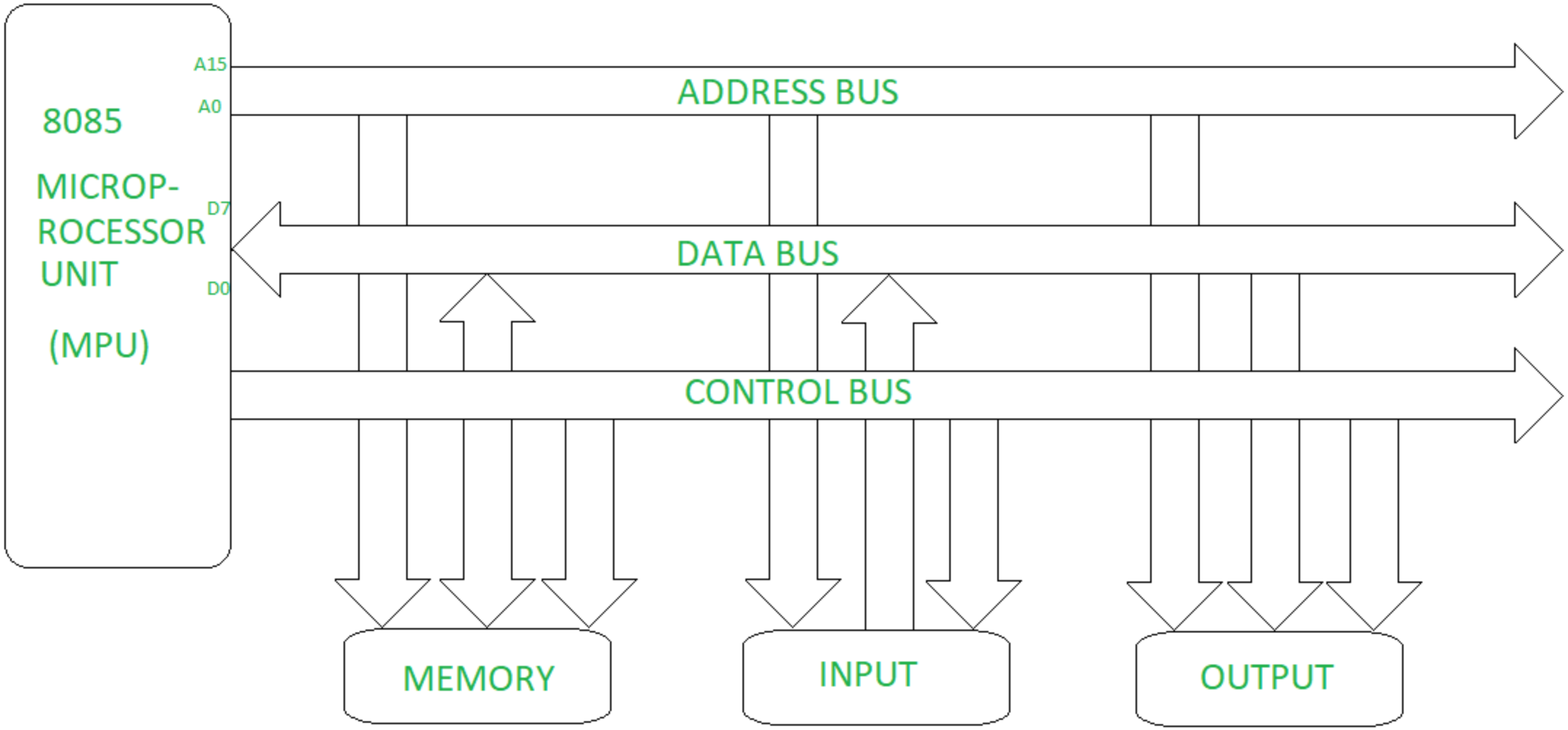
UNIT 1 : CPU ORGANIZATION

PROF. SHANKAR MALI

BUS ORGANIZATION OF 8085 MICROPROCESSOR

- The bus organization of the 8085 microprocessor is the way in which the microprocessor communicates with other devices in a computer system. The 8085 microprocessor has a 16-bit address bus, an 8-bit data bus, and various control signals that are used to manage data transfer and other operations.
- The address bus is used to specify the memory location or device with which the microprocessor wants to communicate. It is 16 bits wide, which allows the microprocessor to address up to 64K bytes of memory. The address bus is unidirectional, which means that data can only flow in one direction from the microprocessor to the addressed device.

- The data bus is used to transfer data between the microprocessor and other devices. It is 8 bits wide, which means that data can be transferred in byte-sized chunks. The data bus is bidirectional, which means that data can flow in either direction between the microprocessor and other devices.
- In addition to the address and data buses, the 8085 microprocessor has various control signals that are used to manage data transfer and other operations. These control signals include the read (RD), write (WR), and hold (HLDA) signals, among others. The RD and WR signals are used to control data transfer to and from memory or other devices, while the HLDA signal is used to indicate that the microprocessor is in a hold state and cannot execute instructions.
- Bus is a group of conducting wires which carries information, all the peripherals are connected to microprocessor through Bus. Diagram to represent bus organization system of 8085 Microprocessor



Bus organization system of 8085 Microprocessor

WHY USE BUS ORGANIZATION IN 8085 MICROPROCESSOR ?

- There are several reasons why bus organization is used in the 8085 microprocessor:

1. Memory access: The bus organization is used for accessing memory by transferring the address of the memory location through the address bus and the data to be stored or retrieved through the data bus. This enables the microprocessor to read and write data to and from memory, which is essential for executing instructions and storing data.

2. I/O operations: The bus organization is used for performing input/output (I/O) operations by transferring the input/output device address through the address bus and the data to be input or output through the data bus. This enables the microprocessor to communicate with peripheral devices such as keyboards, displays, and sensors.

3. Interrupt handling: The bus organization is used for interrupt handling, where the microprocessor uses the address bus to fetch the interrupt vector and the data bus to fetch the interrupt service routine. This enables the microprocessor to respond to external events and perform time-critical operations.

4. DMA operations: The bus organization is used for performing Direct Memory Access (DMA) operations, where the data transfer between the memory and I/O devices takes place without the intervention of the microprocessor. This enables high-speed data transfer between devices and reduces the load on the microprocessor.

5. Control signal transfer: The bus organization is used for transferring control signals between the microprocessor and other components of the system. This enables the microprocessor to control the operation of devices and coordinate the execution of instructions.

