# CHAPTER 4: PROCESSOR FUNDAMENTALS

## 4.1 CENTRAL PROCESSING UNIT (CPU) ARCHITECTURE

### 4.1.1 Von Neumann Model

**Key Concept:**

- Data and programs are indistinguishable
- Can use the same memory for both
- Uses a single processor
- Follows fetch-decode-execute cycle

**Components:**

- Processor (CPU)
- Memory (for data and instructions)
- Input/Output devices

### 4.1.2 Registers

**Definition:** The smallest unit of storage in a microprocessor; allows fast data transfer.

**General Purpose Registers:**

- Used to temporarily store data values
- Can be used by assembly language instructions
- Example: Accumulator (ACC)

**Special Purpose Registers:**

| Register | Function |
| --- | --- |

| | |
|---|---|
| **PC (Program Counter)** | Holds address of next instruction |
| **MDR (Memory Data Register)** | Holds data fetched from memory |
| **MAR (Memory Address Register)** | Holds address of memory cell to access |
| **ACC (Accumulator)** | Holds values processed by ALU |
| **IX (Index Register)** | Stores number to modify address |
| **CIR (Current Instruction Register)** | Holds current instruction for decoding |
| **Status Register** | Holds results of comparisons, arithmetic flags |

## 4.1.3 CPU Components

**ALU (Arithmetic and Logic Unit):**

- Part of processor that processes instructions
- Performs arithmetic operations (add, subtract, multiply, divide)
- Performs logical operations (AND, OR, NOT)

**Control Unit (CU):**

- Fetches instructions from memory
- Decodes instructions
- Synchronizes operations
- Sends signals to memory, ALU, and I/O devices

**System Clock:**

- Timing device connected to processor
- Synchronizes all components
- Generates clock pulses

**IAS (Immediate Access Store):**

- Memory unit the processor can directly access

## 4.1.4 Buses

**Definition:** Set of parallel wires allowing data transfer between components.

**Data Bus:**

- Bidirectional
- Carries data between processor, memory, and I/O devices

**Address Bus:**

- Unidirectional
- Carries memory address from processor to MAR

**Control Bus:**

- Bidirectional
- Transmits control signals from CU
- Ensures components don't conflict

## 4.1.5 Performance Factors

**Clock Speed:**

- Number of pulses the clock sends in a given time
- Usually measured in GHz
- Higher clock speed = more cycles per second = faster execution
- Limitation: Heat generation at high speeds

**Bus Width:**

- Number of bits that can be transferred simultaneously
- Wider bus = more bits transferred at once = faster processing

**Cache Memory:**

- Stores commonly used instructions
- Larger cache = more instructions stored = less waiting = better performance

**Number of Cores:**

- Multi-core processors have multiple processing units
- Each core can process different instructions simultaneously
- Improves performance through parallel processing

## 4.1.6 Ports

| Port Type | Description |
| --- | --- |
| **USB** | Connects input and output devices |
| **HDMI** | High-definition video and audio output |
| **VGA** | Video output only (older displays) |

## 4.1.7 Fetch-Execute Cycle

**Fetch Stage:**

1. PC holds address of next instruction
2. Address copied to MAR
3. PC incremented
4. Instruction loaded to MDR from address in MAR
5. Instruction from MDR loaded to CIR

**Decode Stage:**

- Opcode and operand parts identified

**Execute Stage:**

- Control unit executes instruction
- Return to start

**Register Transfer Notation (RTN):**

**<TEXT>**

```
MAR ← [PC]
PC ← [PC] + 1
MDR ← [[MAR]]
CIR ← [MDR]
Decode
Execute
Return to start
```

## 4.1.8 Interrupts

**Definition:** A signal from a program seeking the processor's attention.

**ISR (Interrupt Service Routine):**

- Handles the interrupt
- Different ISRs for different sources

**Interrupt Handling Process:**

1. Processor checks interrupt register at end of F-E cycle
2. If interrupt flag is set, source detected
3. If low priority, interrupt disabled
4. If high priority:
    - Save register contents to stack
    - Load PC with ISR address
    - Execute ISR
    - Restore registers from stack
    - Continue interrupted program

---

# 4.2 ASSEMBLY LANGUAGE

## 4.2.1 Introduction

**Assembly Language:**

- Low-level programming language
- Instructions consist of opcode and operand

**Machine Code:**

- Binary code the processor executes directly
- One-to-one relationship with assembly language

**Assembler:**

- Software that translates assembly language to machine code
- Replaces mnemonics and labels with binary values

## 4.2.2 Assembler Types

**One-Pass Assembler:**

- Converts source code in one sweep
- Cannot handle forward referencing

**Two-Pass Assembler:**

**Pass 1:**

- Creates symbol table
- Enters symbolic addresses and labels

**Pass 2:**

- Translates to machine code using table
- Reports errors

## 4.2.3 Addressing Modes

| Mode | Description |
| --- | --- |
| **Immediate** | Data is the actual value (e.g., LDM #n) |
| **Direct** | Load contents at given address (e.g., LDD address) |
| **Indirect** | Address to use is at given address (e.g., LDI address) |
| **Indexed** | Address = given address + contents of IX (e.g., LDX address) |
| **Relative** | Next instruction is offset from current instruction |

## 4.2.4 Instruction Types

**Data Movement:**

- LDM#: Load immediate
- LDD: Load direct
- LDI: Load indirect
- LDX: Load indexed
- STO: Store

**Arithmetic:**

- ADD: Add to accumulator
- SUB: Subtract from accumulator
- INC: Increment
- DEC: Decrement

**Comparing:**

- CMP: Compare with contents at address
- CMP#: Compare with immediate value

**Conditional Jumps:**

- JPE: Jump if equal (compare TRUE)
- JPN: Jump if not equal (compare FALSE)

**Unconditional Jumps:**

- JMP: Jump to address

**I/O:**

- IN: Input character
- OUT: Output character

**End:**

- END: Return control to OS

---

# 4.3 BIT MANIPULATION

## 4.3.1 Binary Shifts

**Left Shift (LSL #n):**

- Bits shifted left
- Multiplies by $2^n$
- Zeros fill vacated positions

**Right Shift (LSR #n):**

- Bits shifted right

- Divides by $2^n$
- Zeros fill vacated positions

**Arithmetic Shift:**

- Used for signed integers
- Sign bit (MSB) maintained

**Cyclic Shift:**

- Bit removed from one end added to other end

## 4.3.2 Bit Masking

**Purpose:** Each bit can represent an individual flag. By manipulating bits, flags can be operated upon.

**Operations:**

**Masking to 1:**

- Use OR operation with 1

**Masking to 0:**

- Use AND operation with 0

**Testing Bits:**

- Use AND to mask bits
- Compare result with pattern

**Practical Applications:**

- Setting individual bit positions
- Testing specific bits
- Checking patterns

---

Revision #1
Created 2026-03-16 12:01:11 UTC by Samuel Lee
Updated 2026-03-16 12:01:26 UTC by Samuel Lee