CS_252 Computer Organization and Assembly Language

Course Instructor: Sunbal Rani

Week No 2

Contents

1. A Simple Computer: Hardware Design & Program Execution

- Introduction
- Hardware Design of a Simple Computer
- Registers: The CPU’s Working Memory
- General Purpose Registers (GPRs)
- Special Purpose Registers

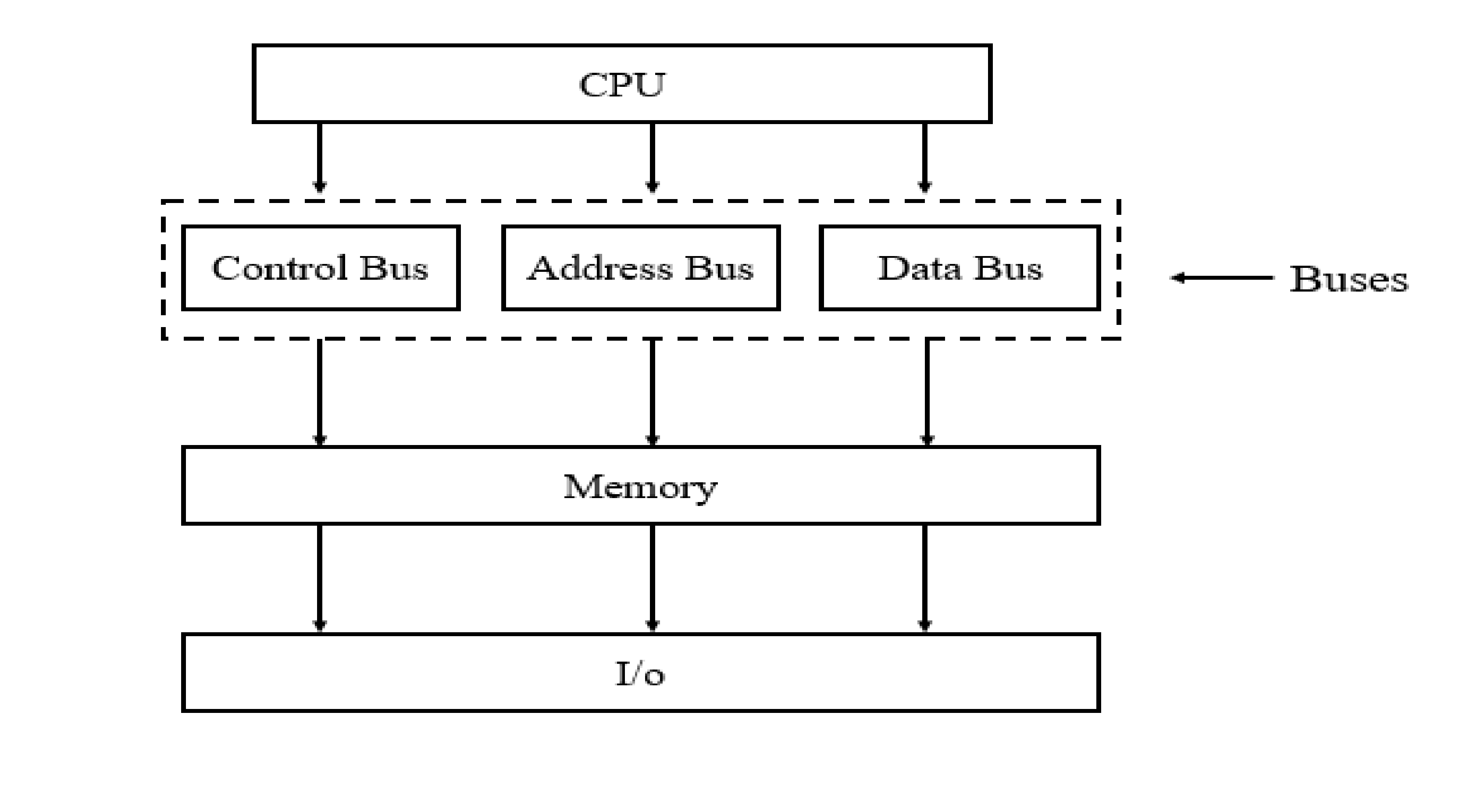
2. Program Execution Cycle

- Input/output in Assembly
- Interrupts & Context Switching
- 16-Bit ASC Computer: Hardware Design

