8086 Microprocessor Complete Notes

1. Evolution of Microprocessor

What is a Microprocessor?

A microprocessor is a programmable digital electronic component that incorporates the functions of a CPU on a single integrated circuit. It processes data according to a set of instructions (programs) stored in memory.

Historical Evolution

1st Generation (4-bit, 1971-1972)

- **Intel 4004**: First commercial microprocessor
- 4-bit data bus, 2,300 transistors
- Clock speed: 740 KHz
- Used in calculators and simple control systems

2nd Generation (8-bit, 1972-1978)

- **Intel 8008**: 8-bit processor, 3,500 transistors
- **Intel 8080**: Improved version with better instruction set
- **Intel 8085**: Single +5V supply, integrated clock generator
- Used in early personal computers

3rd Generation (16-bit, 1978-1985)

- **Intel 8086**: 16-bit processor, introduction of segmented memory
- **Intel 8088**: 8-bit external data bus version of 8086
- **Intel 80286**: Protected mode, virtual memory support
- Marked the beginning of modern PC architecture

4th Generation (32-bit, 1985-1995)

- **Intel 80386**: Full 32-bit processor, virtual 8086 mode
- **Intel 80486**: Integrated cache and math coprocessor
- **Pentium**: Superscalar architecture, dual pipelines

5th Generation and Beyond (1995-present)

- 64-bit processors, multi-core designs
- Advanced features like branch prediction, out-of-order execution
- Modern architectures: Core series, ARM, mobile processors

---

2. 8086 Microprocessor

Architecture

The 8086 is a 16-bit microprocessor with sophisticated architecture designed for efficient programming and memory management.

Key Specifications

- **Data Bus**: 16-bit (can process 16 bits simultaneously)
- **Address Bus**: 20-bit (can address 1MB of memory)
- **Clock Speed**: 5-10 MHz
- **Transistors**: 29,000
- **Packaging**: 40-pin DIP

Internal Architecture

Execution Unit (EU)

- Contains ALU, general-purpose registers, and flag register
- Executes instructions fetched by BIU
- Operates on 16-bit data

Bus Interface Unit (BIU)

- Handles all bus operations
- Contains segment registers and instruction queue
- Fetches instructions ahead of time (prefetching)

Register Set

General Purpose Registers (16-bit each)

- **AX (Accumulator)**: Primary register for arithmetic operations
  - AH (high byte), AL (low byte)
- **BX (Base)**: Base register for addressing

    - BH (high byte), BL (low byte)
- **CX (Count)**: Counter for loops and string operations
    - CH (high byte), CL (low byte)
- **DX (Data)**: Data register, used with AX for 32-bit operations
    - DH (high byte), DL (low byte)

Index Registers

- **SI (Source Index)**: Source pointer for string operations
- **DI (Destination Index)**: Destination pointer for string operations

Pointer Registers

- **SP (Stack Pointer)**: Points to top of stack
- **BP (Base Pointer)**: Base pointer for stack frame

Segment Registers (16-bit each)

- **CS (Code Segment)**: Points to code segment
- **DS (Data Segment)**: Points to data segment
- **ES (Extra Segment)**: Additional data segment
- **SS (Stack Segment)**: Points to stack segment

Flag Register (16-bit) Contains status and control flags:

- **Status Flags**: CF, PF, AF, ZF, SF, OF
- **Control Flags**: TF, IF, DF

Memory Segmentation

The 8086 uses segmented memory model:

- **Physical Address = Segment Address × 16 + Offset**
- Each segment can be up to 64KB
- Allows addressing of 1MB total memory space

Instruction Set

The 8086 instruction set is categorized into several groups:

Data Transfer Instructions

- **MOV**: Move data between registers/memory
- **PUSH/POP**: Stack operations
- **XCHG**: Exchange data between operands
- **IN/OUT**: I/O operations

Arithmetic Instructions

- **ADD/SUB**: Addition and subtraction
- **MUL/DIV**: Multiplication and division
- **INC/DEC**: Increment and decrement
- **CMP**: Compare operands

Logic Instructions

- **AND/OR/XOR**: Bitwise logical operations
- **NOT**: Bitwise complement
- **TEST**: Logical comparison

String Instructions

- **MOVS**: Move string
- **CMPS**: Compare string
- **SCAS**: Scan string
- **LODS/STOS**: Load/store string

Control Transfer Instructions

- **JMP**: Unconditional jump
- **JCC**: Conditional jumps (JZ, JC, etc.)
- **CALL/RET**: Subroutine operations
- **INT**: Software interrupt

Processor Control Instructions

- **CLC/STC**: Clear/set carry flag
- **CLI/STI**: Clear/set interrupt flag
- **HLT**: Halt processor

Interrupts and 8259A

Types of Interrupts

Hardware Interrupts

- **Maskable**: Can be disabled using CLI instruction
- **Non-maskable (NMI)**: Cannot be disabled, highest priority

Software Interrupts

- Generated by INT instruction
- Used for system calls and BIOS functions

Internal Interrupts (Exceptions)

- **Divide by zero**: Type 0 interrupt
- **Single step**: Type 1 interrupt
- **Breakpoint**: Type 3 interrupt

Interrupt Processing

1. Complete current instruction
2. Push flags, CS, and IP onto stack
3. Clear IF and TF flags
4. Load new CS and IP from interrupt vector table
5. Execute interrupt service routine
6. Return using IRET instruction

8259A Programmable Interrupt Controller (PIC)

Features

- Manages up to 8 interrupt sources
- Cascadable (up to 64 interrupts with 8 PICs)
- Programmable priority schemes
- Automatic vectoring

Operating Modes

- **Fully Nested Mode**: Fixed priority system
- **Rotating Priority Mode**: Equal priority rotation

- **Special Mask Mode**: Selective masking
- **Polled Mode**: Software polling instead of interrupts

Programming

- Requires Initialization Command Words (ICWs)
- Operation Command Words (OCWs) for control
- Must be programmed before use

Higher Versions of 8086

80286 (1982)

Key Features

- 16-bit processor with 24-bit address bus
- Can address 16MB of memory
- Introduced protected mode
- Virtual memory support
- Backward compatible with 8086

Protected Mode

- Memory protection mechanisms
- Privilege levels (0-3)
- Segment descriptors instead of simple segments
- Task switching capability

New Instructions

- Additional string and arithmetic instructions
- Protected mode specific instructions
- Enhanced bit manipulation

80386 (1985)

Key Features

- First true 32-bit processor
- 32-bit data and address buses

- Can address 4GB of memory
- Virtual 8086 mode
- Paging support

Architectural Improvements

- Extended 32-bit registers (EAX, EBX, etc.)
- Additional addressing modes
- Enhanced protected mode
- Virtual memory management

Virtual 8086 Mode

- Allows running multiple 8086 programs
- Each program thinks it owns the machine
- Protected from each other

80486 (1989)

Key Features

- Integrated floating-point unit (FPU)
- 8KB on-chip cache
- Enhanced instruction set
- Built-in memory management unit

Performance Improvements

- Instruction pipelining
- Faster execution of common instructions
- Optimized instruction fetch and decode

Variants

- **80486SX**: No integrated FPU
- **80486DX**: With integrated FPU
- **80486DX2**: Double internal clock speed

---

3. Pentium Microprocessor

Architecture

The Pentium represents a major architectural advancement with superscalar design.