ADDRESS BUS –

- The address bus is a unidirectional bus that is used to carry the memory or I/O device address to which the data is to be transferred. The address bus in the 8085 microprocessor is 16-bit wide.
- It is a group of conducting wires which carries address only. Address bus is unidirectional because data flow in one direction, from microprocessor to memory or from microprocessor to Input/output devices (That is, Out of Microprocessor). Length of Address Bus of 8085 microprocessor is 16 Bit (That is, Four Hexadecimal Digits), ranging from 0000 H to FFFF H, (H denotes Hexadecimal). The microprocessor 8085 can transfer maximum 16 bit address which means it can address 65, 536 different memory location. The Length of the address bus determines the amount of memory a system can address.Such as a system with a 32-bit address bus can address 2^32 memory locations.If each memory location holds one byte, the addressable memory space is 4 GB.However, the actual amount of memory that can be accessed is usually much less than this theoretical limit due to chipset and motherboard limitations.

DATA BUS –

- The data bus is an 8-bit bidirectional bus that is used to transfer data between the microprocessor and other components such as memory and I/O devices. It is used to carry data to or from the memory or input/output devices.
- It is a group of conducting wires which carries Data only.Data bus is bidirectional because data flow in both directions, from microprocessor to memory or Input/Output devices and from memory or Input/Output devices to microprocessor. Length of Data Bus of 8085 microprocessor is 8 Bit (That is, two Hexadecimal Digits), ranging from 00 H to FF H. (H denotes Hexadecimal). When it is write operation, the processor will put the data (to be written) on the data bus, when it is read operation, the memory controller will get the data from specific memory block and put it into the data bus. The width of the data bus is directly related to the largest number that the bus can carry, such as an 8 bit bus can represent 2 to the power of 8 unique values, this equates to the number 0 to 255.A 16 bit bus can carry 0 to 65535.

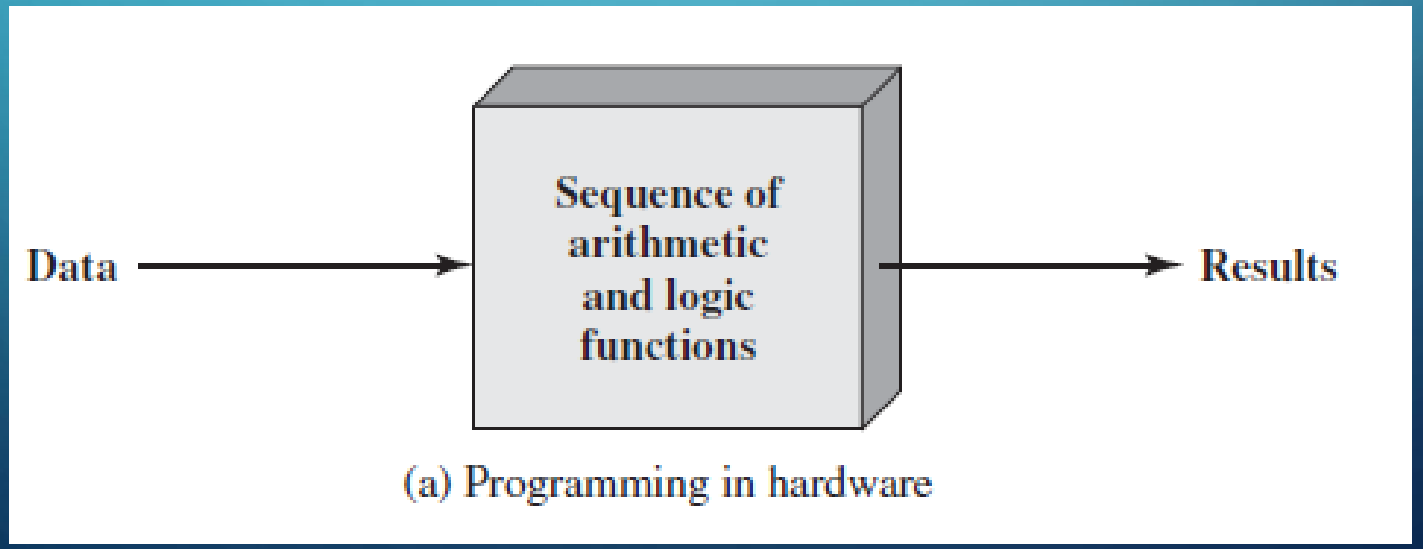
CONTROL BUS –

- The control bus is a bidirectional bus that is used to carry control signals between the microprocessor and other components such as memory and I/O devices. It is used to transmit commands to the memory or I/O devices for performing specific operations.
- It is a group of conducting wires, which is used to generate timing and control signals to control all the associated peripherals, microprocessor uses control bus to process data, that is what to do with selected memory location. Some control signals are:
- Memory read
- Memory write

