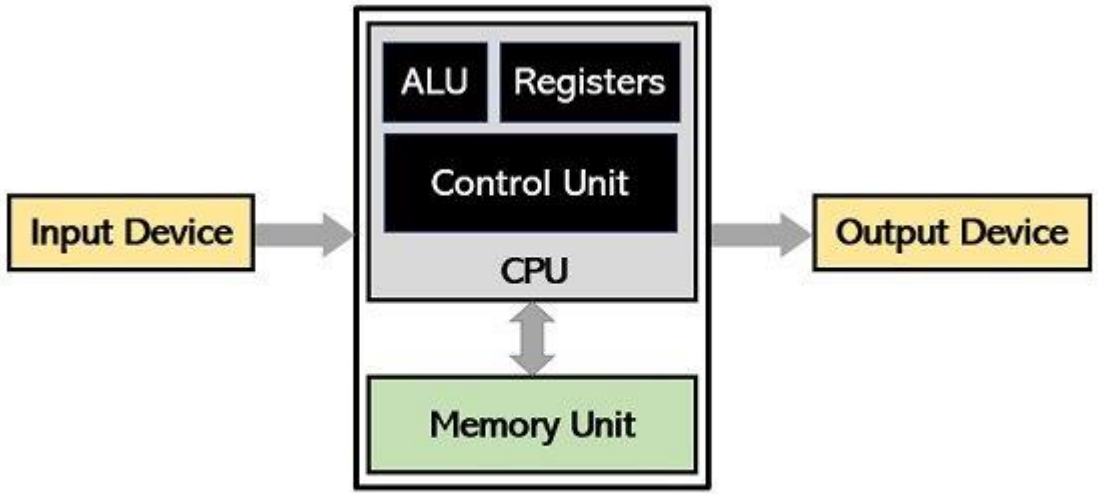
EXAMPLES OF COMPUTER ARCHITECTURE:

- Computer architectures define how a system's memory and processing units interact. Two fundamental architectures used in computing are **Von Neumann Architecture** and **Harvard Architecture**.

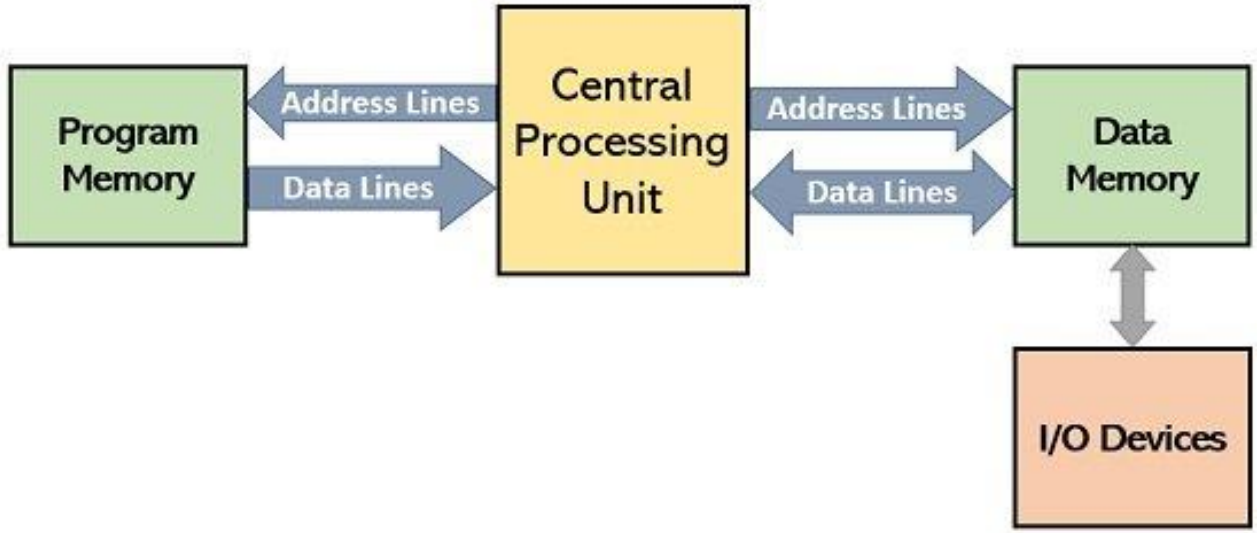


(a) Von Neumann Architecture and (b) Harvard Architecture

VON NEUMANN ARCHITECTURE AND HARVARD ARCHITECTURE



Von Neumann Architecture



Harvard Architecture

EXAMPLES OF COMPUTER ARCHITECTURE:

1. Von Neumann architecture

- The von Neumann architecture, often referred to as the Princeton architecture, is a computer architecture that was established in a 1945 presentation by John von Neumann and his collaborators in the First Draft of a Report on the EDVAC (electronic discrete variable automatic computer).
- This example of computer architecture proposes five components:
  - A processor with connected registers
  - A control unit capable of storing instructions
  - Memory capable of storing information as well as instructions and communicating via buses
  - Additional or external storage
  - Device input as well as output mechanisms

EXAMPLES OF COMPUTER ARCHITECTURE:

2. Harvard architecture

- The Harvard architecture refers to a computer architecture with distinct data and instruction storage and signal pathways.
- In contrast to the von Neumann architecture, in which program instructions and data use the very same memory and pathways, this design separates the two.
- In practice, a customized Harvard architecture with two distinct caches is employed (for data and instruction); X86 and Advanced RISC Machine (ARM) systems frequently employ this instruction.

VON NEUMANN ARCHITECTURE AND HARVARD ARCHITECTURE

- **Von Neumann Architecture:**

➢ **Example: A typical desktop computer running Windows or macOS.**

➢ Key feature: Single memory space for both instructions and data.

➢ Benefit: Flexibility to run various programs with diverse data needs.

- **Harvard Architecture:**

➢ **Example: A microcontroller used in a smart appliance.**

➢ Key feature: Separate memory spaces for instructions and data, allowing simultaneous access.

➢ Benefit: High processing speed for specialized tasks, especially when real-time data processing is required.

COMPARISON OF ARCHITECTURES

VON NEUMANN ARCHITECTURE	HARVARD ARCHITECTURE
It is ancient computer architecture based on stored program computer concept.	It is modern computer architecture based on Harvard Mark I relay based model.
Same physical memory address is used for instructions and data.	Separate physical memory address is used for instructions and data.
There is common bus for data and instruction transfer.	Separate buses are used for transferring data and instruction.
Two clock cycles are required to execute single instruction.	An instruction is executed in a single cycle.
It is cheaper in cost.	It is costly than Von Neumann Architecture.
CPU can not access instructions and read/write at the same time.	CPU can access instructions and read/write at the same time.
It is used in personal computers and small computers.	It is used in micro controllers and signal processing.

BUS

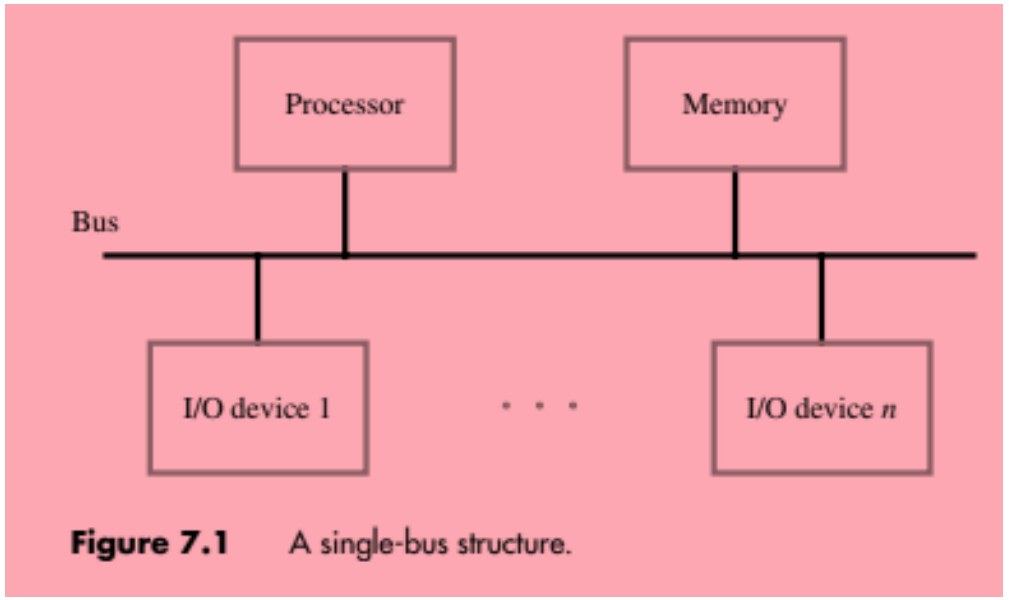


Figure 7.1 A single-bus structure.

- **Bus** is a **subsystem** that is used to **transfer data** and **other information** between devices
- Various devices in computer such as Memory, CPU, I/O etc. are communicate with each other through buses

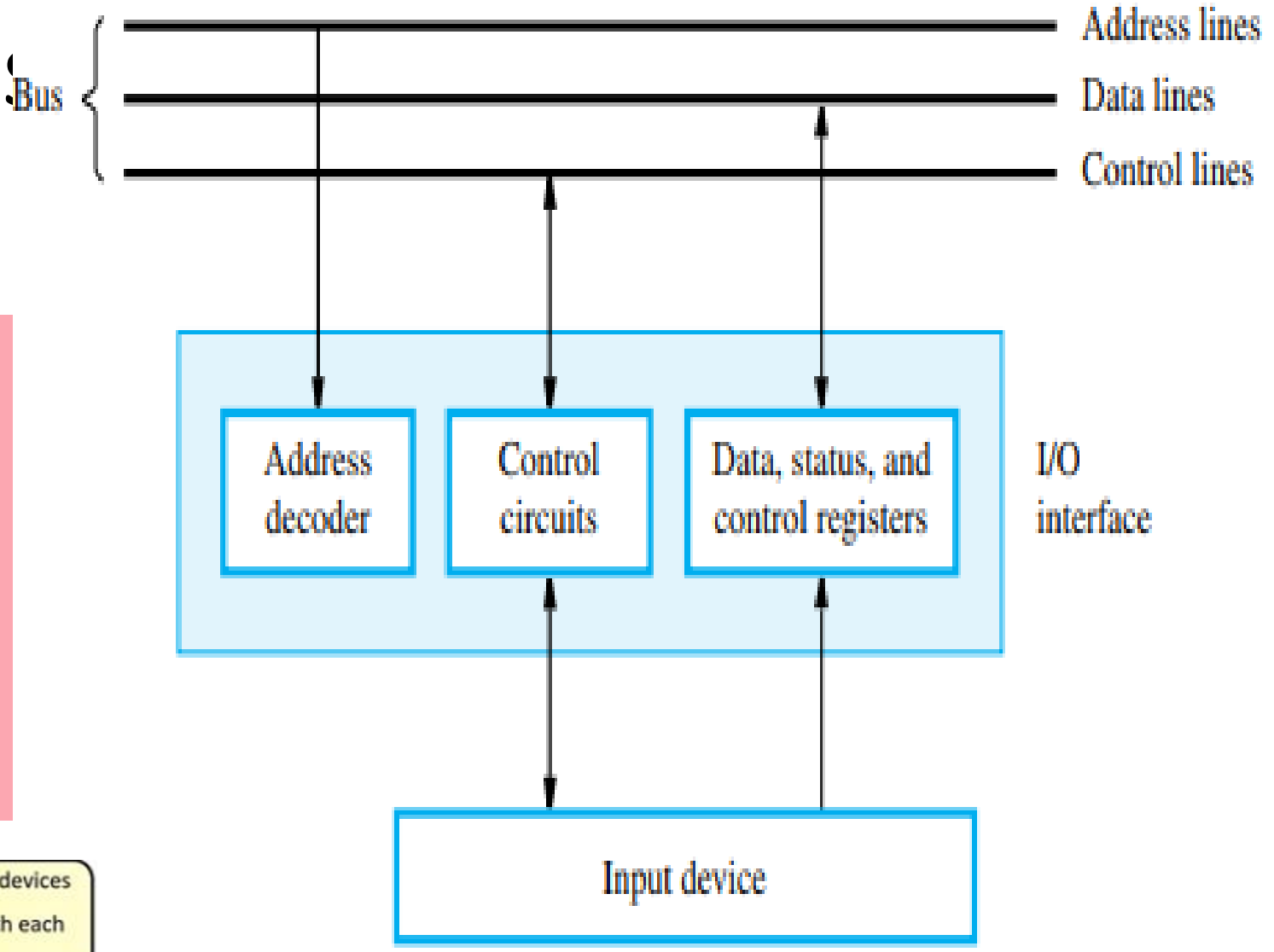
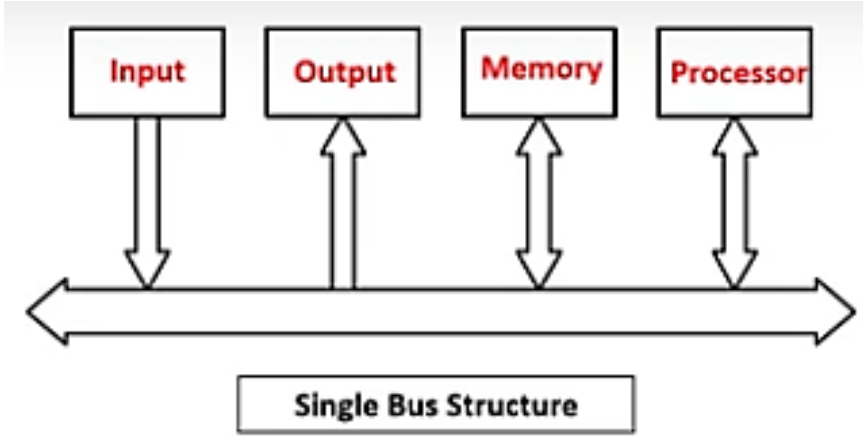


Figure 7.2 I/O interface for an input device.

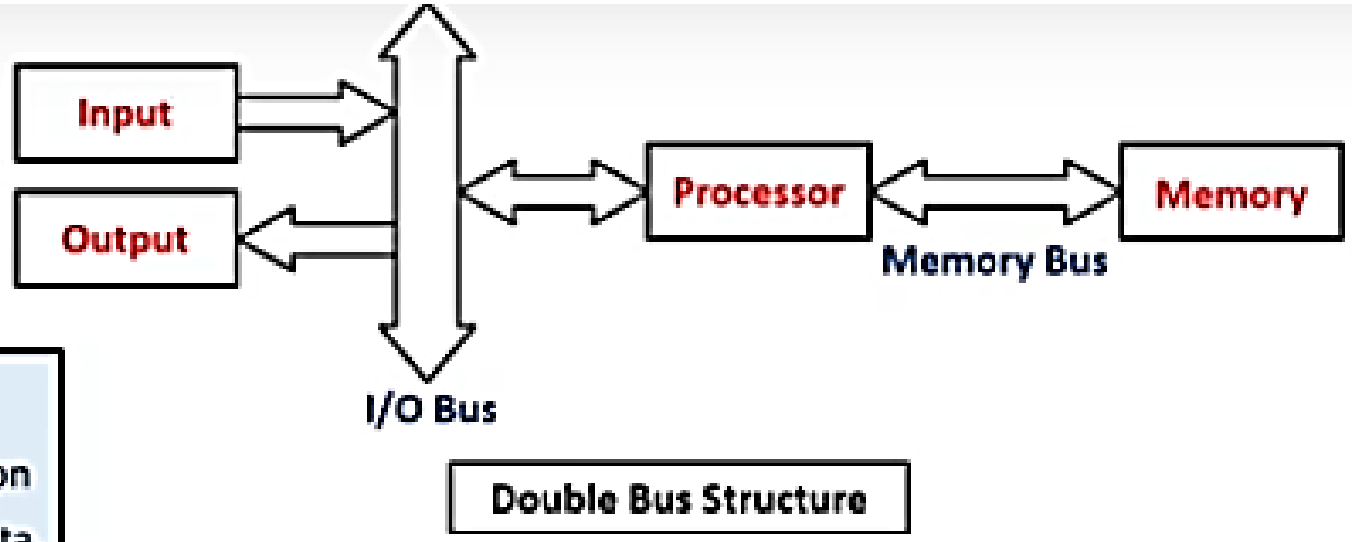
BUS STRUCTURE

Types of Bus Structures:

- Single Bus Structure
- Double Bus Structure
- Multiple Bus Structure

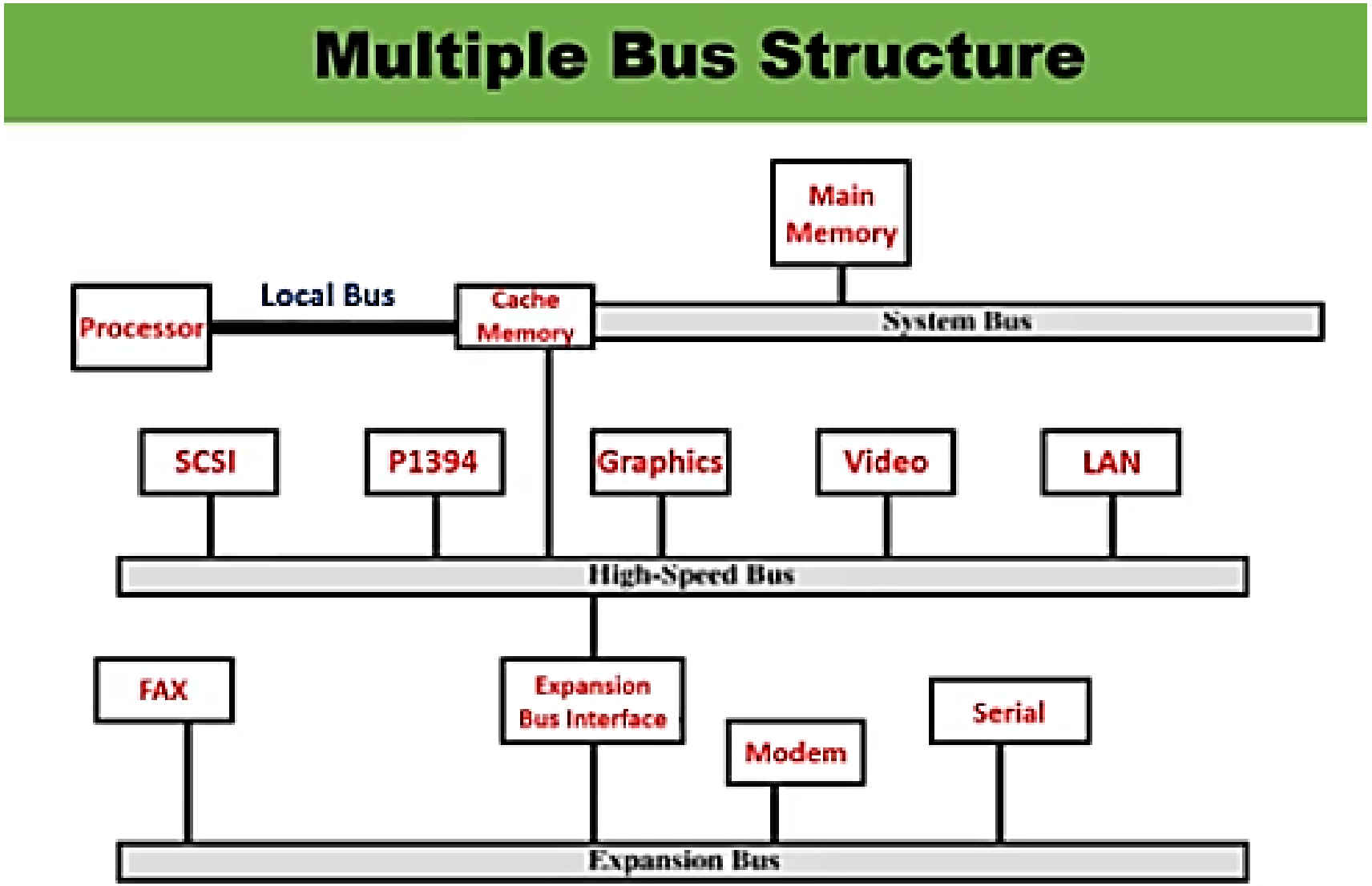


Double bus structure is used to overcome the bottleneck of single bus structure



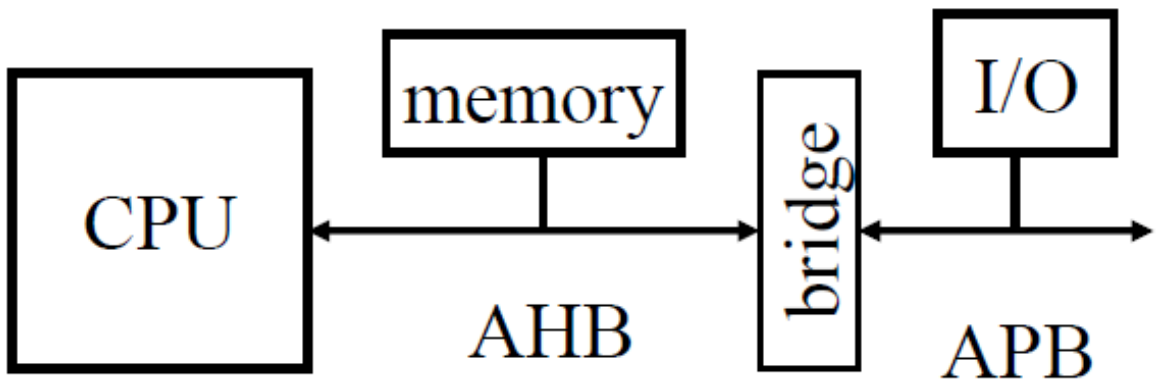
- It uses two buses:
  - one bus is used to fetch instruction
  - other is used to fetch data, required for execution

BUS STRUCTURE



ARM busses

- AMBA:
  - Open standard.
  - Many external devices.
- Two varieties:
  - AMBA High-Performance Bus (AHB).
  - AMBA Peripherals Bus (APB).



BUS STRUCTURE

- The bus consists of three sets of lines used to carry address, data, and control signals.
- I/O device interfaces are connected to these lines, as shown in Figure 7.2 for an input device.
- Each I/O device is assigned a unique set of addresses for the registers in its interface.
- When the processor places a particular address on the address lines, it is examined by the address decoders of all devices on the bus.
- The device that recognizes this address responds to the commands issued on the control lines.
- The processor uses the control lines to request either a Read or a Write operation, and the requested data are transferred over the data lines.
- When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.