Introduction

- Built around CPU, Memory, and I/O units
- **Registers:**
- General-purpose
- Special-purpose
- **Program Execution Cycle**: Fetch → Decode → Execute → Store
- I/O Devices: manage communication with outside world
- **Interrupts:** allow CPU to respond to urgent tasks
- **Context Switching**: saves/restores process states for multitasking

Hardware Design of a Simple Computer



Hardware Design of a Simple Computer

CPU: Brain of the computer (Control Unit + ALU + Registers)

Memory: Stores instructions and data

I/O Devices: Communicate with external environment

System Bus: Connects CPU, memory, and I/O (data, address, control lines)

Registers: The CPU’s Working Memory

Registers = small, high-speed storage in CPU

Two Types:

1.General Purpose Registers (GPRs) – hold data/addresses, intermediate results.

2. Special Purpose Registers – control execution flow

General Purpose Registers (GPRs)

- EAX (Accumulator): Arithmetic operations, function return values
- EBX (Base Register): Addressing memory data
- ECX (Counter Register): Loops and string operations
- EDX (Data Register): I/O operations, extended arithmetic

Special Purpose Registers

- Program Counter (PC): Address of next instruction
- Instruction Register (IR): Holds current instruction
- Stack Pointer (SP): Tracks top of stack (function calls, local data)

Register Usage Example

```
MOV EAX, 5     ; Load 5 into EAX
ADD EAX, 10    ; EAX now holds 15
```

Arithmetic operation with registers

Memory & Register Interaction

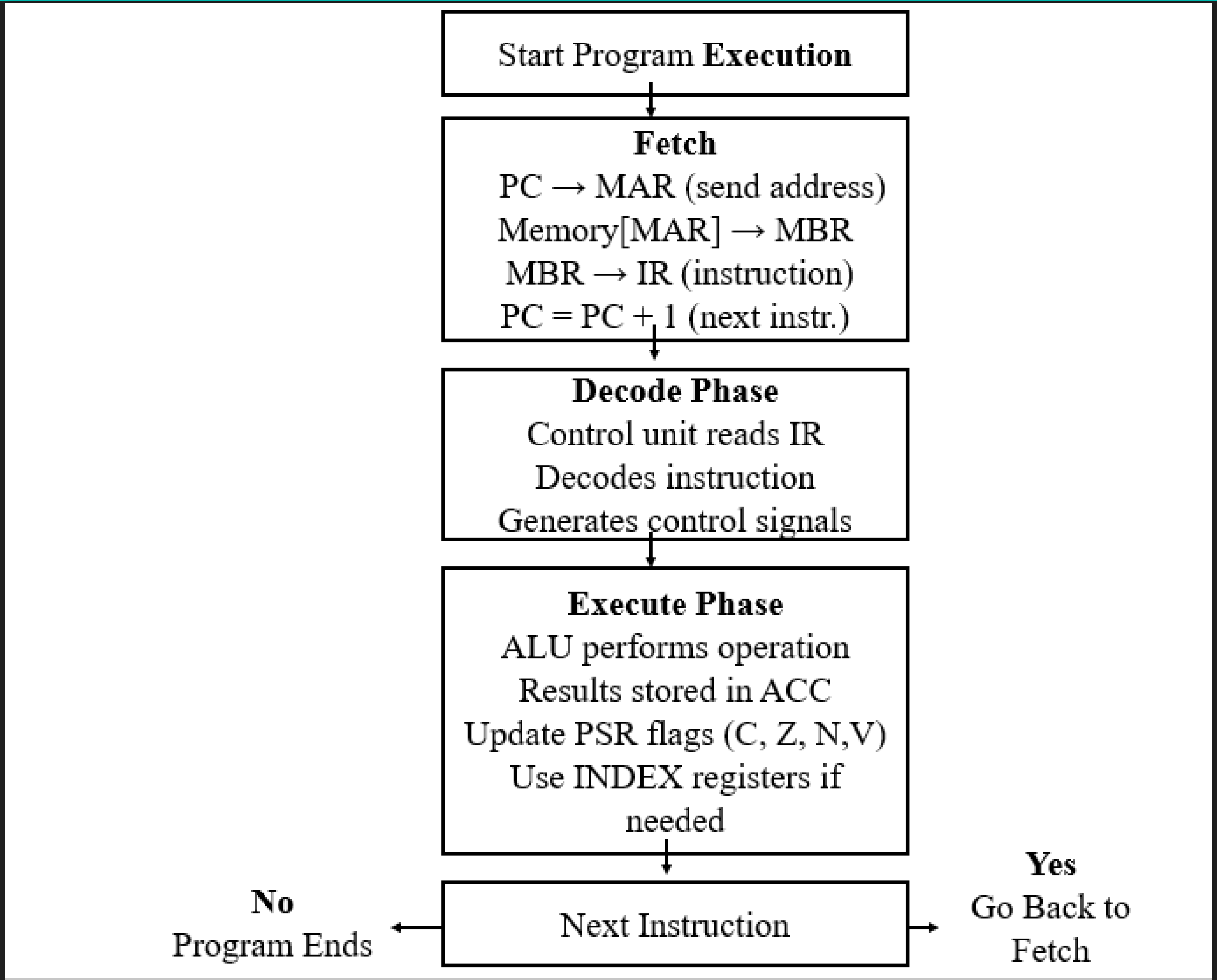
```
MOV [var], AX   ; Store AX into memory
MOV AX, [var]   ; Load from memory into AX
```