Key Features

- **Superscalar Architecture**: Dual instruction pipelines
- **64-bit Data Bus**: Double the bandwidth of 486
- **Branch Prediction**: Reduces pipeline stalls
- **Separate Caches**: Split instruction and data caches
- **Enhanced FPU**: Faster floating-point operations

Pipeline Structure

- **U-pipeline**: Can execute any instruction
- **V-pipeline**: Limited to simple instructions
- **5-stage pipeline**: Prefetch, Decode1, Decode2, Execute, Writeback

Register Sets

General Purpose Registers (Extended from 8086)

- **32-bit Extensions**: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
- **Backward Compatibility**: Can still access 16-bit and 8-bit portions
- **Additional Registers**: Some models include MMX registers

Segment Registers

- Same as 8086: CS, DS, ES, SS, FS, GS
- **FS and GS**: Additional segment registers for flexible addressing

Control Registers

- **CR0**: System control register
- **CR1**: Reserved
- **CR2**: Page fault linear address
- **CR3**: Page directory base register
- **CR4**: Additional system features

Debug Registers

- **DR0-DR7**: Hardware debugging support
- Breakpoint control and status

Cache System

L1 Cache (Primary Cache)

- **Size**: 8KB each for instruction and data
- **Organization**: Two-way set associative
- **Line Size**: 32 bytes
- **Write Policy**: Write-through for data cache

Cache Operation

- **Hit**: Data found in cache (fast access)
- **Miss**: Data not in cache (memory access required)
- **Write-through**: Updates both cache and memory
- **Cache Coherency**: Maintains consistency

Floating Point Operations

Enhanced FPU Features

- **Pipeline**: Overlapped execution
- **Performance**: Up to 5x faster than 80486
- **Precision**: 32, 64, and 80-bit formats
- **Standards Compliance**: IEEE 754 standard

FPU Register Stack

- **Eight 80-bit registers**: ST(0) through ST(7)
- **Stack Organization**: ST(0) is top of stack
- **Data Types**: Integer, single, double, extended precision

Common FP Instructions

- **FADD/FSUB**: Floating-point add/subtract
- **FMUL/FDIV**: Floating-point multiply/divide
- **FLD/FST**: Load/store floating-point values
- **FCOMP**: Compare floating-point values

Addressing Modes

The Pentium supports all 8086 addressing modes plus enhancements:

Basic Addressing Modes

1. **Register**: Operand is in a register
2. **Immediate**: Operand is a constant value
3. **Direct**: Operand address is given directly
4. **Register Indirect**: Address is in a register
5. **Based**: Base register + displacement
6. **Indexed**: Index register + displacement
7. **Based Indexed**: Base + index + displacement

32-bit Enhancements

- **Scale Factor**: Index can be scaled by 1, 2, 4, or 8
- **Extended Registers**: Can use 32-bit registers as base/index
- **Larger Displacements**: 32-bit displacement values

Paging

Virtual Memory Management

- **Page Size**: 4KB standard (4MB pages also supported)
- **Translation**: Linear to physical address conversion
- **Page Tables**: Two-level translation structure

Paging Process

1. **Linear Address**: Generated by processor
2. **Page Directory**: First level of translation
3. **Page Table**: Second level of translation
4. **Physical Address**: Final memory address

Page Directory Entry (PDE)

- **Present Bit**: Page table in memory
- **Read/Write**: Access permissions

- **User/Supervisor**: Privilege level
- **Page Table Address**: Points to page table

Page Table Entry (PTE)

- Similar structure to PDE
- Points to actual physical page
- Contains access and status bits

Instruction Set Enhancements

New Instructions

- **CMPXCHG**: Compare and exchange (atomic operation)
- **XADD**: Exchange and add
- **BSWAP**: Byte swap for endian conversion
- **CPUID**: CPU identification

Enhanced Instructions

- **Bit Scan**: BSF/BSR for bit manipulation
- **Bit Test**: BT/BTC/BTR/BTS operations
- **String Instructions**: Enhanced with repeat prefixes

Opcode Format

Instruction Encoding

Pentium instructions can be 1-15 bytes long:

1. **Prefixes** (0-4 bytes): Optional instruction modifiers
2. **Opcode** (1-2 bytes): Operation specification
3. **ModR/M** (0-1 byte): Addressing mode specification
4. **SIB** (0-1 byte): Scale-Index-Base for complex addressing
5. **Displacement** (0/1/2/4 bytes): Address offset
6. **Immediate** (0/1/2/4 bytes): Constant data

Common Prefixes

- **66h**: Operand size override

- **67h**: Address size override
- **F2h/F3h**: Repeat prefixes
- **26h/2Eh/36h/3Eh**: Segment override

Interrupt System

Interrupt Descriptor Table (IDT)

- **256 Entries**: One for each interrupt type
- **Gate Descriptors**: Define interrupt handlers
- **Privilege Checking**: Based on current privilege level

Types of Gates

- **Interrupt Gate**: Disables interrupts during handling
- **Trap Gate**: Keeps interrupts enabled
- **Task Gate**: Performs task switch

Exception Handling

- **Fault**: Restartable exceptions
- **Trap**: Reported after instruction
- **Abort**: Non-restartable errors

Protected Mode Operations

Privilege Levels

- **Ring 0**: Kernel/OS level (highest privilege)
- **Ring 1**: Device drivers
- **Ring 2**: System services
- **Ring 3**: User applications (lowest privilege)

Segment Descriptors

- **Code Segments**: Executable code with privilege levels
- **Data Segments**: Data storage with access rights
- **System Segments**: Special system structures

Protection Mechanisms

- **Segment Limits**: Prevent out-of-bounds access
- **Access Rights**: Control read/write/execute permissions
- **Privilege Validation**: Ensures proper access levels

Task Management

- **Task State Segment (TSS)**: Saves task context
- **Task Switching**: Hardware-assisted context switch
- **I/O Permission Bitmap**: Controls port access

---

4. Next Generation Microprocessors

Intel Core Architecture

Core Microarchitecture (2006)

The Intel Core architecture marked a significant shift from the NetBurst architecture:

Key Features

- **Lower Power Consumption**: More efficient than Pentium 4
- **Improved Performance per Watt**: Better energy efficiency
- **Enhanced Branch Prediction**: More accurate prediction algorithms
- **Wider Execution**: More execution units working in parallel
- **Advanced Cache Hierarchy**: Improved cache design

Architectural Improvements

- **14-stage Pipeline**: Optimized for performance and power
- **Macro-op Fusion**: Combines simple operations
- **Stack Pointer Tracking**: Optimizes stack operations
- **Enhanced Prefetching**: Better data prediction

Intel Dual Core

Dual Core Concept

- **Two Processing Cores**: Independent execution units on single die
- **Shared Resources**: Cache, memory controller, I/O

- **Parallel Processing**: True simultaneous multitasking
- **Thread-Level Parallelism**: Each core can handle separate threads

Core Communication

- **Shared L2 Cache**: Fast inter-core communication
- **Cache Coherency**: Maintains data consistency between cores
- **Bus Interface**: Single interface to system bus

Benefits

- **Multitasking**: Run multiple applications simultaneously
- **Multithreaded Applications**: Better performance for parallel software
- **System Responsiveness**: One core can handle background tasks