Microprocessor	Microcontroller
A silicon chip representing a Central Processing Unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of Instructions	A microcontroller is a highly integrated chip that contains a CPU, scratch pad RAM, Special and General purpose Register Arrays, On Chip ROM/FLASH memory for program storage, Timer and Interrupt control units and dedicated I/O ports
It is a dependent unit. It requires the combination of other chips like Timers, Program and data memory chips, Interrupt controllers etc for functioning	It is a self contained unit and it doesn't require external Interrupt Controller, Timer, UART etc for its functioning
Most of the time general purpose in design and operation	Mostly application oriented or domain specific
Doesn't contain a built in I/O port. The I/O Port functionality needs to be implemented with the help of external Programmable Peripheral Interface Chips like 8255	Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit Port or as individual port pins
Targeted for high end market where performance is important	Targeted for embedded market where performance is not so critical (At present this demarcation is invalid)
Limited power saving options compared to	Includes lot of power saving features

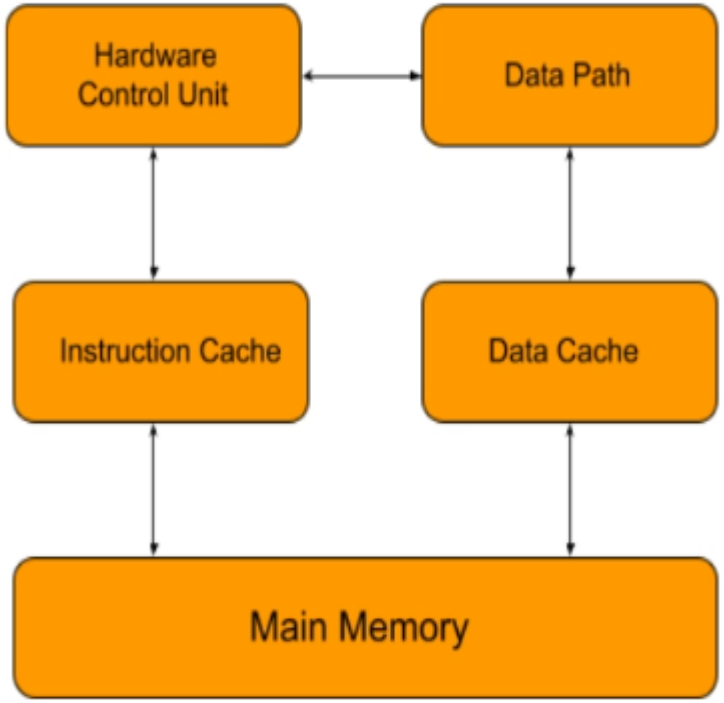
DIFFERENT TYPES OF MICROPROCESSORS

- CISC Microprocessor
- CISC is an acronym for complex instruction set computer.
- This microprocessor is designed to execute complex instructions (a combination of multiple single instructions) which minimize the number of total instructions per program.
- A complex instruction has multiple simple instructions like arithmetic operation, storing in memory, reading from memory, etc.
- The overall length of the program is relatively very small but due to the large size of its instruction set with many addressing modes in a single instruction, it takes multiples machine cycles to execute an instruction.
- Thus it reduces the execution speed of the microprocessor.

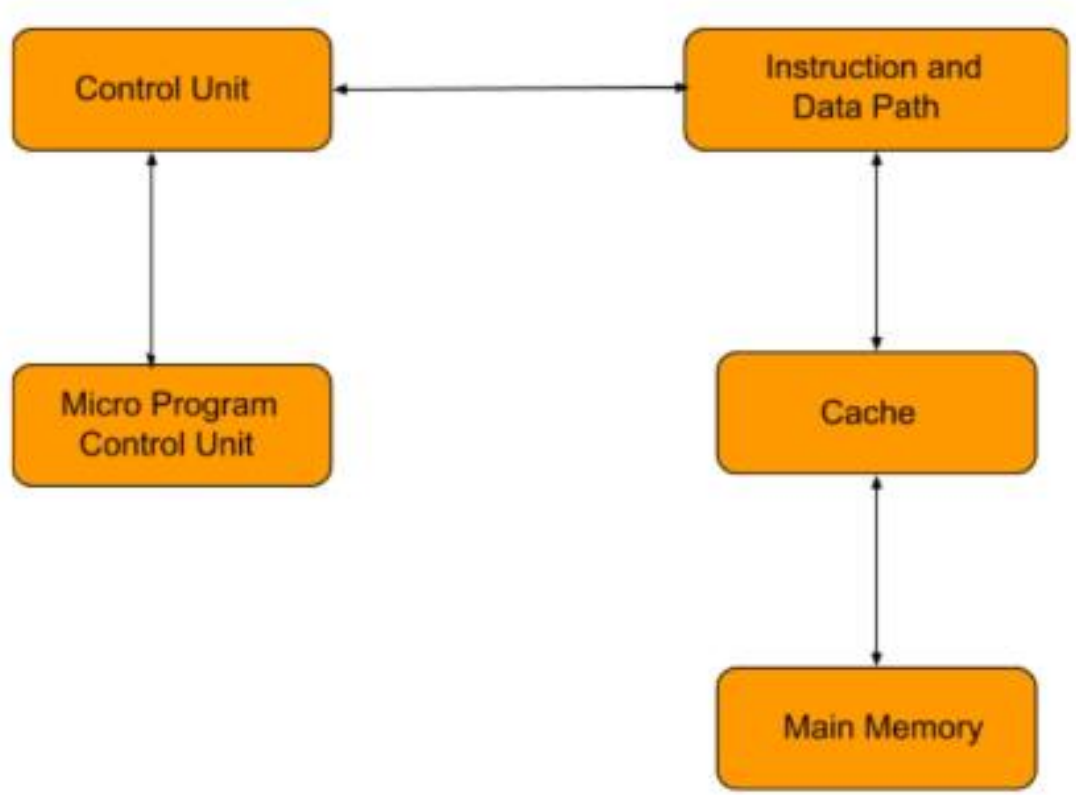
DIFFERENT TYPES OF MICROPROCESSORS

- RISC Microprocessor
- RISC stands for Reduced Instruction Set Computer.
- This type of microprocessor is based on an architecture designed to execute simple instructions.
- The instructions are simplified to reduce the execution time.
- The instruction is usually completed in one clock cycle by using pipelining to execute parts of instruction in parallel.
- Therefore, it requires program code with more lines and more memory to store instruction.
- This architecture offers to execute program efficiently and quickly.

RISC AND CISC



RISC



CISC

Features of CISC Microprocessor:

- The instructions are complex.
- The numbers of instructions are relatively more than RISC microprocessor almost 100 to 200.
- It takes more than four cycles to somewhere around 120 to complete complex instructions.
- The program is executed sequentially thus no feature of pipelining (parallel execution).
- The instructions are executed by micro program where the complexity lies.
- The instruction format and size may vary as opposed to RISC fixed instructions.
- The program code in CISC is simple and short thus it uses less memory or RAM.
- It put emphasis on the hardware and less on the software or programming.

Features of RISC Microprocessor:

- The instructions in RISC microprocessors are simple.
- As suggested by its name, the numbers of instructions are reduced to between 30 and 40.
- The instructions are simple thus it takes only one machine cycle to complete.
- Pipelining (parallel execution) is fairly easy in a RISC microprocessor.
- The format and size of instructions is limited and fixed.
- Due to the low number of instructions, the program code lengthy and require more memory.
- It put emphasis on the software or compiler and less on the load on the hardware.

Feature	RISC	CISC
Instruction Set	Reduced and simple instruction set	Rich and complex instruction set
Instruction Length	Fixed-length instructions	Variable-length instructions
Addressing Modes	Limited addressing modes	Wide range of addressing modes
Instruction Encoding	Simple encoding schemes	Complex encoding schemes
Pipeline Efficiency	Emphasizes pipelining and parallelism	Less emphasis on pipelining and parallelism
Execution Speed	Typically faster due to simpler design	May be slower due to more complex instructions
Memory Access	Registers heavily used for operations	Supports direct memory access and complex modes
Hardware Complexity	Reduced hardware complexity	Higher hardware complexity due to microcoding
Power Consumption	Generally lower power consumption	May consume more power due to complexity
Backward Compatibility	Less emphasis on backward compatibility	Often maintains backward compatibility with older architectures

PERFORMANCE OF THE PROCESSOR

- Integration: Microprocessors are highly integrated electronic devices that combine the functions of a CPU on a single chip. They contain millions to billions of transistors, which allow for the execution of complex instructions and calculations.

- Processing Power: Microprocessors are designed to perform various tasks by executing instructions and manipulating data. They can perform calculations, process data, control devices, and run software applications, making them the heart of modern computing devices.

- Architecture: Microprocessors follow a specific architecture, such as the x86, ARM, or MIPS architectures, which determines the way instructions are processed and data is stored. Different architectures have varying performance, power consumption, and instruction sets.

PERFORMANCE OF THE PROCESSOR

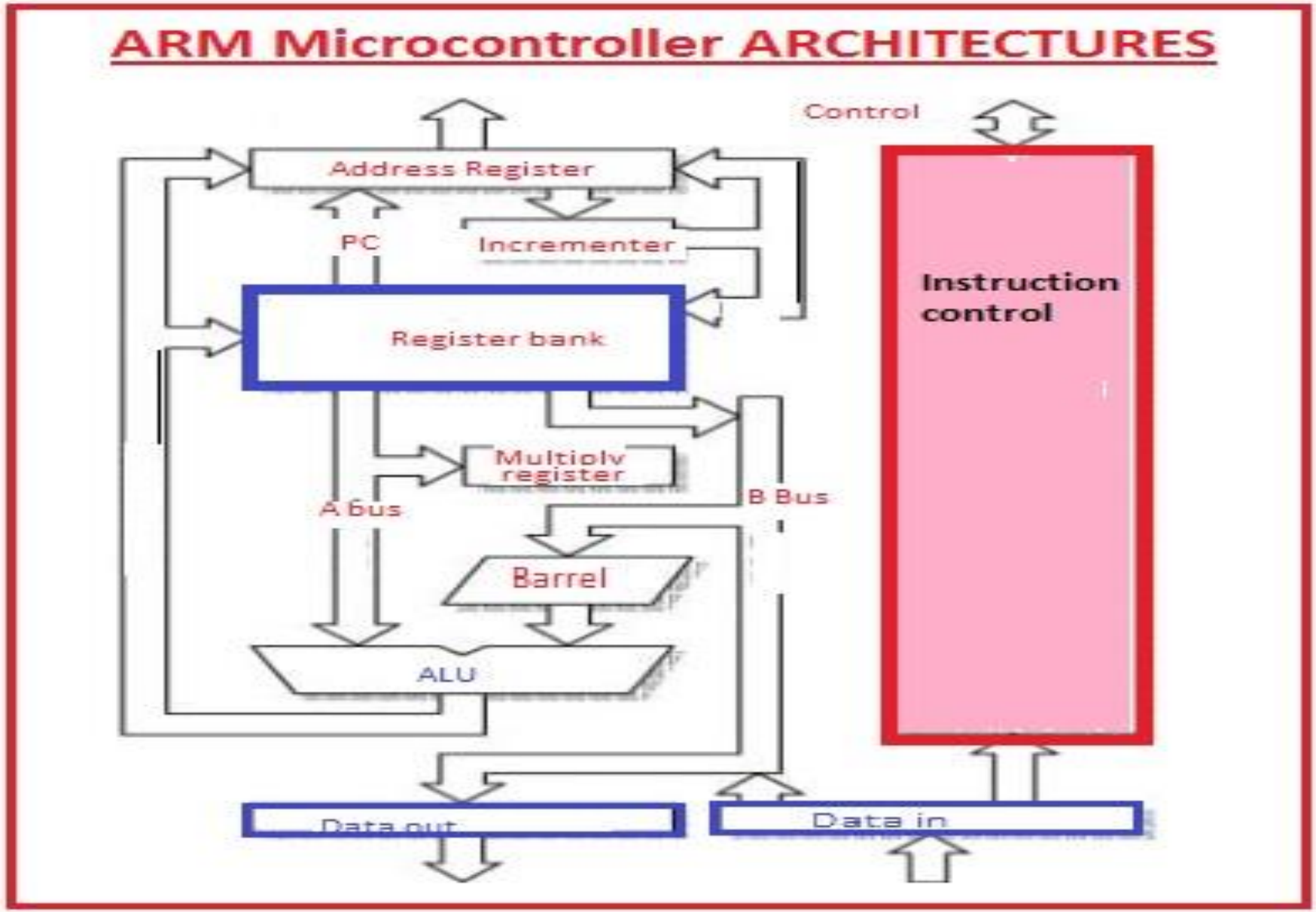
- Clock Speed: Microprocessors operate at a specific clock speed, measured in hertz (Hz). The clock speed determines the number of instructions the processor can execute per second. Higher clock speeds generally result in faster processing, but other factors like architecture and efficiency also play a role.
- Multiple Cores: Many modern microprocessors have multiple cores, which are individual processing units within a single chip. Each core can execute instructions independently, allowing for parallel processing and improved multitasking capabilities.
- Application Diversity: Microprocessors are used in a wide range of devices, including computers, smartphones, tablets, game consoles, embedded systems, and various electronic appliances. They enable the functioning of these devices by providing the necessary computing power and control.

Introduction to ARM Microcontroller

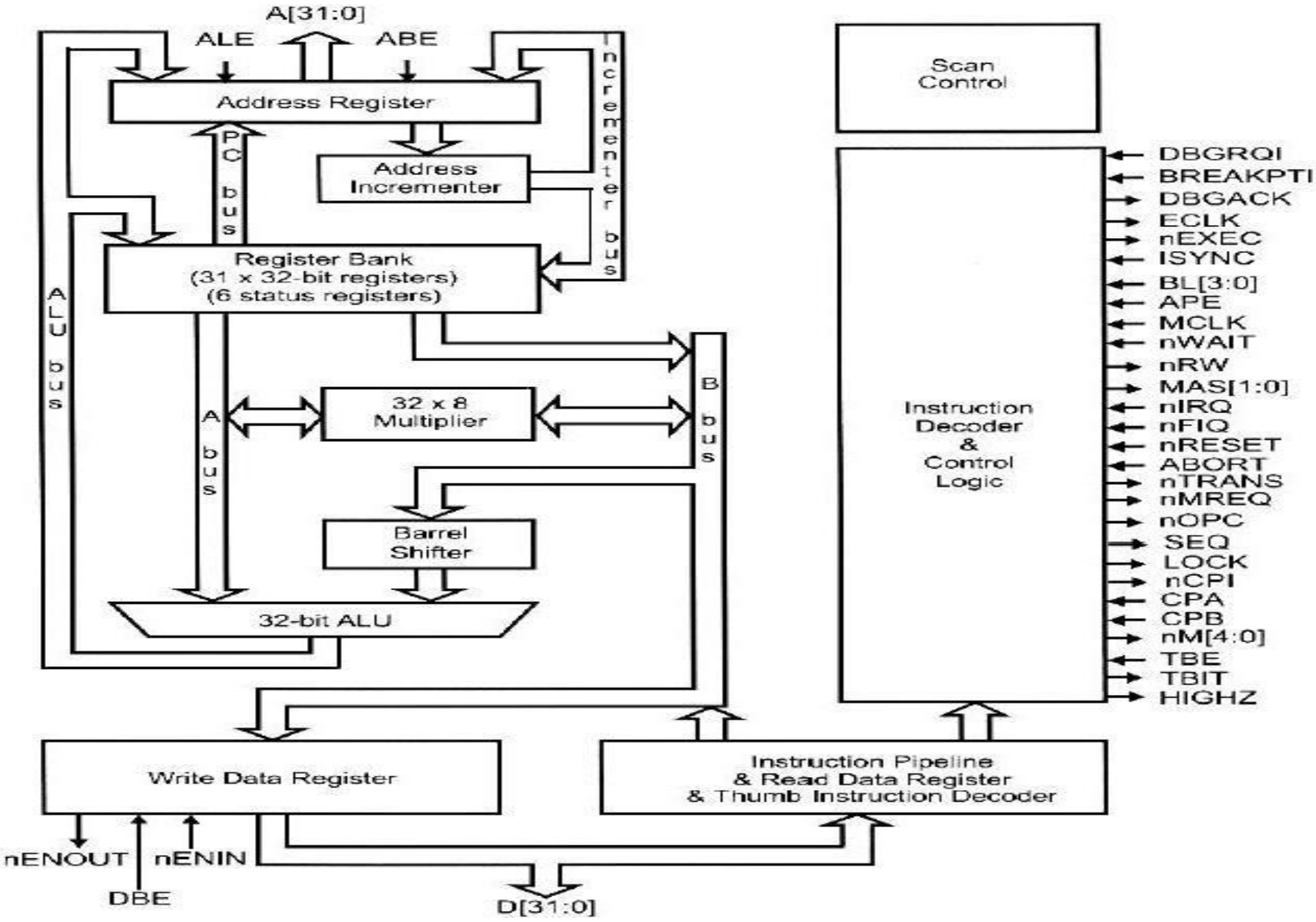
- In ARM controller **RISC** load-store architecture is exists in this board.
- Different types of registers are used in this board which helps in the manipulation of memory.
- The set of instructions is used in the board but the main function is to decrease the time required by every instruction.
- The processor used in the ARM controller is cortex M3 which is a high speed and thirty-two bit, and it offered numerous features to the users.
- The architecture of this board is Harvard architecture has distinct data and instruction buses to transmit data to the random access memory and read-only memory.
- For **execution, fetching, decoding**, different types of commands three-stage pipeline is used.
- The processor of this board uses thumb commands based on the thumb two techniques, so it decreases the memory needed for the program and makes sure a higher density of coding.
- As this model comprises thirty two-bit architecture which provides better performance of the execution of commands.

NXP’s LPC-2148 microcontroller IC in it which belongs to ARM7 family. It is a high performance ARM7 TDMI-S based 32-bit RISC Microcontroller. This board is a good choice for beginners and also can be used in high end applications because of its inbuilt peripherals.

Introduction to ARM Microcontroller



ARM Core Diagram

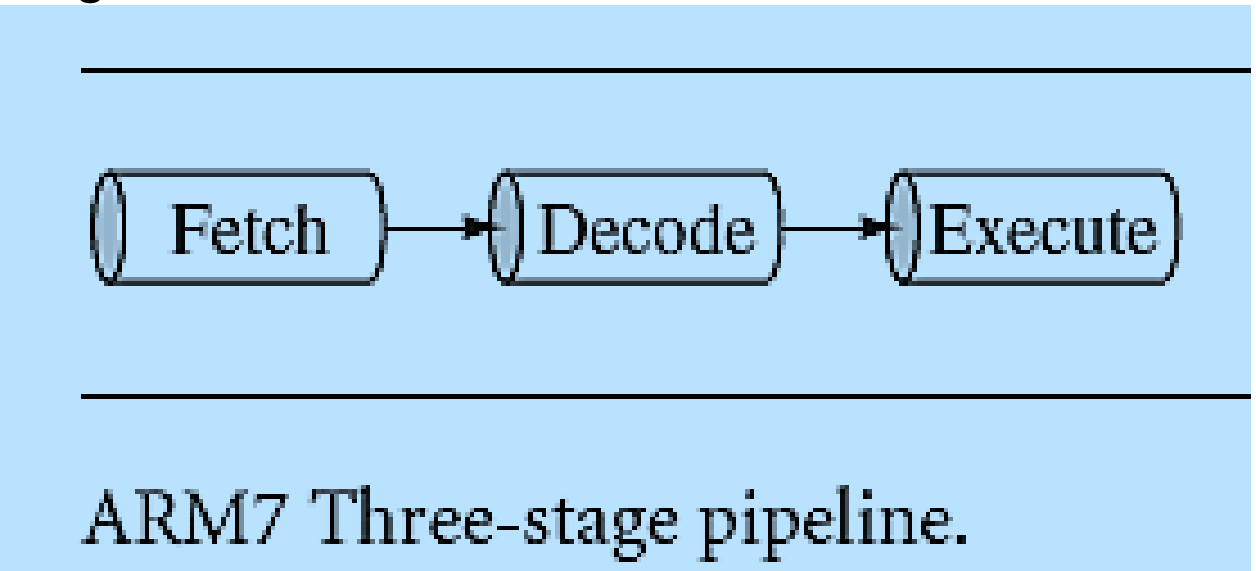


Introduction to ARM Microcontroller

- Some important features of this controller which are described here with the details:
  - ➢ This board comprises of thirty two bit central processing unit which is high speed.
  - ➢ It comprises of the three-stage pipeline.
  - ➢ This board uses the thumb 2 technique.
  - ➢ This module is compatible with the different types of tools and RTOS.
  - ➢ It is compatible with the sleep mode of operation.
  - ➢ It has the ability to control different types of software

ARM Pipeline

- A pipeline is the mechanism a RISC processor uses to execute instructions. Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed.

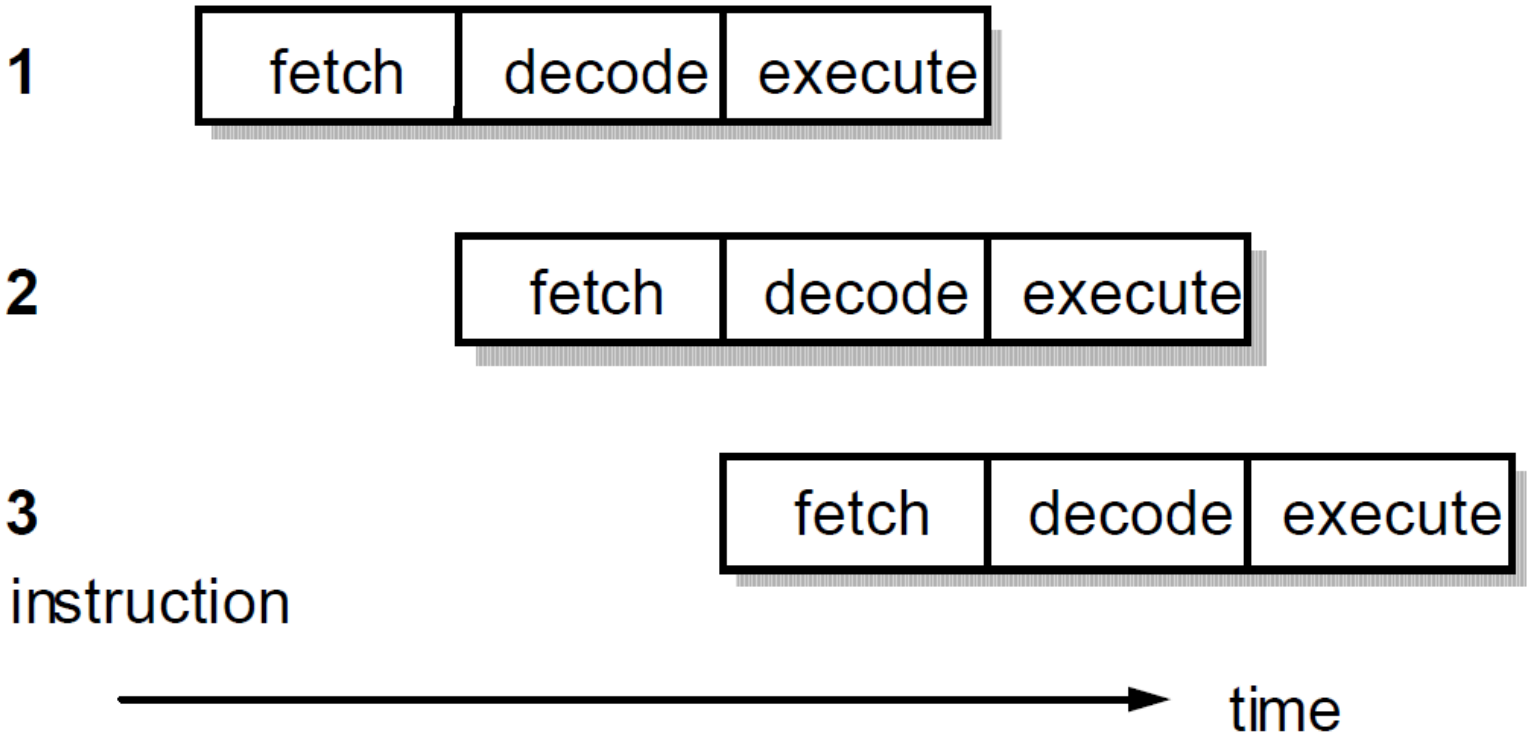


ARM7 Three-stage pipeline.

- ***Fetch*** loads an instruction from memory.
- ***Decode*** identifies the instruction to be executed.
- ***Execute*** processes the instruction and writes the result back to a register.