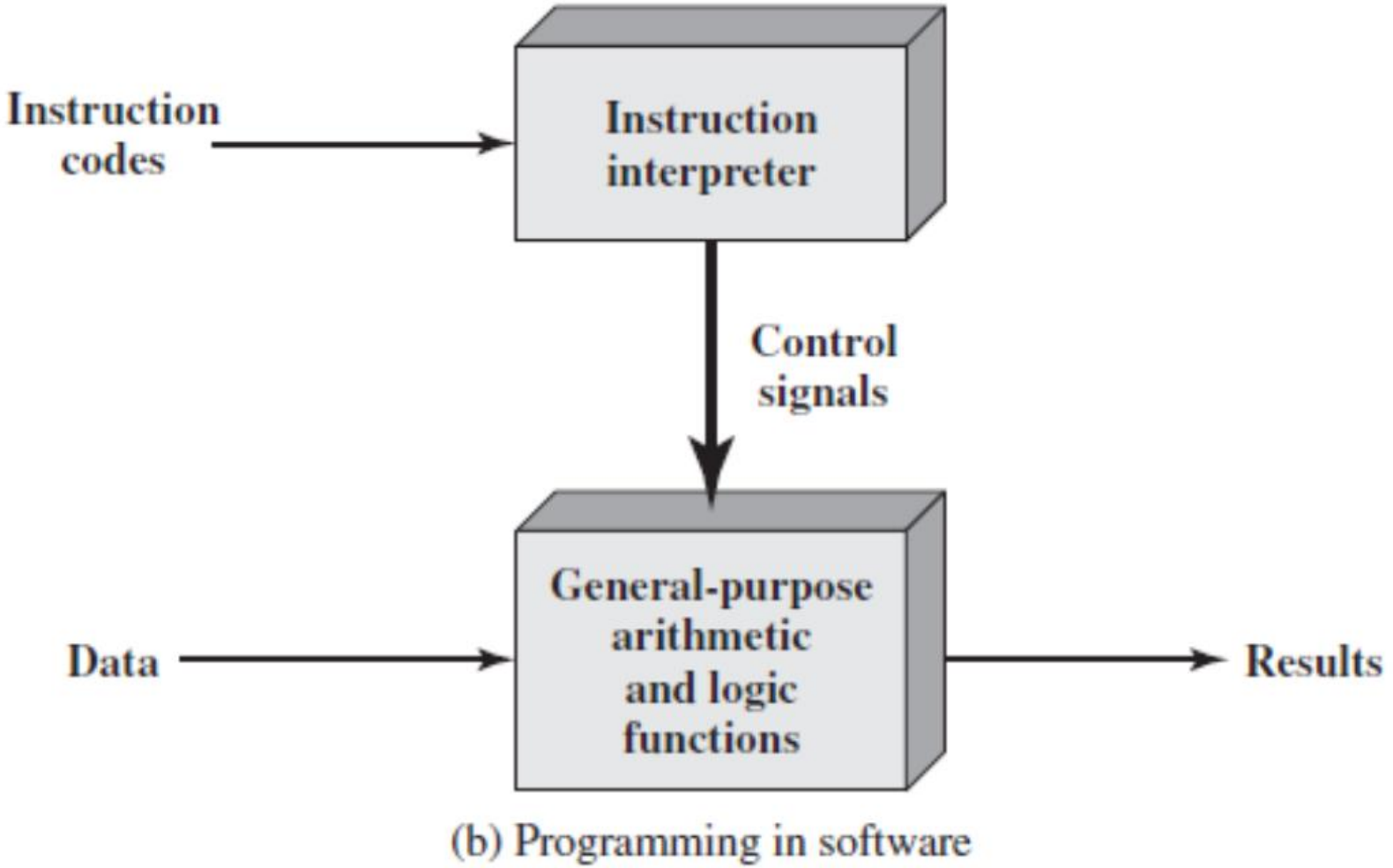
1. I/O read
2. I/O Write
3. Opcode fetch

COMPUTER COMPONENTS

The process of connecting the various components in the desired configuration as a form of programming. the resulting “program” is in the form of hardware and is termed a *hardwired program*



(a) Programming in hardware



(b) Programming in software

Figure 3.1 Hardware and Software Approaches

- Figure 3.1b indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions.
- These two constitute the CPU. Several other components are needed to yield a functioning computer.
- Data and instructions must be put into the system. For this we need some sort of input module.
- This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system. A means of reporting results is needed, and this is in the form of an output module.
- Taken together, these are referred to as I/O components

- One more component is needed. An input device will bring instructions and data in sequentially. But a program is not invariably executed sequentially; it may
- jump around (e.g., jump instruction). Similarly, operations on data may require access to more than just one element at a time in a predetermined sequence.
- Thus, there must be a place to store temporarily both instructions and data.
- That module is called memory, or main memory to distinguish it from external storage or peripheral devices.

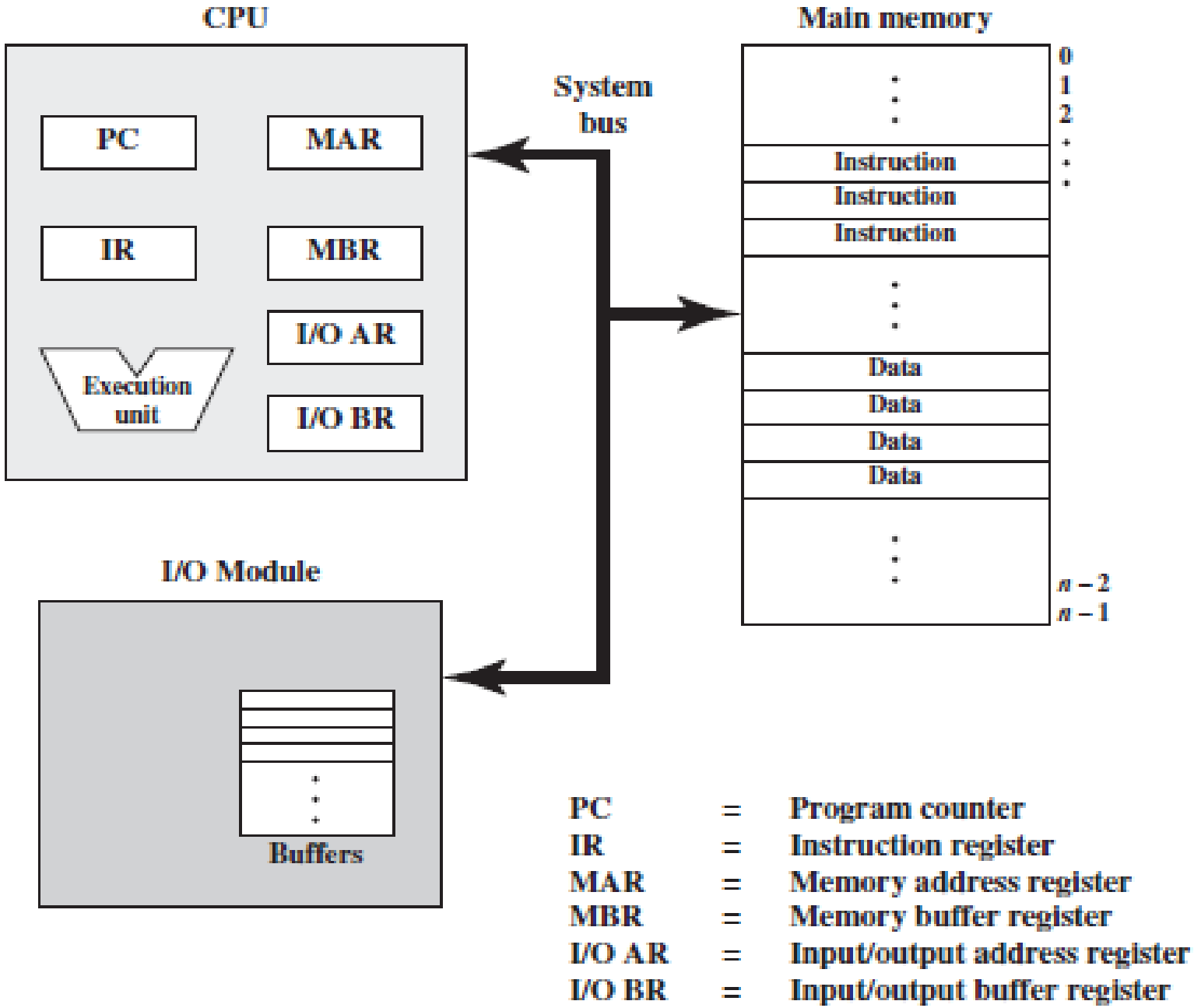


Figure 3.2 Computer Components: Top-Level View

Figure 3.2 illustrates these top-level components and suggests the interactions among them.