Demonstrates data transfer between “CPU registers” and “main memory”

Program Execution Cycle

- **Fetch:** Get the instruction from memory (using PC, MAR, MBR, IR).
- **Decode:** Understands the instruction (done in IR/control unit).
- **Execute:** Perform the task (using ALU, ACC, PSR, and sometimes index registers).
- Repeat until the program ends.

Program Execution Cycle



Start Program **Execution**

Fetch

PC → MAR (send address)

Memory[MAR] → MBR

MBR → IR (instruction)

PC = PC + 1 (next instr.)

Decode Phase

Control unit reads IR

Decodes instruction

Generates control signals

Execute Phase

ALU performs operation

Results stored in ACC

Update PSR flags (C, Z, N,V)

Use INDEX registers if needed

Next Instruction

No

Program Ends

Yes

Go Back to Fetch

Handling Input/output in Assembly

CPU communicates with devices (keyboard, screen, disk) via I/O operations

- Direct access to hardware is not allowed
- Special instructions or system calls are used for I/O
- Example: INT 21h – DOS interrupt for device or file I/O

Handling Input/output in Assembly

Common uses:

File handling (read/write files)

Device input/output (keyboard, display, printers)

Memory management functions

Ensures safe and efficient interaction with hardware

Interrupts & Context Switching

Interrupts = signals that pause normal execution

Context Switching = saving CPU state before handling interrupt

Steps:

Interrupt occurs (e.g., key press)

CPU saves registers + PC

Executes Interrupt Service Routine (ISR)