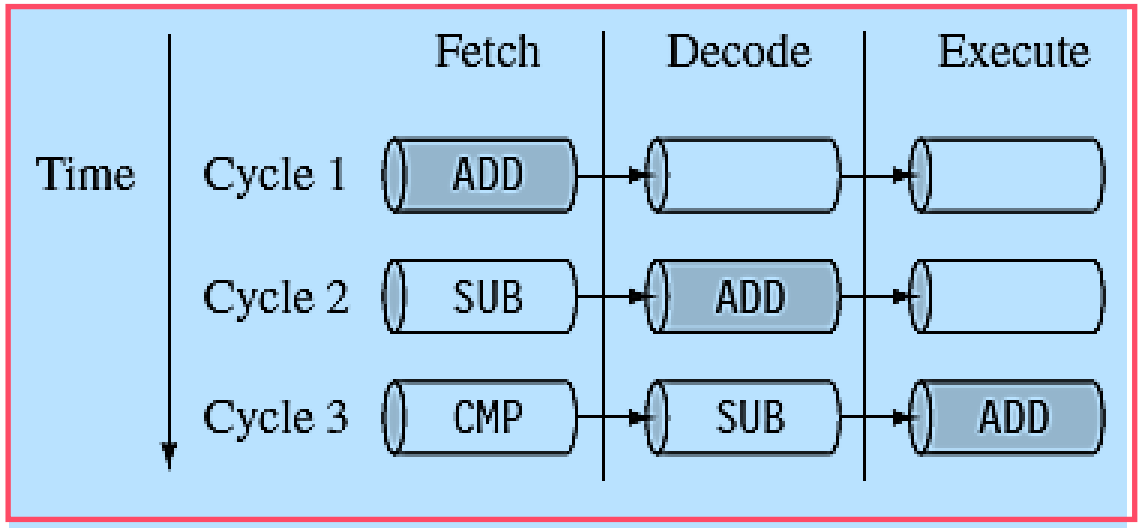
ARM Pipeline

ARM single-cycle instruction 3-stage pipeline operation



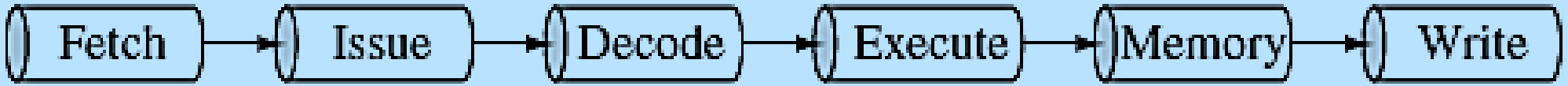
ARM Pipeline

- As the pipeline length increases, the amount of work done at each stage is reduced, which allows the processor to attain a higher operating frequency. This in turn increases the performance.
- The system latency also increases because it takes more cycles to fill the pipeline before the core can execute an instruction.
- The increased pipeline length also means there can be data dependency between certain stages. You can write code to reduce this dependency by using instruction scheduling.
- Pipelined instruction sequence.



ARM Pipeline

ARM9 five-stage pipeline.



ARM10 six-stage pipeline.

ARM Pipeline

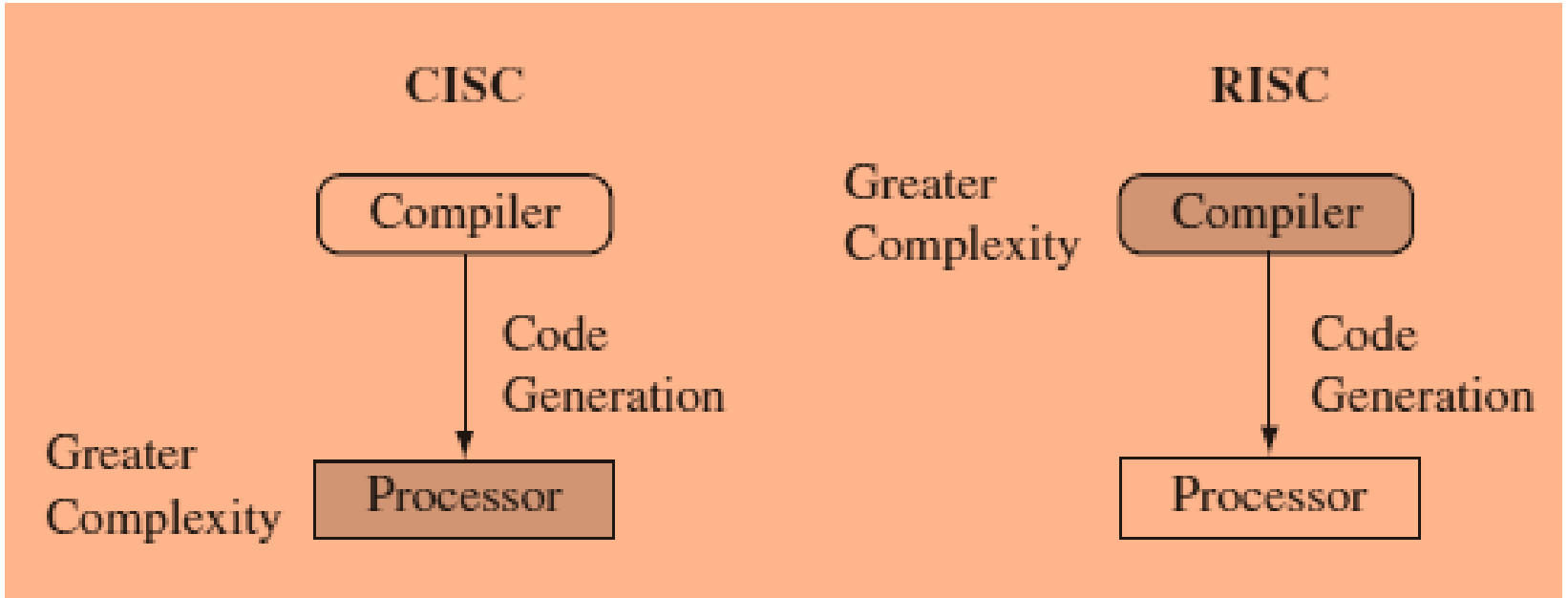
- The pipeline design for each ARM family differs. For example, The ARM9 core increases the pipeline length to five stages.
- The ARM9 adds a memory and writeback stage, which allows the ARM9 to process on average **1.1 million instructions per second for every MHz of clock speed**

—an increase in instruction throughput by around 13% compared with an ARM7.

- The maximum core frequency attainable using an ARM9 is also higher.
- The ARM10 increases the pipeline length still further by adding a sixth stage.
- The ARM10 can process on average **1.3 million instructions per second per MHz of clock speed**, about 34% more throughput than an ARM7 processor core, but again at a higher latency cost.
- Even though the ARM9 and ARM10 pipelines are different, they still use the same *pipeline executing characteristics* as an ARM7.
- Code written for the ARM7 will execute on an ARM9 or ARM10.

The RISC Design Philosophy

- The ARM core uses a RISC architecture. RISC is a design philosophy aimed at delivering simple but powerful instructions that execute within a single cycle at a high clock speed.
- The RISC philosophy concentrates on reducing the complexity of instructions performed by the hardware because it is easier to provide greater flexibility and intelligence in software rather than hardware. As a result, a RISC design places greater demands on the compiler.
- In contrast, the traditional complex instruction set computer (CISC) relies more on the hardware for instruction functionality, and consequently the CISC instructions are more complicated.



The RISC Design Philosophy

- The RISC philosophy is implemented with four major design rules:

1. **Instructions**—RISC processors have a reduced number of instruction classes. These classes provide simple operations that can each execute in a single cycle. The compiler or programmer synthesizes complicated operations (for example, a divide operation) by combining several simple instructions. Each instruction is a fixed length to allow the pipeline to fetch future instructions before decoding the current instruction. In contrast, in CISC processors the instructions are often of variable size and take many cycles to execute.

2. **Pipelines**—The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines. Ideally the pipeline advances by one step on each cycle for maximum throughput. Instructions can be decoded in one pipeline stage. There is no need for an instruction to be executed by a miniprogram called microcode as on CISC processors.

The RISC Design Philosophy

3. **Registers**—RISC machines have a large general-purpose register set. Any register can contain either data or an address. Registers act as the fast local memory store for all data processing operations. In contrast, CISC processors have dedicated registers for specific purposes.

4. **Load-store architecture**—The processor operates on data held in registers. Separate load and store instructions transfer data between the register bank and external memory. Memory accesses are costly, so separating memory accesses from data processing provides an advantage because you can use data items held in the register bank multiple times without needing multiple memory accesses. In contrast, with a CISC design the data processing operations can act on memory directly.

The ARM Design Philosophy

- The ARM processor has been specifically designed to be small to reduce power consumption and extend battery operation—essential for applications such as mobile phones and personal digital assistants (PDAs).
- High code density is another major requirement since embedded systems have limited memory due to cost and/or physical size restrictions. High code density is useful for applications that have limited on-board memory, such as mobile phones and mass storage devices.
- Embedded systems are price sensitive and use slow and low-cost memory devices. For high-volume applications like digital cameras, every cent has to be accounted for in the design. The ability to use low-cost memory devices produces substantial savings.

The ARM Design Philosophy

- Another important requirement is to reduce the area of the die taken up by the embedded processor. For a single-chip solution, the smaller the area used by the embedded processor, the more available space for specialized peripherals. This in turn reduces the cost of the design and manufacturing since fewer discrete chips are required for the end product.
- ARM has incorporated hardware debug technology within the processor so that software engineers can view what is happening while the processor is executing code. With greater visibility, software engineers can resolve issues faster, which has a direct effect on the time to market and reduces overall development costs.
- The ARM core is not a pure RISC architecture because of the constraints of its primary application—the embedded system.

Instruction Set for Embedded Systems

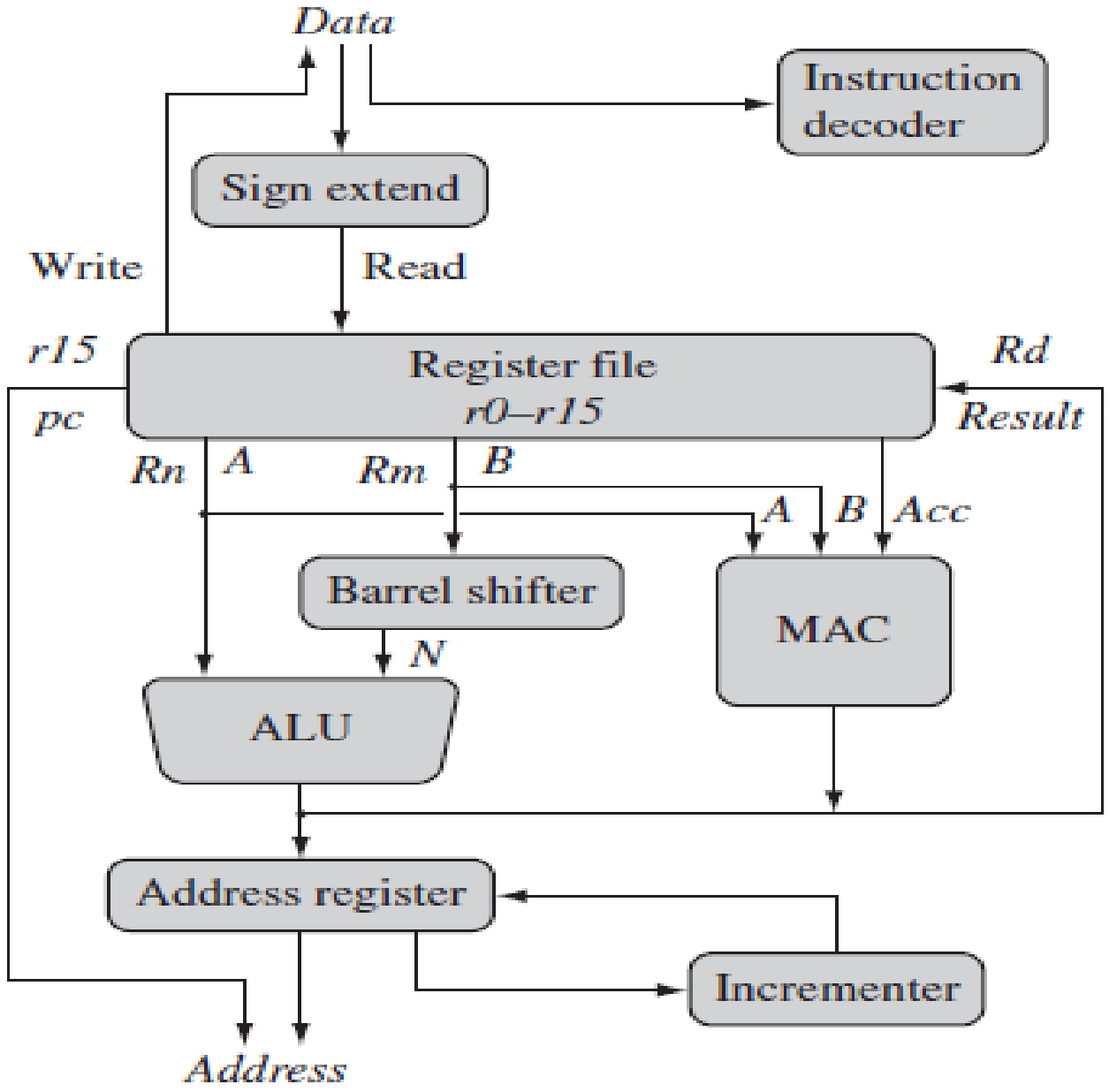
- The ARM instruction set differs from the pure RISC definition in several ways that make the ARM instruction set suitable for embedded applications:
- **Variable cycle execution for certain instructions** — Not every ARM instruction executes in a single cycle.
- For example, load-store-multiple instructions vary in the number of execution cycles depending upon the number of registers being transferred.
- The transfer can occur on sequential memory addresses, which increases performance since sequential memory accesses are often faster than random accesses.
- Code density is also improved since multiple register transfers are common operations at the start and end of functions.

Instruction Set for Embedded Systems

- **Inline barrel shifter leading to more complex instructions** — The inline barrel shifter is a hardware component that pre-processes one of the input registers before it is used by an instruction. This expands the capability of many instructions to improve core performance and code density.
- **Thumb 16-bit instruction set** — ARM enhanced the processor core by adding a second 16-bit instruction set called Thumb that permits the ARM core to execute either 16- or 32-bit instructions. The 16-bit instructions improve code density by about 30% over 32-bit fixed-length instructions.
- **Conditional execution** — An instruction is only executed when a specific condition has been satisfied. This feature improves performance and code density by reducing branch instructions.
- **Enhanced instructions**—The enhanced digital signal processor (DSP) instructions were added to the standard ARM instruction set to support fast 16×16-bit multiplier operations and saturation. These instructions allow a faster-performing ARM processor in some cases to replace the traditional combinations of a processor plus a DSP.

ARM core Dataflow model

- Data enters the processor core through the *Data* bus. The data may be an instruction to execute or a data item.
- Figure shows a Von Neumann implementation of the ARM— data items and instructions share the same bus.
- In contrast, Harvard implementations of the ARM use two different buses.
- The instruction decoder translates instructions before they are executed.
- Each instruction executed belongs to a particular instruction set.



ARM core Dataflow model

- **The ARM processor, like all RISC processors, uses a *load-store architecture*.**
- This means it has two instruction types for transferring data in and out of the processor: **load instructions copy data from memory to registers in the core**, and conversely **the store instructions copy data from registers to memory**.
- There are no data processing instructions that directly manipulate data in memory.
- Thus, data processing is carried out solely in registers.
- Data items are placed in the *register file*—a storage bank made up of 32-bit registers.
- Since the ARM core is a 32-bit processor, most instructions treat the registers as holding signed or unsigned 32-bit values.
- The sign extend hardware converts signed 8-bit and 16-bit numbers to 32-bit values as they are read from memory and placed in a register.

ARM core Dataflow model

- ARM instructions typically have two source registers, *Rn* and *Rm*, and a single result or destination register, *Rd*.
- Source operands are read from the register file using the internal buses *A* and *B*, respectively.
- The ALU (arithmetic logic unit) or MAC (multiply-accumulate unit) takes the register values *Rn* and *Rm* from the *A* and *B* buses and computes a result.
- Data processing instructions write the result in *Rd* directly to the register file.
- Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the *Address* bus.
- One important feature of the ARM is that register Rm alternatively can be pre processed in the barrel shifter before it enters the ALU.
- Together the barrel shifter and ALU can calculate a wide range of expressions and addresses.

ARM core Dataflow model

- After passing through the functional units, the result in *Rd* is written back to the register file using the *Result* bus.
- For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location.
- The processor continues executing instructions until an exception or interrupt changes the normal execution flow.

Programming Model

- ARM Data Types
- ARM Processor Modes
- ARM Operating States
- ARM Registers

ARM Data Type

- ARM processor support the following data types
  - Byte: 8 bits
  - Halfword: 16 bits
    - Halfwords must be aligned to two-byte boundaries
  - Word: 32 bits
    - Words must be aligned to four-byte boundaries

ARM Data Type

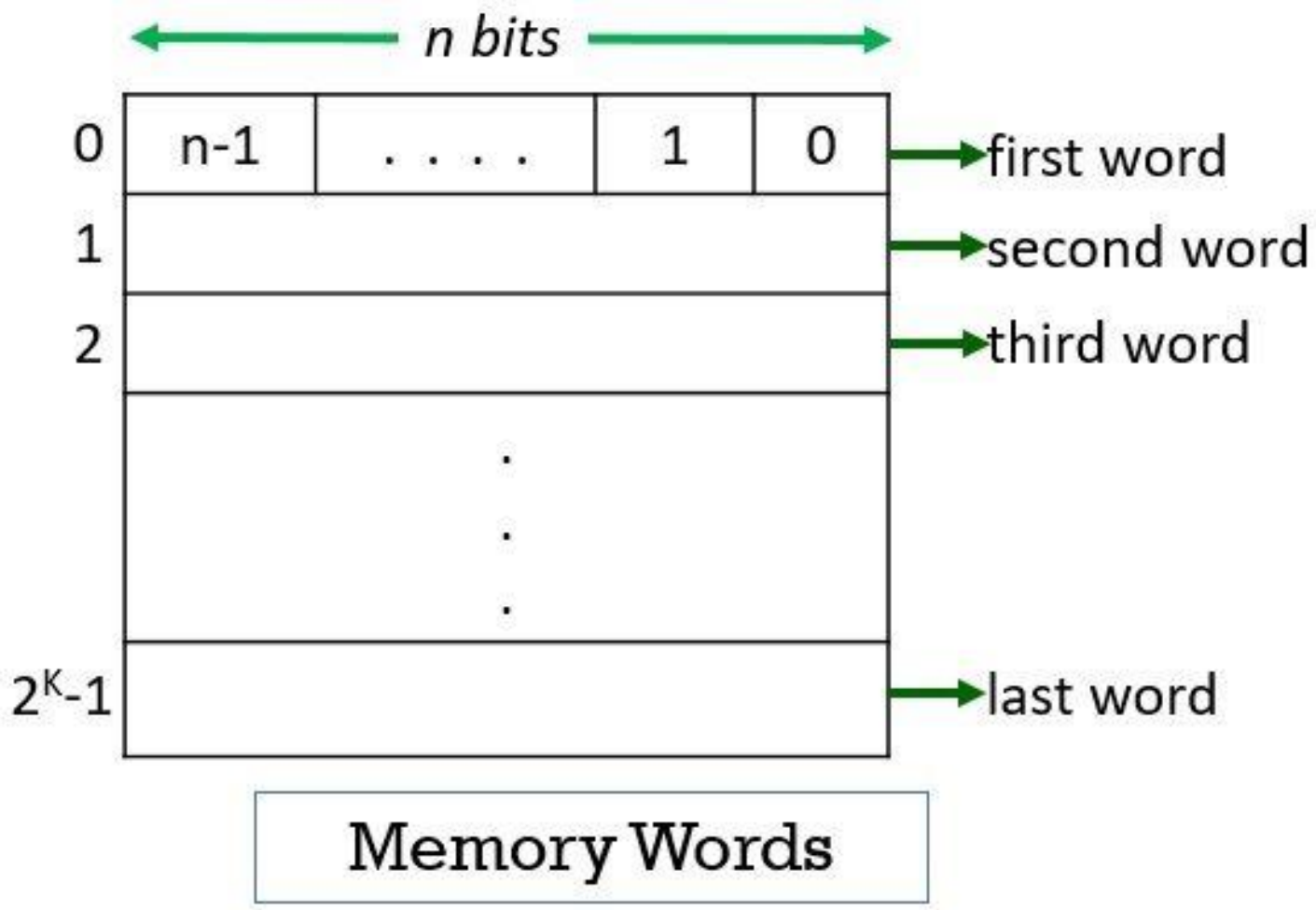
- ARM7 primarily deals with integer-based data types, which can be of different sizes.

Data Type	Size	Registers Used	Instruction Used
Byte (8-bit)	1 byte	R0 to R15 (LSB used)	LDRB, STRB
Halfword (16-bit)	2 bytes	R0 to R15 (lower 16 bits)	LDRH, STRH
Word (32-bit)	4 bytes	R0 to R15	LDR, STR
**Doubleword (64-bit, only in ARMv5+)	8 bytes	Two registers (R0-R1)	LDRD, STRD

ARM Data Type (Cont.)

- ARM allow addresses to be 32 bits long
  - An address refer to a byte, not a word
  - Word 0 is at location 0
  - Word 1 is at location 4
  - PC is incremented by 4 in sequential access
- Can be configured at power-up to address the bytes in a word in either *little-endian* or *bit-endian* mode

Memory location and addresses



- The group of n bit is termed as word where n is termed as the word length.
- The word length of the computer has evolved from 8, 16, 24, 32 to 64 bits.
- General-purpose computers nowadays have **32 to 64 bits**. The group of 8 bit is called a byte.
- The memory locations are addressed from 0 to 2^K-1 i.e. a memory has 2^K addressable locations. And thus the address space of the computer has 2^K addresses.
- 2^{10} = 1024 = 1K (Kilobyte)
  2^{20} = 1,048,576 = 1M (Megabyte)
  2^{30} = 1073741824 = 1G (Gigabyte)
  2^{40} = 1.0995116e+12 = 1T (Terabyte)

Memory location and addresses

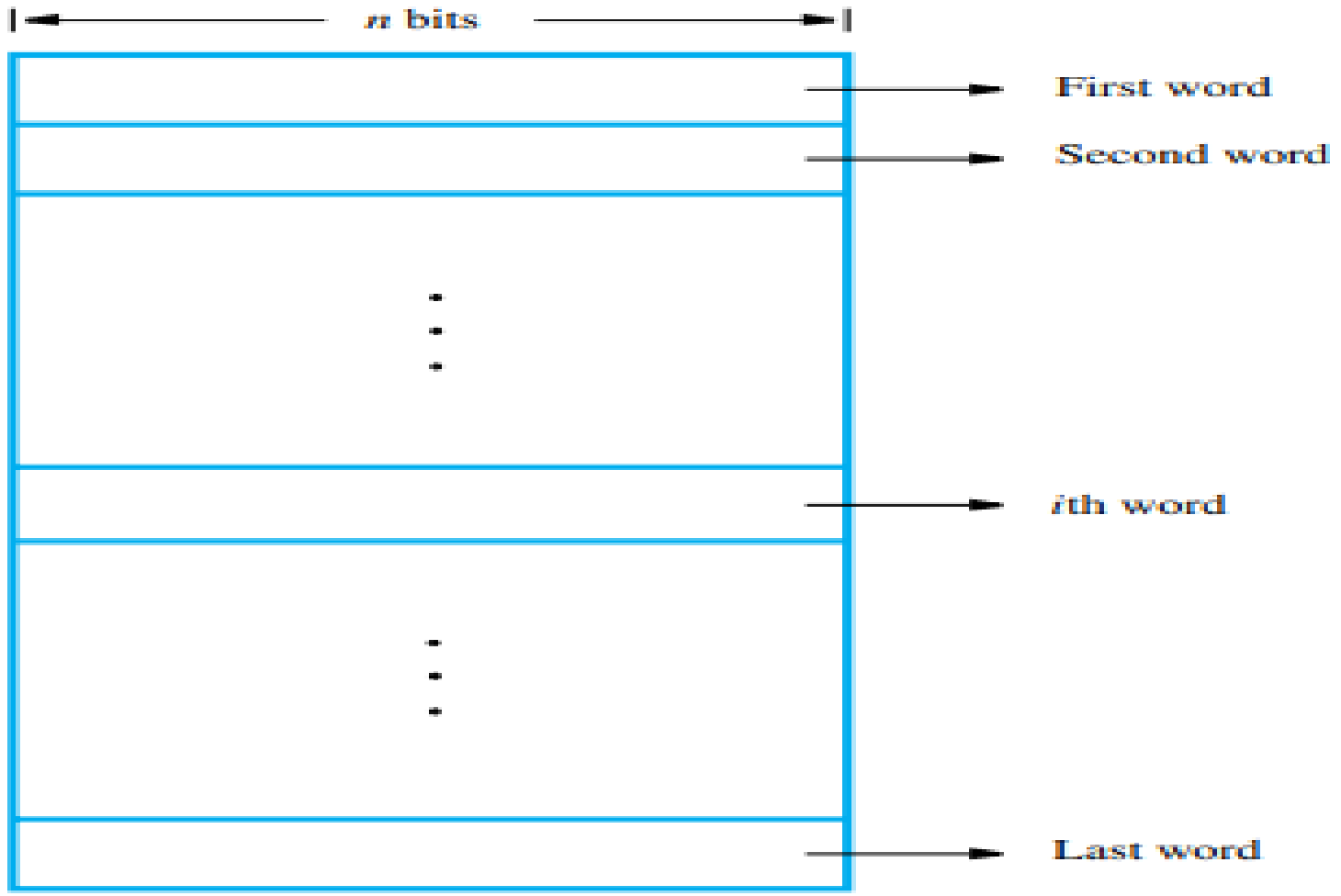


Figure 2.1 Memory words.