- The CPU exchanges data with memory. For this purpose, it typically makes use of two internal (to the CPU) registers: a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory or receives the data read from memory.
- Similarly, an I/O address register (I/O AR) specifies a particular I/O device. An I/O buffer (I/O BR) register is used for the exchange of data between an I/O module and the CPU.
- A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data. An I/O module transfers data from external devices to
- CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.

COMPUTER FUNCTION

- The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.
- The processor does the actual work by executing instructions specified in the program
- In its simplest form, instruction processing consists of two steps: The processor reads (*fetches*) instructions from memory one at a time and executes each instruction.
- Program execution consists of repeating the process of instruction fetch and instruction execution.

- The processing required for a single instruction is called an instruction cycle.
- Using the simplified two-step description given previously, the instruction cycle is depicted in Figure 3.3.
- The two steps are referred to as the fetch cycle and the execute cycle.
- Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered

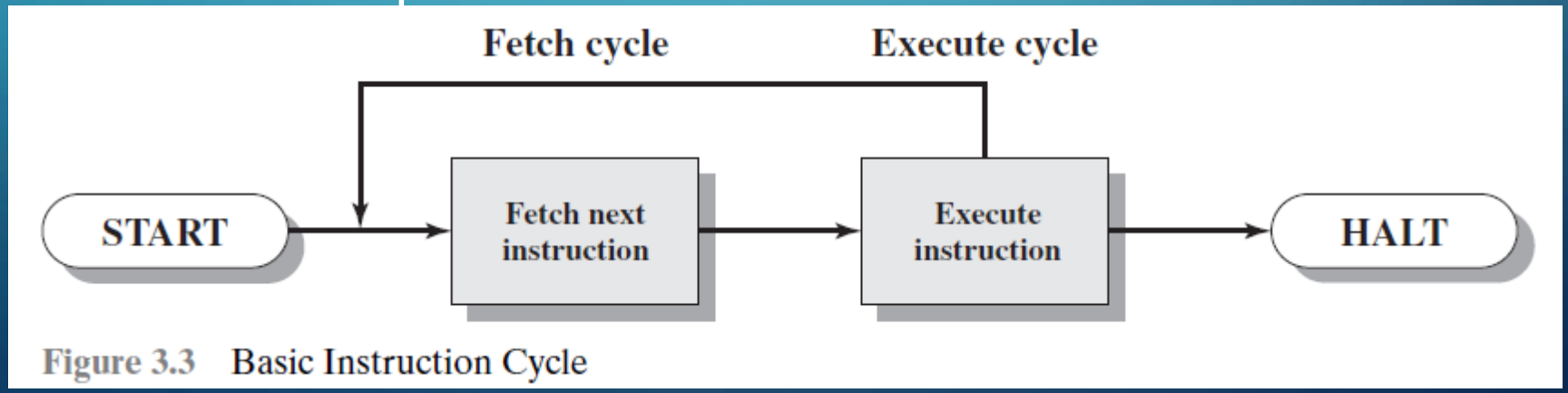


Figure 3.3 Basic Instruction Cycle

INSTRUCTION FETCH AND EXECUTE

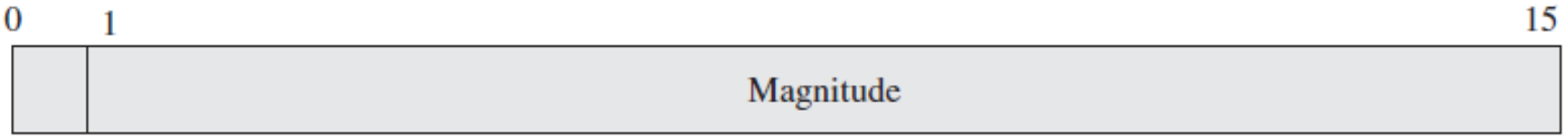
- At the beginning of each instruction cycle, the processor fetches an instruction from memory. **In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next.**
- The processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence (i.e., the instruction located at the next higher memory address).
- So, for example, consider a computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to location 300.The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on.

The fetched instruction is loaded into a register in the processor known as the instruction register (IR). The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action. In general, these actions fall into four categories:

- **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
- **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- **Data processing:** The processor may perform some arithmetic or logic operation on data.
- **Control:** An instruction may specify that the sequence of execution be altered. For example, the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor will remember this fact by setting the program counter to 182. Thus, on the next fetch cycle, the instruction will be fetched from location 182 rather than 150.

Consider a simple example using a hypothetical machine that includes the characteristics listed in Figure 3.4. The processor contains a single data register, called an accumulator (AC). Both instructions and data are 16 bits long. Thus, it is convenient to organize memory using 16-bit words. The instruction format provides 4 bits for the opcode, so that there can be as many as $2^4 = 16$ different opcodes, and up to $2^{12} = 4096$ (4K) words of memory can be directly addressed.