Core 2 Duo

Architecture Features

- **65nm Process Technology**: Smaller, more efficient transistors
- **Up to 4MB Shared L2 Cache**: Large shared cache between cores
- **1066 MHz FSB**: High-speed system bus
- **Enhanced Digital Media**: Improved multimedia processing

Performance Improvements

- **Intel Wide Dynamic Execution**: More instructions per clock
- **Intel Intelligent Power Capability**: Dynamic power management
- **Intel Advanced Smart Cache**: Optimized cache sharing
- **Intel Smart Memory Access**: Improved memory bandwidth

Variants

- **Core 2 Duo E6000 Series**: Desktop processors
- **Core 2 Duo T7000 Series**: Mobile processors
- **Different Cache Sizes**: 2MB and 4MB versions

Core 2 Quad

Quad-Core Design

- **Four Processing Cores**: Maximum parallelism for the time
- **Multi-Core Multi-Cache (MCMC)**: Dual shared cache design
- **Two Dies**: Each die contains two cores with shared cache
- **High Bandwidth**: Fast inter-core and core-to-memory communication

Cache Hierarchy

- **L1 Cache**: 32KB per core (16KB instruction + 16KB data)
- **L2 Cache**: 4MB or 6MB shared between pairs of cores
- **Cache Line**: 64 bytes for efficient data transfer

Applications

- **Content Creation**: Video editing, 3D rendering
- **Gaming**: Physics processing, AI calculations
- **Server Applications**: Database, web servers
- **Scientific Computing**: Parallel algorithms

Core i3, i5, i7 Series

Core i3

Target Market: Budget-conscious consumers, basic computing

Key Features

- **Dual-Core**: Two processing cores
- **Hyper-Threading**: 4 logical processors
- **Integrated Graphics**: Built-in GPU
- **Smart Cache**: Shared L3 cache
- **Basic Turbo**: Limited frequency scaling

Core i5

Target Market: Mainstream users, gamers

Key Features

- **Quad-Core**: Four processing cores (varies by generation)
- **No Hyper-Threading**: (in most desktop versions)

- **Turbo Boost**: Automatic overclocking
- **Integrated Graphics**: Enhanced GPU performance
- **Larger Cache**: More L3 cache than i3

Core i7

Target Market: Enthusiasts, professionals, content creators

Key Features

- **Higher Core Count**: 4-8+ cores depending on generation
- **Hyper-Threading**: Double the logical processors
- **Advanced Turbo Boost**: Maximum performance scaling
- **Larger Cache**: Maximum L3 cache
- **Higher TDP**: More power for better performance

Common Technologies Across Series

- **Intel Turbo Boost**: Automatic overclocking when possible
- **Hyper-Threading**: Simultaneous multithreading technology
- **Intel Quick Sync**: Hardware video encoding/decoding
- **AES-NI**: Hardware encryption acceleration
- **Virtualization**: VT-x and VT-d support

Mobile Microprocessors

Design Considerations

Power Efficiency

- **Lower TDP**: Thermal Design Power optimized for battery life
- **Dynamic Voltage/Frequency Scaling**: Adjusts power based on load
- **Sleep States**: Multiple power-saving modes
- **Clock Gating**: Disables unused circuits

Thermal Management

- **Lower Heat Generation**: Prevents overheating in confined spaces
- **Throttling**: Reduces performance to manage temperature
- **Package Design**: Optimized for thin form factors

Mobile-Specific Features

- **Integrated Graphics**: Reduces component count and power
- **Wireless Integration**: WiFi and cellular modems
- **Sensor Support**: Accelerometers, gyroscopes
- **Quick Wake**: Fast resume from sleep states

ARM Architecture

RISC Philosophy

Reduced Instruction Set Computer (RISC)

- **Simple Instructions**: Fixed-length, easy to decode
- **Load-Store Architecture**: Memory access only through specific instructions
- **Large Register File**: 16 general-purpose registers
- **Conditional Execution**: Most instructions can be conditional

Key Characteristics

- **Low Power**: Designed for battery-operated devices
- **High Performance per Watt**: Excellent efficiency
- **Scalable**: From microcontrollers to server processors
- **Licensing Model**: ARM licenses designs to manufacturers

ARM Instruction Set

- **32-bit Instructions**: Fixed length for easy decoding
- **Barrel Shifter**: Built-in shifting for one operand
- **Conditional Flags**: N, Z, C, V flags
- **Multiple Data Processing**: Single instruction, multiple operations

Modern ARM Features

- **ARMv8 Architecture**: 64-bit support (AArch64)
- **Advanced SIMD**: NEON technology for parallel processing
- **TrustZone**: Hardware security extensions
- **Big.LITTLE**: Combines high-performance and efficiency cores

MediaTek Helio Series

Overview

MediaTek Helio processors are system-on-chip (SoC) solutions primarily designed for smartphones and tablets.

Key Features

- **ARM-based Cores**: Uses ARM Cortex designs
- **Integrated GPU**: Mali or PowerVR graphics
- **AI Processing Unit**: Dedicated neural processing
- **Advanced Camera ISP**: Image signal processor
- **Connectivity**: 4G/5G modems, WiFi, Bluetooth

Helio Variants

- **Helio A Series**: Entry-level processors
- **Helio G Series**: Gaming-focused processors
- **Helio P Series**: Mid-range processors
- **Helio X Series**: High-performance processors (discontinued)

Intel Atom

Target Applications

- **Netbooks**: Ultra-portable laptops
- **Tablets**: x86-based tablet computers
- **Embedded Systems**: Industrial and IoT applications
- **Set-top Boxes**: Media streaming devices

Architecture Features

In-Order Execution

- **Simpler Design**: Reduces power consumption
- **Smaller Die Size**: Lower manufacturing cost
- **Predictable Performance**: Consistent execution time

Hyper-Threading

- **Single Core**: Appears as two logical processors
- **Better Utilization**: Improves performance with threaded applications
- **Minimal Hardware**: Efficient implementation

Power Optimization

- **Enhanced SpeedStep**: Dynamic frequency scaling
- **Deeper Sleep States**: C-states for power saving
- **Package Integration**: System controller on same package
- **Low Voltage**: Operates at reduced voltages

---

5. Microcontrollers

Microcontroller vs Microprocessor

Microprocessor Characteristics

- **CPU Only**: Requires external components
- **External Memory**: RAM and ROM separate
- **External I/O**: Ports and interfaces external
- **High Power**: Generally consumes more power
- **Complex System**: Requires supporting circuitry

Microcontroller Characteristics

- **Complete System**: CPU, memory, I/O on single chip
- **Integrated Memory**: Built-in RAM and ROM/Flash
- **Built-in Peripherals**: Timers, UART, ADC, etc.
- **Low Power**: Optimized for embedded applications
- **Cost Effective**: Single chip solution

Embedded Systems

Definition

An embedded system is a computer system with dedicated functions within a larger mechanical or electrical system.

Characteristics

- **Specific Purpose**: Designed for particular applications
- **Real-Time Operation**: Must respond within time constraints
- **Resource Constraints**: Limited memory, processing power
- **Reliability**: Must operate continuously without failure
- **Cost Sensitive**: Optimized for production costs

Examples

- **Automotive**: Engine control units, ABS systems
- **Consumer Electronics**: Washing machines, microwaves
- **Industrial**: Process control, automation systems
- **Medical**: Pacemakers, insulin pumps
- **Telecommunications**: Routers, switches

Design Considerations

- **Performance Requirements**: Speed and throughput needs
- **Power Consumption**: Battery life and heat generation
- **Memory Requirements**: Program and data storage
- **Real-time Constraints**: Deadline requirements
- **Cost Targets**: Manufacturing and development costs

8051 Microcontroller Architecture

Overview

The 8051 is an 8-bit microcontroller developed by Intel in 1980, widely used in embedded applications.