Memory location and addresses

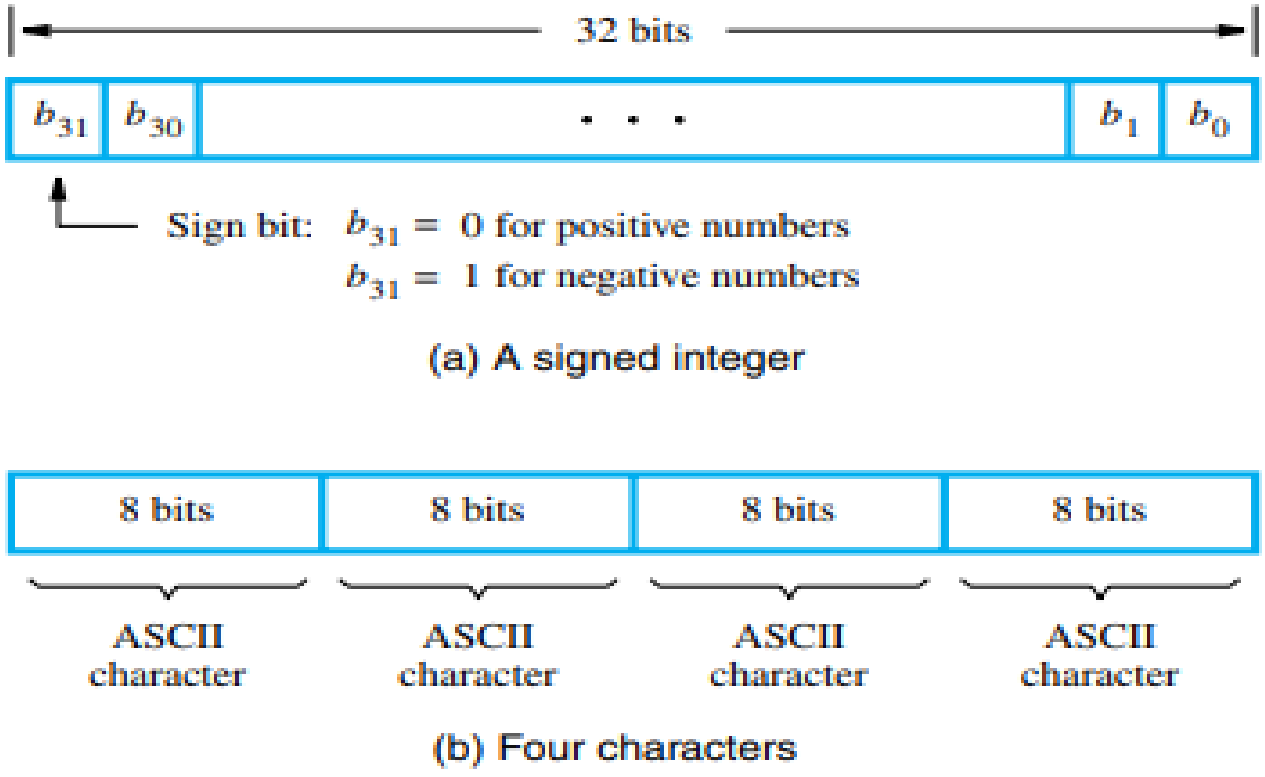
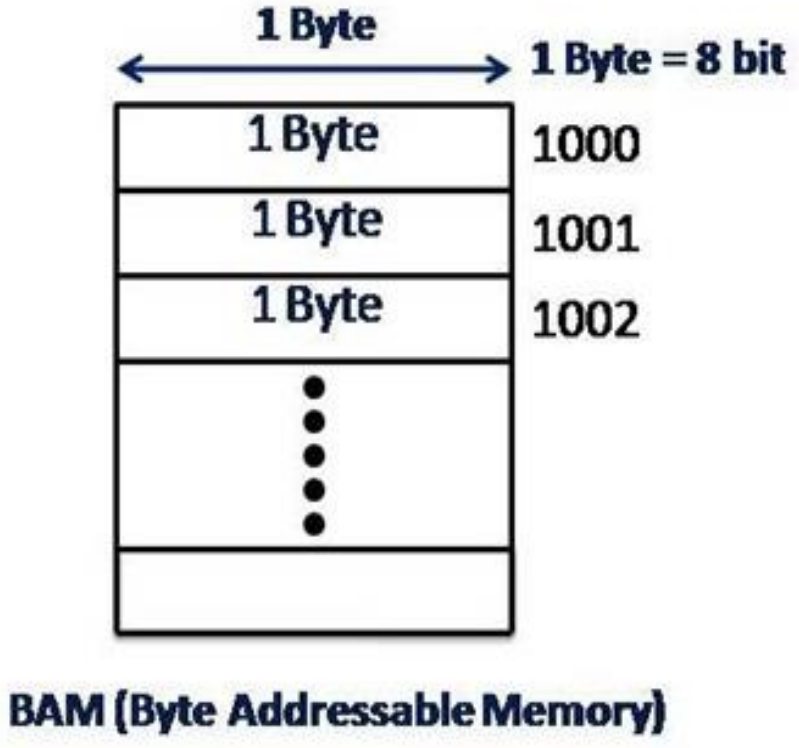


Figure 2.2 Examples of encoded information in a 32-bit word.

- Modern computers have word lengths that typically range from 16 to 64 bits.
- If the word length of a computer is 32 bits, a single word can store a 32-bit signed number or four ASCII-encoded characters, each occupying 8 bits, as shown in Figure.
- A unit of 8 bits is called a byte.
- Machine instructions may require one or more words for their representation.



Byte Addressability

- It is impossible to allot a unique address to each bit in memory.

- As a solution, most modern computers assign successive addresses to successive byte locations in memory.
- This assignment of addresses to individual byte locations is termed byte addressability and memory is referred to as byte-addressable memory.
- If we assign an address to individual byte locations in the memory like 0, 1, 2, 3…. .Now if the **word length of the machine is 16 bit** then the successive words are located at addresses 0, 2, 4, 6… where each word would have 2 bytes of information.
- Similarly, if we have a machine with a **word length of 32 bit then the successive words are located at the addresses 0, 4, 8, 12…** where each word would have 4 bytes of information and it could store or retrieve 4 bytes of instruction or data in a single and basic operation.

Big-Endian and Little-Endian Assignments in Byte Addresses

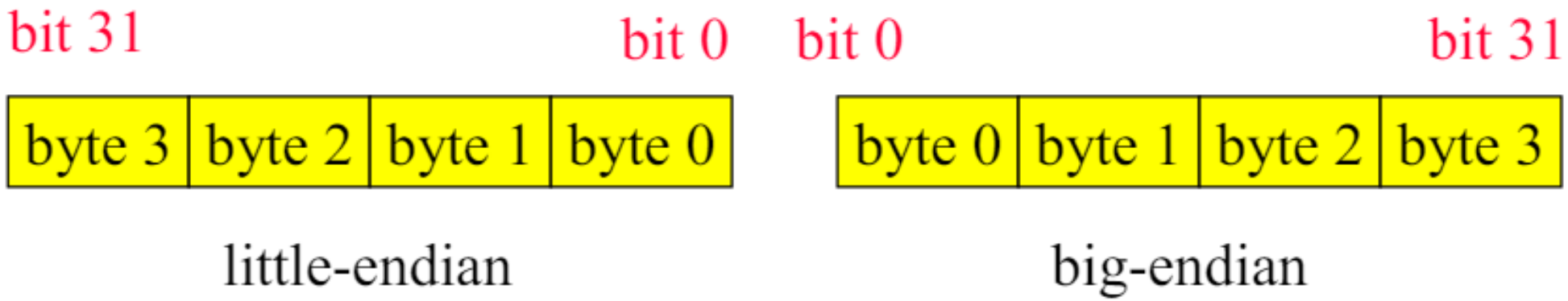
- The big-endian and little-endian are two methods of assigning byte addresses across the words in the memory.
- In the big-endian assignment, the lower byte addresses are used for the most significant bytes (MSB).
- In the little-endian assignment, the lower byte addresses are used for the least significant bytes (LSB).

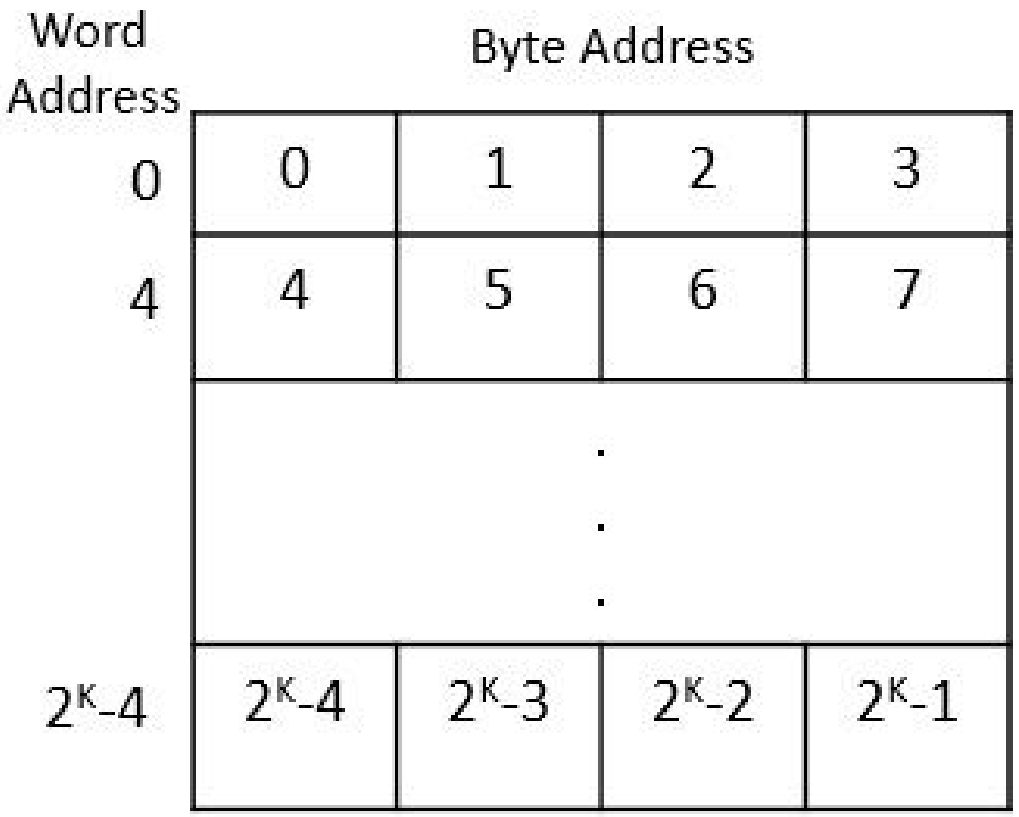
Little-endian - means the lower-order byte of the data is stored in memory at the lowest address, and the higher-order byte at the highest address. (The little end comes first)

Big-endian - means the higher-order byte of the data is stored in memory at the lowest address, and the lower-order byte at the highest address. (The big end comes first.)

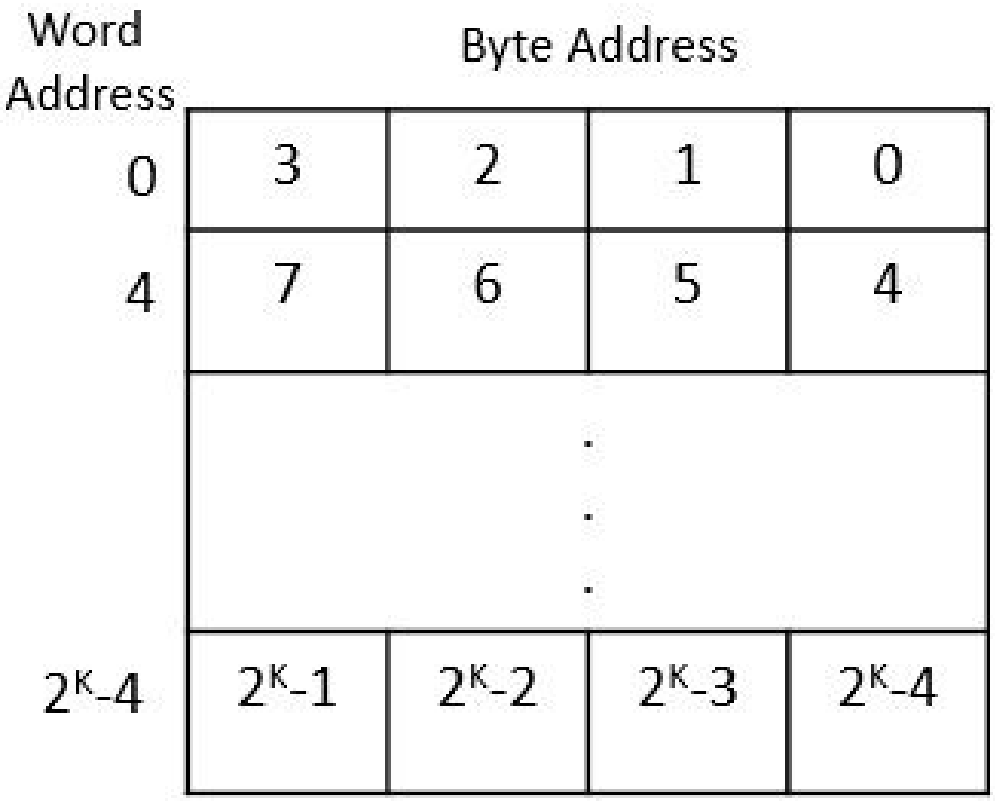
Byte Organization Within an ARM Word

- Little-endian mode
  - The lowest-order byte residing in the low-order bits of the word
- Big-endian mode
  - The lowest-order byte stored in the highest bits of the word





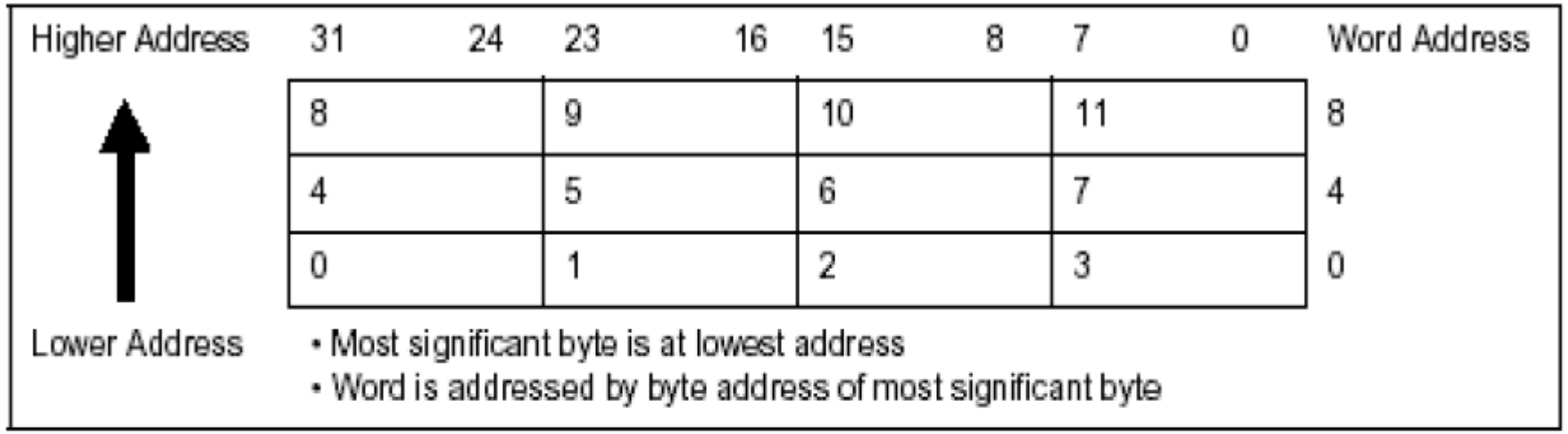
Big-Endian Assignment



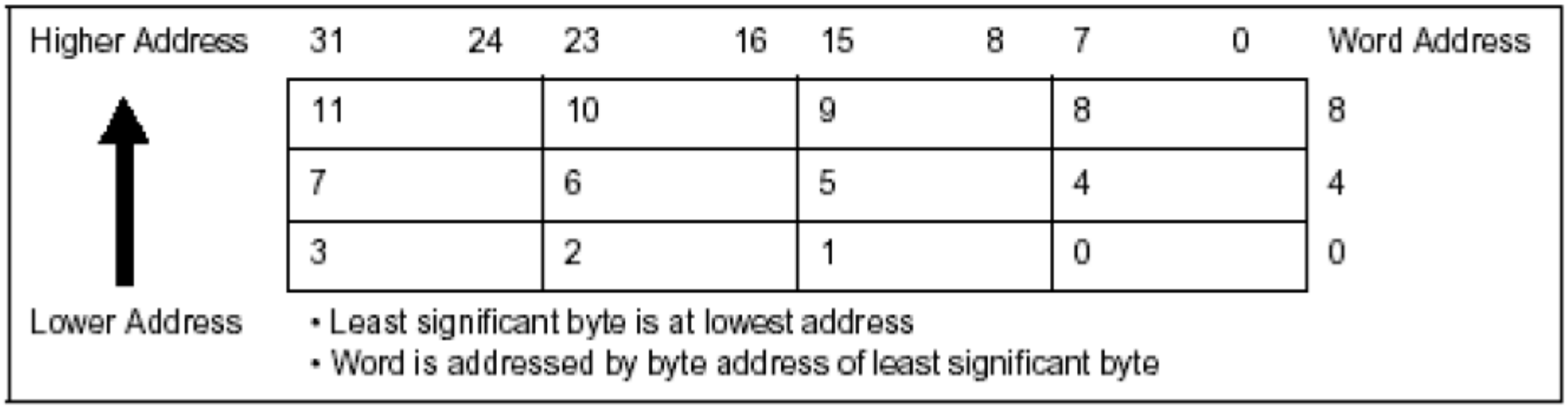
Little-Endian Assignment

Big Endian and Little Endian

Big endian



Little endian



Example-1

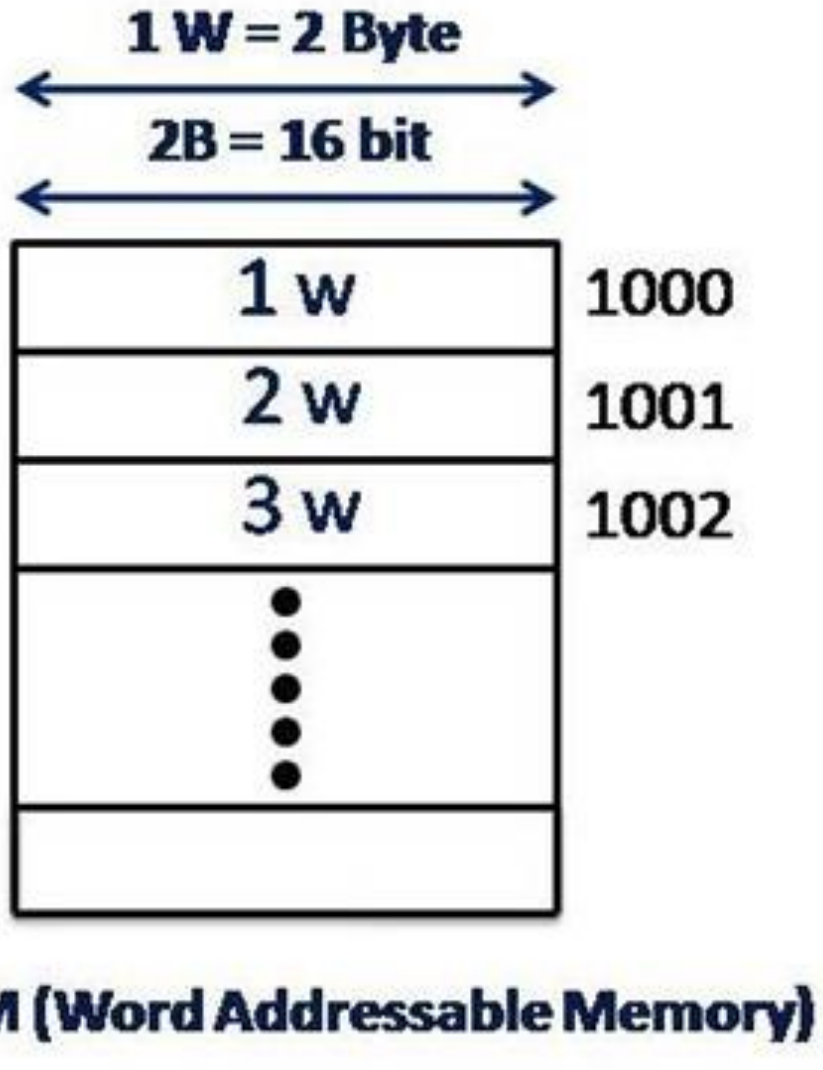
- Consider a computer that has a byte-addressable memory organized in 32-bit words according to the Little-endian scheme. A program reads Hexadecimal numbers entered at a keyboard and stores them in successive byte locations, starting at location 1000. Show the contents of the two memory words at locations 1000 and 1004 after the number “5678ABCD” has been entered. Also, show the contents of the same two memory words if it is a Big-endian scheme.

Memory Address	Big-Endian	Little-Endian
1000,1001	0x56 0x78	0xCD 0xAB
1002, 1003	0xAB 0xCD	0x78 0x56

The bits must be labelled inside the byte or a word and the most common way of labelling bits in a byte or word is as shown in the figure below

- i.e. labelling the bits as b7, b6,…….,b1, b0 from left to write as we do in little-endian assignment.

Labelling bits within a Byte



WAM (Word Addressable Memory)

- In a machine with word length 32-bit, the word boundaries occur at the bytes addresses 0, 4, 8…
- It is said that the word has aligned addresses if they begin with the byte address that is multiple of the number of bytes present in that word.
- For example, the word address 4 has four bytes in it with byte address 4, 5,6 and 7. The word address 4 starts with the byte address 4 which is multiple of the number of bytes in word 4.
- In case if the word address begins with the arbitrary byte address the word is said to have unaligned addresses.
- But conventionally the words have aligned addresses as this lets the access of memory operand more efficiently.