(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

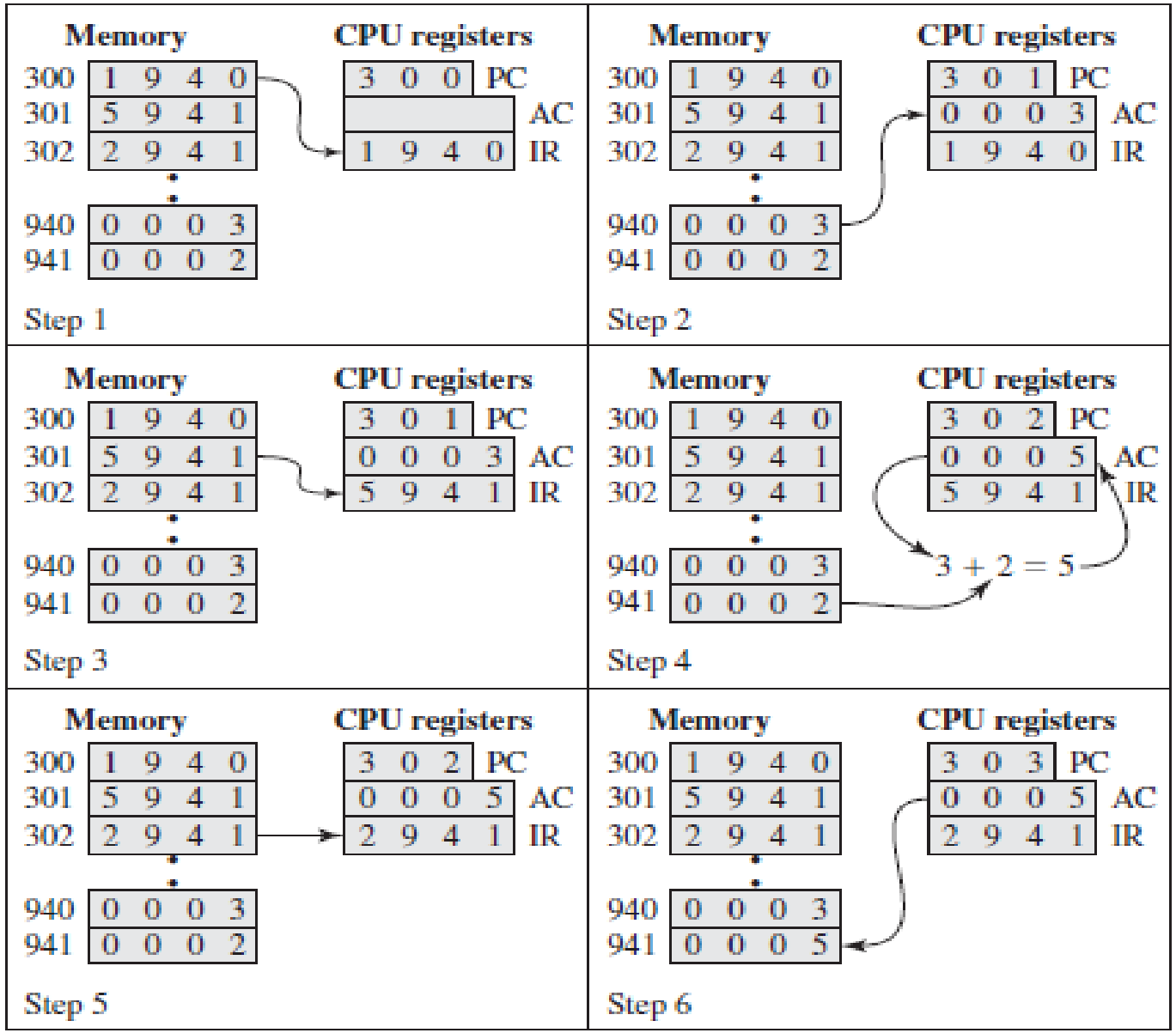


Figure 3.5 Example of Program Execution (contents of memory and registers in hexadecimal)

HYPOTHETICAL OPCODES

Arithmetic Instructions	
Instruction	Opcode
ADD	1
SUB	10
MUL	11
DIV	100
Data Transfer Instructions	
LOAD	101
STORE	110
MOVE	111

Control Instructions	
JMP	1000
JZ	1001
CALL	1010
RET	1011
Logical Instructions	
AND	1100
OR	1101
XOR	1110
NOT	1111

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IR and the PC is incremented. Note that this process involves the use of a memory address register (MAR) and a memory buffer register (MBR). For simplicity, these intermediate registers are ignored.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hexadecimal digits) specify the address (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

INTERRUPTS

- Interrupt is an interruption of the normal sequence of execution.\
- When the interrupt processing is completed, execution resumes.
- The processor and the operating system are responsible for suspending the user program and then resuming it at the same point.
- To accommodate interrupts, an interrupt cycle is added to the instruction cycle.
- In the interrupt cycle, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program.

INTERRUPTS

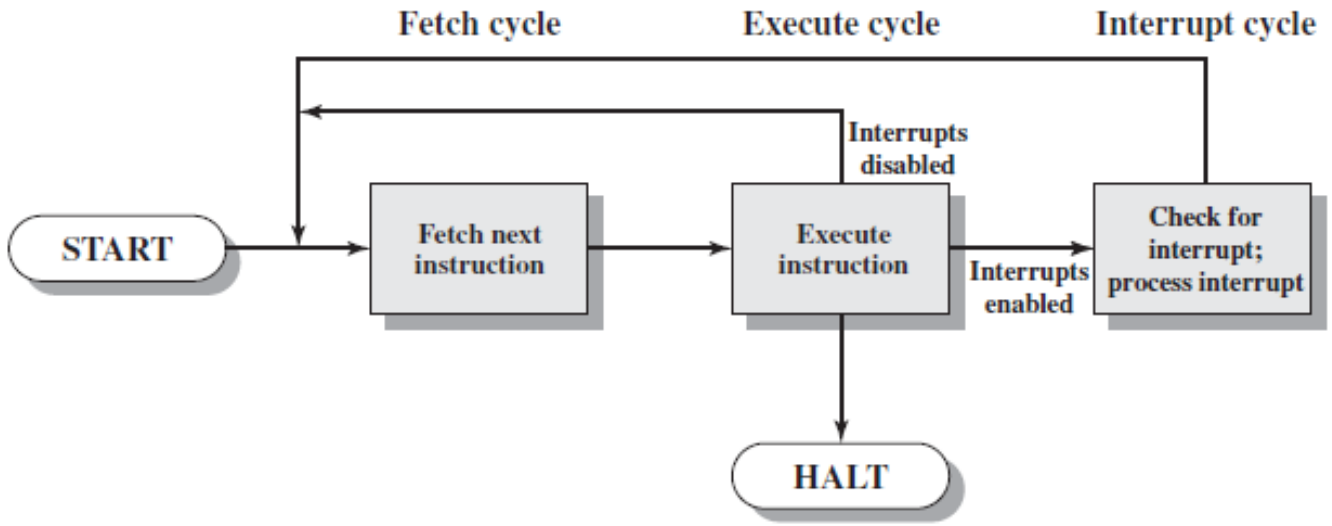


Figure 3.9 Instruction Cycle with Interrupts

INTERRUPTS

- If an interrupt is pending, the processor does the following:
  - It suspends execution of the current program being executed and saves its context. This means saving the address of the next instruction to be executed (current contents of the program counter) and any other data relevant to the processor's current activity.
  - It sets the program counter to the starting address of an *interrupt handler* routine.
- The processor now proceeds to the fetch cycle and fetches the first instruction in the interrupt handler program, which will service the interrupt.
- When the interrupt handler routine is completed, the processor can resume execution of the user program at the point of interruption.