Key Specifications

- **8-bit CPU**: Harvard architecture
- **Program Memory**: 4KB ROM (internal)
- **Data Memory**: 128 bytes RAM (internal)
- **I/O Ports**: 4 × 8-bit ports (P0, P1, P2, P3)
- **Timers**: 2 × 16-bit timers
- **Serial Port**: Full duplex UART
- **Interrupts**: 5 interrupt sources

- **Instruction Set**: 111 instructions

Harvard Architecture

Separate Memory Spaces

- **Program Memory**: Contains executable code
- **Data Memory**: Contains variables and data
- **Advantages**: Simultaneous access to program and data
- **Bus Structure**: Separate buses for program and data

Memory Organization

Program Memory (ROM/EPROM/Flash)

- **Address Range**: 0000H to FFFFH (64KB max)
- **Internal ROM**: 0000H to 0FFFH (4KB)
- **External ROM**: Can extend to full 64KB
- **Reset Vector**: Program starts at 0000H
- **Interrupt Vectors**: Located at specific addresses

Data Memory (RAM) Internal RAM (128 bytes)

- **Address Range**: 00H to 7FH
- **Bank 0**: Registers R0-R7 (00H to 07H)
- **Bank 1**: Registers R0-R7 (08H to 0FH)
- **Bank 2**: Registers R0-R7 (10H to 17H)
- **Bank 3**: Registers R0-R7 (18H to 1FH)
- **Bit Addressable**: 20H to 2FH (16 bytes = 128 bits)
- **General Purpose**: 30H to 7FH (80 bytes)

Special Function Registers (SFRs)

- **Address Range**: 80H to FFH
- **Key Registers**: ACC, B, PSW, SP, DPTR
- **Port Registers**: P0, P1, P2, P3
- **Control Registers**: TCON, TMOD, SCON, PCON

Register Set

Accumulator (ACC)

- **Address**: E0H
- **Function**: Primary register for arithmetic and logic operations
- **Bit Addressable**: Individual bits can be accessed

B Register

- **Address**: F0H
- **Function**: Used with ACC for multiplication and division
- **Bit Addressable**: Can access individual bits

Program Status Word (PSW)

- **Address**: D0H
- **Flags**: CY, AC, F0, RS1, RS0, OV, P
- **Register Bank Selection**: RS1, RS0 bits select active bank

Stack Pointer (SP)

- **Address**: 81H
- **Function**: Points to top of stack
- **Initial Value**: 07H (stack starts at 08H)

Data Pointer (DPTR)

- **Components**: DPH (83H) and DPL (82H)
- **Function**: 16-bit pointer for external data access
- **Usage**: Addressing external memory and lookup tables

I/O Ports

Port 0 (P0)

- **Address**: 80H
- **Function**: 8-bit bidirectional I/O or external address/data bus
- **Open Drain**: Requires external pull-up resistors
- **Dual Role**: I/O port or multiplexed address/data bus

Port 1 (P1)

- **Address**: 90H
- **Function**: 8-bit bidirectional I/O
- **Internal Pull-ups**: Built-in pull-up resistors
- **General Purpose**: Typically used for general I/O

Port 2 (P2)

- **Address**: A0H
- **Function**: 8-bit bidirectional I/O or high-order address bus
- **Internal Pull-ups**: Built-in pull-up resistors
- **Dual Role**: I/O port or high address byte

Port 3 (P3)

- **Address**: B0H
- **Function**: 8-bit bidirectional I/O with alternate functions
- **Alternate Functions**:
  - P3.0 (RXD): Serial receive
  - P3.1 (TXD): Serial transmit
  - P3.2 (INT0): External interrupt 0
  - P3.3 (INT1): External interrupt 1
  - P3.4 (T0): Timer 0 input
  - P3.5 (T1): Timer 1 input
  - P3.6 (WR): External write strobe
  - P3.7 (RD): External read strobe

Timers/Counters

Timer/Counter 0 and 1

- **16-bit Resolution**: Can count up to 65536
- **Modes of Operation**: 4 different modes each
- **Clock Source**: Internal clock or external pulses
- **Control Registers**: TMOD and TCON

Timer Modes Mode 0: 13-bit timer/counter

- **TH**: 8 bits, TL: 5 bits

- **Overflow**: After count 1FFFH
- **Compatibility**: With 8048 microcontroller

Mode 1: 16-bit timer/counter

- **TH**: 8 bits, TL: 8 bits
- **Overflow**: After count FFFFH
- **Most Common**: Standard 16-bit operation

Mode 2: 8-bit auto-reload

- **TL**: 8-bit counter
- **TH**: 8-bit reload value
- **Automatic**: TH copied to TL on overflow

Mode 3: Split timer mode

- **Timer 0**: Split into two 8-bit timers
- **Timer 1**: Stops counting
- **Special Use**: When three timers needed

Serial Communication

UART Features

- **Full Duplex**: Simultaneous send and receive
- **Baud Rate**: Programmable using Timer 1
- **Modes**: 4 different modes of operation
- **Control Register**: SCON for configuration

Serial Modes Mode 0: Shift register mode

- **Synchronous**: Clock provided
- **8-bit Data**: No start/stop bits
- **Fixed Baud Rate**: Fosc/12

Mode 1: 8-bit UART

- **Asynchronous**: No external clock
- **10-bit Frame**: 1 start + 8 data + 1 stop

- **Variable Baud Rate**: Set by Timer 1

Mode 2: 9-bit UART

- **9th Bit**: Programmable
- **11-bit Frame**: 1 start + 8 data + 1 programmable + 1 stop
- **Fixed Baud Rate**: Fosc/32 or Fosc/64

Mode 3: 9-bit UART

- **Same as Mode 2**: But variable baud rate
- **Baud Rate**: Controlled by Timer 1

Interrupt System

Interrupt Sources

1. **External Interrupt 0** (INT0): P3.2 pin
2. **Timer 0 Overflow**: Timer 0 overflow flag
3. **External Interrupt 1** (INT1): P3.3 pin
4. **Timer 1 Overflow**: Timer 1 overflow flag
5. **Serial Interrupt**: Serial transmit/receive complete

Interrupt Priorities

- **High Priority**: Can interrupt low priority routines
- **Low Priority**: Cannot interrupt high priority routines
- **Priority Control**: IP register sets priorities

Interrupt Processing

1. **Finish Current Instruction**: Complete execution
2. **Save Program Counter**: Push PC onto stack
3. **Jump to Vector**: Load PC with interrupt vector
4. **Execute ISR**: Run interrupt service routine
5. **Return**: RETI instruction restores PC

Interrupt Vectors

- **External INT0**: 0003H

- **Timer 0**: 000BH
- **External INT1**: 0013H
- **Timer 1**: 001BH
- **Serial**: 0023H

8051 Operation and Instruction Set

CPU Operation

Instruction Cycle The 8051 executes instructions in machine cycles, where each machine cycle consists of 6 states (S1-S6), and each state consists of 2 oscillator periods.