Memory Representation

Word Address	Byte Address	Stored Bytes
0	0, 1, 2, 3	4 bytes (Word 0)
4	4, 5, 6, 7	4 bytes (Word 1)
8	8, 9, 10, 11	4 bytes (Word 2)

Word Alignment

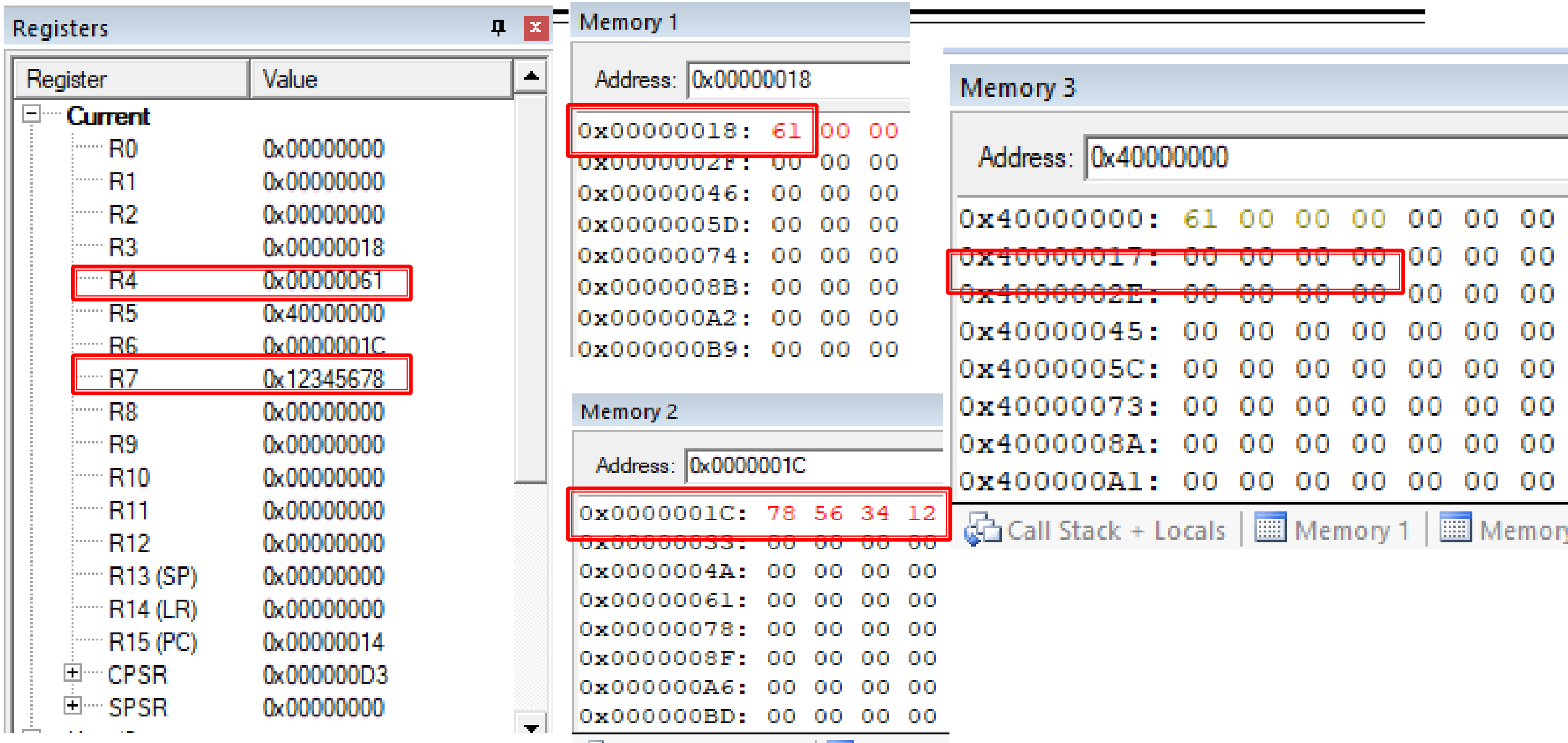
- In the case of a 32-bit word length, natural word boundaries occur at addresses 0, 4, 8,...,
- The word locations have aligned addresses if they begin at a byte address that is a multiple of the number of bytes in a word.
- if the word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4,..., and for a word length of 64 (8 bytes), aligned words begin at byte addresses 0, 8, 16,....
- There is no fundamental reason why words cannot begin at an arbitrary byte address.
- In that case, words are said to have unaligned addresses. But, the most common case is to use aligned addresses, which makes accessing of memory operands more efficient

Example:

```
            AREA prg1,CODE,READONLY
            EXPORT __main
            ENTRY

__main
            LDR R3,=array                ;loads the address of array into R3
            LDR R4,[R3]
            LDR R6,=array1               ;loads the address of array into R3
            LDR R7,[R6]
            LDR R5,=num1
            STR R4,[R5]
array     DCB   'A'
array1 DCD     0x12345678
          AREA DATA1,DATA,READWRITE
num1 DCD 0
          end
```

Output


Registers
Register
Value
Current
R0 0x00000000
R1 0x00000000
R2 0x00000000
R3 0x00000018
R4 0x00000061
R5 0x40000000
R6 0x0000001C
R7 0x12345678
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x00000000
R14 (LR) 0x00000000
R15 (PC) 0x00000014
CPSR 0x000000D3
SPSR 0x00000000
Memory 1
Address: 0x00000018
0x00000018: 61 00 00
0x00000046: 00 00 00
0x0000005D: 00 00 00
0x00000074: 00 00 00
0x0000008B: 00 00 00
0x000000A2: 00 00 00
0x000000B9: 00 00 00
Memory 2
Address: 0x0000001C
0x0000001C: 78 56 34 12
0x0000004A: 00 00 00 00
0x00000061: 00 00 00 00
0x00000078: 00 00 00 00
0x0000008F: 00 00 00 00
0x000000A6: 00 00 00 00
0x000000BD: 00 00 00 00
Memory 3
Address: 0x40000000
0x40000000: 61 00 00 00 00 00 00
0x40000017: 00 00 00 00 00 00 00
0x4000002E: 00 00 00 00 00 00 00
0x40000045: 00 00 00 00 00 00 00
0x4000005C: 00 00 00 00 00 00 00
0x40000073: 00 00 00 00 00 00 00
0x4000008A: 00 00 00 00 00 00 00
0x400000A1: 00 00 00 00 00 00 00
Call Stack + Locals
Memory 1
Memory

Registers

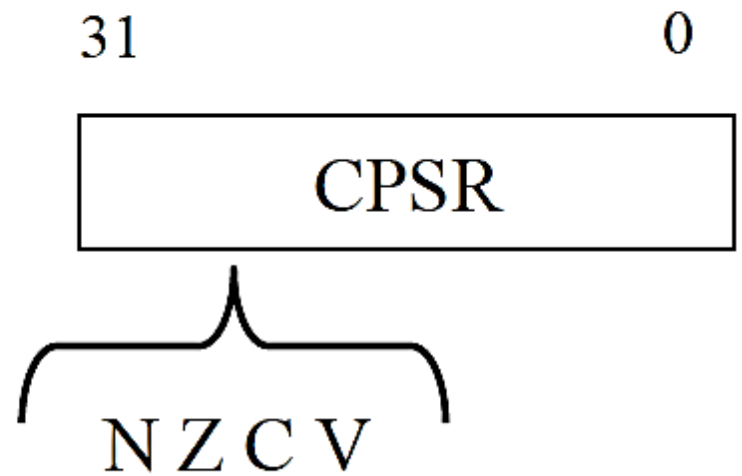
- ARM has 37 registers all of which are 32-bits long.
- 1 dedicated program counter
  - 1 dedicated current program status register
  - 5 dedicated saved program status registers
  - 30 general purpose registers
- The current processor mode governs which of several banks is accessible. Each mode can access
  - a particular set of **R0-R12** registers
  - a particular R13 (the stack pointer, sp) and R14 (the link register)
  - the program counter, R15 (pc)
  - the current program status register, cpsr

  Privileged modes (except System) can also access
  - a particular spsr (saved program status register)

ARM Registers

r0
r1
r2
r3
r4
r5
r6
r7

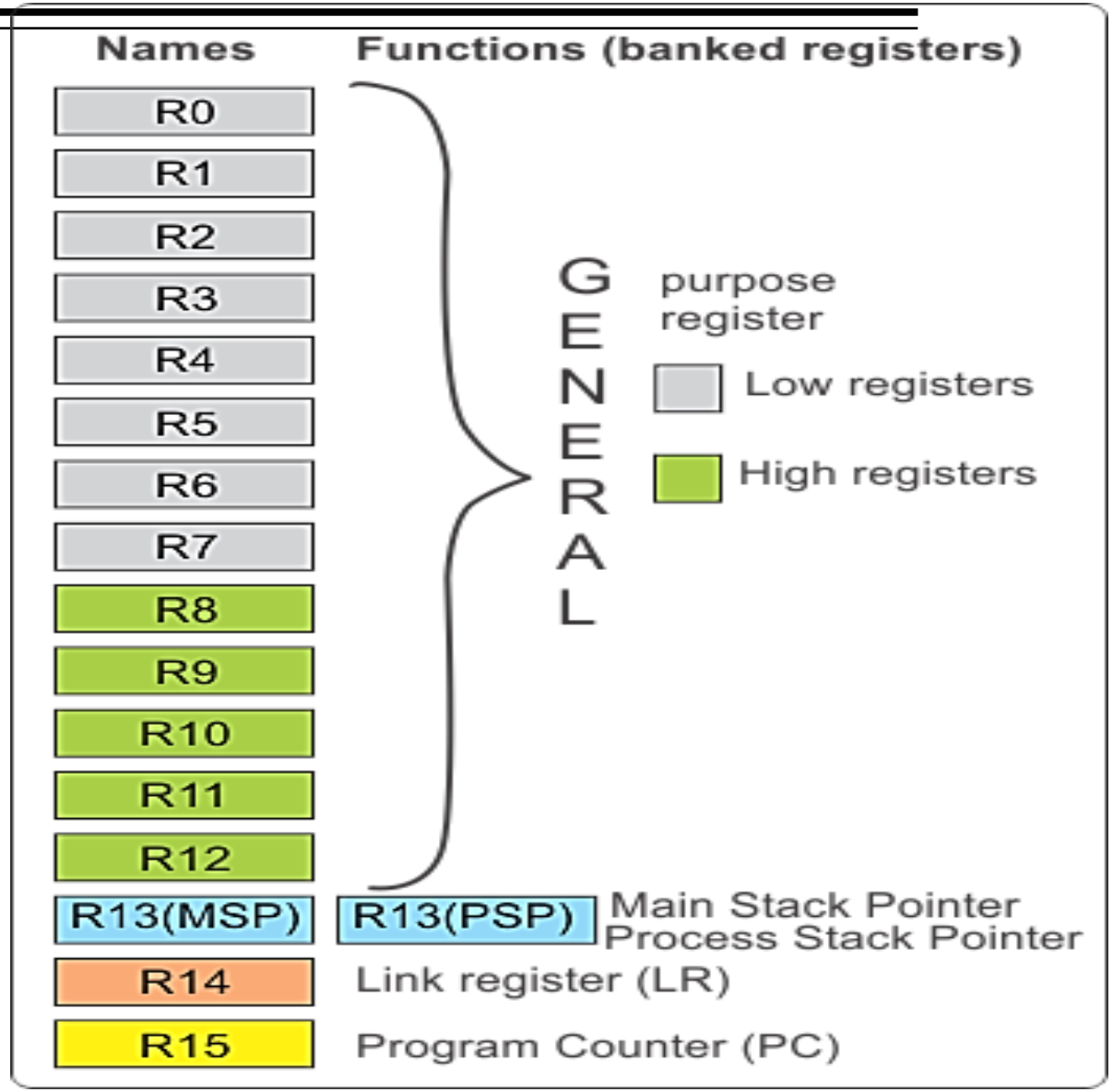
r8
r9
r10
r11
r12
r13
r14
r15 (PC)



CPSR: Current Program Status Register
SPSR: Saved Program Status Register

Registers

- General-purpose registers hold either data or an address.
- They are identified with the letter ***'R'*** prefixed to the register number.
- For example, register 4 is given the label ***R4.***
- Active registers available in user mode—a protected mode normally used when executing applications.
- *All the registers shown are 32 bits in size*



Registers

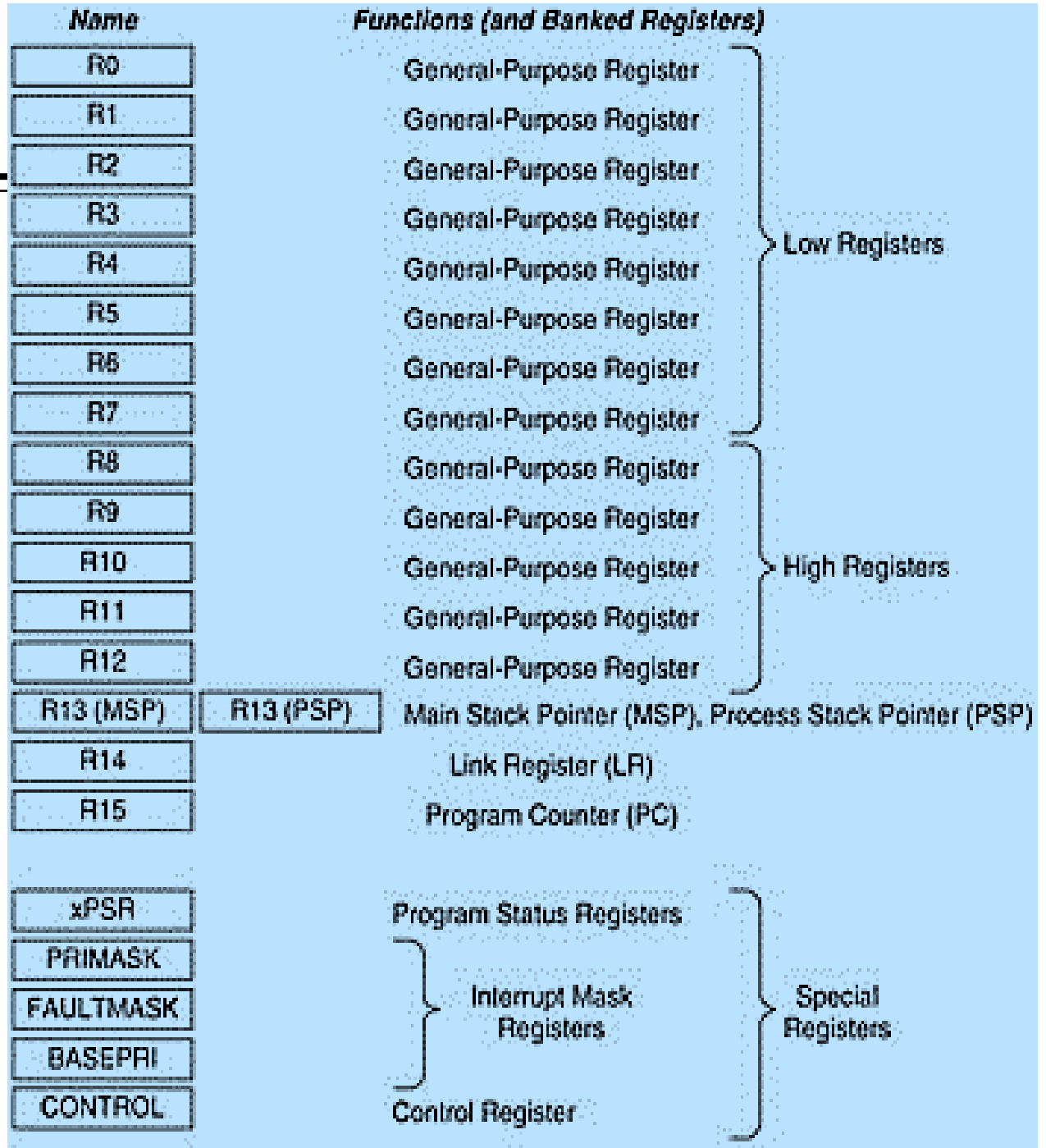
- There are up to 18 active registers: 16 data registers and 2 processor status registers.
- The data registers are visible to the programmer as *R0* to *R15.*
- **The ARM processor has three registers assigned to a particular task or special function: R*13*, *R14*, and *R15*.**
- They are frequently given different labels to differentiate them from the other registers.
  - **Register *R13*** is traditionally used as the stack pointer (*sp*) and stores the head of the stack in the current processor mode.
  - **Register *R14*** is called the link register (*lr*) and is where the core puts the return address whenever it calls a subroutine.
  - **Register *R15*** is the program counter (*pc*) and contains the address of the next instruction to be fetched by the processor.

Registers

- Depending upon the context, registers *R13* and *R14* can also be used as general-purpose registers, which can be particularly useful since these registers are banked during a processor mode change.
- However, it is dangerous to use *R13* as a general register when the processor is running any form of operating system because operating systems often assume that *R13* always points to a valid stack frame.
- In ARM state the registers *R0* to R*13* are *orthogonal*—any instruction that you can apply to *R0* you can equally well apply to any of the other registers.
- In addition to the 16 data registers, there are two program status registers: ***cpsr* and *spsr*** (the current and saved program status registers, respectively).
- The register file contains all the registers available to a programmer.
- Which registers are visible to the programmer depend upon the current mode of the processor.

Registers

- **R0 through R7:**
  - These R0 to R7 registers are also called as Low registers.
  - These registers can access by both 16-bit Thumb instructions and 32-bit Thumb-2 instructions.
- **R8 through R12:**
  - These registers are also called as High registers.
  - They are accessible by all Thumb-2instructions but not by all 16-bit Thumb instructions.



Registers

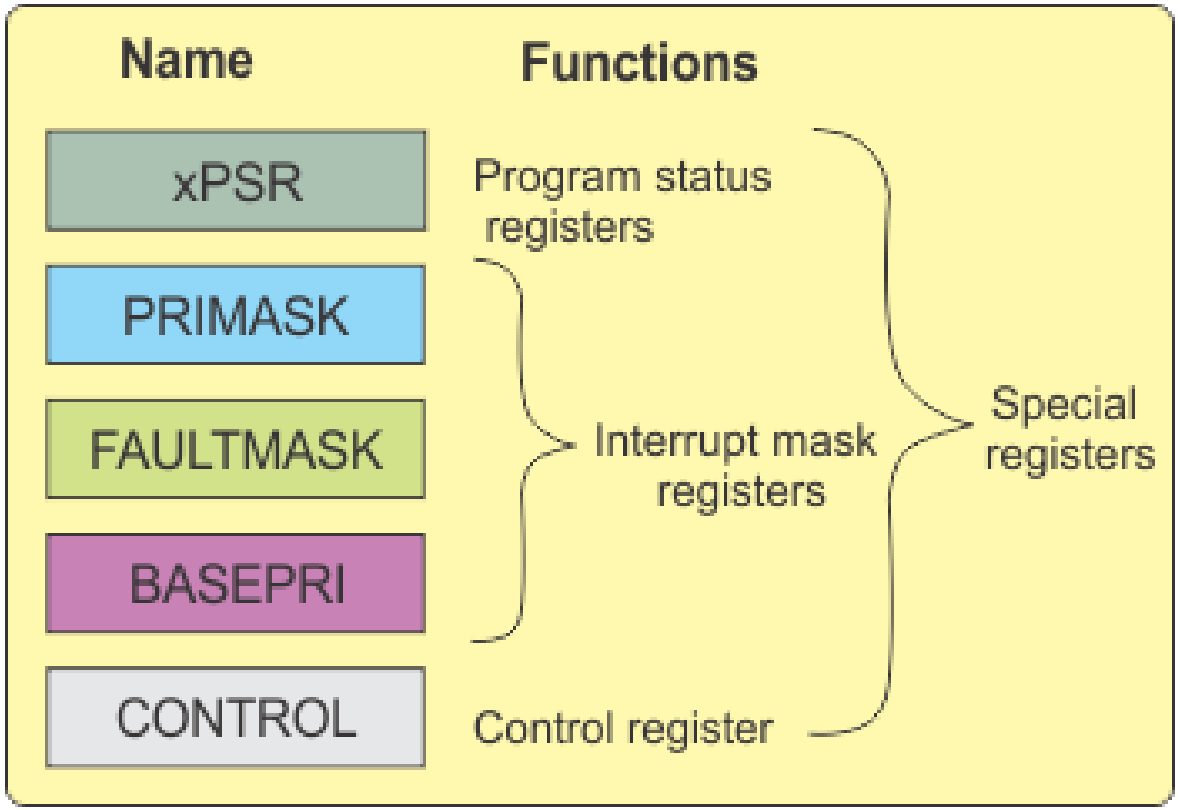
- **Stack Pointer R13:**
  - This R13 is stack pointer (SP).
  - In Cortex-M3 there are two separate stack pointers.
  - This two stack pointers are allowed to set up two separate stack memories.
- **Main Stack Pointer (MSP):**
  - This is default stack pointer and this is used by OS kernel and exception handlers.
- **Process Stack Pointer (PSP):**
  - Used by use application mode only.
  - This is used by the base-level application code (when not running an exception handler)

Registers

- Link Register: R14 is Link Register.
  - When we call subroutine, the return address is stored in the link register.
  - Inside an assembly program, you can write it as either *R14* or *LR*.
  - LR is used to store the return program counter (PC) when a subroutine or function is called.
- Program Counter: R15 is program counter.
  - This stores the current program address.
  - You can access PC in assembler code by either R15 or PC.
  - Because of the pipelined nature of the Cortex-M3 processor, when you read this register, you will find that the value is different than the location of the executing instruction.

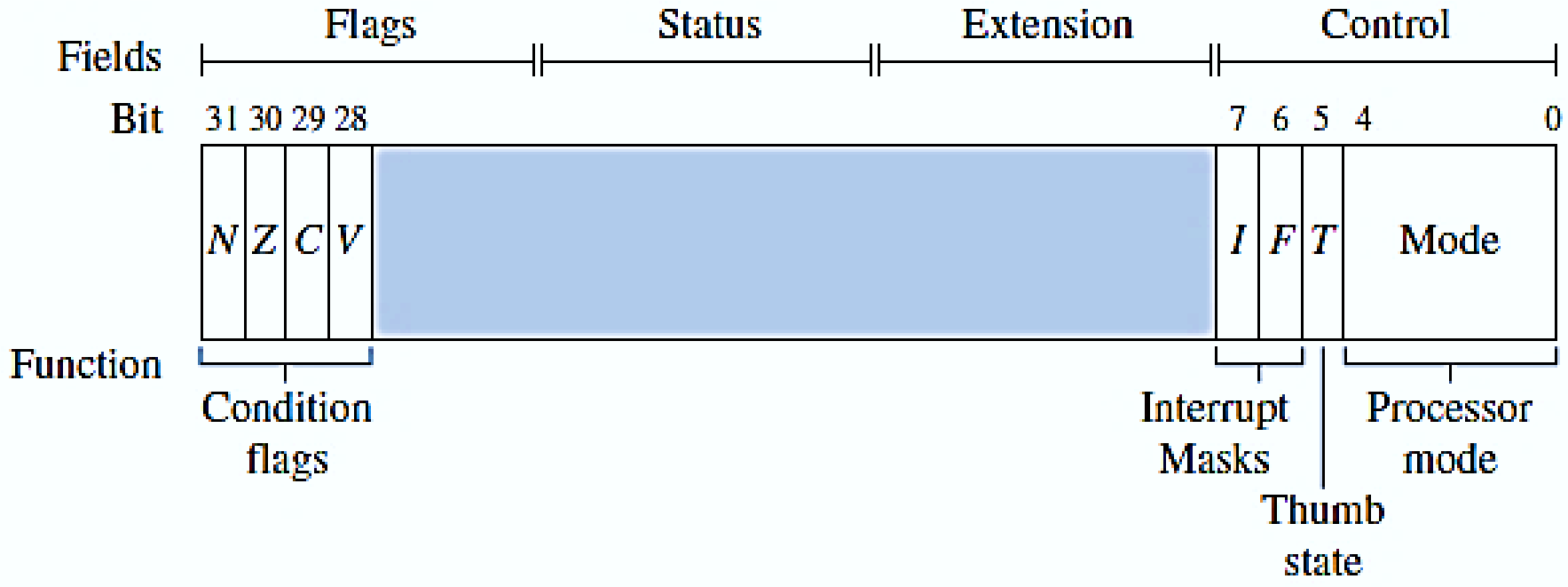
Registers

- Special Registers:
  - **PSRs**
    - Application Program Status register (APSR)
    - Interrupt Program Status register (IPSR)
    - Execution Program Status register (EPSR)
  - Interrupt Mask Registers:
    - PRIMASK,
    - FAULTMASK and
    - BASEPRI
  - Control Register(CONTROL)



Current Program Status Register

- The ARM core uses the cpsr to monitor and control internal operations.
- The cpsr is a dedicated 32-bit register and resides in the register file.