MULTIPLE INTERRUPTS

- Two approaches can be taken to dealing with multiple interrupts.

1. The first is to disable interrupts while an interrupt is being processed.

- A disabled interrupt simply means that the processor can and will ignore that interrupt request signal.
- If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has enabled interrupts.
- Thus, when a user program is executing and an interrupt occurs, interrupts are disabled immediately.
- After the interrupt handler routine completes, interrupts are enabled before resuming the user program, and the processor checks to see if additional interrupts have occurred.

WORKFLOW OF INTERRUPTS

1. Interrupt Generation: A hardware device (e.g., keyboard, mouse, disk) or software raises an interrupt signal. This signal is sent to the CPU via the interrupt controller (e.g., Programmable Interrupt Controller, or PIC).
2. Interrupt Request (IRQ): The CPU checks for interrupt signals at specific points, usually after completing an instruction. If interrupts are enabled, the CPU acknowledges the signal.
3. Saving Context: Before handling the interrupt, the CPU saves the current execution state (program counter, registers, flags) to a stack or memory.
4. ISR Execution: The CPU locates the ISR by using: Vector Table: Contains addresses of ISRs for vectored interrupts and Fixed Address: Used for non-vectored interrupts.
5. Restoring Context: Once the ISR completes, the CPU restores the saved execution state from the stack.
6. Resuming Program: The CPU resumes execution of the interrupted program.

STACK BASED CPU ORGANIZATION

In a stack-based CPU organization, the central processing unit (CPU) uses a stack as the primary structure to perform operations. All instructions operate implicitly on the stack, and operands are pushed to or popped from the stack rather than being accessed directly from registers or memory.

Stack as the Core:

All operations use a stack to store operands and results.

Operands are pushed (stored) onto the stack and popped (retrieved) from it.

Implicit Addressing:

Instructions do not specify operands explicitly.

Operands are implicitly at the top of the stack (TOS).

No General-Purpose Registers:

Unlike register-based architectures, stack CPUs rely entirely on the stack for computation.

WORKFLOW OF INTERRUPTS

Simple Instruction Set:

Instructions are compact and fewer in number, as they operate implicitly on the stack.

Basic Operations

PUSH:

Adds (pushes) a value onto the stack.

Example: PUSH 5 → Stack becomes [5].

POP:

Removes (pops) the top value from the stack.

Example: Stack [5, 10] → POP → Stack becomes [5].

Arithmetic Operations:

Operate on the top elements of the stack.

Example: ADD pops two values, adds them, and pushes the result.

LOAD and STORE:

LOAD: Pushes a value from memory to the stack.

STORE: Pops a value from the stack and stores it in memory.

WORKFLOW OF INTERRUPTS

Instruction Format

Stack-based CPUs typically have a simple instruction format, as the stack eliminates the need for operand addressing. For example:

Instruction	Description
PUSH X	Push value X onto the stack.
POP	Pop the top value from the stack.
ADD	Pop two values, add them, and push the result.
SUB	Pop two values, subtract them, and push the result.
MUL	Pop two values, multiply them, and push the result.
DIV	Pop two values, divide them, and push the result.

ADDITION OF TWO NUMBERS

Goal:

Add two numbers (5 and 3) and store the result in memory.

Stack Operations:

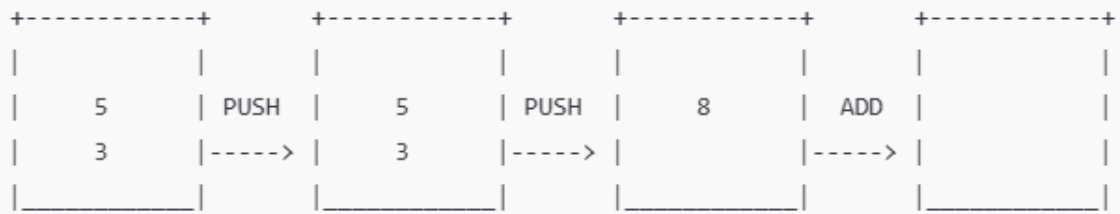
1. PUSH 5 ⟶ Stack: [5]
2. PUSH 3 ⟶ Stack: [5, 3]
3. ADD ⟶ Stack: [8]
4. POP ⟶ Result is removed from the stack and stored in memory.

STACK BASED EXECUTION

Diagram of Stack-Based Execution

plaintext

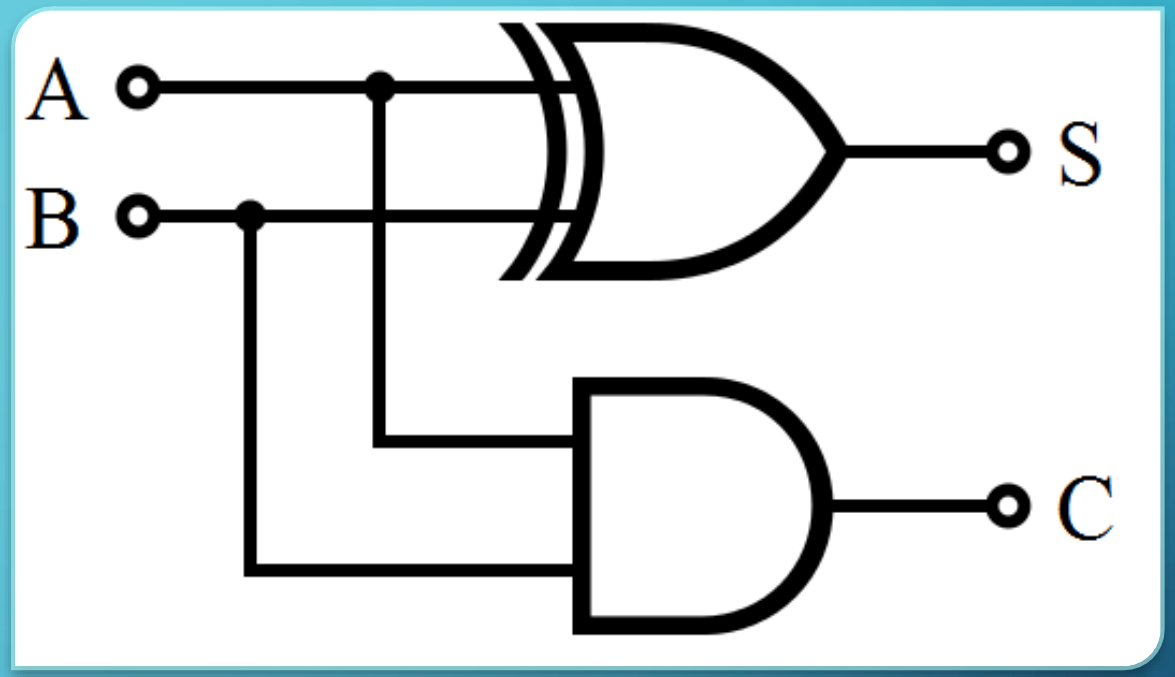
Copy code



ARITHMETIC AND LOGIC UNIT (ALU)