Timing

- **Oscillator Period**: Basic timing unit
- **State**: 2 oscillator periods
- **Machine Cycle**: 6 states = 12 oscillator periods
- **Instruction Time**: 1-4 machine cycles depending on instruction

Fetch-Execute Cycle

1. **Fetch**: Read instruction from program memory
2. **Decode**: Determine instruction type and operands
3. **Execute**: Perform the operation
4. **Store**: Save results if required

Addressing Modes

Immediate Addressing

- **Format**: #data
- **Example**: MOV A, #55H
- **Description**: Data is part of instruction
- **Usage**: Loading constants

Register Addressing

- **Format**: Rn (where n = 0-7)
- **Example**: MOV A, R0
- **Description**: Data is in specified register

- **Usage**: Register to register operations

Direct Addressing

- **Format**: address
- **Example**: MOV A, 30H
- **Description**: Address specifies memory location
- **Usage**: Accessing specific memory locations

Register Indirect Addressing

- **Format**: @Ri (where i = 0 or 1)
- **Example**: MOV A, @R0
- **Description**: Register contains address of data
- **Usage**: Array access, pointer operations

Indexed Addressing

- **Format**: A+DPTR or A+PC
- **Example**: MOVC A, @A+DPTR
- **Description**: Base address plus offset in accumulator
- **Usage**: Table lookups, accessing arrays

Instruction Set Categories

Data Transfer Instructions

MOV Instructions

- **MOV A, Rn**: Move register to accumulator
- **MOV Rn, A**: Move accumulator to register
- **MOV A, direct**: Move direct byte to accumulator
- **MOV direct, A**: Move accumulator to direct byte
- **MOV A, @Ri**: Move indirect RAM to accumulator
- **MOV @Ri, A**: Move accumulator to indirect RAM
- **MOV A, #data**: Move immediate data to accumulator
- **MOV Rn, #data**: Move immediate data to register
- **MOV direct, #data**: Move immediate data to direct byte

External Memory Access

- **MOVX A, @DPTR**: Move external RAM (16-bit address) to accumulator
- **MOVX @DPTR, A**: Move accumulator to external RAM (16-bit address)
- **MOVX A, @Ri**: Move external RAM (8-bit address) to accumulator
- **MOVX @Ri, A**: Move accumulator to external RAM (8-bit address)

Program Memory Access

- **MOVC A, @A+DPTR**: Move code byte relative to DPTR to accumulator
- **MOVC A, @A+PC**: Move code byte relative to PC to accumulator

Stack Operations

- **PUSH direct**: Push direct byte onto stack
- **POP direct**: Pop direct byte from stack

Data Exchange

- **XCH A, Rn**: Exchange accumulator with register
- **XCH A, direct**: Exchange accumulator with direct byte
- **XCH A, @Ri**: Exchange accumulator with indirect RAM
- **XCHD A, @Ri**: Exchange low-order digit indirect RAM with accumulator

Arithmetic Instructions

Addition

- **ADD A, Rn**: Add register to accumulator
- **ADD A, direct**: Add direct byte to accumulator
- **ADD A, @Ri**: Add indirect RAM to accumulator
- **ADD A, #data**: Add immediate data to accumulator
- **ADDC A, Rn**: Add register to accumulator with carry
- **ADDC A, direct**: Add direct byte to accumulator with carry
- **ADDC A, @Ri**: Add indirect RAM to accumulator with carry
- **ADDC A, #data**: Add immediate data to accumulator with carry

Subtraction

- **SUBB A, Rn**: Subtract register from accumulator with borrow

- **SUBB A, direct**: Subtract direct byte from accumulator with borrow
- **SUBB A, @Ri**: Subtract indirect RAM from accumulator with borrow
- **SUBB A, #data**: Subtract immediate data from accumulator with borrow

Increment/Decrement

- **INC A**: Increment accumulator
- **INC Rn**: Increment register
- **INC direct**: Increment direct byte
- **INC @Ri**: Increment direct RAM
- **INC DPTR**: Increment data pointer
- **DEC A**: Decrement accumulator
- **DEC Rn**: Decrement register
- **DEC direct**: Decrement direct byte
- **DEC @Ri**: Decrement indirect RAM

Multiplication and Division

- **MUL AB**: Multiply A and B (result in A and B)
- **DIV AB**: Divide A by B (quotient in A, remainder in B)

Decimal Adjust

- **DA A**: Decimal adjust accumulator after addition

Logical Instructions

Bitwise Operations

- **ANL A, Rn**: AND register with accumulator
- **ANL A, direct**: AND direct byte with accumulator
- **ANL A, @Ri**: AND indirect RAM with accumulator
- **ANL A, #data**: AND immediate data with accumulator
- **ANL direct, A**: AND accumulator with direct byte
- **ANL direct, #data**: AND immediate data with direct byte
- **ORL A, Rn**: OR register with accumulator
- **ORL A, direct**: OR direct byte with accumulator
- **ORL A, @Ri**: OR indirect RAM with accumulator

- **ORL A, #data**: OR immediate data with accumulator
- **ORL direct, A**: OR accumulator with direct byte
- **ORL direct, #data**: OR immediate data with direct byte
- **XRL A, Rn**: XOR register with accumulator
- **XRL A, direct**: XOR direct byte with accumulator
- **XRL A, @Ri**: XOR indirect RAM with accumulator
- **XRL A, #data**: XOR immediate data with accumulator
- **XRL direct, A**: XOR accumulator with direct byte
- **XRL direct, #data**: XOR immediate data with direct byte

Other Logical Operations

- **CLR A**: Clear accumulator
- **CPL A**: Complement accumulator
- **SWAP A**: Swap nibbles within accumulator
- **RL A**: Rotate accumulator left
- **RLC A**: Rotate accumulator left through carry
- **RR A**: Rotate accumulator right
- **RRC A**: Rotate accumulator right through carry

Control Transfer Instructions

Unconditional Jumps

- **LJMP addr16**: Long jump (3 bytes)
- **AJMP addr11**: Absolute jump (2 bytes)
- **SJMP rel**: Short jump (2 bytes, -128 to +127)
- **JMP @A+DPTR**: Jump indirect relative to DPTR

Conditional Jumps

- **JZ rel**: Jump if accumulator is zero
- **JNZ rel**: Jump if accumulator is not zero
- **JC rel**: Jump if carry flag is set
- **JNC rel**: Jump if carry flag is not set
- **JB bit, rel**: Jump if bit is set

- **JNB bit, rel**: Jump if bit is not set
- **JBC bit, rel**: Jump if bit is set then clear bit

Compare and Jump

- **CJNE A, direct, rel**: Compare direct byte with accumulator and jump if not equal
- **CJNE A, #data, rel**: Compare immediate with accumulator and jump if not equal
- **CJNE Rn, #data, rel**: Compare immediate with register and jump if not equal
- **CJNE @Ri, #data, rel**: Compare immediate with indirect RAM and jump if not equal

Loop Instructions

- **DJNZ Rn, rel**: Decrement register and jump if not zero
- **DJNZ direct, rel**: Decrement direct byte and jump if not zero

Subroutine Instructions

- **LCALL addr16**: Long subroutine call
- **ACALL addr11**: Absolute subroutine call
- **RET**: Return from subroutine
- **RETI**: Return from interrupt

Bit Manipulation Instructions

Bit Data Transfer

- **MOV C, bit**: Move bit to carry flag
- **MOV bit, C**: Move carry flag to bit