A generic program status register (*psr*).

Current Program Status Register

- The cpsr is divided into four fields, each 8 bits wide: flags, status, extension, and control.
- In current designs the extension and status fields are reserved for future use. The control field contains the processor mode, state, and interrupt mask bits.
- The flags field contains the condition flags.
- Some ARM processor cores have extra bits allocated. For example, the J bit, which can be found in the flags field, is only available on Jazelle-enabled processors, which execute 8-bit instructions.

xPSR

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception number				
EPSR						ICI/IT	T			ICI/IT						

- The PSRs are subdivided into three status registers: APSR,IPSR,EPSR
- The three PSRs can be accessed together or separately using the special register access instructions MSR and MRS.
- One can also change the APSR using the MSR instruction, but EPSR and IPSR are read-only.
- For example:
  - MRS R0, APSR ; Read Flag state into R0
  - MRS R0, IPSR ; Read Exception/Interrupt state.
  - MRS R0, EPSR ; Read Execution state
  - MSR APSR, R0 ; Write Flag state
- **MRS - Move PSR status/flags to register and MSR - Move register to PSR status/flags**

The Program Status Registers (CPSR and SPSRs)

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

*Copies of the ALU status flags (latched if the instruction has the "S" bit set).

* **Condition Code Flags**

N = **N**egative result from ALU flag.
Z = **Z**ero result from ALU flag.
C = ALU operation **C**arried out
V = ALU operation o**V**erflowed
Q = Indicates overflow or saturation

* **Interrupt Disable bits.**
  I = 1, disables the IRQ.
  F = 1, disables the FIQ.

* **T Bit (Architecture v4T only)**
  T = 0, Processor in ARM state
  T = 1, Processor in Thumb state

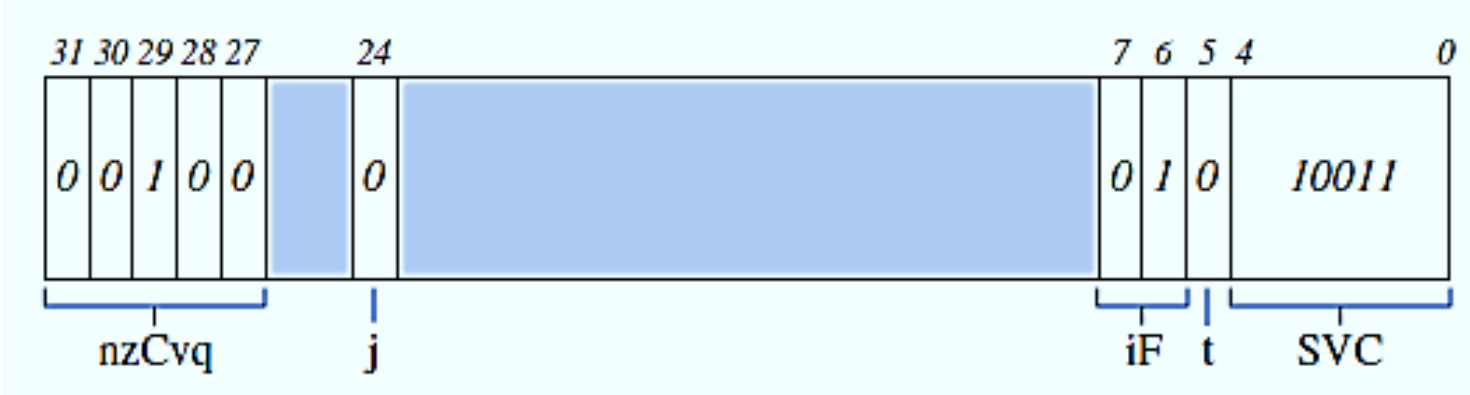
* **Mode Bits**
  M[4:0] define the processor mode.

Condition Flags

Flag	Logical Instruction	Arithmetic Instruction
Negative (N='1')	No meaning	Bit 31 of the result has been set Indicates a negative number in signed operations
Zero (Z='1')	Result is all zeroes	Result of operation was zero
Carry (C='1')	After Shift operation '1' was left in carry flag	Result was greater than 32 bits
oVerflow (V='1')	No meaning	Result was greater than 31 bits Indicates a possible corruption of the sign bit in signed numbers

Condition Flags

- For the condition flags a capital letter shows that the flag has been set. For interrupts a capital letter shows that an interrupt is disabled.
- Example: *cpsr = nzCvqjiFt_SVC*.



- The *C* flag is the only condition flag set. The rest *nzvq* flags are all clear.
- The processor is in ARMstate because neither the Jazelle *j* or Thumb *t* bits are set.
- The IRQ interrupts are enabled, and FIQ interrupts are disabled.
- The processor is in *supervisor* (*SVC*) mode since the mode[4:0] is equal to binary 10011.

Conditional Execution

- Conditional execution controls whether or not the core will execute an instruction.
- Most instructions have a condition attribute that determines if the core will execute it based on the setting of the condition flags.
- Prior to execution, the processor compares the condition attribute with the condition flags in the cpsr.
- If they match, then the instruction is executed; otherwise the instruction is ignored.
- The condition attribute is postfixed to the instruction mnemonic, which is encoded
- into the instruction.
- When a condition mnemonic is not present, the default behaviour is to set it to always (AL) execute.

Conditional Execution

Condition mnemonics.

Mnemonic	Name	Condition flags
EQ	equal	*Z*
NE	not equal	*z*
CS HS	carry set/unsigned higher or same	*C*
CC LO	carry clear/unsigned lower	*c*
MI	minus/negative	*N*
PL	plus/positive or zero	*n*
VS	overflow	*V*
VC	no overflow	*v*
HI	unsigned higher	*zC*
LS	unsigned lower or same	*Z* or *c*
GE	signed greater than or equal	*NV* or *nv*
LT	signed less than	*Nv* or *nV*
GT	signed greater than	*NzV* or *nzv*
LE	signed less than or equal	*Z* or *Nv* or *nV*
AL	always (unconditional)	ignored

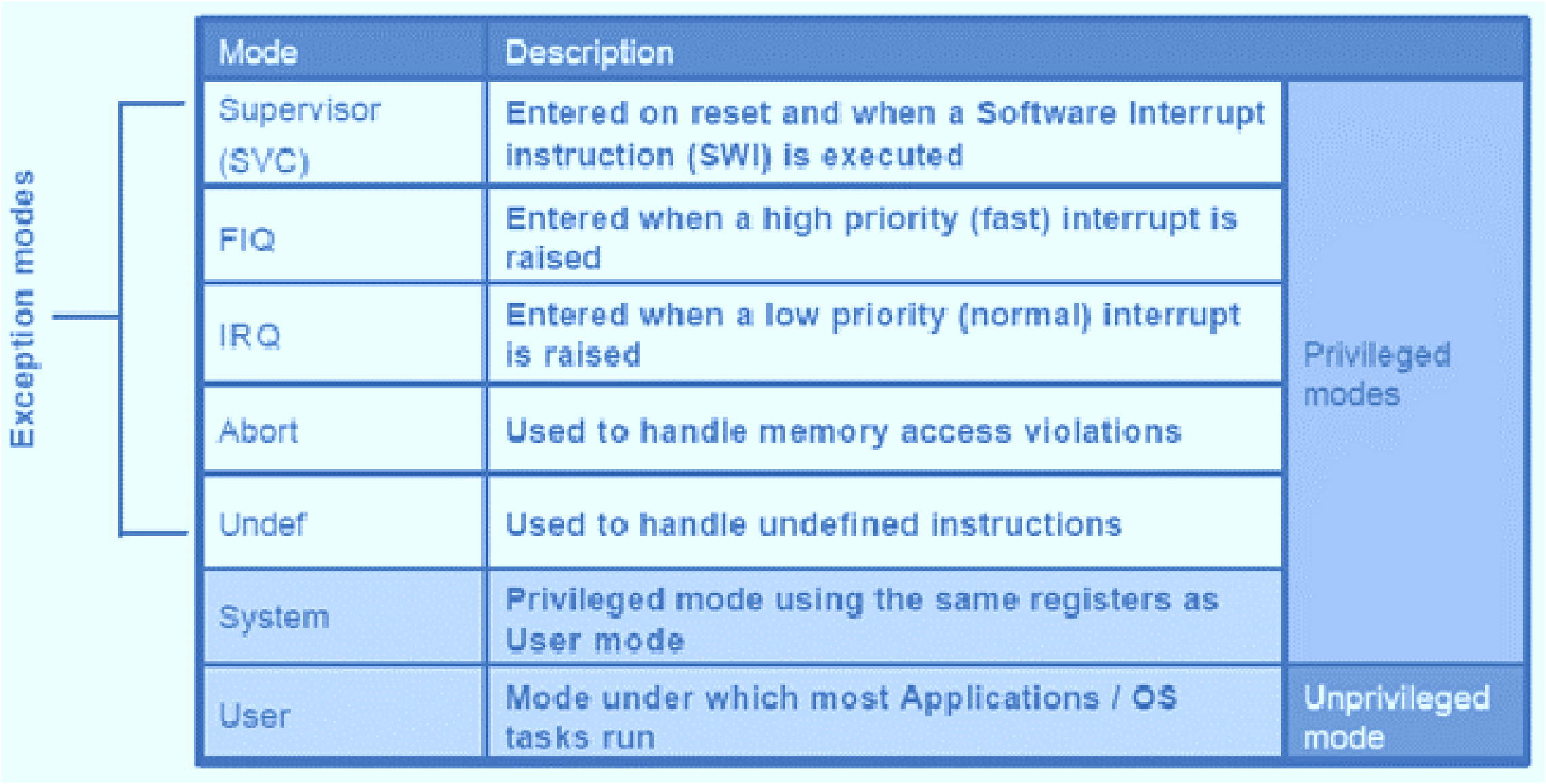
Processor Modes

- The processor mode determines which registers are active and the access rights to the *cpsr* register itself.
- Each processor mode is either privileged or nonprivileged.
- A privileged mode allows full read-write access to the *cpsr*.
- Conversely, a nonprivileged mode only allows read access to the control field in the *cpsr* but still allows read-write access to the condition flags.
- There are seven processor modes in total: Six privileged modes (***abort***, ***fast interrupt request***, ***interrupt request***, ***supervisor***, ***system***, **and** ***undefined***) and one nonprivileged mode (*user*).

Processor Modes

- **The ARM has seven basic operating modes:**
  - **User** : **unprivileged mode under which most tasks run**
  - **FIQ** : entered when a high priority (fast) interrupt is raised
  - **IRQ** : entered when a low priority (normal) interrupt is raised
  - **Supervisor** : entered on reset and when a **Software Interrupt instruction** is executed
  - **Abort** : used to handle memory access violations
  - **Undef** : used to handle undefined instructions
  - **System** : privileged mode using the same registers as user mode

Processor Modes



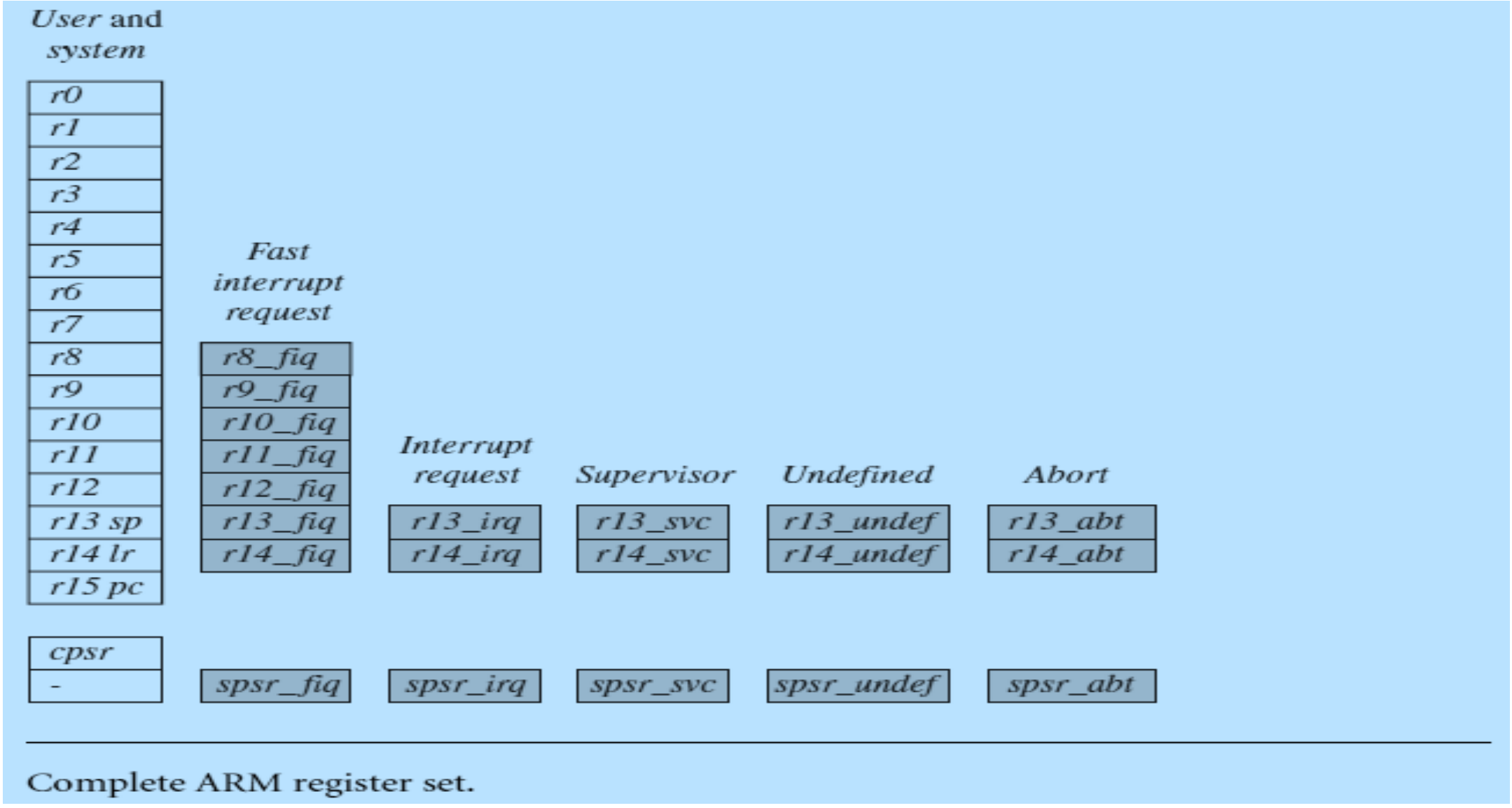
Exception modes

Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

Processor Modes

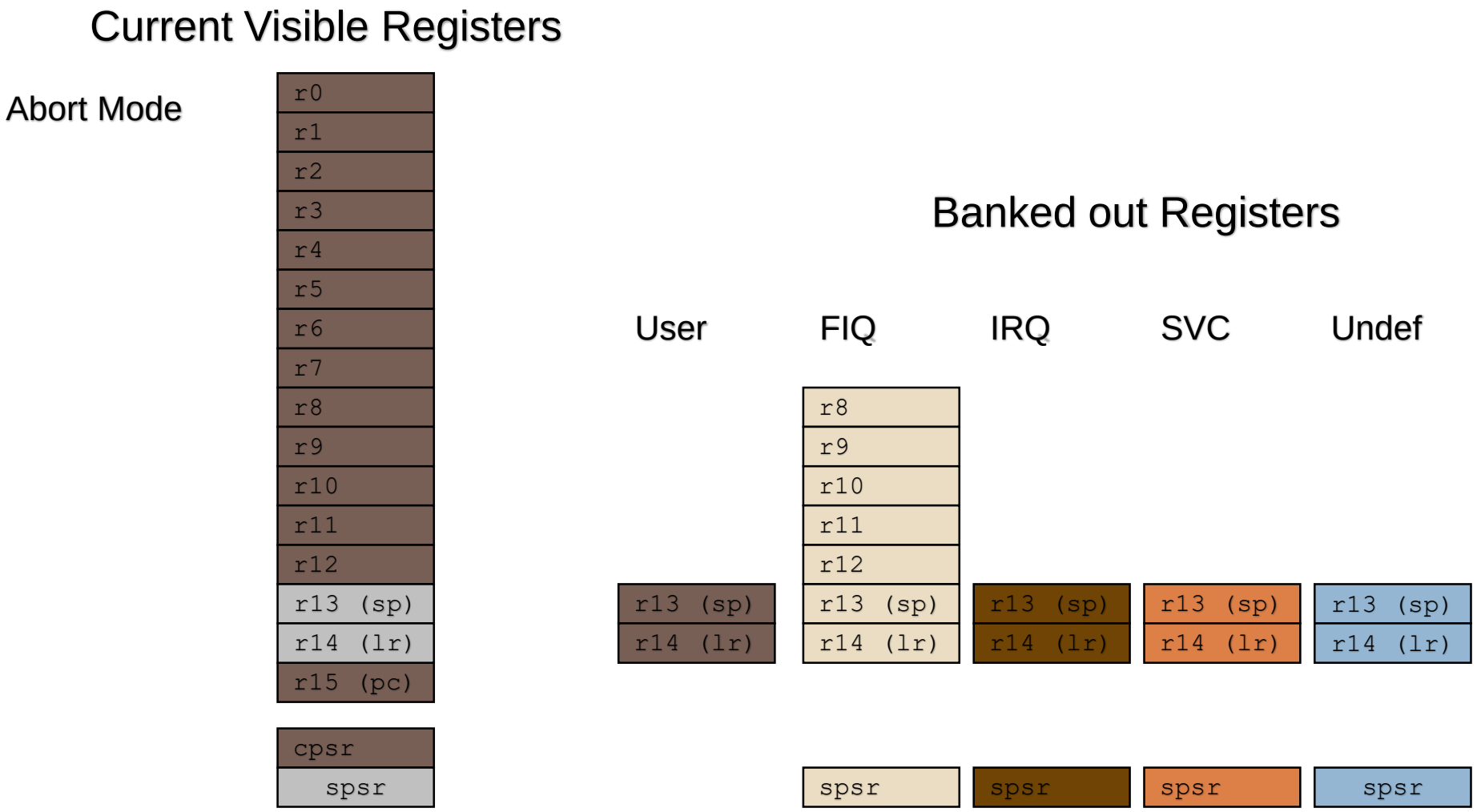
- The processor enters *abort* mode when there is a failed attempt to access memory.
- *Fast interrupt request* and *interrupt request* modes correspond to the two interrupt levels available on the ARM processor.
- *Supervisor* mode is the mode that the processor is in after reset and is generally the mode that an operating system kernel operates in.
- *System* mode is a special version of *user* mode that allows full read-write access to the *cpsr.*
- *Undefined* mode is used when the processor encounters an instruction that is undefined or not supported by the implementation.
- *User* mode is used for programs and applications.

Processor Modes



Complete ARM register set.

Processor Modes : The ARM Register Set



Processor Modes

- Every processor mode except *user* mode can change mode by writing directly to the mode bits of the *cpsr.*
- All processor modes except *system* mode have a set of associated banked registers that are a subset of the main 16 registers.
- A banked register maps one-to one onto a *user* mode register. If you change processor mode, a banked register from the new mode will replace an existing register.
- The processor mode can be changed by a program that writes directly to the *cpsr* (the processor core has to be in privileged mode) or by hardware when the core responds to an exception or interrupt.
- The following exceptions and interrupts cause a mode change:
  - *reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort, and undefined instruction.*

Processor Modes

Processor mode.

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

Register Organization in ARM States

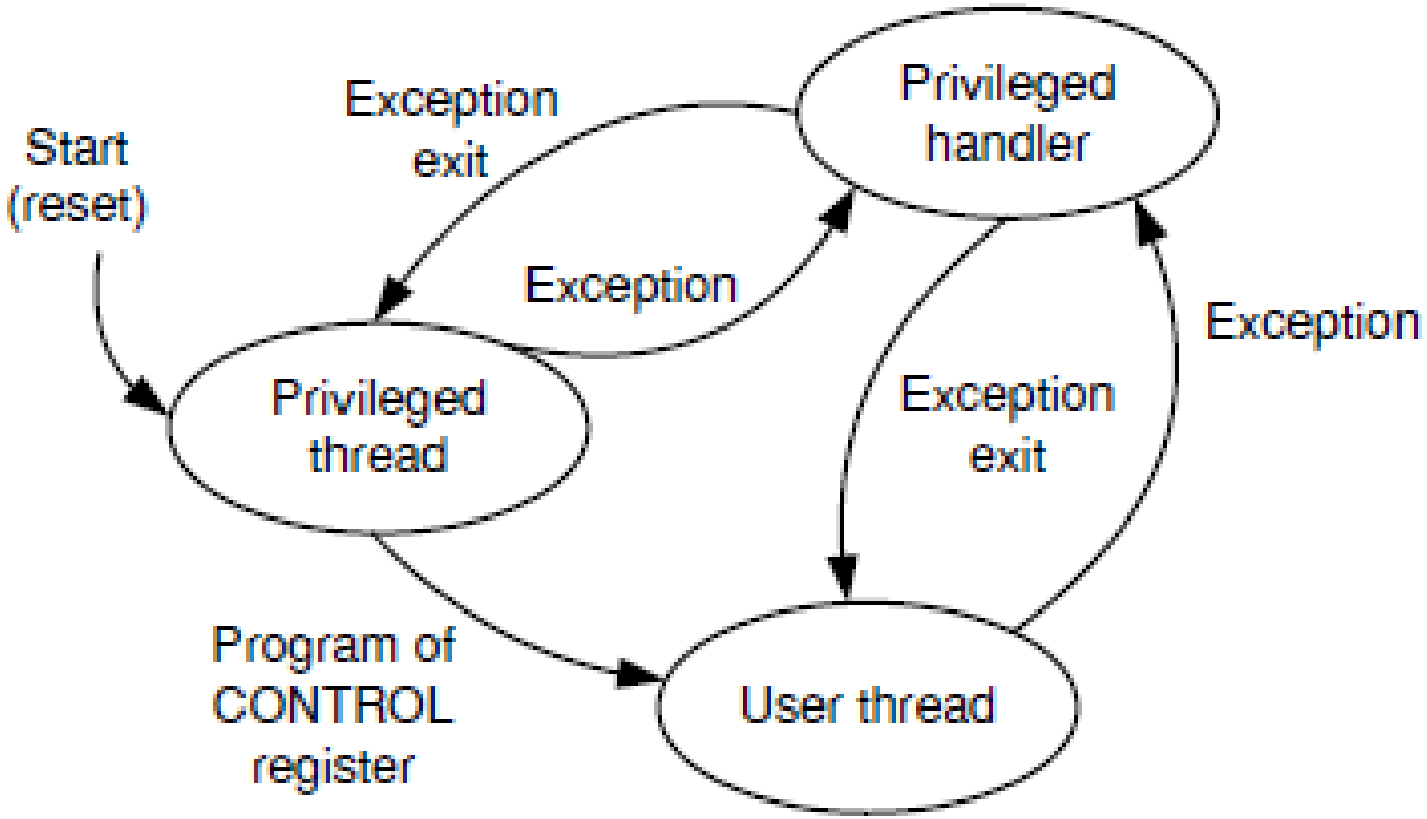
Mode	Registers
User Mode & System Mode	r0~r13, LR(r14), PC (r15), CPSR
FIQ Mode	r0~r7, r8_fiq~r13_fiq, r14_fiq, PC, CPSR, SPSR_fiq
IRQ Mode	r0~r12, r13_irq, r14_irq, PC, CPSR, SPSR_irq
Supervisor Mode	r0~r12, r13_svc, r14_svc, PC, CPSR, SPSR_svc
Abort Mode	r0~r12, r13_abt, r14_abt, PC, CPSR, SPSR_abt
Undefined Mode	r0~r12, r13_und, r14_und, PC, CPSR, SPSR_und

ARM Cortex-M3: Operation Modes

- The Cortex-M3 processor supports two operating modes, Thread and Handler and two levels of access for the code, privileged and unprivileged, enabling the implementation of complex and open systems without sacrificing the security of the application.
- Unprivileged code execution limits or excludes access to some resources like certain instructions and specific memory locations.
- The Thread mode is the typical operating mode and supports both privileged and unprivileged code.
- The Handler mode is entered when an exception occurs and all code is privileged during this mode.
- In addition, all operation is categorized under two operating states, Thumb for normal execution and Debug for debug activities.