Half Adder

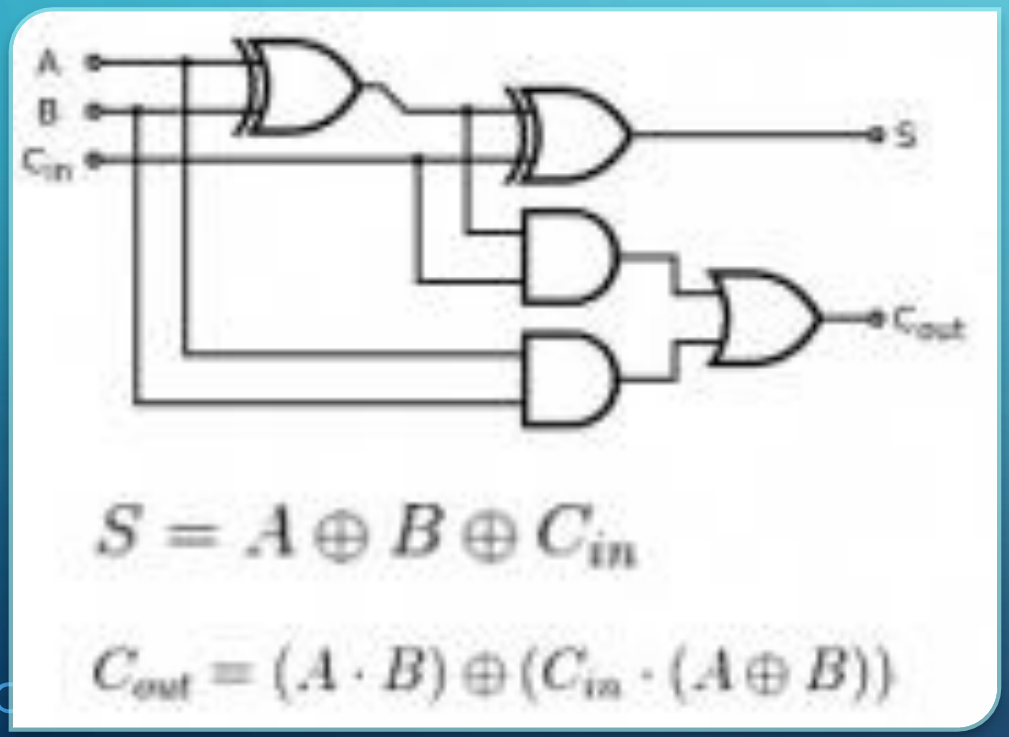
- Lets start with a simple half adder. **Half adder** adds two single binary digits *A* and *B*. It has two outputs, sum (*S*) and carry (*C*). The carry signal represents an overflow into the next digit of a multi-digit addition. Figures below illustrate a simple half adder constructed from logic gates



INPUT		OUTPUT	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

FULL ADDER

Full Adder is an extension of half adder to include the Cin input as well. The truth table can be implemented to form the logic diagram as shown below.



$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = (A \cdot B) \oplus (C_{in} \cdot (A \oplus B))$$

INPUT			OUTPUT	
A	***B***	***C*^{in}**	***C*^{out}**	***S***
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

THE BASIC UNIT: 1 BIT ALU

- Lets construct a simple ALU that performs a arithmetic operation (1 bit addition)and does 3 logical operations namely AND, NOR and XOR as shown below. The multiplexer selects only one operation at a time. The operation selected depends on the selection lines of the multiplexer as shown in the truth table.