Bit Logic

- **CLR C**: Clear carry flag
- **CLR bit**: Clear bit
- **SETB C**: Set carry flag
- **SETB bit**: Set bit
- **CPL C**: Complement carry flag
- **CPL bit**: Complement bit
- **ANL C, bit**: AND bit with carry flag
- **ANL C, /bit**: AND complement of bit with carry flag

- **ORL C, bit**: OR bit with carry flag
- **ORL C, /bit**: OR complement of bit with carry flag

Other Instructions

- **NOP**: No operation
- **HALT**: Halt mode (power down)

Memory and I/O Interfacing

External Memory Interfacing

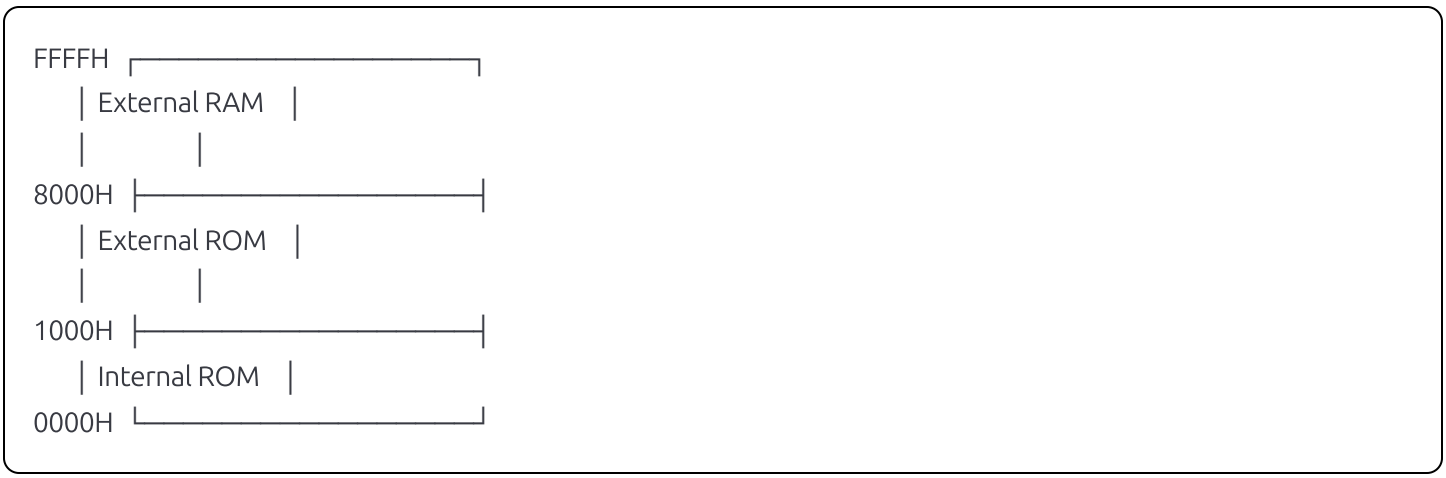
Program Memory Expansion

- **Address Range**: 0000H-FFFFH (64KB)
- **Control Signals**: PSEN (Program Store Enable)
- **Address Bus**: Port 0 (low byte) + Port 2 (high byte)
- **Data Bus**: Port 0 (time multiplexed)

External RAM Interfacing

- **Address Range**: 0000H-FFFFH (64KB)
- **Control Signals**: RD, WR
- **16-bit Address**: Using DPTR
- **8-bit Address**: Using R0 or R1

Memory Map



I/O Interfacing Techniques

Memory-Mapped I/O

- **Concept**: I/O devices appear as memory locations
- **Access**: Use MOVX instructions
- **Address Decoding**: Part of memory address space
- **Advantage**: Full instruction set available

Port-Mapped I/O

- **Concept**: Separate I/O address space
- **Access**: Use dedicated I/O instructions (not available in 8051)
- **8051 Method**: Uses memory-mapped approach

Parallel I/O

- **8255 PPI**: Programmable Peripheral Interface
- **Configuration**: Mode 0, 1, 2 operations
- **Port Usage**: 24 I/O lines in three 8-bit ports
- **Control**: Programming through control register

Serial I/O

- **Built-in UART**: Full duplex communication
- **External Devices**: Additional serial interfaces
- **Standards**: RS-232, RS-485 compatibility
- **Applications**: Communication with PCs, other microcontrollers

Interfacing to External Devices

LED Display Interface

- **Direct Connection**: Through current limiting resistors
- **7-Segment Display**: Common cathode/anode configuration
- **Multiplexing**: For multiple digits
- **Driver ICs**: For high current requirements

Matrix Keyboard Interface

- **Scanning Method**: Row-column scanning
- **Debouncing**: Software or hardware debouncing
- **Key Encoding**: Converting position to key code

- **Multiple Keys**: Handling simultaneous key presses

LCD Interface

- **HD44780 Compatible**: Standard LCD controller
- **Control Signals**: RS, R/W, E
- **Data Lines**: 4-bit or 8-bit mode
- **Initialization**: Power-on initialization sequence
- **Character Display**: ASCII character set

ADC Interface

- **ADC0808/0809**: 8-channel 8-bit ADC
- **Control Signals**: ALE, START, EOC
- **Channel Selection**: Address lines A, B, C
- **Timing**: Conversion time considerations

DAC Interface

- **DAC0808**: 8-bit DAC
- **Current Output**: Requires op-amp for voltage output
- **Reference Voltage**: Sets full-scale output
- **Applications**: Waveform generation, motor control

Stepper Motor Interface

- **Control Sequences**: Full step, half step
- **Driver Circuits**: ULN2003 or similar
- **Direction Control**: Clockwise/counterclockwise
- **Speed Control**: Timing between steps

Sensor Interfaces

- **Temperature Sensors**: LM35, DS18B20
- **Pressure Sensors**: Analog output types
- **Light Sensors**: LDR, photodiodes
- **Signal Conditioning**: Amplification, filtering

Communication Interfaces

- **SPI**: Serial Peripheral Interface (bit-banged)
- **I2C**: Inter-Integrated Circuit (bit-banged)
- **CAN**: Controller Area Network (external controller)
- **Ethernet**: External Ethernet controller

---

6. Programmable Logic Controller (PLC)

Basic Structure

What is a PLC?

A Programmable Logic Controller is an industrial digital computer designed for control of manufacturing processes, such as assembly lines, robotic devices, or any activity requiring high reliability control and ease of programming.