Operation Modes

Allowed Operation Mode Transitions.



	Privileged	*User*
When running an exception handler	Handler mode	
When not running an exception handler (e.g., main program)	Thread mode	Thread mode

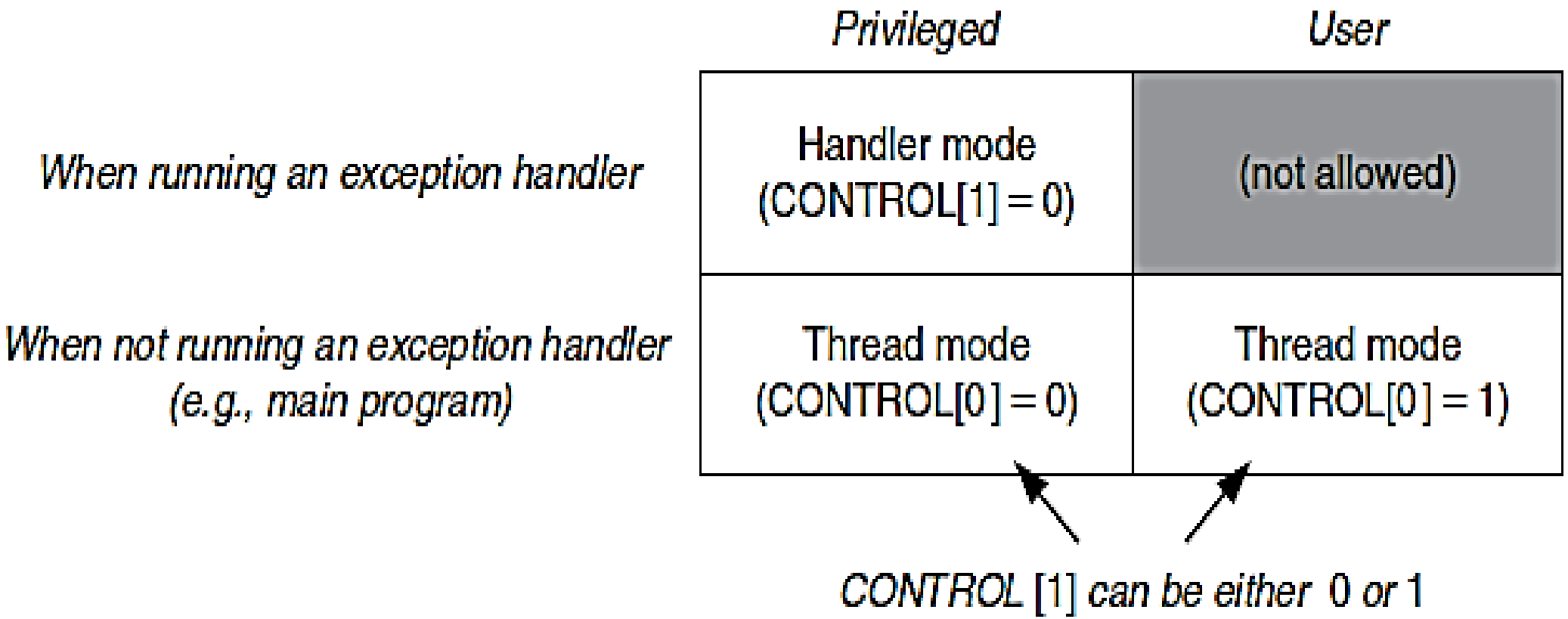
Operation Modes

- When the processor is running a main program (thread mode), it can be either in a privileged state or a user state, but exception handlers can only be in a privileged state.
- When the processor exits reset, it is in thread mode, with privileged access rights.
- In the privileged state, a program has access to all memory ranges (except when prohibited by MPU settings) and can use all supported instructions.
- Software in the privileged access level can switch the program into the user access level using the control register.
- When an exception takes place, the processor will always switch back to the privileged state and return to the previous state when exiting the exception handler.
- A user program cannot change back to the privileged state by writing to the control register.
- It has to go through an exception handler that programs the control register to switch the processor back into the privileged access level when returning to thread mode.

Operation Modes

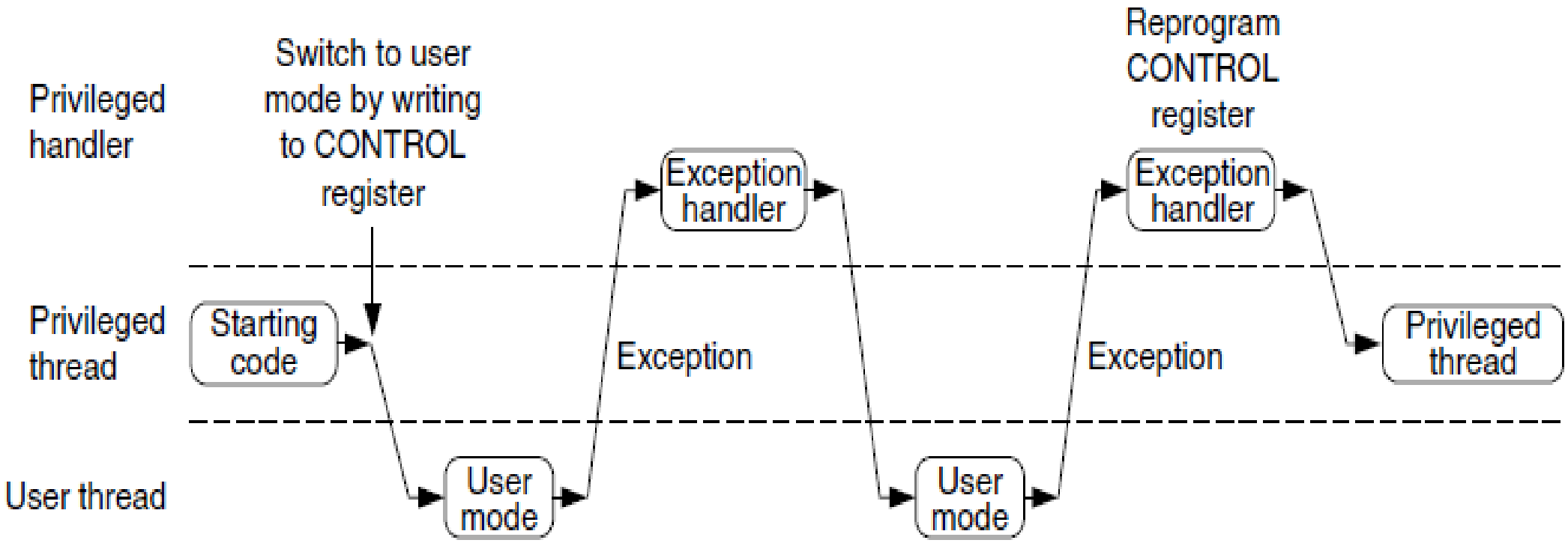
- In the user access level (thread mode), access to the system control space (SCS)—a part of the memory region for configuration registers and debugging components—is blocked.
- Furthermore, instructions that access special registers (such as MSR, except when accessing APSR) cannot be used.
- If a program running at the user access level tries to access SCS or special registers, a fault exception will occur.
- Software in a privileged access level can switch the program into the user access level using the control register.
- When an exception takes place, the processor will always switch to a privileged state and return to the previous state when exiting the exception handler.
- A user program cannot change back to the privileged state directly by writing to the control register. It has to go through an exception handler that programs the control register to switch the processor back into privileged access level when returning to thread mode.

Operation Modes



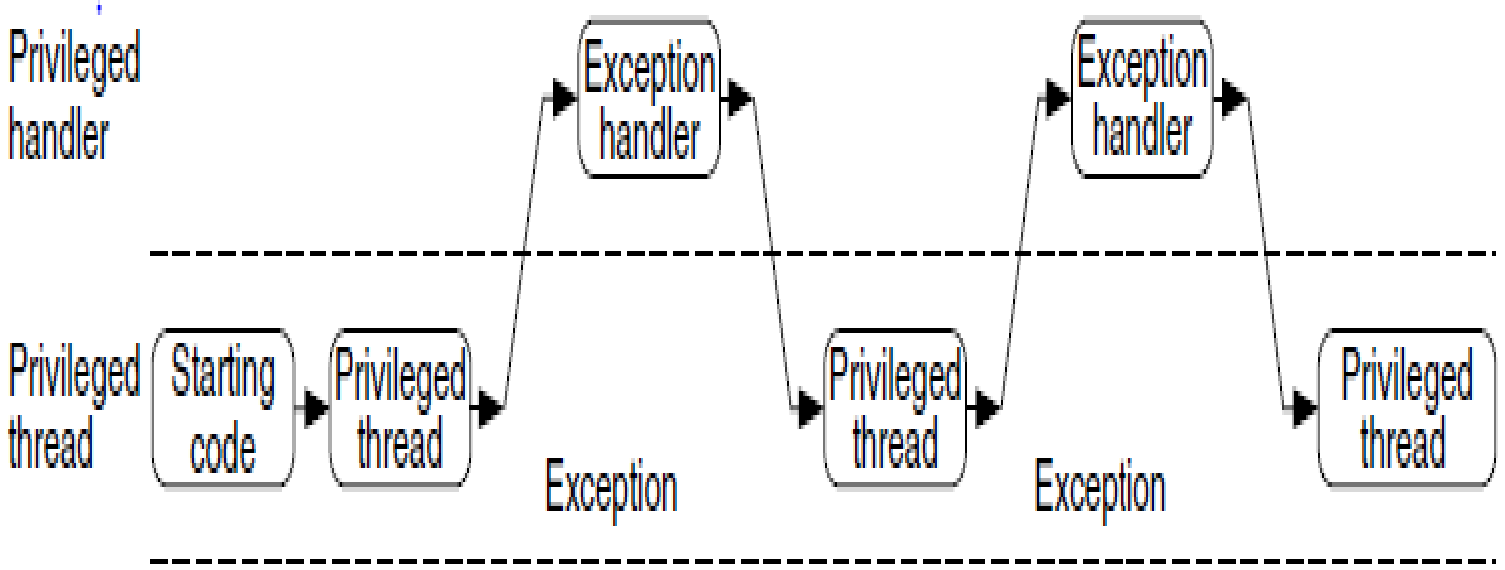
Operation Modes

- Switching of Operation Mode by Programming the Control Register or by Exceptions.



Operation Modes

- Simple applications do not require user access level in thread mode.



Operation Modes

- In simple applications, there is no need to separate the privileged and user access levels. In these cases, there is no need to use user access level and no need to program the control register.
- You can separate the user application stack from the kernel stack memory to avoid the possibility of crashing a system caused by stack operation errors in user programs.
- With this arrangement, the user program (running in thread mode) uses the PSP, and the exception handlers use the MSP.
- The switching of SPs is automatic upon entering or leaving the exception handlers.
- The mode and access level of the processor are defined by the control register. When the control register bit 0 is 0, the processor mode changes when an exception takes place.
- When control register bit 0 is 1 (thread running user application), both processor mode and access level change when an exception takes place.
- Control register bit 0 is programmable only in the privileged level. For a user-level program to switch to privileged state, it has to raise an interrupt (for example, supervisor call [SVC]) and write to CONTROL[0] within the handler.

Operation Modes

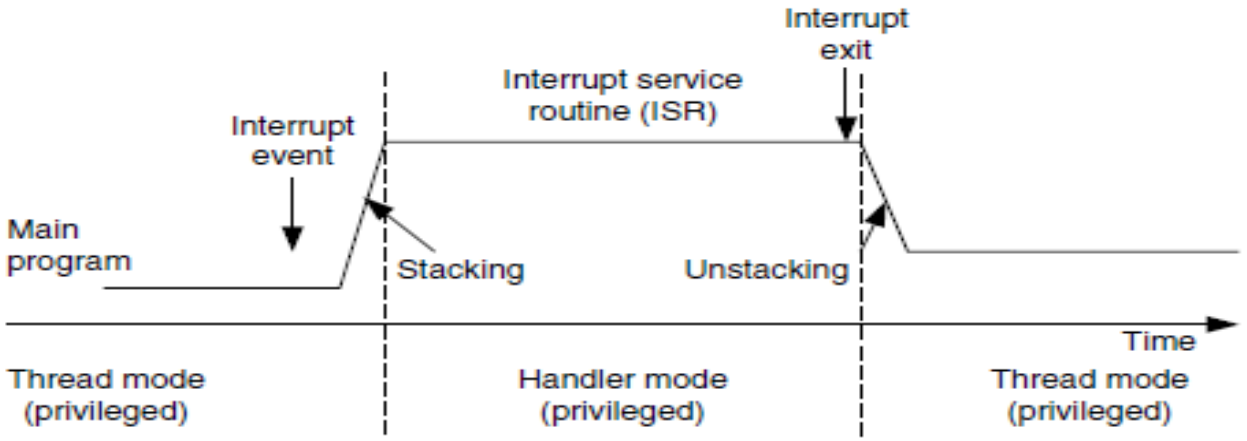


FIGURE 3.9

Switching Processor Mode at Interrupt.

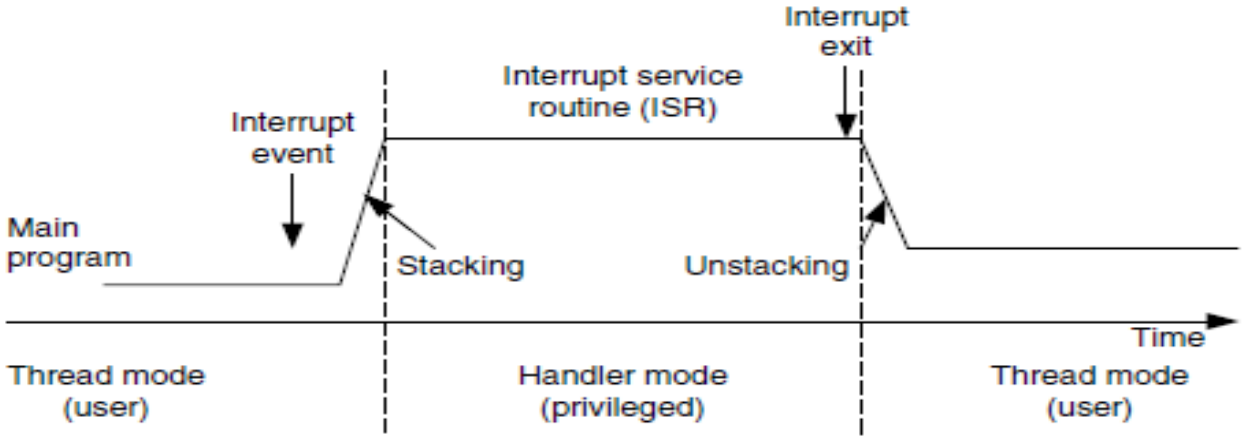


FIGURE 3.10

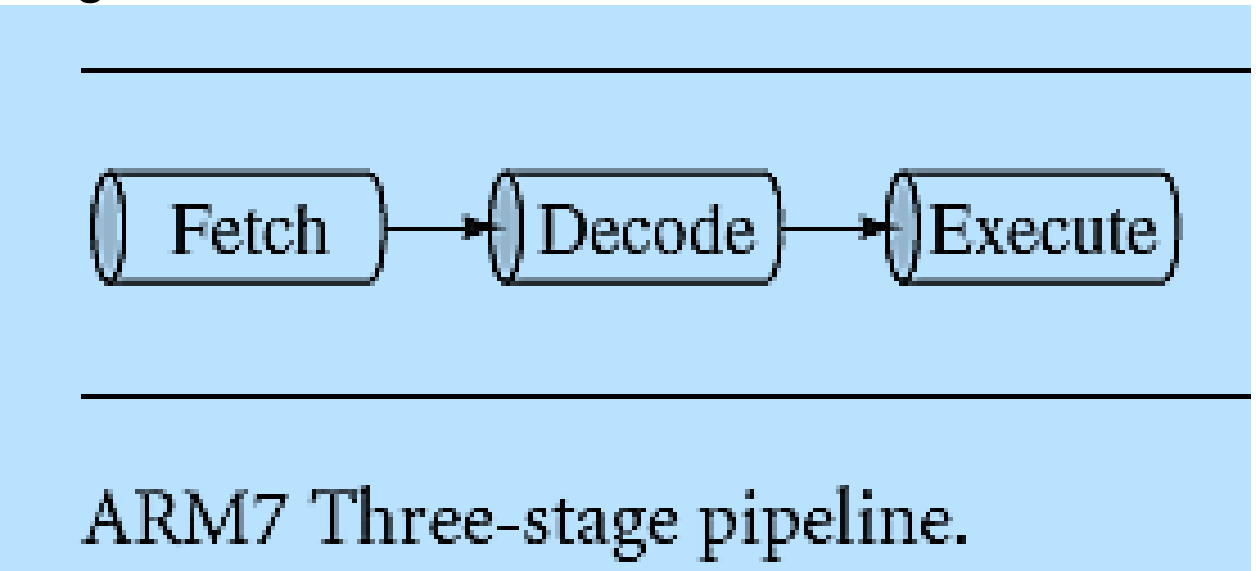
Switching Processor Mode and Privilege Level at Interrupt.

Operation Modes

- The separation of privilege and user levels improves system reliability by preventing system configuration registers from being accessed or changed by some untrusted programs.
- If an MPU is available, it can be used in conjunction with privilege levels to protect critical memory locations, such as programs and data for OSs.
- With privileged accesses, usually used by the OS kernel, all memory locations can be accessed (unless prohibited by MPU setup).
- When the OS launches a user application, it is likely to be executed in the user access level to protect the system from failing due to a crash of untrusted user programs.

ARM Pipeline

- A pipeline is the mechanism a RISC processor uses to execute instructions. Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed.

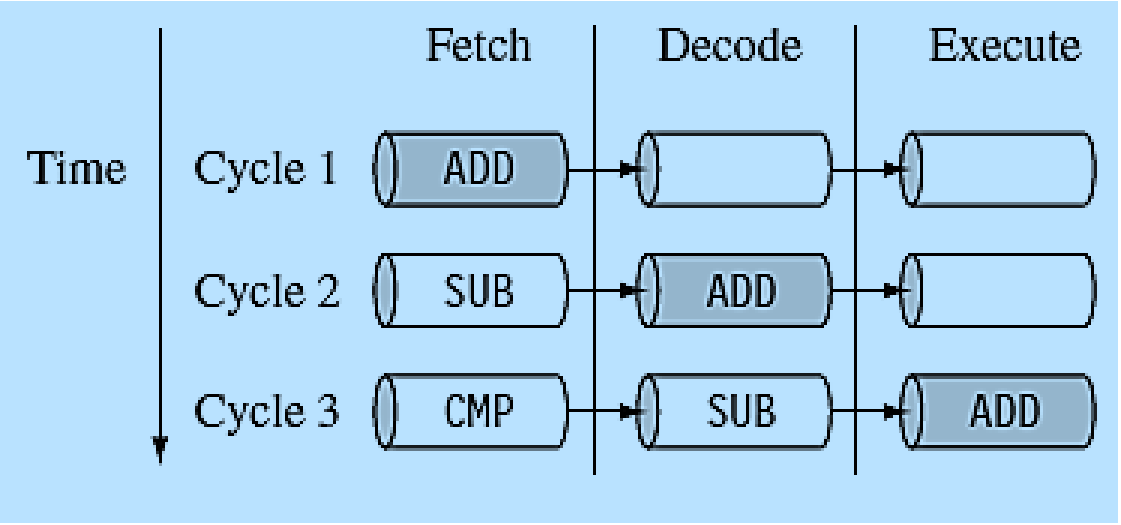


ARM7 Three-stage pipeline.

- ***Fetch*** loads an instruction from memory.
- ***Decode*** identifies the instruction to be executed.
- ***Execute*** processes the instruction and writes the result back to a register.

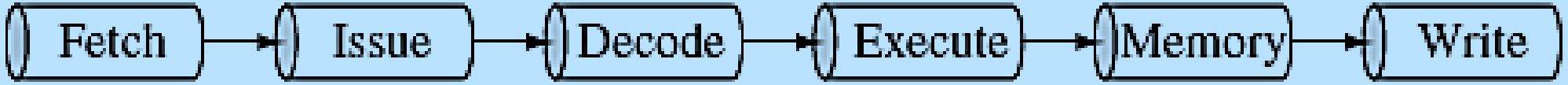
ARM Pipeline

- As the pipeline length increases, the amount of work done at each stage is reduced, which allows the processor to attain a higher operating frequency. This in turn increases the performance.
- The system latency also increases because it takes more cycles to fill the pipeline before the core can execute an instruction.
- The increased pipeline length also means there can be data dependency between certain stages. You can write code to reduce this dependency by using instruction scheduling.
- Pipelined instruction sequence.



ARM Pipeline

ARM9 five-stage pipeline.

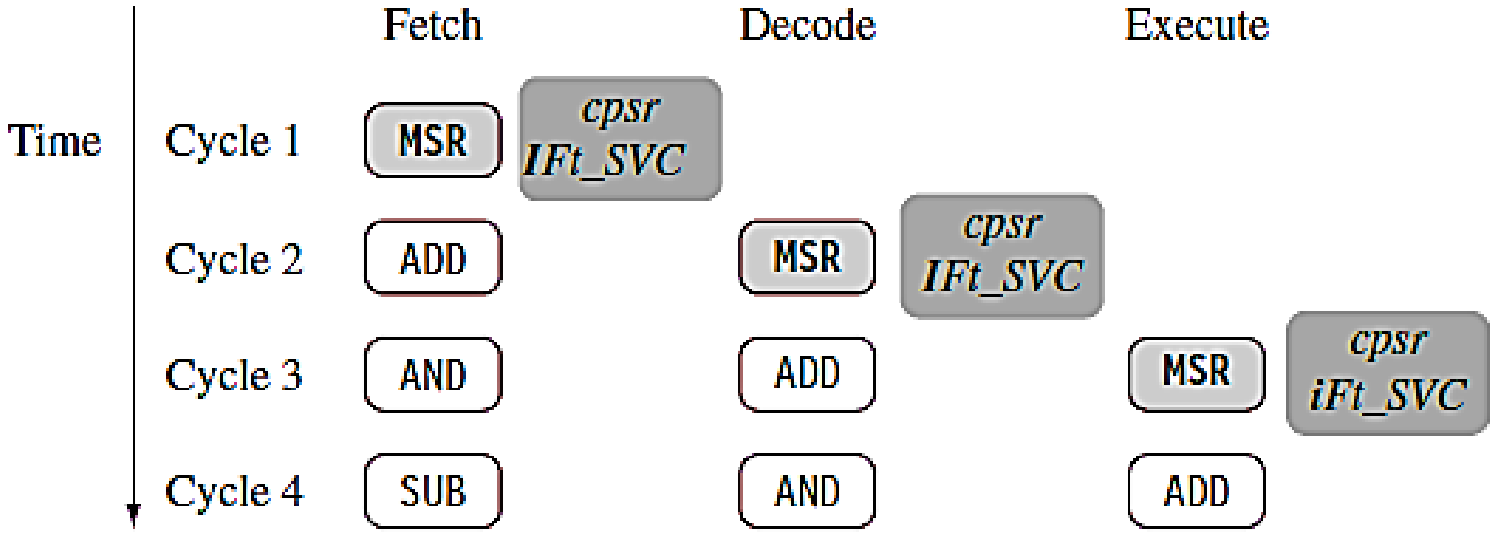


ARM10 six-stage pipeline.