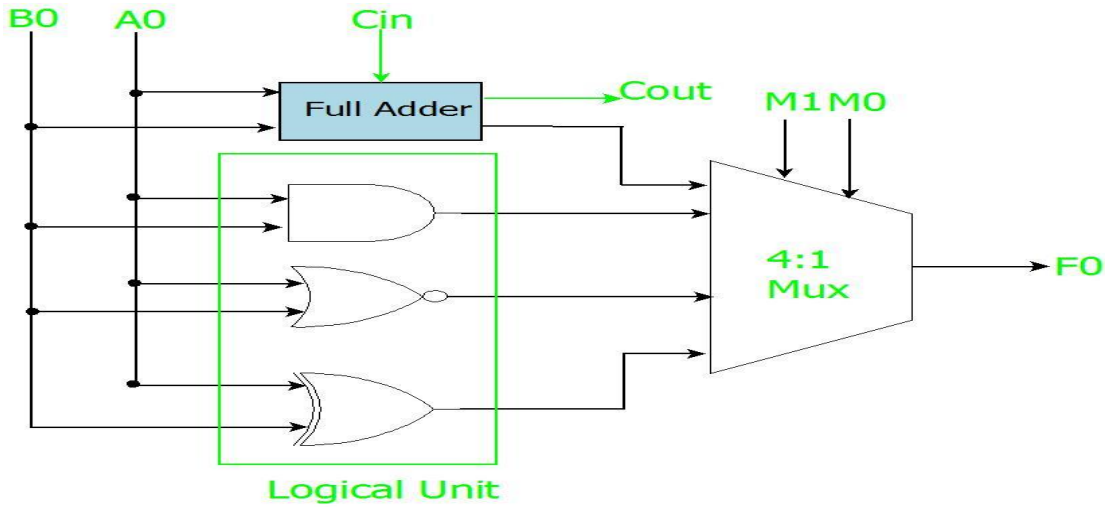
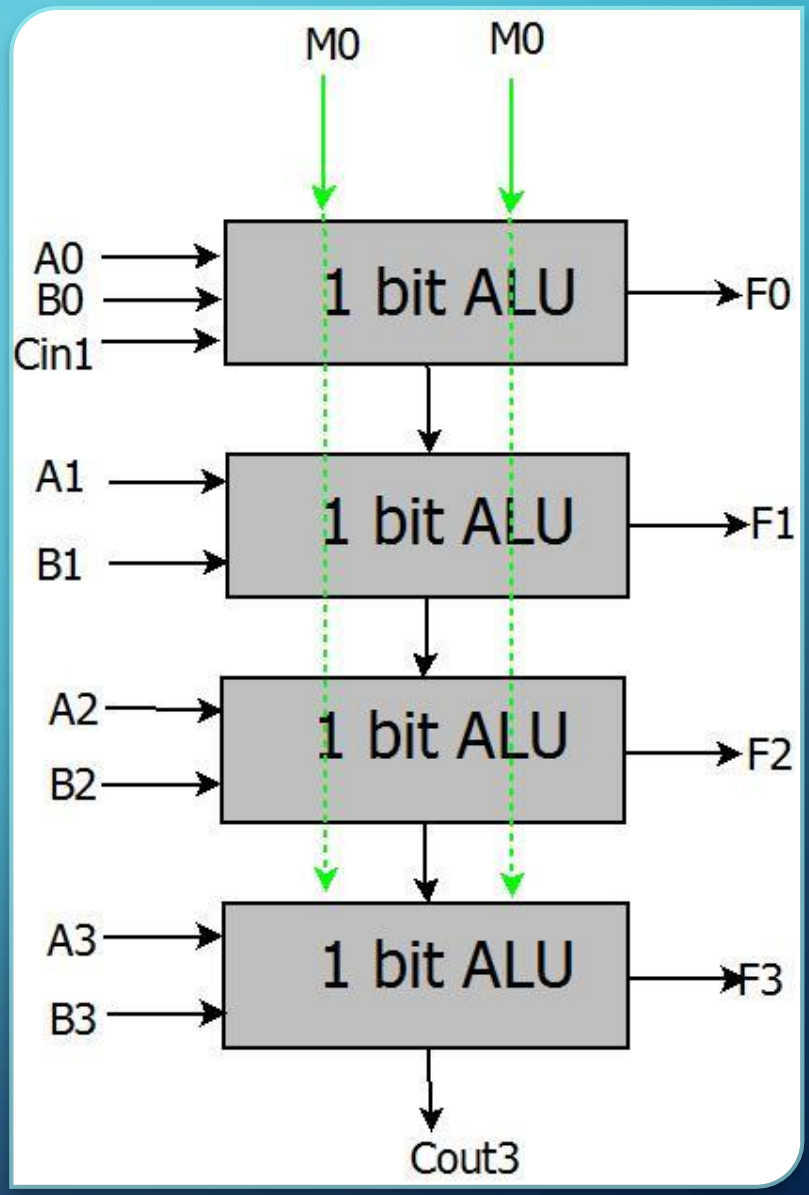


Figure: 1 bit ALU

INPUT		OUTPUT
M1	*M0*	*Operation*
0	0	SUM
1	0	AND
0	1	OR
1	1	XOR

4 BIT ALU

Now we can take up the 1 bit ALU as block and construct a 4 bit ALU, which performs all the functions of the 1 bit ALU on the 4 bit inputs. Thus a single building block can be constructed and used recursively. The inputs A and B are four bits and the output is 4 bit as well. Figure illustrates it:



IMPORTANT TAKEAWAYS

- The selection lines **MO** and **M1** select the function ALU performs. These selection lines combined with the input arguments and desired functions a ***Instruction Set*** can be formed.
- These Instructions can used to create meaningful programs. Since these are required to be easily available they can be stored on ***ROM*** unit.
- The input arguments **A** and **B** are often stored in **Internal Registers**. These along with other special purpose register form the **registers** of the microcontroller.
- ROM memories are slower in speed, hence an intermediate high speed *RAM* is often used.

ADDRESSING MODES

- Addressing modes are methods used by a CPU to access operands (data) for operations during program execution. Different addressing modes allow the processor to interpret the address of an operand in various ways. The choice of addressing mode impacts the efficiency and flexibility of the system.
- The various addressing modes can be categorized into **Data Addressing Modes, Memory Addressing Modes**, and **Stack Addressing Modes** based on how the operand is accessed. Here's how they can be classified:

DATA ADDRESSING MODE

These modes directly involve registers or immediate values where the data can be stored, retrieved, or manipulated.

- Immediate Addressing Mode: The operand is directly specified in the instruction.

  Example: MOV R0, #5 (Load the immediate value 5 into register R0)

- Register Addressing Mode: The operand is located in a register.

  Example: ADD R1, R2 (Add contents of register R1 and R2)

- Register Indirect Addressing Mode: The register contains the memory address of the operand.

  Example: MOV R1, (R2) (Fetch the operand from the address in register R2 and load it into R1)

DATA ADDRESSING MODE

- Indexed Addressing Mode: The operand's address is calculated by adding a constant index to the contents of a register.

  Example: MOV R1, [R2 + 5] (Fetch the operand from address R2 + 5)

- Base-Register Addressing Mode: The operand's address is calculated by adding an offset to a base address stored in a register.

  Example: MOV R1, [R2 + offset] (Fetch the operand from the address R2 + offset)

MEMORY ADDRESSING MODE

These modes are primarily focused on how data is accessed from memory locations.

- Direct Addressing Mode: The operand is at a specific memory address given directly in the instruction.

Example: MOV R1, [1000] (Fetch the operand from memory location 1000 into register R1)

- Indirect Addressing Mode: The instruction specifies a register that contains the address of the operand in memory.

Example: MOV R1, [R2] (Fetch the operand from the address stored in register R2)

- Absolute Addressing Mode: The operand is located at a fixed memory address directly specified in the instruction.
- Example: MOV R1, 0x2000 (Fetch the operand from memory address 0x2000 into R1)
- Memory-mapped I/O Addressing Mode: A specific memory address is mapped to an I/O device for communication with peripherals.
- Example: MOV R1, [I/O Port] (Read the value from a memory-mapped I/O port into register R1)

STACK ADDRESSING MODE

- These modes are focused on stack-based memory operations, where data is accessed in a Last-In, First-Out (LIFO) manner.
- Stack Addressing Mode: The operand is located at the top of the stack.

Example: POP R1 (Pop the top value from the stack and load it into register R1)

- Relative Addressing Mode: Used in stack operations for branch instructions where the operand address is determined by adding an offset to the Program Counter (PC).

Example: BEQ 100 (Branch to the address PC + 100)

Thank You