Key Components

Central Processing Unit (CPU)

- **Function**: Executes the control program
- **Types**: 8-bit, 16-bit, or 32-bit processors
- **Memory**: Program memory (ROM/Flash) and data memory (RAM)
- **Processing**: Scan cycle execution
- **Communication**: Handles I/O and network communication

Power Supply

- **Input Voltage**: Typically 24VDC or 120/240VAC
- **Output Voltage**: Regulated DC for internal circuits
- **Isolation**: Electrical isolation from input power
- **Backup**: Battery backup for memory retention

Input/Output (I/O) System

- **Input Modules**: Interface with field input devices
- **Output Modules**: Interface with field output devices
- **Isolation**: Optical isolation for safety
- **Signal Conditioning**: Level conversion and filtering

Programming Device

- **Types**: Dedicated programmer, PC with software
- **Connection**: Serial, USB, or Ethernet
- **Functions**: Program development, debugging, monitoring
- **Software**: Ladder logic editors, simulation tools

PLC Architecture Types

Compact PLCs

- **Integration**: CPU, I/O, and power supply in one unit
- **I/O Count**: Limited number of I/O points
- **Applications**: Small control systems
- **Cost**: Lower cost for simple applications

Modular PLCs

- **Rack-Based**: Separate modules in a rack system
- **Expandability**: Easy to add I/O modules
- **Flexibility**: Mix different types of I/O modules
- **Applications**: Large, complex control systems

Distributed I/O

- **Remote Modules**: I/O modules at remote locations
- **Network**: Connected via industrial networks
- **Wiring**: Reduces field wiring
- **Applications**: Geographically distributed systems

I/O System

Input Types

Digital Inputs

- **Voltage Levels**: 24VDC, 120VAC, 240VAC
- **Devices**: Switches, sensors, pushbuttons
- **Signal Types**: Sinking or sourcing
- **Isolation**: Optical isolation from CPU

Common Input Devices

- **Limit Switches**: Position sensing
- **Proximity Sensors**: Non-contact detection
- **Photoelectric Sensors**: Light-based detection
- **Pressure Switches**: Pressure monitoring
- **Temperature Switches**: Temperature monitoring

Analog Inputs

- **Signal Types**: 4-20mA, 0-10VDC, ±10VDC
- **Resolution**: 12-bit, 16-bit ADC
- **Devices**: Temperature transmitters, pressure transmitters
- **Scaling**: Engineering unit conversion

Output Types

Digital Outputs

- **Types**: Relay, transistor, triac
- **Voltage**: 24VDC, 120VAC, 240VAC
- **Current**: Varies by module type
- **Devices**: Motors, solenoids, indicators

Relay Outputs

- **Advantages**: High voltage/current capability, isolation
- **Disadvantages**: Slower switching, wear out
- **Applications**: AC loads, high current loads

Transistor Outputs

- **Advantages**: Fast switching, long life
- **Disadvantages**: DC only, lower current
- **Applications**: DC loads, fast switching requirements

Triac Outputs

- **Advantages**: AC switching, no moving parts

- **Disadvantages**: AC only, heat generation
- **Applications**: AC loads, heating elements

Analog Outputs

- **Signal Types**: 4-20mA, 0-10VDC
- **Resolution**: 12-bit, 16-bit DAC
- **Applications**: Variable speed drives, control valves

I/O Addressing

Rack and Slot Addressing

- **Format**: Rack.Slot.Terminal
- **Example**: 1.2.3 (Rack 1, Slot 2, Terminal 3)
- **Organization**: Physical location based

Tag-Based Addressing

- **Format**: Descriptive names
- **Example**: Motor_1_Start, Pump_A_Status
- **Advantages**: Self-documenting, easier maintenance

Programming

Programming Languages (IEC 61131-3)

Ladder Logic (LD)

- **Format**: Graphical, resembles electrical schematics
- **Elements**: Contacts, coils, function blocks
- **Execution**: Left to right, top to bottom
- **Popularity**: Most widely used in industry

Function Block Diagram (FBD)

- **Format**: Graphical blocks with connections
- **Elements**: Function blocks, connections
- **Applications**: Complex control algorithms
- **Advantages**: Shows signal flow clearly

Structured Text (ST)

- **Format**: High-level text language
- **Syntax**: Similar to Pascal
- **Applications**: Complex calculations, algorithms
- **Features**: Loops, conditionals, functions

Instruction List (IL)

- **Format**: Assembly-like text
- **Instructions**: Load, store, and, or operations
- **Applications**: Low-level programming
- **Usage**: Less common in modern systems

Sequential Function Chart (SFC)

- **Format**: Graphical state machine
- **Elements**: Steps, transitions, actions
- **Applications**: Sequential processes
- **Features**: Parallel branches, synchronization

Ladder Logic Fundamentals

Basic Elements

Contacts (Inputs)

- **Normally Open (NO)**: ──] [── (closes when input is true)
- **Normally Closed (NC)**: ──]/[── (opens when input is true)
- **Representation**: Input devices, internal bits

Coils (Outputs)

- **Output Coil**: ──( )── (energizes when logic is true)
- **Negated Coil**: ──(/)── (energizes when logic is false)
- **Set Coil**: ──(S)── (latches on when logic is true)
- **Reset Coil**: ──(R)── (turns off when logic is true)

Logic Operations

- **Series (AND)**: Contacts in series
- **Parallel (OR)**: Contacts in parallel
- **Mixed Logic**: Combination of series and parallel

Programming Rules

- **Power Flow**: Left to right
- **Rung Logic**: Each rung is a complete logic statement
- **Outputs**: Only on the right side
- **Comments**: Document each rung

Mnemonics and Timers

Instruction Mnemonics

Basic Instructions

- **LD**: Load (start a new logic path)
- **LDN**: Load NOT (start with normally closed contact)
- **AND**: AND with current path
- **ANDN**: AND NOT with current path
- **OR**: OR with current path
- **ORN**: OR NOT with current path
- **OUT**: Output result to specified address

Example Program

```
LD I:1/0      // Load input I:1/0
AND I:1/1     // AND with input I:1/1
OUT O:2/0     // Output to O:2/0
```

Timer Types

On-Delay Timer (TON)

- **Function**: Turns on after specified delay
- **Applications**: Motor start delays, alarm delays
- **Operation**: Starts timing when input becomes true
- **Reset**: Resets when input becomes false

Off-Delay Timer (TOF)

- **Function**: Turns off after specified delay
- **Applications**: Fan run-on, cooling delays
- **Operation**: Starts timing when input becomes false
- **Reset**: Resets when input becomes true

Retentive Timer (RTO)

- **Function**: Accumulates time across multiple periods
- **Applications**: Maintenance scheduling, hour meters
- **Operation**: Continues timing from last value
- **Reset**: Requires separate reset instruction

Timer Programming

Timer Structure

- **Timer Address**: T4:0 (Timer file 4, element 0)
- **Preset Value**: PRE (time duration)
- **Accumulated Value**: ACC (current time)
- **Status Bits**: EN (enable), TT (timing), DN (done)

Time Base

- **0.01 seconds**: High resolution timing
- **0.1 seconds**: Standard resolution
- **1.0 seconds**: Long duration timing

Relays and Counters

Internal Relays (Control Relays)

Control Bits

- **Function**: Internal memory bits for logic control
- **Applications**: Intermediate logic, flags, status bits
- **Addressing**: B3:0/0 (bit file, word, bit)
- **Retention**: Can be retentive or non-retentive

One-Shot (OSR)

- **Function**: Provides single pulse on rising edge
- **Applications**: Initialization, pulse generation
- **Operation**: True for one scan when input goes true
- **Storage**: Requires storage bit for edge detection

Master Control Relay (MCR)

- **Function**: Controls power to groups of outputs
- **Applications**: Emergency stops, zone control
- **Operation**: When false, forces outputs in zone off
- **Safety**: Should not be used for safety functions

Counter Types

Count-Up Counter (CTU)

- **Function**: Increments on rising edge of input
- **Applications**: Part counting, batch counting
- **Operation**: Increments ACC when input transitions false to true
- **Done Bit**: Sets when ACC ≥ PRE

Count-Down Counter (CTD)