Restores previous state and continues

Example: Keyboard Interrupt

User presses a key: interrupt signal generated

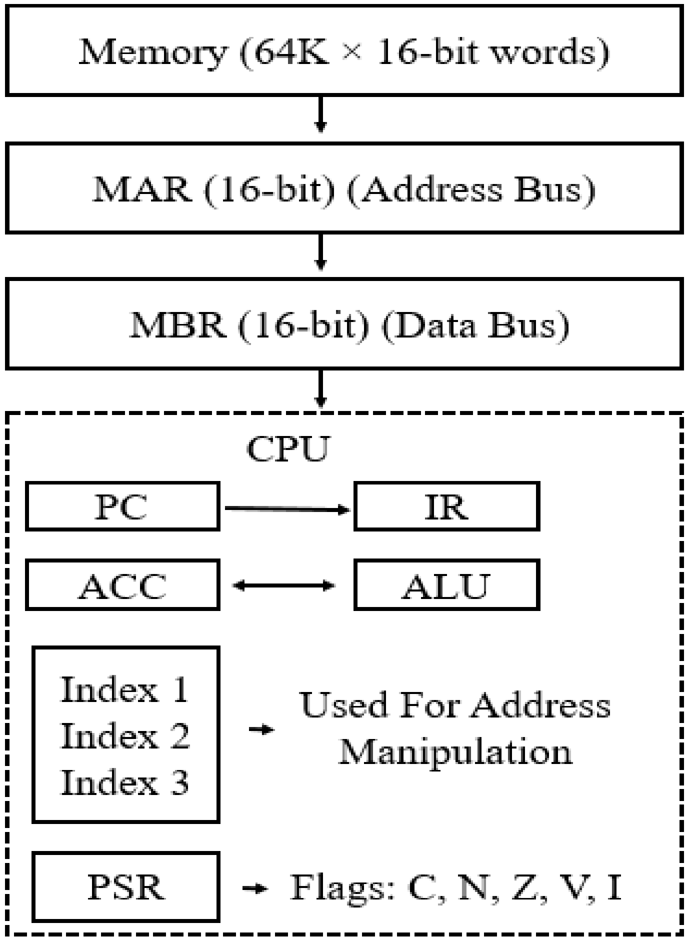
CPU:

Saves current execution context

Runs ISR to read key input

Restores program state: continues running original program

16-Bit ASC Computer: Hardware Design



Memory (64K × 16-bit words)
MAR (16-bit) (Address Bus)
MBR (16-bit) (Data Bus)
CPU
PC
IR
ACC
ALU
Index 1
Index 2
Index 3
Used For Address Manipulation
PSR
Flags: C, N, Z, V, I

Holds memory address

Holds data read/written

Word Size (16-bit machine)

- The ASC computer works with 16-bit data.
- Every register, memory word, and operation deals with 16 bits at a time.
- Negative numbers are stored using 2’s complement (a standard way to handle negatives in binary).

Memory

- Memory size = 64K words (where each word is 16 bits).

- To point to any memory location, you need a 16-bit address.
- Start Memory Location: 0000H
- End Memory Location: FFFFH

Registers for Memory Access

- MAR: Holds the address (where in memory to read/write).
- MBR : Holds the data being read from or written to memory.
- These registers are not directly accessible to the programmer—they are used internally by the CPU.

Program Execution (Stored-program computer

- Programs are stored in memory.
- Execution cycle is Fetch → Decode → Execute:
- Fetch: Bring the instruction from memory.
- Decode: Figure out what the instruction means.
- Execute: Perform the required operation.

Registers for Program Execution

- PC (Program Counter, 16-bit): Holds the address of the next instruction. After fetching one instruction, it automatically moves to the next.
- IR (Instruction Register, 16-bit): Holds the current instruction being executed.
- Together, PC + IR control program flow.

Accumulator (ACC, 16-bit)

- Used for arithmetic and logic operations.
- Example: When you add or subtract, the result is stored in the ACC.
- It "accumulates" results of operations.

Index Registers

- ASC has 3 index registers (Index1, Index2, Index3).
- They are used to modify addresses (e.g., for loops, arrays, or shifting memory access).

Note: Details are usually explained later when addressing modes are discussed.

Processor Status Register (PSR, 5-bit)

- Stores flags that show the outcome of operations:

- C (Carry) → set if an operation produces a carry out.
- N (Negative) → set if the result is negative.
- Z (Zero) → set if the result is zero.
- V (Overflow) → set if numbers are too big to fit in 16 bits.
- I (Interrupt Enable) → allows the CPU to accept interrupts.

Thank You