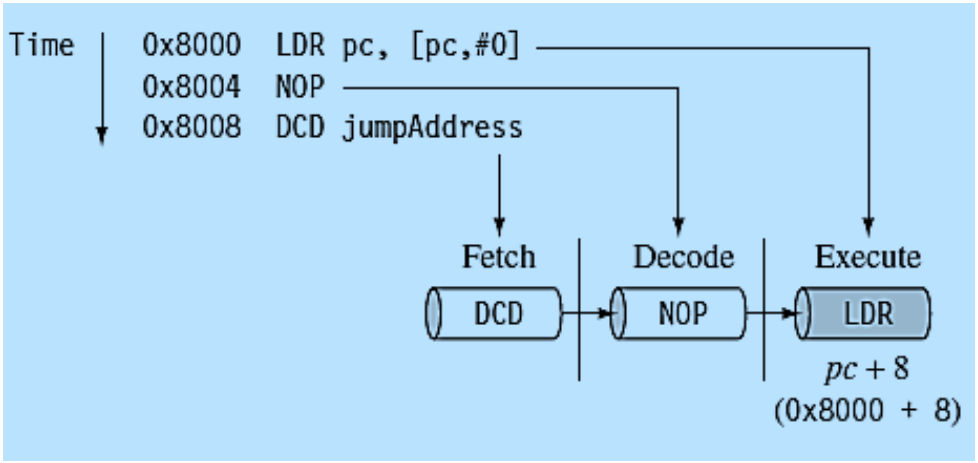
ARM Pipeline

- The pipeline design for each ARM family differs. For example, The ARM9 core increases the pipeline length to five stages.
- The ARM9 adds a memory and writeback stage, which allows the ARM9 to process on average 1.1 Dhrystone MIPS per MHz—an increase in instruction throughput by around 13% compared with an ARM7.
- The maximum core frequency attainable using an ARM9 is also higher.
- The ARM10 increases the pipeline length still further by adding a sixth stage.
- The ARM10 can process on average 1.3 Dhrystone MIPS per MHz, about 34% more throughput than an ARM7 processor core, but again at a higher latency cost.
- Even though the ARM9 and ARM10 pipelines are different, they still use the same *pipeline executing characteristics* as an ARM7.
- Code written for the ARM7 will execute on an ARM9 or ARM10.

Pipeline Executing Characteristics



ARM instruction sequence.



Pipeline Executing Characteristics

- In the execute stage, the *pc* always points to the address of the instruction plus 8 bytes.
- In other words, the *pc* always points to the address of the instruction being executed plus two instructions ahead. When the *pc* is used for calculating a relative offset and is an architectural characteristic across all the pipelines.
- Note: when the processor is in Thumb state the *pc* is the instruction address plus 4.
- There are three other characteristics of the pipeline worth mentioning.
- **First,** the execution of a branch instruction or branching by the direct modification of the *pc* causes the ARM core to flush its pipeline.
- **Second,** ARM10 uses branch prediction, which reduces the effect of a pipeline flush by predicting possible branches and loading the new branch address prior to the execution of the instruction.
- **Third,** an instruction in the execute stage will complete even though an interrupt has been raised. Other instructions in the pipeline will be abandoned, and the processor will start filling the pipeline from the appropriate entry in the vector table.

PIPELINING

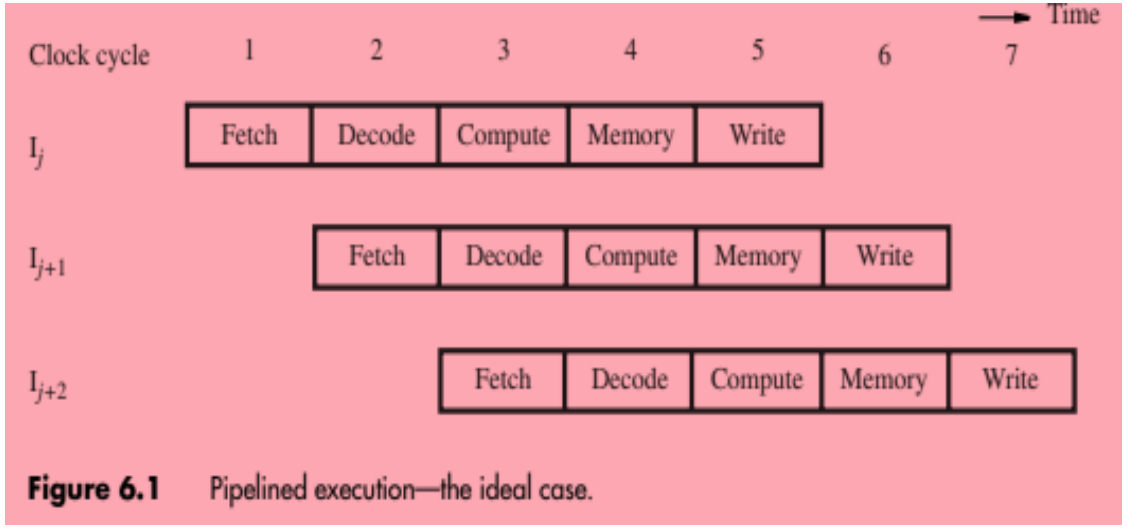
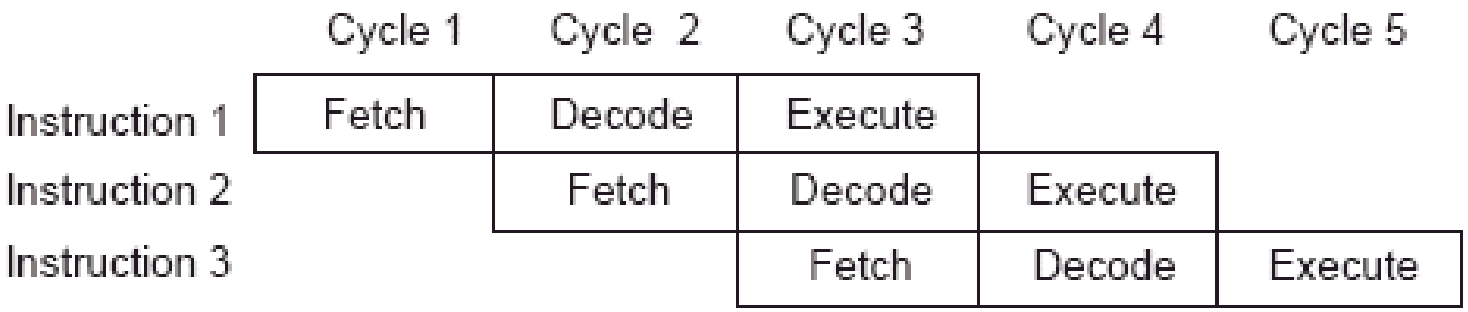


Figure 6.1 Pipelined execution—the ideal case.

- Pipelining is a particularly effective way of organizing concurrent activity in a computer system. The basic idea is very simple. It is frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly-line operation.
- The five-stage processor, takes five clock cycles to complete the execution of each instruction. Rather than wait until each instruction is completed, instructions can be fetched and executed in a pipelined manner, as shown in Figure.
- The five stages are labelled as Fetch, Decode, Compute, Memory, and Write.

PIPELINING – ARM 7

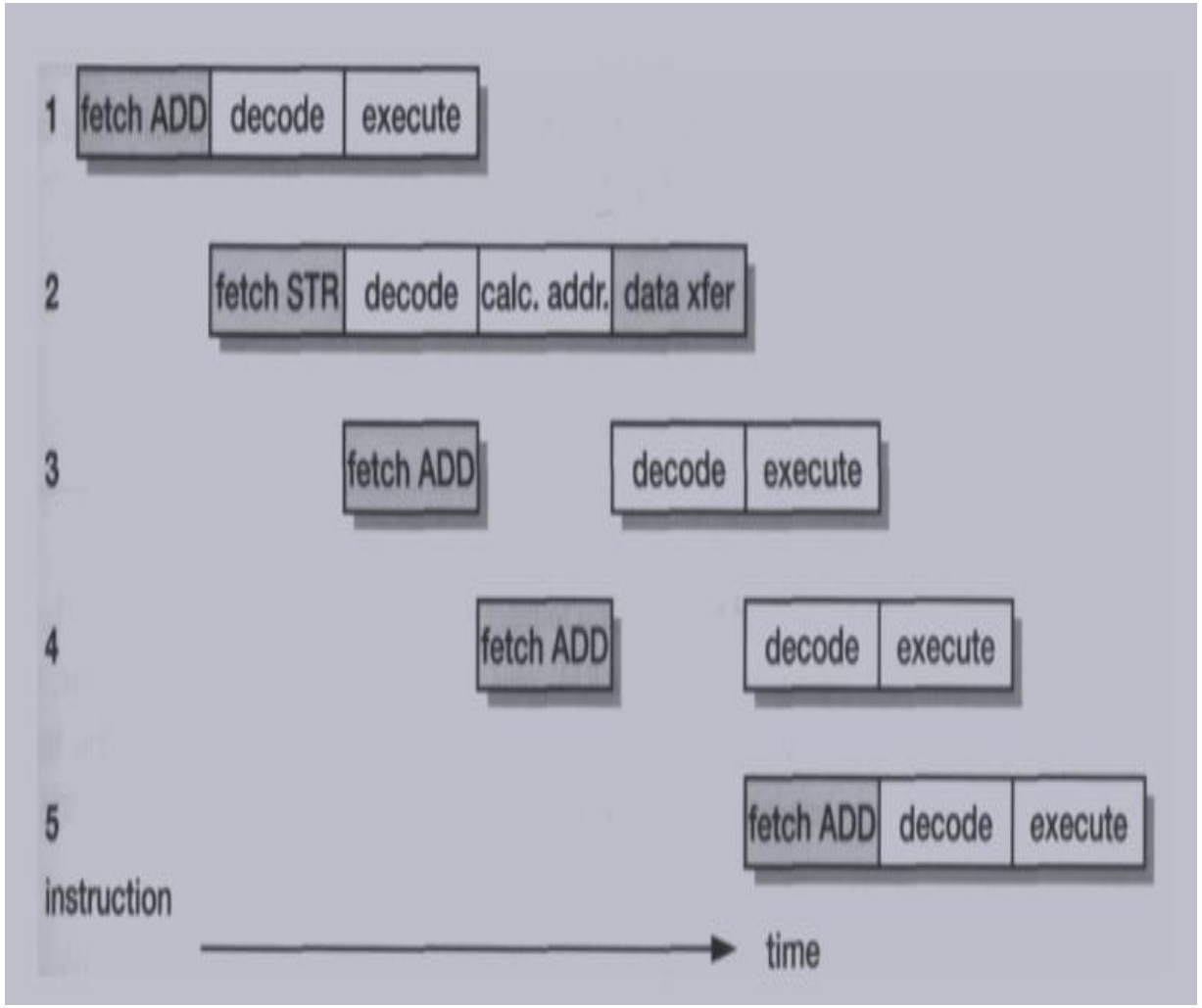


Three stage pipeline

There are three stages in this pipeline method:

1. **Fetch** – The instruction is fetched from the memory and stored in the instruction register.
2. **Decode** – The instruction is moved to the decoder which decodes the instruction. It activates the appropriate control signals and takes the necessary steps for the the next execution stage.
3. **Execute** – The instruction is executed. Data transfer, logical and arithmetic operations all take place during this stage.

PIPELINING – ARM 7



- For simple instructions, the execute stage last only one clock cycle. But there are some instructions which has a multi-cycle execution stage. For example. STR
- In this example, the first instruction (ADD) is fetched during the first cycle. The second instruction (STR) is fetched during the second cycle. During the third cycle, there is simultaneous decoding of the second instruction and the fetching of the third instruction (ADD). But the second instruction STR is a multi-cycle execute instruction. Hence there is a temporary stalling of the pipeline during the fourth cycle. During the fifth cycle, the pipeline resumes. But because both the third and fourth instructions are in their decode stage, the pipeline stalls for the fourth instruction. The decode of the third instruction takes place during the fifth cycle and the decode of the fourth instruction takes place during the sixth cycle. The rest of the pipeline happens normally.

Advantages of pipe-lining:

1. It increases the instruction throughput. The time it takes to complete an instruction doesn't change but the number of simultaneous instructions that can be processed increases with increase in pipe-lining. It reduces the delay between completed instructions.
2. CPU's ALU can be designed to work faster. But this requires complex hardware.
3. It increases the performance of the processor.

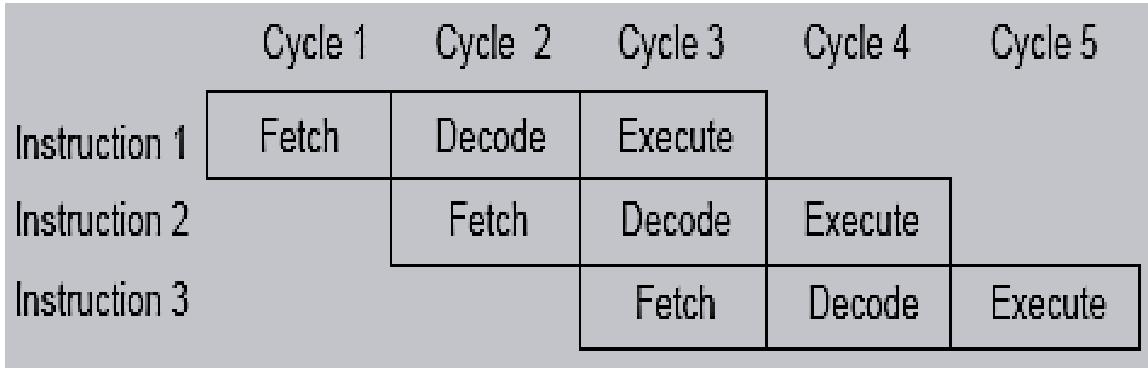
Disadvantages of pipe-lining:

1. It is more complex and more expensive to build.
2. In a pipelined processor, insertion of flip flops between modules increases the instruction latency compared to a non-pipelining processor.
3. The instruction throughput is difficult to predict.
4. The occurrence of a branching instruction flushes (erases) the entire pipeline.
5. Not all instructions are independent of each other. The output of one instruction maybe the input to another instruction. In such cases, stalling of the pipe-lining is required so that one instruction has completed execution. The output of this instruction is needed for the subsequent dependent instruction.
6. When writing assembly code, it is assumed that the one instruction is executed after another. But when this assumption is not validated by the pipelining, the program behaves unexpectedly or incorrectly causing situations known as **hazards**.

EXAMPLE:

```
        LDR R1, =0x01
        ADD R1, R1, #4
        SUBEQ R1, R1, #4
        SUB R1, R1, #7
        ADDMI R1, R1, #10
        ADDS R1, R1, #2
        ADDEQ R1, R1, #3
LOOP    B LOOP
```

- **Interpret and evaluate the three-stage pipeline for the processing of following sequence of instructions.**



EXAMPLE:

- The first instruction (LDR R1, =0x01) is fetched from memory.
- The second instruction (ADD R1, R1, #4) is fetched while the first instruction is decoded.
- The third instruction (SUBEQ R1, R1, #4) is fetched, the second instruction is decoded, and the first instruction is executed.
- This pattern continues such that in each clock cycle, each stage of the pipeline processes a different instruction. By Cycle 3, the pipeline is fully loaded, meaning that each stage of the pipeline is processing a different instruction simultaneously.

EXAMPLE:

- pipeline

Fetch	Decode	Execute
LDR R1, =0x01	-	-
ADD R1,R1,#4	LDR R1,=0x01	-
SUBEQ R1,R1,#4	ADD R1,R1,#4	LDR R1,=0x01
SUB R1,R1,#7	SUBEQ R1,R1,#4	ADD R1,R1,#4
ADDMI R1,R1,#10	SUB R1,R1,#7	SUBEQ R1,R1,#4
ADDS R1,R1,#2	ADDMI R1,R1,#10	SUB R1,R1,#7
ADDEQ R1,R1,#3	ADDS R1,R1,#2	ADDMI R1,R1,#10
B DN	ADDEQ R1, R1,#3	ADDS R1,R1,#2
-	B DN	ADDEQ R1, R1,#3
-	-	B DN

EXAMPLE:

- pipeline

Fetch	Decode	Execute
LDR R1, =0x01	-	-
ADD R1,R1,#4	LDR R1,=0x01	-
SUBEQ R1,R1,#4	ADD R1,R1,#4	LDR R1,=0x01
SUB R1,R1,#7	SUBEQ R1,R1,#4	ADD R1,R1,#4
ADDMI R1,R1,#10	SUB R1,R1,#7	SUBEQ R1,R1,#4
ADDS R1,R1,#2	ADDMI R1,R1,#10	SUB R1,R1,#7
ADDEQ R1,R1,#3	ADDS R1,R1,#2	ADDMI R1,R1,#10
B DN	ADDEQ R1, R1,#3	ADDS R1,R1,#2
-	B DN	ADDEQ R1, R1,#3
-	-	B DN

- The first instruction, LDR R1, =0x01, loads the value 1 into register R1. This instruction takes one cycle to complete.
- The second instruction, ADD R1, R1, #4, adds 4 to the value in register R1. This instruction also takes one cycle to complete.
- The third instruction, SUBEQ R1, R1, #4, subtracts 4 from the value in register R1 if the value in register R1 is equal to 0.
- This instruction takes two cycles to complete, because it has to check the value in register R1 before it can subtract 4.

EXAMPLE:

- The fourth instruction, SUB R1, R1, #7, subtracts 7 from the value in register R1. This instruction takes one cycle to complete.
- The fifth instruction, ADDMI R1, R1, #10, adds 10 to the value in register R1 if the value in register R1 is less than 0. This instruction takes two cycles to complete, because it has to check the value in register R1 before it can add 10.
- The sixth instruction, ADDS R1, R1, #2, adds 2 to the value in register R1. This instruction takes one cycle to complete.
- The seventh instruction, ADDEQ R1, R1, #3, adds 3 to the value in register R1 if the value in register R1 is equal to 0. This instruction also takes two cycles to complete, because it has to check the value in register R1 before it can add 3.
- The eighth instruction, LOOP, is a branch instruction that jumps back to the beginning of the loop. This instruction takes one cycle to complete. The ninth instruction, B LOOP, is another branch instruction that jumps back to the beginning of the loop. This instruction also takes one cycle to complete.

EXAMPLE:

- The total number of cycles required to execute the instructions is 14. This is because some of the instructions have to wait for the results of other instructions before they can be executed. For example, the third instruction has to wait for the results of the second instruction before it can subtract 4.
- The three-stage pipeline can improve the performance of the processor by overlapping the execution of multiple instructions. For example, while the third instruction is waiting for the results of the second instruction, the fourth instruction can be fetched and decoded. This can significantly reduce the amount of time it takes to execute a sequence of instructions.

PIPELINING

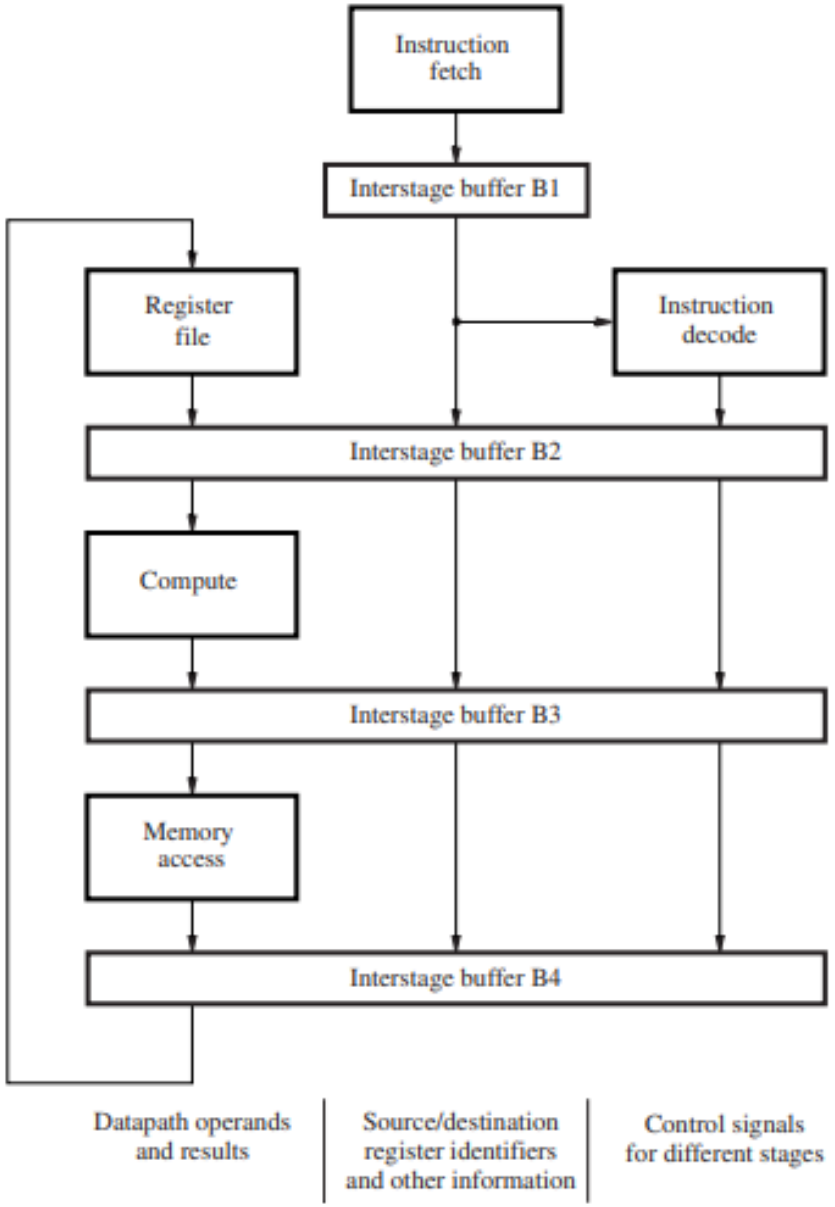


Figure 6.2 A five-stage pipeline.

- Interstage buffer B1 feeds the Decode stage with a newly-fetched instruction.
- Interstage buffer B2 feeds the Compute stage with the two operands read from the register file, the source/destination register identifiers, the immediate value derived from the instruction, the incremented PC value used as the return address for a subroutine call, and the settings of control signals determined by the instruction decoder. The settings for control signals move through the pipeline to determine the ALU operation, the memory operation, and a possible write into the register file.
- Interstage buffer B3 holds the result of the ALU operation, which may be data to be written into the register file or an address that feeds the Memory stage. In the case of a write access to memory, buffer B3 holds the data to be written. These data were read from the register file in the Decode stage. The buffer also holds the incremented PC value passed from the previous stage, in case it is needed as the return address for a subroutine-call instruction.
- Interstage buffer B4 feeds the Write stage with a value to be written into the register file. This value may be the ALU result from the Compute stage, the result of the Memory access stage, or the incremented PC value that is used as the return address for a subroutine-call instruction.


Thank you!