- **Function**: Decrements on rising edge of input
- **Applications**: Inventory tracking, countdown operations
- **Operation**: Decrements ACC when input transitions false to true
- **Done Bit**: Sets when ACC ≤ 0

Up-Down Counter (CTUD)

- **Function**: Can count up or down
- **Applications**: Position tracking, bidirectional counting
- **Inputs**: Count up and count down inputs
- **Flexibility**: Single counter for bidirectional operations

Counter Programming

Counter Structure

- **Counter Address**: C5:0 (Counter file 5, element 0)
- **Preset Value**: PRE (target count)
- **Accumulated Value**: ACC (current count)
- **Status Bits**: CU (count up enable), CD (count down enable), DN (done), OV (overflow), UN (underflow)

Reset Function

- **RES Instruction**: Resets accumulator and status bits
- **Applications**: Batch reset, system initialization
- **Usage**: Separate from counter instruction

Master and Jump Control

Master Control Reset (MCR)

Purpose

- **Zone Control**: Controls execution of ladder logic zones
- **Emergency Stop**: Quick shutdown of process sections
- **Conditional Execution**: Enable/disable logic sections

Operation

- **MCR Zones**: Code between MCR instructions
- **True Condition**: Normal execution of zone
- **False Condition**: All non-retentive outputs in zone turn off
- **Nested MCRs**: Not recommended, can cause confusion

Programming Guidelines

- **Balanced Pairs**: Each MCR must have matching end MCR
- **Safety Considerations**: Not suitable for safety-critical stops
- **Documentation**: Clearly mark MCR zones

Jump Instructions

Jump (JMP)

- **Function**: Unconditionally skips program sections

- **Applications**: Conditional program execution, error handling
- **Label**: Jumps to specified label
- **Direction**: Can jump forward or backward

Label (LBL)

- **Function**: Destination for jump instructions
- **Naming**: Alphanumeric labels
- **Placement**: Can be anywhere in program
- **Multiple Jumps**: Multiple JMPs can go to same label

Jump to Subroutine (JSR)

- **Function**: Calls subroutine programs
- **Applications**: Reusable code, modular programming
- **Return**: SBR instruction starts subroutine, RET returns
- **Parameters**: Can pass input parameters

Subroutine (SBR) and Return (RET)

- **SBR**: First instruction in subroutine file
- **RET**: Last instruction in subroutine file
- **File Numbers**: Subroutines in separate program files
- **Local Variables**: Can use local tags in subroutines

Program Control Applications

Mode Selection

- **Manual/Auto**: Different control algorithms
- **Production/Setup**: Different operating parameters
- **Implementation**: Use JMP instructions based on mode bits

Error Handling

- **Fault Detection**: Monitor system conditions
- **Recovery Procedures**: Automatic or manual recovery
- **Implementation**: Jump to error handling routines

Data Control

Data Types

Basic Data Types

- **BOOL**: Boolean (true/false)
- **SINT**: Short integer (8-bit, -128 to 127)
- **INT**: Integer (16-bit, -32,768 to 32,767)
- **DINT**: Double integer (32-bit, ±2.1 billion)
- **REAL**: Floating point (32-bit, ±10^±38)

Structured Data Types

- **Arrays**: Multiple elements of same type
- **User-Defined**: Custom data structures
- **Timer**: Predefined timer structure
- **Counter**: Predefined counter structure

Data Manipulation Instructions

Move Instructions

- **MOV**: Copy data from source to destination
- **MVM**: Move with mask (bitwise masking)
- **AND**: Bitwise AND operation
- **OR**: Bitwise OR operation
- **XOR**: Bitwise exclusive OR operation

Math Instructions

- **ADD**: Addition
- **SUB**: Subtraction
- **MUL**: Multiplication
- **DIV**: Division
- **NEG**: Negate
- **SQR**: Square root

Comparison Instructions

- **EQU**: Equal to

- **NEQ**: Not equal to
- **LES**: Less than
- **LEQ**: Less than or equal to
- **GRT**: Greater than
- **GEQ**: Greater than or equal to

Conversion Instructions

- **TOD**: Convert to BCD
- **FRD**: Convert from BCD
- **DEG**: Convert radians to degrees
- **RAD**: Convert degrees to radians

File Operations

File Copy (COP)

- **Function**: Copy data files
- **Applications**: Recipe loading, data backup
- **Parameters**: Source, destination, length

File Fill (FLL)

- **Function**: Fill file with constant value
- **Applications**: Array initialization, clearing data
- **Parameters**: Source value, destination, length

File Search (FSC)

- **Function**: Search for values in data files
- **Applications**: Finding specific data, validation
- **Parameters**: Control, expression, position

Sequencer Instructions

- **SQO**: Sequencer output
- **SQI**: Sequencer input
- **SQL**: Sequencer load
- **Applications**: Machine sequencing, recipe control

Analog I/O Control

Analog Input Processing

Signal Scaling

- **Raw Counts**: ADC conversion result (0-4095 for 12-bit)
- **Engineering Units**: Process values (temperature, pressure, flow)
- **Scaling Formula**: EU = ((Raw - Raw_Min) × (EU_Max - EU_Min)) / (Raw_Max - Raw_Min) + EU_Min

Scale (SCL) Instruction

- **Function**: Convert raw ADC counts to engineering units
- **Parameters**: Input, input minimum, input maximum, output minimum, output maximum
- **Applications**: Temperature, pressure, flow measurement

Input Filtering

- **Digital Filtering**: Software filtering for noise reduction
- **Types**: Moving average, low-pass filter
- **Implementation**: Average multiple samples

Analog Output Control

Output Scaling

- **Engineering Units**: Process control values
- **Raw Counts**: DAC input values
- **Reverse Scaling**: Convert from EU to raw counts

Control Algorithms

Proportional Control

- **Formula**: Output = Kp × Error
- **Characteristics**: Fast response, steady-state error
- **Applications**: Level control, temperature control

Proportional-Integral (PI) Control

- **Formula**: Output = Kp × Error + Ki × ∫Error dt
- **Characteristics**: Eliminates steady-state error

- **Applications**: Most process control loops

PID Control

- **Formula**: Output = Kp × Error + Ki × ∫Error dt + Kd × (dError/dt)
- **Characteristics**: Best dynamic response
- **Applications**: Critical control loops

PID Instruction

- **Auto/Manual**: Operator control capability
- **Setpoint**: Desired process value
- **Process Variable**: Actual measured value
- **Control Variable**: Output to final control element
- **Tuning Parameters**: Kp, Ki, Kd, loop update time

Advanced Analog Control

Cascade Control

- **Primary Loop**: Main process variable control
- **Secondary Loop**: Manipulated variable control
- **Applications**: Temperature control with steam flow

Feedforward Control

- **Concept**: Compensate for measurable disturbances
- **Implementation**: Adjust output based on disturbance measurement
- **Applications**: Load disturbance compensation

Ratio Control

- **Purpose**: Maintain ratio between two flows
- **Implementation**: Master flow controls slave flow ratio
- **Applications**: Blending, combustion air/fuel ratio

Split Range Control

- **Purpose**: Use multiple final control elements
- **Implementation**: Output signal split between elements

- **Applications**: Heating/cooling with separate elements

This completes your comprehensive 8086 microprocessor syllabus notes covering all the topics from evolution of microprocessors through modern architectures, microcontrollers, and PLCs. Each section provides thorough explanations suitable for study and reference.